BACHELORTHESIS
Alex Fuhr

Aufbau und Implementierung einer Infrastruktur für die Kommunikation zwischen einem Lademanagementsystem und einem Elektrofahrzeug am Energie-Campus Bergedorf.

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

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

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Alex Fuhr

Aufbau und Implementierung einer Infrastruktur für die Kommunikation zwischen einem Lademanagementsystem und einem Elektrofahrzeug am Energie-Campus Bergedorf.

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Regenerative Energiesysteme und Energiemanagement*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Hans Schäfers
Zweitgutachter: Prof. Dr. Pawel Buczek

Eingereicht am: 06. Dezember 2021

Alex Fuhr

Thema der Arbeit

Aufbau und Implementierung einer Infrastruktur für die Kommunikation zwischen einem Lademanagementsystem und einem Elektrofahrzeug am Energie-Campus Bergedorf.

Stichworte

Elektrofahrzeug, Kommunikation, Ladestatus, Lademanagement, Mikrocontroller

Kurzzusammenfassung

In dieser Arbeit soll ein Architekturansatz für das bestehende Lademanagementsystem am Technologiezentrum der HAW entwickelt und implementiert werden. Dieses soll die Übertragung der Ladestatusinformationen eines batterieelektrischen Fahrzeugs ermöglichen. Dadurch soll die gezielte Berechnung des Ladeplans für das Fahrzeug gewährleistet werden.

Alex Fuhr

Title of Thesis

Setup and implementation of an infrastructure for communication between a charging management system and an electric vehicle on the Energy Campus Bergedorf

Keywords

Electric vehicle, communication, charging status, charging management, Microcontroller

Abstract

In this work, an architectural approach for the existing charging management system at the HAW Technology Center shall be developed and implemented. This shall enable the transmission of the charging status information of a battery operated electrical vehicle. That is to ensure the targeted calculation of the charging plan for the vehicle.

Danksagung

An dieser Stelle möchte ich mich bei einer Vielzahl von Personen bedanken. Ohne sie wäre die Erstellung dieser Arbeit viel mühsamer gewesen. In erster Linie möchte ich mich bei Herrn Professor Dr. Hans Schäfers, Herrn Simon Decher, Frau Kaja Aniol und Herrn Sebastian Farrenkopf für die Unterstützung und Betreuung bei der Bachelorarbeit bedanken.

Ich danke vor allem meiner Freundin Jessica Laroche für die unermüdliche Begleitung bei der Korrektur dieser Arbeit. Mein Dank geht ebenfalls an meine Freunde Frau Kristina Schuller und Herrn Kristian Morkel für die formalen Verbesserungen und Berichtigungen meiner Bachelorarbeit.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

SPI	Serial Peripheral Interface	v
MCU	Mikrocontroller	viii
AP	Access Point	ix
GPIO	General Purpose Input/Output	1
BEV	Battery Electrical Vehicle	2
CC4E	Competence Center für Erneuerbare Energien und Energieeffizienz	3
SoC	State of Charge	3
CAN	Controller Area Network	3
Bus	Binary Unit System	3
API	Application Programming Interface	4
App	Applikation Software	4
EMSA	E-Mobility Systems Architecture	7
SGAM	Smart Grid Architecture Model	11
IEEE	Institute of Electrical and Electronics Engineers	10
A	Ampere	12
ISO	International Organization for Standardization	12
UML	Unified Modeling Language	12
IEC	International Electrotechnical Commission	12
ETSI	European Telecommunications Standards Institute,	12
ITU	International Telecommunication Union	12
SAE	Society of Automotive Engineers	12
XML	Extensible Markup Language	13
COSEM	Companion Specification for Energy Metering	12
DLMS	Device Language Message Specification	12
HTTP	Hypertext Transfer Protocol	12
SSL	Secure Socket Layer	12
TCP	Transmission Control Protocol/Internet Protocol	12
IP	Internet Protocol	12

1 Einleitung

Der am schnellsten wachsende Bereich in der Entwicklung von digitalen Elektronikkomponenten ist das Design von seriellen Signalen. Ein gigantischer Anstieg des Bedarfs an Kommunikation zwischen den Modulen, FPGA's (englisch: Field Programmable Gate Array, integrierter Schaltkreis) und Prozessoren herrscht in vielen Bereichen der Unterhaltungs- und Industrieelektronik. Für die Effizienz des Designs und die Markteinführung des Produkts sind die Busse und spezifischen Kommunikationsprotokolle unabdingbar [8].

Die Produkte von Arduino, Raspberry pi und ESP32 sind momentan die meist verbreiteten Hardware- und Softwareprodukte in der Welt der Mikrocontroller. Deren Funktionalitäten reichen von einfachen General Purpose Input/Output (GPIO) Kommunikationen bis hin zu selbstständigen Computern mit integriertem Betriebssystem. Mittlerweile kommunizieren die Geräte miteinander über verschiedene Schnittstellen. Die Kombination der verfügbaren Geräte eröffnet eine neue Dimension an Möglichkeiten [5].

Da die heutigen Möglichkeiten eines kleinen Mikrocontrollers oder -computers fast unbegrenzt sind und die Entwicklung so rasant voranschreitet, werden jedoch die dabei entstehenden Gefahren von der Gesellschaft viel zu spät wahrgenommen [Vgl. 29, S.27ff]. Die Digitalisierung der vergangenen Dekaden, wie das Internet, die zunehmenden Geräte, zunehmende Produktionsautomatisierung sowie die Produktionsvernetzung tragen teilweise dazu bei, dass viele Ressourcen des Planeten in einem rasanten Tempo rücksichtslos abgebaut und benutzt werden [43]. Ein Beispiel dafür ist die Erdöl-Ressource, die für die gewaltigen Mengen an CO_2-Ausstoß in die Atmosphäre und damit für die Klimaerwärmung mitverantwortlich ist. Dabei kann man die entwickelten Technologien umweltbewusster einsetzen. So werden im Bereich der E-Autos die Elektronikkomponenten bereits für das intelligente Laden der E-Autos verwendet [17].

Die Folgen des Anstiegs des CO_2-Gehalts in der Atmosphäre sind in den letzten Jahren spürbarer als je zuvor, daher ist das Ziel der Bundesregierung den CO_2-Ausstoß bis 2050 auf 50 Prozent zu reduzieren. Die Dringlichkeit zur Findung innovativer Lösungswege ist hoch. Das große Ziel der Weltbevölkerung ist es den Übergang von einer fossilen zu einer nachhaltigen Industriegesellschaft zu gewährleisten [7]. Anhand der Daten über den primären Energieverbrauch in Deutschland nach Sektoren, weist alleine der Verkehrssektor 30,6 Prozent (770 tWh) der Primärenergie auf (Abbildung 1.1).

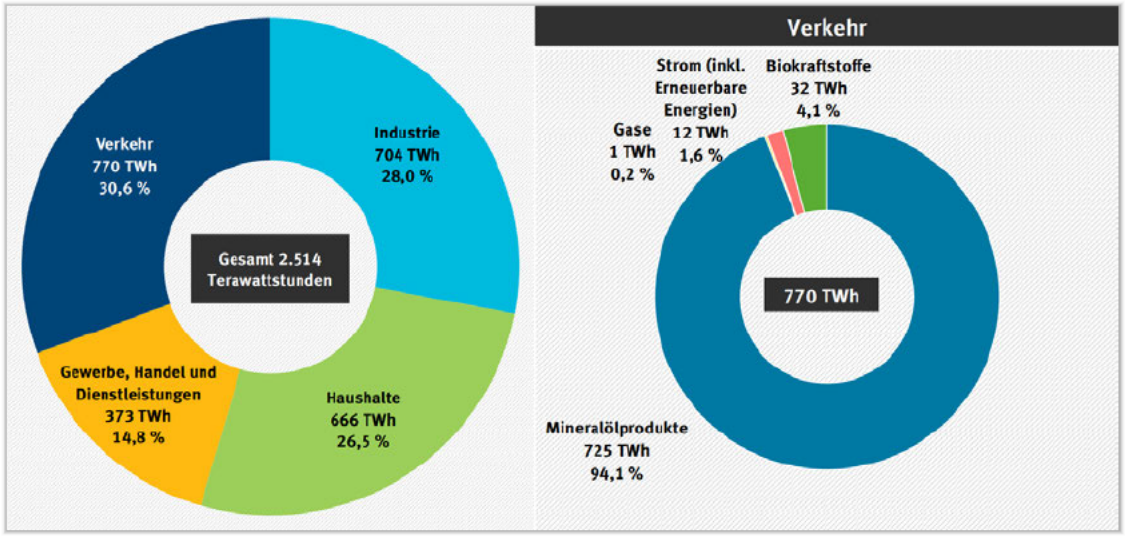


Abbildung 1.1: Links in der Abbildung ist der Energieverbrauch nach Sektoren und rechts ist der Energiesektor „Verkehr", unterteilt nach Energieträgern in Deutschland im Jahr 2019 [41].

Es existiert dementsprechend ein großes Potential für die CO_2-Einsparung. Ein Elektrofahrzeug bietet eine gute Basis zur Nutzung der regenerativen Energien und der Dekarbonisierung in der modernen Welt. Aufgrund ihres steigenden Marktanteils bilden sie eine stark wachsende Verbrauchergruppe, wodurch die Relevanz zur Einbindung der Fahrzeuge in ein Lastenmanagement zunehmend an Bedeutung gewinnt [41]. Durch die langsam verminderten Batteriekosten und gleichzeitig steigender Reichweite der batterieeletrischen Fahrzeuge, auch Battery Electrical Vehicle (BEV) genannt, wächst auch deren Rentabilität [15].

Momentan sind die BEV in der Lage die elektrische Leistung nach Bedarf aus dem Netz zu beziehen und somit das Netz in vielen Fällen zu entlasten. So hat sich nach den ersten Versuchen das Potenzial des gezielten Aufladens der Fahrzeuge gezeigt [27]. Um jedoch bessere Aussagen erzielen zu können, sind weitere langfristige Daten von dem batterieelektrischen Fahrzeug notwendig. Allerdings besteht, aufgrund einiger Änderungen des Herstellers, ein erschwerter Zugang zu den benötigten Daten am Competence Center für Erneuerbare Energien und Energieeffizienz (CC4E), welcher daher für weitere Forschungszwecke überbrückt werden muss.

Eine der Lösungen zur Fahrzeugkommunikation bietet ein Controller Area Network (CAN)-Binary Unit System (Bus)-Shield-Modul, welches das CAN-Bus-Protokoll des Fahrzeugs lesen und verarbeiten kann. Der Mikrocontroller ist in der Lage die ausgelesenen Information des Fahrzeugs an das Netzwerk des Energie-Campus weiter zu leiten. Somit geht es in dieser Bachelorarbeit darum, ein Smart-Grid-Device zu entwickeln, das mit Hilfe eines oder mehreren Mikrocontrollern und eines CAN-Bus-Shields die Fahrzeuginformationen, wie z.B. State of Charge (SoC) an das Lademanagement weiterleitet.

2 Aufgabenstellung und Aufbau der Arbeit

2.1 Aktuelle Fragenstellung

Angelehnt an die Bachelorarbeit von Mathias Röper aus dem Jahr 2018 [27] zeigt sich ein großes Potential an Stromnetzentlastung durch einen geregelten Ladeplan eines batterieelektrischen Fahrzeugs am Energie-Campus in Bergrdorf. Durch die zukünftige Erhöhung des Fuhrparks an BEV am Energie-Campus steigt gleichzeitig das Entlastungspotential der Stromnetze um ein Vielfaches. Um das entstehende Potential nutzen zu können, muss das Lademanagementsystem vor Beginn des Ladevorgangs über die Ladestatusinformation des batterieelektrischen Fahrzeugs verfügen.

Momentan ist diese Möglichkeit zur Abfrage des Ladestatus am Energie-Campus in Bergedorf, aufgrund der ersetzten Application Programming Interface (API), durch eine Applikation Software (App) nicht gegeben. Die Programmierschnittstelle, über die es direkten Zugang zu dem Webportal gab, wurde von dem Hersteller eingestellt. Eine Möglichkeit gäbe es eventuell über die existierende App, die Schnittstelle mit Hilfe von Reverseengineering herauszufinden. Allerdings könnte die Schnittstelle durch weitere Schutzmaßnahmen gesichert sein. Eine weitere Möglichkeit zur Abfrage des Ladestatus über das Ladegerät, fällt aufgrund fehlender Funktionalität der am CC4E vorhandenen Wallbox aus [Vgl. 42]. Aus diesem Grund stellt sich die Frage, ob es eine unkomplizierte und kostengünstige Möglichkeit gibt, die fehlenden Informationen des BEV an das interne Lademanagementsystem zu übertragen. Im Rahmen dieser Bachelorarbeit wird versucht Antworten auf die aktuelle Fragestellung mit Hilfe wissenschaftlicher Methoden zu finden.

2.1.1 Ziel der Arbeit

Aus dem Masterprojekt von Niclas Meyer [Vgl. 28, S.52ff] werden Verbesserungen des Lademanagements am CC4E in Bergedorf vorgeschlagen, zu der u.a. die Automatisierung der Abfrage erwähnt wird. Die Verbesserungen können erst dann realisiert werden, wenn der Datenaustausch mit dem BEV wieder hergestellt ist. Nach Überprüfung weiterer Alternativen zur Kommunikation mit dem Fahrzeug, hat eine der Möglichkeiten die Machtbarkeitsprüfung bestanden und wird als Hauptlösung näher betrachtet. Darauf basiert das klare Ziel dieser Bachelorarbeit und zwar der Aufbau sowie die Implementierung eines, auf Mikrocontroller basierten, Smart-Grid's, welches die Übertragung der Daten (z.B. Ladestatus) von batterieelektrischen Fahrzeugen an das Lademanagement des Technologiezentrums ermöglicht. Diese Daten werden für die Errechnung des optimalen Ladeplans der Fahrzeuge benutzt, bei dem ein größtmöglicher Anteil der erneuerbaren Energien benutzt werden soll. In der Abbildung 2.1 ist das Konzept der Datenübertragung vorgestellt. Der zu entwickelnde Prototyp sieht vorerst die Kommunikation zwischen dem Lademanagement-System und nur einem Verbraucher (einem Elektrofahrzeug) vor. Ein langfristiges Ziel besteht darin, mehrere Verbraucher innerhalb des Technologiezentrums in einem Lastenmanagement anzusprechen und die notwendigen Informationen aller Fahrzeuge strukturell übertragen zu können.

2.1.2 Muss-Anforderungen

Die folgenden Muss-Anforderungen beinhalten das essentielle Gerüst eines Smart-Grids für die gewünschte Funktionalität.

Um die Zeit für die Entwicklung eines neuen Konzepts zu sparen, wird die Hardware in die existierende Abfrage im Lademanagementsystem am Energie-Campus integriert und die Daten an dieses übertragen. Dabei muss die Datenübertragung automatisch, ohne Interaktion mit dem Benutzer, unmittelbar nach der Ankunft stattfinden. Durch die Kombination mehrerer Mikrocontroller-Module soll, mit Hilfe mehrerer Kommunikationsprotokolle die Ladestatusinformation an das Lademanagement weitergegeben werden.

Die beschriebenen Muss-Anforderungen implizieren vorausgesetzte Prozesse:

- Die Kommunikationseinheit (die Hardware) muss den Ladestatus des batterieelektrischen Fahrzeugs erfassen und verarbeiten können.

- Um den Forschungsaufwand zu optimieren und Fehler möglichst zu vermeiden, soll die Verbindung zu dem Lademanagement automatisch aufgebaut werden.
- Aufgrund des fest verankerten Netzwerks am CC4E müssen die Daten automatisch übergeben werden, sobald sich das Fahrzeug am Ladeplatz befindet.
- Die Arbeitsweise des Smart-Grid-Geräts soll möglichst kosten- und energieeffizient gestaltet werden, um den Batteriespeicher des Elektrofahrzeugs nicht zu beeinträchtigen.
- Zur Umsetzung der Kommunikationsinfrastruktur soll das Smart-Grid-System mit entsprechenden Hardware-Komponenten entwickelt und implementiert werden.

2.1.3 Kann-Anforderungen

Durch die Kann-Anforderungen wird das Ergebnis detaillierter und umfassender, für die Grundfunktion sind sie jedoch irrelevant. Der Grad der Realisierung dieser Anforderungen hängt stark vom zeitlichen Aufwand ab. Die Aufzeichnung des Akkustandes des Fahrzeugs während der Fahrt und während des Ladevorgangs (ununterbrochen) wäre ein weiteres Ziel, um den Ladestatus zu jedem Zeitpunkt überwachen zu können. Die Erfüllung dieser Anforderung wird eine Grundlage zur präziseren Aussage über die Batteriekapazität, deren Verluste und Abnutzung schaffen. Die nächste Anforderung bezieht sich auf die Übergabe zusätzlicher Daten des Fahrzeugs für weitere Zwecke, wie z.B. die technische Überwachung und damit die Organisation rechtzeitiger Reparaturen des Fahrzeugs.

In der Abbildung 2.1 ist ein Konzept der Datenübertragung eines batterieelektrischen Fahrzeugs grob dargestellt. Die roten Pfeile stehen für Muss-Anforderungen, grüne Pfeile für Kann-Anforderungen und der grau markierte Bereich stellt das bereits vorhandene Kommunikationssystem dar.

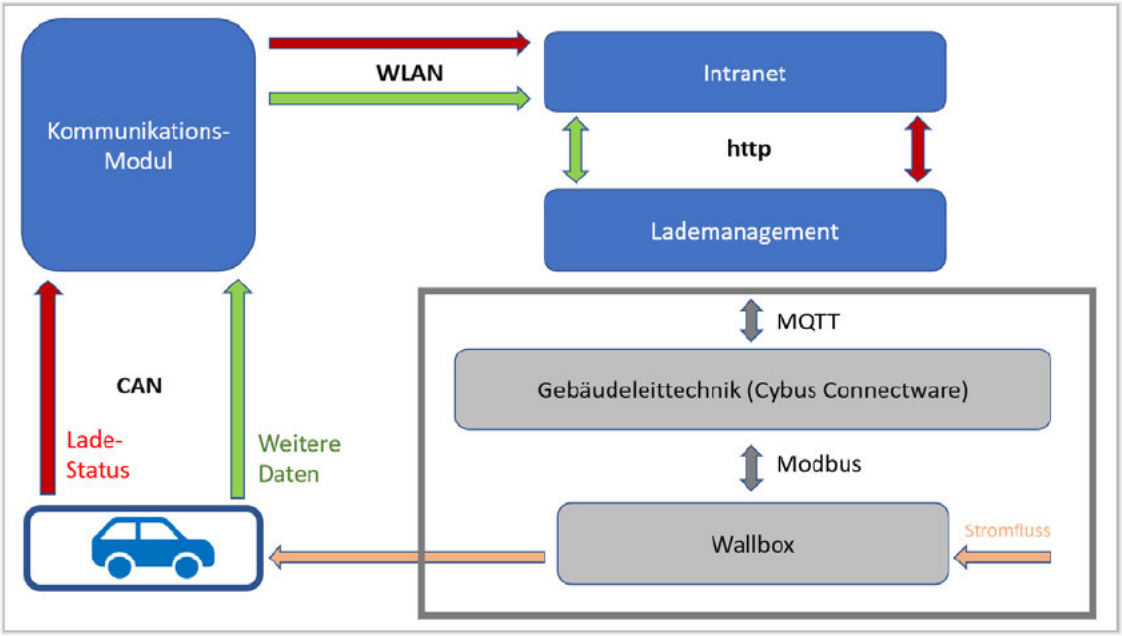


Abbildung 2.1: Konzept der Datenübertragung eines batterieelektrischen Fahrzeugs (angelehnt an [27, S.41ff]).

2.1.4 Resultierende sekundäre Anforderungen

Aufgrund der bevorstehenden Entwicklung einer Architektur im Bereich der Elektromobilität müssen entsprechende wissenschaftliche Werkzeuge für die Konzeptentwicklung angewendet werden. Um dem Standard der Entwicklung gerecht zu werden, wird die Entwicklung des aktuellen Systems nach dem E-Mobility Systems Architecture (EMSA)-Modell vorausgesetzt. Da die Entwicklung sowohl software- als auch hardwaretechnisch umgesetzt werden soll, ist es unverzichtbar hierfür die Systemerstellung nach dem V-Modell zu nutzen, welche speziell für Software- und Systemengineering vorgesehen ist. Nähere Details zur EMSA-Architektur und zum V-Modell sind im Kapitel „Grundlagen" beschrieben.

2.2 Aufbau der Arbeit

In diesem Unterkapitel wird, zum besseren Verständnis des Projektes, der Aufbau der Arbeit erläutert und jeweils eine kurze Einführung der einzelnen Kapitel vorgestellt. Nachdem in Kapitel 1 eine Themeneinführung erfolgte, wurden in Kapitel 2 die Anforderungen und die Ziele der Arbeit für die bestehende Problematik konkreter beschrieben. In Kapitel 3 werden Grundlagen zum besseren Verständnis des Vorgehens erläutert. In Kapitel 4 wird ein Überblick über die technisch relevante Ausstattung an der wissenschaftlichen Einrichtung am CC4E-Labor in Bergedorf erstellt. Kapitel 5 beschreibt anhand der in Kapitel 4 vorgestellten Kriterien, die entwickelte Kommunikationsstruktur zwischen einem Elektrofahrzeug und einem Lademanagementsystem am CC4E-Labor. Dabei werden die Architektur, die Hardware, die Kommunikationsschnittstellen, die Funktion sowie die optimale Datenübertragung nach dem EMSA-Modell verfeinert. In Kapitel 6 wird die Implementierung des Entwurfs aufgezeichnet. Dabei wird die Umsetzung von Soft- und Hardware parallel nach dem V-Modell entwickelt. Das Kapitel 7 beinhaltet die erzielten Ergebnisse der entwickelten Architektur von Software und Hardware und eine kritische Auseinandersetzung mit diesen. In Kapitel 8 erfolgt ein Fazit mit abschließendem Ausblick auf weitere Handlungsbedarfe im Bereich der Kommunikation mit dem Lademanagement.

3 Grundlagen und Stand der Technik

3.1 Energie-Campus CC4E in Bergedorf

„Das Competence Center für Erneuerbare Energien und Energieeffizienz (CC4E) der HAW Hamburg entwickelt nachhaltige Lösungen für die Energieprobleme der Gesellschaft. Dadurch soll die HAW Hamburg zur führenden Hochschule des Nordens für Erneuerbare Energien und Energieeffizienz werden.“ - Prof. Dr. Werner Beba [20].

Das Technologiezentrum Energie-Campus wurde im Jahr 2015 in Hamburg-Bergedorf eröffnet. Es stehen viele innovative Anlagen, u.a. Photovoltaikanlagen, Wärmespeicher, ein Elektrolyseur, ein Blockheizkraftwerk sowie ein batterieelektrisches Fahrzeug, zur Erforschung und technologischen Erprobung der Energiewende, zur Verfügung. Fast alle technischen Komponenten sind intelligent miteinander vernetzt. [Vgl. 19].

Abbildung 3.1: Der Energie-Campus in Curslack vom CC4E [18]

3.2 Wahl der wissenschaftlichen Methoden

Im Rahmen dieser Bachelorarbeit wird zunächst die theoretische wissenschaftliche Methode für das tiefere Verständnis der relevanten Systeme und der Zusammenfassung des aktuellen Forschungsstandes genutzt. Zur Generierung eines neuen Smart-Grid Devices im Bereich der E-Mobilität wird zusätzlich ein EMSA-Modell (E-Mobility Systems Architecture) angewandt, welches im nächsten Kapitel näher beschrieben wird. Dieser erleichtert den Umgang mit der Komplexität und sorgt gleichzeitig für die Interoperabilität des neuen Systems [Vgl. 25, S.6]. Um das System physikalisch zu entwickeln und zu realisieren wurde das spezielle Vorgehensmodell für Software- und Systemengineering, das V-Modell benutzt.

3.3 Interoperabilität

Die Interoperabilität ist im Institute of Electrical and Electronics Engineers (IEEE) definiert als eine Fähigkeit von zwei oder mehreren Systemen oder Komponenten, die Informationen auszutauschen und diese zu nutzen. Dadurch sind interoperable Systeme in der Lage, Dienste anzubieten, diese Dienste untereinander bereitzustellen und zu unterstützen. Die Elemente, wie Kompatibilität und Konnektivität zählen dabei zu den schwächeren Zuständen im Vergleich zur Interoperabilität. Für die Interoperabilität sind gemeinsame Standards notwendig. Die Interoperabilität kann hauptsächlich auf drei Ebenen definiert werden: Technisch (Syntax), informatorisch (Semantik) und organisatorisch (Pragmatik). Wenn ein System an ein zweites, nicht von offenen Standards abhängiges System funktionell angepasst ist, stellt diese Anpassung im Grunde eine Kompatibilität und keine Interoperabilität dar [Vgl. 25, S.5].

3.4 E-Mobility Systems Architecture

EMSA ist eine Anpassung der SGAM-Architektur mit Fokus auf die Elektromobilität. SGAM ist eine Abkürzung für das Smart Grid Architecture Model (SGAM), es umfasst ein Framework für die einheitliche Beschreibung von Systemarchitekturen und eine umfassende Methodik für das Anwendungsfall-Management im Bereich der Smart-Grids. Dazu gehört der Entwurf, die Entwicklung, die Visualisierung, die Validierung und die Analyse des Smart-Grid-Systems auf Interoperabilität. Der Begriff „Smart-Grid" beschreibt die kommunikative Anbindung der Akteure des Energieversorgungssystems an das Energieversorgungsnetz. SGAM ist eine Weiterentwicklung des EU-Mandats „M/490", welche u.a. die „Architekturbeschreibung" und das „Use Case Management" beinhaltet. Die Nutzung von SGAM vereinfacht den Austausch an Informationen zwischen verschiedenen Projekten mit vergleichbaren Anwendungsfällen, jedoch mit unterschiedlicher technischer Umsetzung [40]. Um eine maximale Kompatibilität des EMSA Modells mit SGAM zu erreichen, sind sowohl die Anzahl der Interoperabilitätsschichten (Business, Funktion, Information, Kommunikation, Komponente) als auch die Anzahl der Zonen (Prozess, Feld, Station, Betrieb, Unternehmen, Markt) im EMSA-Modell gleich, wie bei SGAM. Elektromobilität ist im Rahmen der EMSA nicht auf batterieelektrische Fahrzeuge beschränkt, sondern umfasst auch alle anderen elektrisch betriebenen Fahrzeuge, wie Elektrostraßenbahnen oder Wasserstoff-Brennstoffzellen-Lkw. Die EMSA wurde erfolgreich durch das SGAM evaluiert und wird deshalb in dieser Arbeit als verlässliches wissenschaftliches Werkzeug eingesetzt [Vgl. 25, S.25ff].

3.4.1 Dimension: Interoperabilitätsschichten

Das EMSA Modell besteht aus drei Dimensionen (siehe Abbildung 3.2): Zonen, Domänen und den fünf abstrakten und komprimierten Interoperabilitätskategorien, die durch Trennung der Substanzen in der Lage sind die Komplexität zu bewältigen. Auf jeder einzelnen Ebene werden verschiedene Mittel zur Standardisierung und Harmonisierung implementiert, um die Interoperabilität zu gewährleisten und die Wiederverwendbarkeit der Fragmente zu unterstützen. Zusätzlich werden alle relevanten Module entsprechend der EMSA-Diemensionen (Schichten, Domänen, Zonen) eingruppiert [Vgl. 25, S.15].

Die erste Interoperabilitätsschicht heißt **Business-Schicht** (Business Layer) und ist für die Darstellung der regulatorisch-ökonomischen Marktstrukturen und Geschäftsmodelle verantwortlich. Die zweite Interoperabilitätsschicht heißt **Funktionsschicht** (Function

Layer) und beschreibt aus einer architektonischen Sichtweise die Dienste, Funktionen und deren Beziehung untereinander. Abhängig von der Abstraktionsebene können die Funktionen unterschiedlich beschrieben, gruppiert und klassifiziert werden. Die funktionale Architektur kann durch den Einsatz von Unified Modeling Language (UML)-Aktivitäts- oder Sequenzdiagrammen detaillierter dargestellt werden [Vgl. 25, S.15].

Die dritte Interoperabilitätsschicht heißt **Informationsschicht** (Information Layer). Sie ist eng mit dem Communication Layer verbunden und bildet den Informationsfluss zwischen Funktionen, Diensten und Komponenten mit dazugehörigen Objekten ab. Dabei liegt der Schwerpunkt bei drei Aspekten: Integrationskonzepte, notwendige Schnittstellen für den Informationsaustausch und Datenmanagement. Diese Zuordnung hilft, Lücken in der Standardisierung zu erkennen. Es können verschiedene Standards von International Electrotechnical Commission (IEC), International Organization for Standardization (ISO), European Telecommunications Standards Institute, (ETSI), International Telecommunication Union (ITU), IEEE und Society of Automotive Engineers (SAE) den verschiedenen Zonen und Domänen zugeordnet werden. So kann beispielsweise der Informationsaustausch zwischen Fahrzeug und Ladesäule unter dem IEC 15118 oder dem IEC 61851-1 eingeordnet werden [Vgl. 25, S.15ff].

Die vierte Interoperabilitätsschicht ist die **Kommunikationsschicht** (Communication Layer), die für die Darstellung der Technologien und Kommunikationsprotokolle mit den dazugehörigen Datenmodellen und Informationsobjekten zuständig ist. In den höher liegenden Zonen (Betrieb, Unternehmen und Markt, sowie für die Kommunikation mit Endgeräten) kommen oft die Web Services Hypertext Transfer Protocol (HTTP) über Secure Socket Layer (SSL) und Transmission Control Protocol/Internet Protocol (TCP)/Internet Protocol (IP) zum Einsatz. In den Energiemärkten wird die IEC 61968-100 und für allgemeinen Marktplätze die Europäischen Normen (EN) 82325-450/451 abgebildet. Es können domänenspezifische Protokolle im Bereich des Netzmanagements, z.B. IEC 62056 Device Language Message Specification (DLMS)/Companion Specification for Energy Metering (COSEM) eingesetzt werden. Im Bereich der Smart-Meter-Datenaustauschkommunikation wird meist die DIN Norm CLC/prTS 50568-5 verwendet. Sie beinhaltet mehrere Anforderungen für den Anschluss von Stromerzeugungsanlagen über 16 Ampere (A) je Phase an das Niederspannungsverteilungsnetz oder an das Mittelspannungsverteilungsnetz [Vgl. 25, S.16].

Eine wichtige Rolle spielt die ISO 9506 aus der IEC 61850-Normgruppe, die bei der Kommunikation mit übergeordneten Zonen zum Einsatz kommt. Für die Interaktion von einem BEV mit einer Ladesäule wird entweder ein Pulsweitenmodulation (PWM)-Signal nach IEC 61851-1 (SAE J 1772) oder der neuere Kommunikationsstandard ISO/IEC 15118-2 Extensible Markup Language (XML) TCP/IP verwendet [Vgl. 25, S.16ff]. Für die Kommunikation zwischen einer elektrischen Schnittstelle eines Batteriemanagementsystems „CHAdeMO" und dem Elektroauto werden zwei CAN Busse verwendet. Für die Ladesäulen werden generell ETSI TS 101 556 ASN.1 und zukünftige Standards für Open Charge Point Protocol (OCPP)/Open Clearing House Protocol (OCHP) verwendet. Die Standards ISO 11898 CAN-Bus, der frühere Industriestandard FlexRay (jetzt ISO 17458-1 bis 17458-5) oder Ethernet (ISO/Draft International Standard (DIS) 8802/3) werden für den Informationsaustausch der verschiedenen Komponenten innerhalb eines Fahrzeug verwendet [Vgl. 25, S.17].

Die letzte Interoperabilitätsschicht repräsentiert die physikalische Komponente in Smart-Grid und heißt **Komponentenebene** (Component Layer). Zu dieser Schicht gehören Geräte, Anlagen und Systeme, die Netzinfrastruktur, die Ausrüstung des Stromnetzes, Hardware-/Softwarekomponenten für die batterieelektrische Mobilität und alle Arten von Rechenhardware. Diese Schicht bildet die Grundlage für die oberen vier Schichten [Vgl. 25, S.18].

3.4.2 Dimension: Domänen

Die Dimension „Domänen" repräsentiert alle Schritte in der Energieumwandlungskette. Die Definition der Domänen wurde dem Kontext der Elektromobilität besser angepasst. Dazu gehört *Energieumwandlung, Transfer vom/zum BEV, Elektrofahrzeug und BEV-Nutzer-Territorium* (siehe unten mittig in der Abbildung 3.2) [25, S.13]. Die Domäne wird grundsätzlich nach zwei Gruppen unterschieden: die *immobile* Gruppe und die *mobile* Gruppe. Die immobile Gruppe umfasst die Kategorien Energieumwandlung und Energietransfer vom/zum BEV. Die mobile Gruppe umfasst die Kategorien Elektrofahrzeug und Räumlichkeiten des BEV-Nutzers [25, S.13].

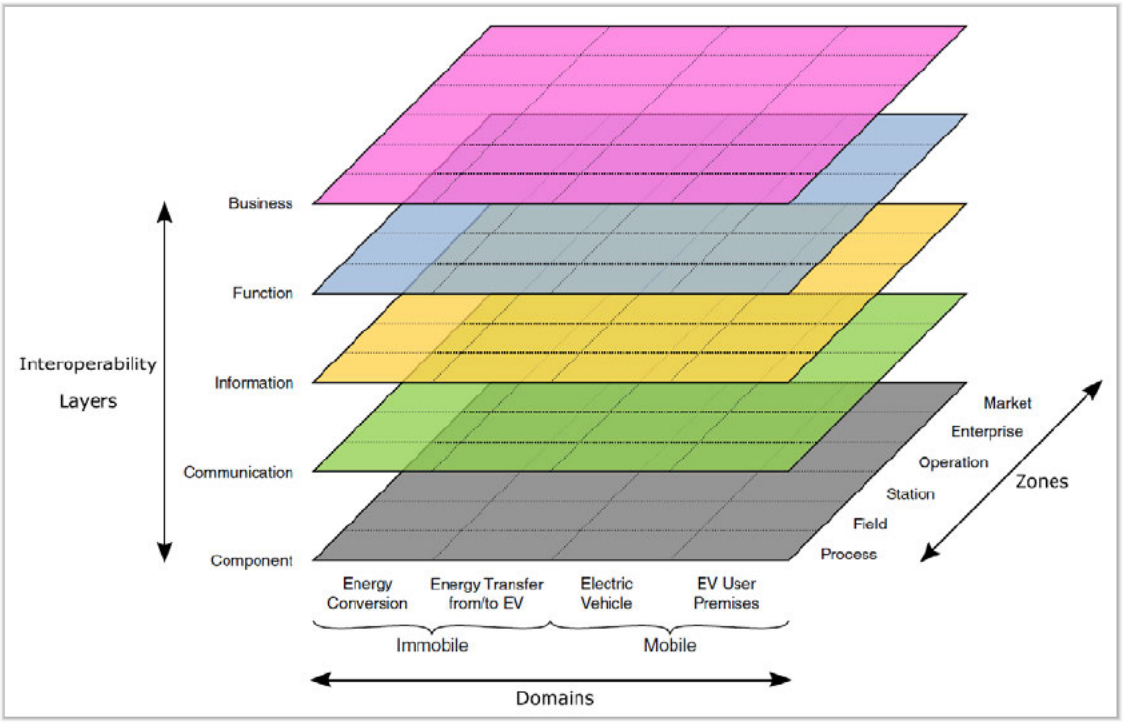


Abbildung 3.2: Visualisierung der Systemarchitektur des Elektromobilitäts (EMSA) Modell, bestehend aus drei Dimensionen: Interoperabilitätsschichten, Zonen und Domänen [25, S.13].

Das Fragment der **Energieumwandlung** beinhaltet Energiequellen und die Energieumwandlungskette. Dazu gehört das Elektrizitätssystem mit allen Ebenen, einschließlich Erzeugung, Übertragungsnetz, Verteilungsnetz und lokale Stromerzeugung, wie Photovoltaikanlagen. Es kann aber auch die Primärenergie, die in elektrische Energie transformierbar ist, dargestellt werden. So im Beispiel von Wasserstoff, der lokal erzeugt oder über ein Rohrleitungssystem transportiert werden kann [25, S.17ff]. Der **Energietransfer vom/zum BEV** beinhaltet die notwendige Infrastruktur für die Übertragung von der Energie zum BEV und umgekehrt. Es können z.B. Oberleitungen für Züge oder Wasserstofftankstellen eingetragen werden. Auch die Lademanagementsysteme und viele weitere Komponenten, die zwischen dem Netz oder Heim und dem Elektroauto richtungsunabhängig agieren, sind in dieser Kategorie enthalten [Vgl. 25, S.13].

Zu der mobilen Gruppe in der Domänen-Dimension zählen die Komponenten Elektrofahrzeug und BEV-Nutzer-Territorium. Das **Elektrofahrzeug** besteht aus Einheiten, die für die Durchführung des elektrischen Fahrens verantwortlich sind. Beispiele hierfür sind Transportmittel, wie E-Bikes, E-Scooter, E-Autos, E-Busse und E-Schienenfahrzeuge. Zu diesem Bereich gehören ebenfalls alle Komponenten und Systeme, die zum elektronischen Fahrzeug gehören, wie die Batterie, das Batteriemanagementsystem und Überwachungssysteme, sowie das Management von Elektrofahrzeugflotten. Der Sektor **Räumlichkeiten des BEV-Benutzer** umfasst Schnittstellen für die Benutzer. Zu den Schnittstellen gehören Geräte, wie Computer, Radio-Frequency Identification (RFID)-Ladekarten oder mobile Geräte. Zu diesem Sektor gehören auch Schnittstellen, die zur Verwaltung des Elektrofahrzeugs dienen, durch beispielsweise Smartphone-App's, aber auch die Systeme, die für Suche, Reservierung oder Buchung von Ladesäulen und elektrischen Transportmitteln nützlich sind, befinden sich in diesem Bereich. Nicht zuletzt gehören alle Arten von E-Mobilitätsdiensten, intelligente Routenplanung und Navigation für die Benutzer zu diesem Bereich dazu [Vgl. 25, S.13ff].

3.4.3 Dimension: Zonen

Die letzte Dimension „Zonen" stellt die natürliche Hierarchie des E-Mobilitätsmanagements dar. Sie beinhaltet sechs verschiedene Sektoren: *Prozess, Feld, Station, Betrieb, Unternehmen und Markt* (siehe unten rechts in der Abbildung 3.2) [Vgl. 25, S.14]. Dabei wird die Idee der Aggregation und funktionalen Trennung benutzt. Bei der Aggregation kann zwischen räumlicher Aggregation (einzelne Standorte im Feld und in der Station) und Datenaggregation (z.B. Datenkonzentration vom Feld zur Stationszone) unterschieden werden. Die funktionale Trennung ist teilweise durch die räumliche Aggregation implementiert, da lokale Funktionen, wie die Kommunikation in dem Fahrzeug überwiegend in Station- und Feldzonen eingeordnet werden. Das gleiche Prinzip gilt auch für die globalen Funktionen, wie z.B. Monitoring, die in den Zonen Markt, Unternehmen und Betrieb vorhanden sind. Es werden Zonen ähnlich des SGAM-Architekturmodells definiert, allerdings angepasst an den E-Mobilitätssektor [Vgl. 25, S.14].

Der Zonenbereich ***Prozess*** beinhaltet die Umwandlung der chemischen oder physikalischen Energie und den dazugehörigen Informationsfluss in allen beteiligten physischen Ausrüstungen. Es handelt sich dabei um Stromnetzeinheiten, Ladesysteme, Elektrofahrzeuge, Endnutzergeräte oder jede Art von Sensoren und Aktoren, die direkt mit dem E-Mobilitätsprozess verbunden sind. Der Zonenbereich ***Feld*** enthält Geräte zur Überwachung, zum Schutz, zur Steuerung und zur Unterstützung des Prozesses der E-Mobilität. Dazu zählen Schutzrelais in einem Ladesystem, Stromnetz oder im BEV. Auch Messgeräte und alle Arten von intelligenten elektronischen Geräten zur Datenerfassung, Datenverarbeitung und Datennutzung, wie z.B. die RFID-Authentifizierungsmethode werden in diesem Rahmen zu finden sein. Der Zonenbereich ***Station*** ist für die Darstellung der flächenhaften Aggregation für die Feldzone verantwortlich. Zu diesen zählen lokale Sensorsysteme, funktionale Aggregationen oder Datenkonzentrationen. Als Aggregationsstufe kann eine Ladestation mit mehreren Ladesäulen oder ein internes Kommunikations- und Steuerungssystem eines Fahrzeugs dienen (z.B. CAN-Bus, FlexRay oder fahrzeuginternes Ethernet) [Vgl. 25, S.14ff].

Der Zonenbereich ***Betrieb*** beinhaltet Managementeinheiten in der jeweiligen Domäne für die Verarbeitung aggregierter Daten. Dazu gehören BEV-Management-Systeme, lokale oder Netz-Energiemanagementsysteme, Lademanagement-Systeme, Mensch-Maschine-Schnittstellen für Eingaben des Nutzers oder des Datenbereitstellungsdienstes. Der Zonenbereich ***Unternehmen*** stellt kommerzielle und organisatorische Infrastrukturen, Dienstleistungen und Prozesse für Unternehmen (Energiehändler, Dienstleister, Versorgungsunternehmen usw.) dar. Zu diesen zählen Kundenbetreuung, Personalverwaltung, Logistik, Anlagenverwaltung, Rechnungsstellung, Mitarbeiterschulung und Beschaffung. Der Zonenbereich ***Markt*** reflektiert die möglichen Marktaktivitäten entlang der Elektromobilitätskette, wie z.B. Energiehandel, BEV-Sharing, Datenhandelsplattformen, Dienstleistungen von E-Mobilitätsanbietern sowie Ladeservicenetzwerke [Vgl. 25, S.14ff].

Die Zerlegung eines Systems auf diese Weise ermöglicht die Darstellung der Interaktionen zwischen Zonen und Domänen. Es realisiert die Darstellung des aktuellen Stands der Implementierungen im Stromnetz, aber auch die Entwicklung zukünftiger Smart-Grid-Szenarien unter Anwendung der Prinzipien der Universalität, Lokalisierung, Konsistenz, Flexibilität und Interoperabilität [Vgl. 25, S.7].

3.5 V-Modell

Das V-Modell ist ein Vorgehensmodell für Software und System Engineering und definiert den Standard für ein systematisches Vorgehen. Es steigert die Produktivität, Wartung, Kosteneffizienz und damit die Erfolgswahrscheinlichkeit eines Projekts. Dieser Entwicklungsstandard ist bei zivilen und militärischen Bundesprojekten zum Teil verbindlich vorgeschrieben [Vgl. 10, S.2].

Neben verschiedenen Aufgaben, wie die Regelung der Arbeit zwischen verschiedenen Organisationen, Beteiligtenkommunikation, Auftragerstellung, Qualitätsmanagement und Projektführung, gibt es eine Vorgabe für den Vorgang bei der Systemerstellung. Die Systementwicklung beschäftigt sich sowohl mit einem zu erstellenden System als auch mit Systemen, die der Unterstützung dienen. Der Vorgang bildet im Grunde die hierarchische Zerlegung eines Systems in immer kleinere Entitäten, bis diese den Umfang von Modulen erreicht haben, bei denen die Umsetzung machbar wird [Vgl. 10, S.5].

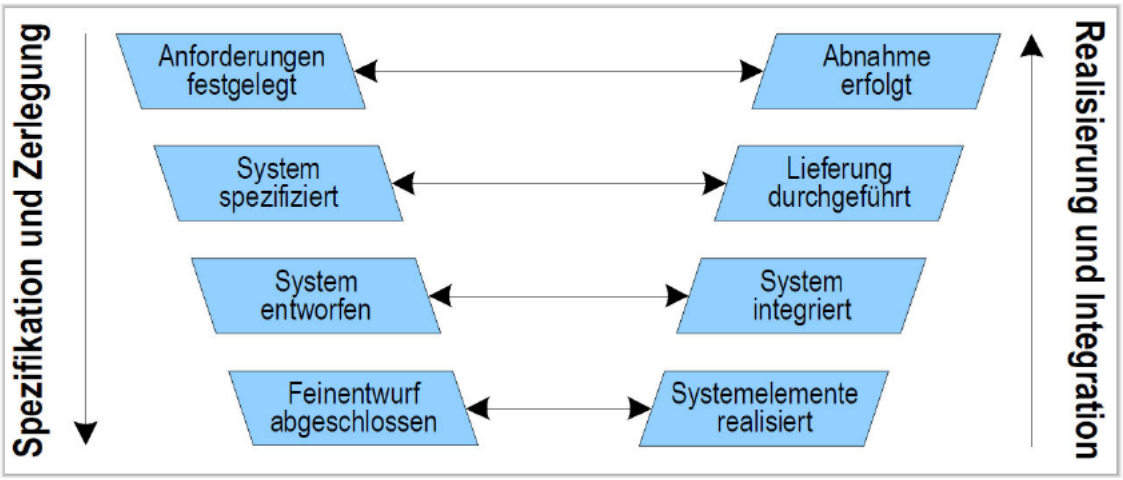


Abbildung 3.3: Vorgehen bei der Systemerstellung nach dem V-Modell [10, S.6]

Im V-Modell sind die Systeme hierarchisch in Segmente, Hardware- und Software Einheiten, Komponenten, Module und externe Einheiten gegliedert. Die Entscheidungspunkte sind wichtige Etappen und stellen die Verfeinerung der Spezifikation und der Zerlegung des Systems dar (Abbildung 3.3). Jeder Zerlegungsschritt ist festgelegt, um die vollständige Umsetzung der Anforderung zu garantieren. Hierbei werden in jedem Schritt die Anforderungen aus dem übergerordneten Systemobjekt entnommen, die Aufteilung entworfen, die Umsetzung des Systemobjektes spezifiziert und anschließend die funktionellen Forderungen der nächsten Ebene der Systemobjekte einsortiert. Die Umsetzung

ins reale System geschieht in umgekehrter Reihenfolge im Vergleich zur Spezifikation und Zerlegung. Die Validierung und Verifikation des Systems findet dabei auf jeder Konstruktionsstufe statt [Vgl. 10, S.5].

3.6 SPI-Bus Kommunikation

Das serielle Schnittstellenprotokoll SPI ist ein sehr einfaches, zuverlässiges und universell einsetzbares Protokoll. Es verwendet das First In - First Out (FIFO)-Prinzip zur schnelleren Informationsübertragung und ermöglicht das Senden und Empfangen von Daten zur gleichen Zeit (Vollduplex). Die Vorteile des Protokolls sind seine hohe Übertragungsgeschwindigkeit und wenige Pins, die es für die Kommunikation benötigt. Zu dem Entwurf zählt eine Mikrocontroller-Schnittstelle, die Multi-Master-Konflikterkennung und -Unterbrechung. Die SPI unterstützt acht externe Slaves, vier Übertragungsprotokolle und verschiedene SPI-Sende- und Empfangsregister, die den jeweiligen FIFOs zugeordnet sind. [26]

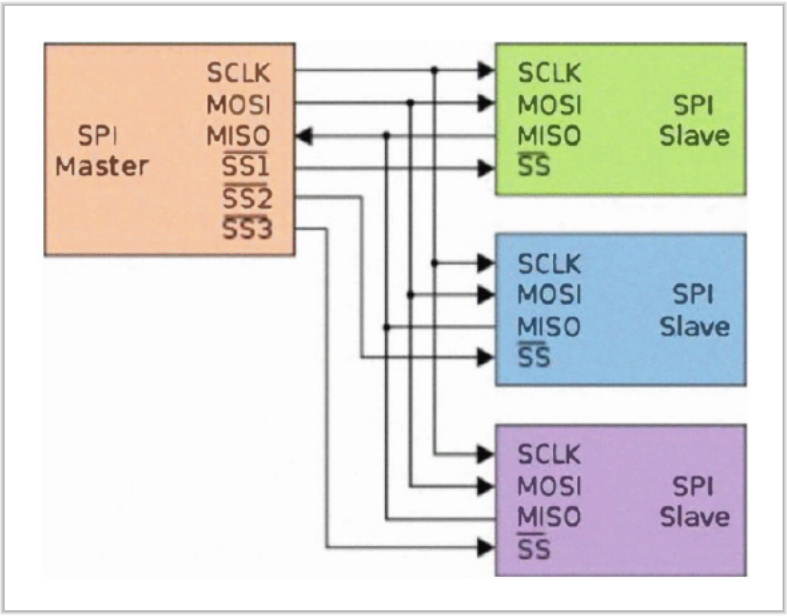


Abbildung 3.4: Kaskadierung mehrerer SPI-Bausteine [16]

SPI-Protokoll

Das SPI besteht aus zwei wichtigen Elementen, dem SPI-Master und SPI-Slave. Im praktischen Einsatz wird mindestens ein Master und einer oder mehrere Slaves benötigt. Zudem werden vier Leitungen für die Kommunikation benötigt: Master Output, Slave Input (MOSI), Master Input, Slave Output (MISO), Serial Clock (SCK), Slave Select (SS)/Chip Select (CS). Der Kommunikationsvorgang ist in der Abbildung 3.4 dargestellt. Die Datenbits an MOSI und MISO werden bei der fallenden Serial Clock (SCLK)-Flanke getoggelt und bei der steigenden SCLK-Flanke abgetastet. Der SPI-Modus legt fest, welche SCLK-Flanke zum Toggeln von Daten und welche SCLK-Flanke zum Abtasten von Daten verwendet wird [26].

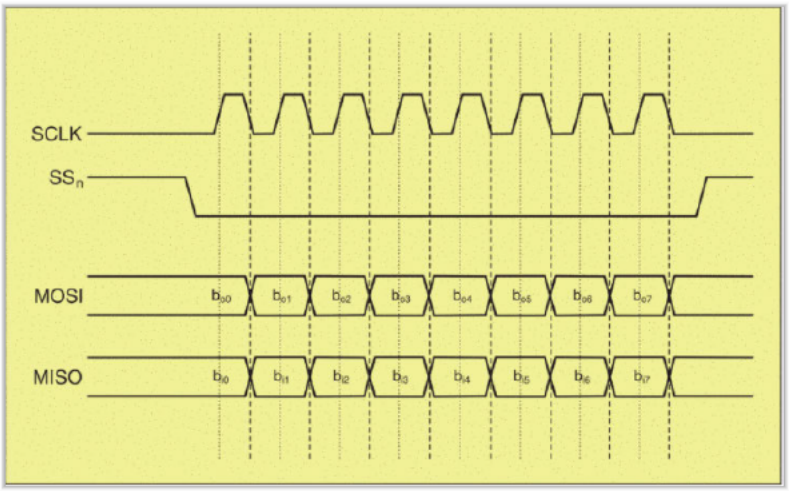


Abbildung 3.5: Einfache SPI-Kommunikation [26].

Zum Senden der Daten setzt der SPI-Master die SS-Leitung auf low, um die gewünschten Slaves auszuwählen. Als Nächstes wird das Taktsignal (SCK) aktiviert, dabei wird die zu übertragende Information auf dem Datenausgangsanschluss MOSI übergeben und die empfangene Information Bit für Bit vom Dateneingangsanschluss MISO abgetastet (Abbildung 3.5) [26].

Übertragungssequenz des SPI-Busses

Die Clock Polarity (CPOL) können high-Pegel oder low-Pegel bei dem SPI-Bus gewählt werden. Wird dem CPOL die 0 zugewiesen, hat das CLK-Signal im Ruhezustand einen niedrigen Pegel. Wird dem CPOL die 1 zugewiesen, verfügt das CLK-Signal im Ruhezustand über einen hohen Pegel (hellblaue Spur in der Abbildung 3.6). Im Übertragungsmodus kann auch die Taktphase mit dem Clock Phase (CPHA)-Wert geändert werden. Wenn CPHA=0 werden die Daten wirksam ab der ersten, und wenn CPHA=1 bei der zweiten steigenden Flanke bei Aktivierung des SS-Pins (pinke Spur in der Abbildung 3.6) [38].

Um die Aktivierungsfehler zu vermeiden, wird in dem CPHA=0 Modus der SS-Pin nach jeder 1-Bit-Übertragung zurückgesetzt. Im CPHA=1 Modus wird der Start der Flanke vom low-Pegel zum high-Pegel definiert [26], aus diesem Grund kann der SS-Pin im low-Pegel-Zustand bleiben und muss nicht zurückgesetzt werden (Abbildung 3.6) [38].

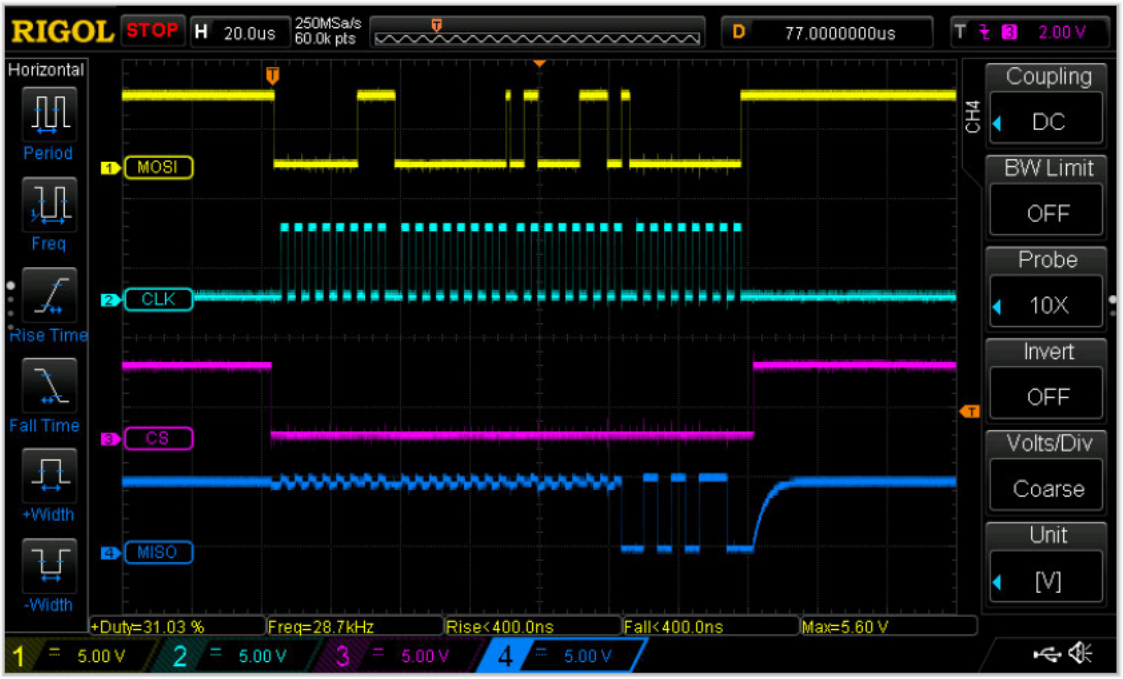


Abbildung 3.6: Simulation der Übertragungssequenz eines SPI-Busses mit gleichzeitiger Sende- und Empfangsfunktion von Daten [38].

3.7 CAN-Bus

3.7.1 CAN-Bus Signal

Das CAN steht für Controller Area Network und ist im Grunde ein Datenbus, der seit dem Jahr 2008 als einzig erlaubtes Diagnoseprotokoll für alle Fahrzeuge mit einer neuen Typzulassung vorgeschrieben ist. Das CAN basiert auf einem Zweileitersystem mit Differenzsignal, welche untereinander gegen die Störanfälligkeit verdrillt und mit „CAN Low“ und „CAN High“ benannt sind. Zu der Besonderheit zählt der entgegengesetzte Signalpegel mit einem Offset von 1,5 Volt (Volt). Während an einer Leitung eine hohe Spannung liegt, wird gleichzeitig eine niedrige Spannung an die andere Leitung gesetzt. Die „CAN Low“ Spannung liegt zwischen ca. 1,5 Volt und 2,5 V und bei „CAN High“ wechselt die Spannung zwischen ca. 2,5 V und 3,5 V (Abbildung 3.7) [35]. Die Summe aus beiden Spannungen ergibt stets 5 V. Das Differenzsignal der beiden Leitungen (CAN High minus CAN Low) wechselt zwischen 0 V und ca. 2,0 V. Durch die Verdrillung der beiden Leitungen wirken die elektromagnetischen Störungen stets auf beide Leitungen in gleichem Maße, so dass sich die Wirkung aufhebt [35].

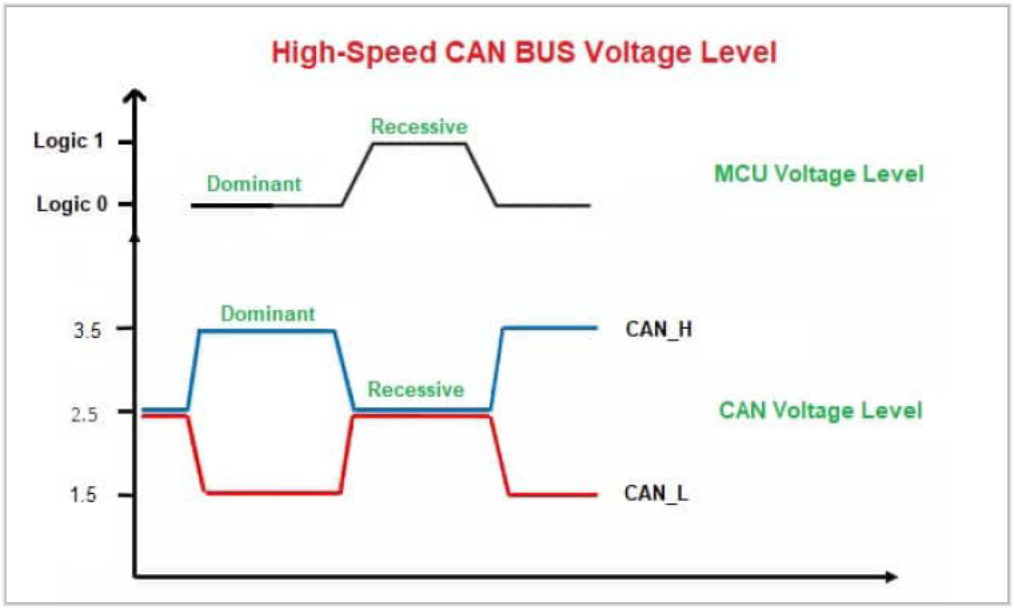


Abbildung 3.7: CAN-Bus Spannungsebene und oberhalb ist der dazugehörige logische MCU-Zustand (rezessiv oder dominant) dargestellt [35].

Im Falle eines Ruhezustandes führen beide Leitungen ca. 2,5 V, wobei das Differenzsignal der Leitungen 0 V ist. Dieser Signalzustand wird als *rezessives Signal* bezeichnet. In diesem Zustand kann jedes Gerät am Bus den Signalpegel beider Leitungen ändern und die Daten transportieren. Die Phase mit dem Signal „CAN High“ bei einer Differenz von 2 V wird als *dominanter Zustand* bezeichnet [14].

3.7.2 CAN-Bus Telegramm

Die Abbildung 3.8 zeigt das komplette CAN-Bus-Telegramm. In dem unteren Teil der Abbildung sind die Spannungsebenen und der daraus resultierende dominante oder rezessive Zustand abgebildet. In der oberen Hälfte der Abbildung sind die einzelnen Bits in Bereiche fest aufgeteilt, die durch die Bitzahl-Angabe unterstützt werden. Nach dem Start-Bit beginnt die Phase der Arbitrierung, in der alle CAN-Bus Teilnehmer die Wichtigkeit ihrer Nachricht miteinander vergleichen. Der Sender mit der wichtigsten Nachricht startet den Sendevorgang. Die Voraussetzung hierfür ist die festgelegte und nur einmalig vorkommende Zuweisung aller Nachrichten auf dem CAN-Bus Netzwerk [35]. Als Nächstes folgen einige Informationen zu der Nachrichten-Identity (ID) und Größe, der zu erwartenden Daten. Der wichtigste Bereich ist in der Abbildung 3.8 rot markiert und stellt den Datenbereich dar, der von Sensoren gesendeten Informationen, die maximal 64 Bit oder 8 Byte groß sein kann. Anschließend folgt im Telegramm die Information zur Überprüfung der Richtigkeit der Nachricht und ein Signal zur Beendigung des aktuellen Telegramms [24].

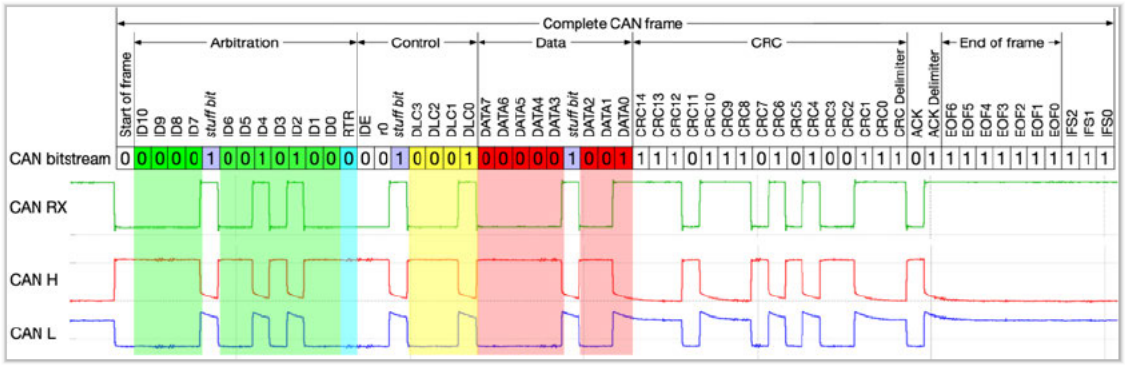


Abbildung 3.8: Komplettes CAN-Daten-Frame, rot markiert sind die Nutzdaten des gesamten Telegramms. [24].

3.8 HTTP

Das Hypertext Transfer Protocol (HTTP, englisch für Hypertext-Übertragungsprotokoll) wurde ursprünglich von der World-Wide-Web-Iniative als Software Prototyp verwendet und ist derzeit als allgemeines Datenprotokoll sehr verbreitet. Das Protokoll basiert auf der TCP/IP-Verbindung, welches den Namen der Domäne oder der IP-Adresse und der Portnummer für den Verbindungsaufbau zwischen dem Client und dem Host benötigt. Ist keine Portnummer angegeben, wird die standardisierte Portnummer 80 angenommen. Die Kommunikationsnachrichten erfolgen in American Standard Code for Information Interchange (ASCII)-Zeichen und bestehen aus Anfragen (Request) vom Client an den Server und Antworten (Response) als Reaktion vom Server zum Client (siehe Abbildung 3.9). Die Antwort eines Servers ist in der Hypertext Markup Language (HTML)-Sprache (Hypertext Markup Language) aufgebaut, die als Bytestrom von ASCII-Zeichen zusammen gefasst werden. Anschließend wird die Nachricht durch das Schließen der Verbindung durch den Server beendet [39].

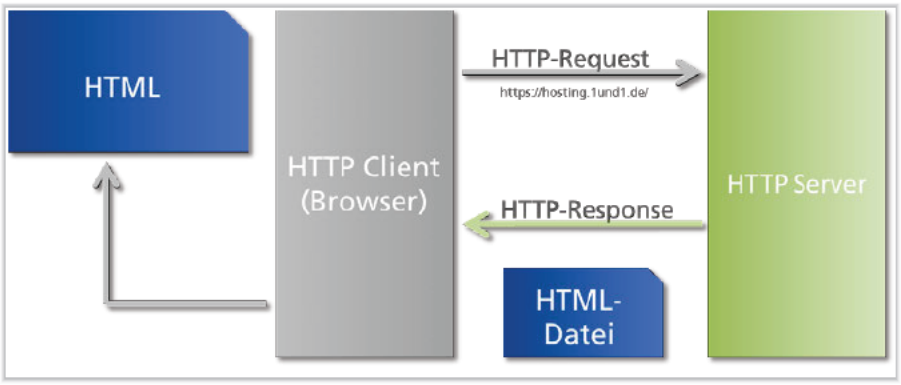


Abbildung 3.9: Kommunikationsprozess eines HTTP-Protokolls [21].

3.9 IEEE 802.11

Immer öfter werden in realen Anwendungsszenarien mit dem wachsenden Bedarf an drahtlosen LANs höhere Datenraten benötigt. Das Wireless Local Area Network (WLAN) basiert auf dem Standard IEEE 802.11 und wird auch als Wireless Fidelity (WiFi) bezeichnet. Die Standards der 802.11-Gruppe, haben mehrere Erweiterungen der ursprünglichen Spezifikation eingeführt. Die bekanntesten Ergänzungen der 802.11-Spezifikation

sind 802.11a, 802.11b, 802.11g und 802.11n [36]. Im Folgenden werden die Hauptunterschiede zwischen verschiedenen IEEE 802.11-Standards erläutert.

Dieser Standard definiert Datenraten von 1 Mbps bis 2 Mbps für drei verschiedene physikalische Schichten. Diese sind Direct Sequence Spread Spectrum (DSSS), Frequency Hopping Spread Spectrum Technik (FHSS) und Infrarot (IR). Dabei ist DSSS eine der am häufigsten verwendeten Technologien, um höhere Datenraten zu ermöglichen [36].

Die Spezifikation „a“ wurde im Jahre 1997 eingeführt. Es beinhaltet ein Orthogonal Frequency Division Multiplexing (OFDM)-Verfahren, welches die Datenrate bis 54 Mbit/s im 5-GHz-Band ermöglicht (siehe Abbildung 3.10) [36].

Product	Spectrum	Maximum physical rate	T_x	Compatible with	Major Disadvantages	Major Advantages
802.11 a	5.0 GHz	54 Mbps	OFDM	None	Smallest range of all 802.11 standard	High bit rate in less-Crowded spectrum
802.11 b	2.4 GHz	11 Mbps	DSSS	802.11	Bit rate too low for many emerging applications	Widel deploy-ed, higher range
802.11 g	2.4 GHz	54 Mbps	OFDM	802.11/ 802.11b	Limited number of collocated WLANs	High bit rate in 2.4 GHz spectrum
802.11 n	5 or 2.4 GHz	600 Mbps	OFDM/ MIMO	802.11a/ b/g	Difficult to Implement	Highest bit rate

Abbildung 3.10: Technische Übersicht der WLAN standardisierten Verbindungen [36].

Die Spezifikation „b“ erweitert den ursprünglichen Standard IEEE 802.11 mit dem Direct Sequence Spread Spectrum (DSSS), das die Datenrate bis zu 11 Mbps im unlizenzierten 2,4-GHz-Spektrum erhöht. Zum Nachteil des IEEE 802.11b-Standards gehören die Störungen, die durch gemeinsam genutzte Frequenzen wie Drahtlostelefonie oder Bluetooth verursacht werden [36].

Die Spezifikation „g“ des IEEE 802.11-Standards wurde im Jahre 2001 eingeführt. Sie erweitert die Datenraten von bis zu 54 Mbps unter Verwendung desselben Frequenzbandes wie bei der „b“-Spezifikation und ist mit dieser kompatibel. Beim Kombinieren der Spezifikationen wird als Folge der Gesamtdurchsatz des Netzwerks minimiert [36].

Die Spezifikation „n“ verfügt über die Möglichkeit mit breiteren Kanälen und verzögerten Bestätigungen zu arbeiten. Die physikalische Schicht von IEEE 802.11n kann in drei verschiedenen Modi agieren. Der „Non-HT“ (Legacy)-Modus unterstützt keine älteren Produkte. Der AP arbeitet mit den alten IEEE 802.11a/b/g Dimensionen, somit sind alle neuen Funktionen deaktiviert. Dieser Modus verwendet eine Kanalbreite von 20MHz [36].

Der „HT Mixed Mode“ ist ein Modus zum Vereinigen der alten IEEE 802.11a/b/g mit den neuen, verbesserten IEEE 802.11n Modi. Dieser Modus erlaubt es den Stationen mit dem AP zu kommunizieren, auch wenn diese nur die Legacy Kommunikation unterstützen. Der „High Throughput Mode“ wird nur bei der MAC-Schicht-Rahmenformat-Übertragung in den APs der neuen IEEE 802.11n-Standards verwendet. Dieses Format ist als „Greenfield“ bekannt. Es sorgt für die Bereitstellung der vollen IEEE 802.11n-Funktionalität, auch bei den APs, die nur IEEE 802.11a/b/g Standards unterstützen [36].

3.10 OBD

On-Board-Diagnose (OBD) bedeutet soviel wie Selbstdiagnose- und Meldefähigkeit eines Fahrzeugs. Es ist ein Standard, mit dessen Hilfe über einen OBD-Anschluss die Sensordaten eines Fahrzeugs abgefragt werden können. Die am häufigsten verwendeten Sensordaten eines konventionellen Fahrzeugs sind Geschwindigkeit, Kühlmitteltemperatur und Kraftstoffmenge. Die aktuelle Version des Standards heißt OBDII und wird in den meisten neuen Fahrzeugen installiert. Die Analyse der Fahrzeugdaten erweist sich als immer nützlicher für neue intelligente Systeme im Fahrzeug, z.B. für das Navigationssystem. Interessanter Weise ist die Bedeutung der Funktionen mit Fehlererkennung in den letzten Jahren erheblich gestiegen [1].

3.11 OBD PIDs

Function		REQUEST (e.g.: from ScanGauge)			RESPONSE (from battery)			
ScanGauge	Name	ID	Len	Data	ID	Len	Data	Units
SOC	SOC	0745h	8	03 22 49 23 55 55 55 55	074Dh	8	05 62 49 23 xx xx 00 00	xx xx [100/2^16 %] (1)
TBV	Voltage	0745h	8	03 22 49 0B 55 55 55 55	074Dh	8	05 62 49 0B xx xx 00 00	xx xx [1/2^17 KV] (2)
TBV	Voltage (alt)	07E1h	8	03 22 49 0B 55 55 55 55	07E8h	8	05 62 49 0B xx xx 00 00	xx xx [1/2^15 KV] (3)
MDV	Module Delta Voltage	0745h	8	03 22 A9 11 55 55 55 55	074Dh	8	04 62 A9 11 xx 00 00 00	xx [50 mV] (4)
BTM	Temperature	0745h	8	03 22 A9 14 55 55 55 55	074Dh	8	04 62 A9 14 xx 00 00 00	xx [C -40] (5)
Tmx, Tmn, Tav, Txc	Module temperatures	0745h	8	03 22 49 11 55 55 55 55	074Dh	8	05 62 49 11 xx xx xx xx	xx [C -40] (5) (6)
MxC (CCL)	Charge Limit	0745h	8	03 22 A9 12 55 55 55 55	074Dh	8	04 62 A9 12 xx 00 00 00	xx [500 mA] (7)
MxD (DCL)	Discharge Limit	0745h	8	03 22 A9 0F 55 55 55 55	074Dh	8	04 62 A9 0F xx 00 00 00	xx [500 mA] (7)

Abbildung 3.11: Übersicht der Beispiel-PIDs und deren Funktionsbeschreibung mit Request- und Responseinformation im hexadezimalen Zahlenformat [13].

OBDII-Parameter-IDs (PIDs) sind in dem ISO 15031-5 Standard definierte Identifizierungsnummern im hexadezimalen Format (Anhang A). Die PIDs von 00 bis 59 sind fest im Standard definiert. Die PIDs im Bereich ab 5A bis 5F sind für SAE/ISO reserviert. Für eventuelle Erweiterungen ist der Bereich ab 60 bis FF vorbehalten. Die PIDs liefern bestimmte Informationen über das Fahrzeug, hierzu gehören Geschwindigkeit, Motordrehzahl, Kraftstoffverbrauch und Fehlercodes [17].

Neben den primären Standardservice der SAE J1979 gibt es weitere Diagnosedienste in der Automobilindustrie. Die Abbildung 3.12 zeigt eine Übersicht dieser. In der folgenden Abbildung 3.11 ein Ausschnitt der Abfrage einiger PIDs eines Hybridfahrzeugs mit Hilfe erweiterter Diagnosedienste dargestellt. Links in der Abbildung ist die Funktion der Werte. Rechts sind die Einheiten und eine eventuelle Berechnung aufgeführt. In der Spalte „REQUEST" sind, nach dem im Standard [9] festgelegten Abfragemuster, ausgewählte Identifikationsnummern „ID", Datenlänge „Len" und sensorspezifische Daten „Data". Es ist das gleiche Datenmuster in der Antwort „ECU-Response-Message" des Steuergeräts, wie bei der Anfrage „ECU-Request-Message" [17].

Function group	Request SID	Response SID	Service	Available in Default Session	Available for RoE	Has Sub-Function
Diagnostic and Communications Management	$10	$50	Diagnostic Session Control	X	-	X
	$11	$51	ECU Reset	X	-	X
	$27	$67	Security Access	X	-	X
	$28	$68	Communication Control	X	-	X
	$3E	$7E	Tester Present	X	-	X
	$83	$C3	Access Timing Parameters	X	-	X
	$84	$C4	Secured Data Transmission	X	-	X
	$85	$C5	Control DTC Settings	X	-	X
	$86	$C6	Response On Event	X	-	X
	$87	$C7	Link Control	X	-	X
Data Transmission	$22	$62	Read Data By Identifier	X	X	-
	$23	$63	Read Memory By Address	X	X	-
	$24	$64	Read Scaling Data By Identifier	X	X	-
	$2A	$6A	Read Data By Identifier Periodic	X	X	-
	$2C	$6C	Dynamically Define Data Identifier	X	X	X
	$2E	$6E	Write Data By Identifier	X	X	X
	$3D	$7D	Write Memory By Address	X	X	X
Stored Data Transmission	$14	$54	Clear Diagnostic Information	X	-	-
	$19	$59	Read DTC Information	X	X	X
Input / Output Control	$2F	$6F	Input Output Control By Identifier	-	X	-
Remote Activation of Routine	$31	$71	Routine Control	X	X	X
Upload / Download	$34	$74	Request Download	-		
	$35	$75	Request Upload	-		
	$36	$76	Transfer Data	-		
	$37	$77	Request Transfer Exit	-		
	$38	$78	Request File Transfer	-		

Abbildung 3.12: Weitere Diagnosedienste der Automobilindustrie [4].

3.12 ECU

Ein Fahrzeug besitzt in der Regel mehrere elektronische Steuergeräte Electronic Controll Unit (ECU)'s, die getrennt für verschiedene Bereiche verantwortlich sind. Die ECU's sind in der Lage die notwendige Information nach Anfrage über den CAN-Bus an einen Diagnosetester zu senden. Dafür werden die im ISO Standard 15031-5 [17] definierten Adressen für Anfrage und Antwort bestimmter ECU's genutzt (Abbildung 3.13).

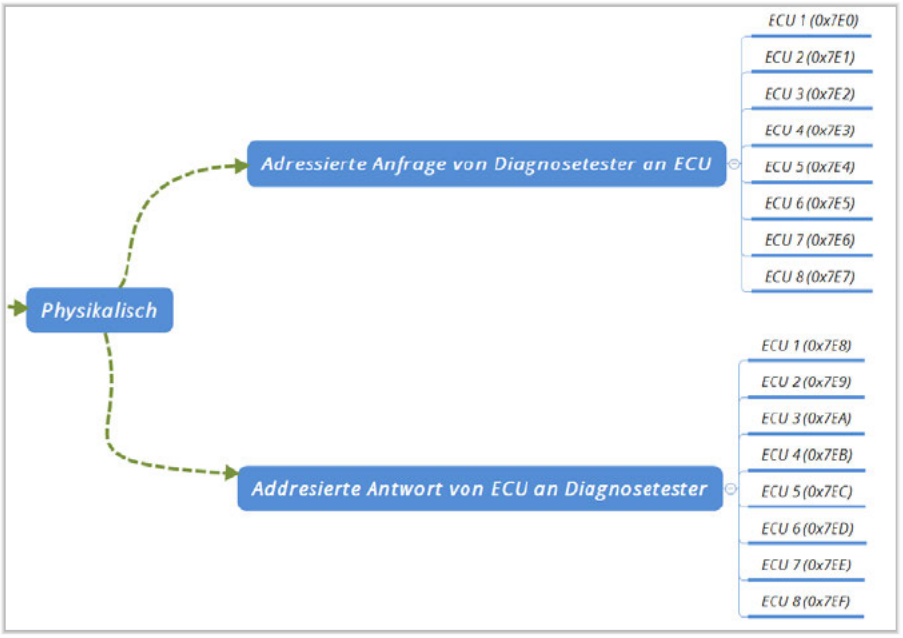


Abbildung 3.13: Physikalische Adressen für Anfrage und Antwort bestimmter Steuergeräte, die im ISO 15031-5 festgelegt sind [17].

3.13 Demand Side Management

Das Demand Side Management (DMS) ist im Grunde eine weitere Bezeichnung für das Lastenmanagement und beinhaltet Maßnahmen der Energieversorger oder der Konsumenten zur Steuerung der elektrischen Energiemenge oder den Zeitpunkt des Energiekonsums. Es ist insbesondere im Zuge der weiteren Marktdurchdringung des volatilen Stroms aus erneuerbaren Energien ein wichtiger Bestandteil eines zukünftigen Energiesystems. Die Implementierung der größeren industriellen Elektizitätskunden hat bereits statt gefunden. Eine indirekte DSM-Maßnahme ist die automatisierte oder manuelle Beeinflussung der Last z.B. aufgrund eines Preissignals seitens der Kunden, wodurch zusätzlich neue Lastspitzen auftreten können [Vgl. 34].

4 Technische Analyse

In diesem Kapitel werden die technisch relevanten Gegebenheiten des Energie-Campus in Bergedorf dargestellt, die zur Lösung des Problems in direktem Zusammenhang beitragen können.

4.1 Wireless Access Point

Ein AP ist eine Hardware, das als Schnittstelle für kabellose Kommunikationsgeräte dient. Das AP-Gerät ist meistens über ein Kabel mit einem definierten Kommunikationsnetz verbunden (rechts in der Abbildung 4.1). Es wird das IEEE-802-LAN Standard zur Kommunikation zwischen authentifizierten und autorisierten Geräten verwendet. Dieser Standard spezifiziert eine gemeinsame Architektur, Protokolle und funktionale Elemente, die eine bidirektionale Authentifizierung zwischen den Clients und Ports in dem selben lokalen Netzwerk ermöglichen. Es ist u.a. eine generelle Methode zur Erbringung einer portbasierten Netzwerkzugangskontrolle, die den Zugang zum Netz regelt und vor der Übertragung und dem Empfang durch nicht identifizierte oder nicht autorisierte Parteien schützt [23].

4.2 Unified Services Router

Am CC4E in Bergedorf stehen mehrere AP's zur Verfügung, um die WLAN-Kommunikation im gesamten Gebäude aufrecht zu erhalten. Eines dieser ist der USR-1000N Router (links in der Abbildung 4.1) der sich momentan neben der Wallbox des batterieelektrischen Fahrzeugs befindet. Dieser Router ermöglicht eine vielseitige Kommunikation, darunter auch über WLAN, dessen Anwendung in dieser Bachelorarbeit eine wichtige Rolle zur Datenübertragung spielt.

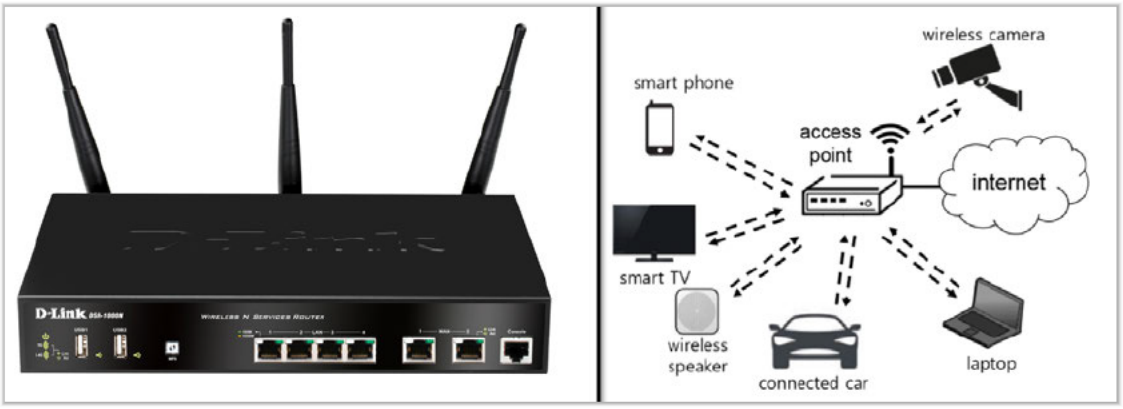


Abbildung 4.1: Links in der Abbildung ist ein Unified Services Router (USR-1000N) von der Firma D-Link [12]. Rechts in der Abbildung ist ein drahtloses lokales Netz AP als Bindeglied für verschiedene Geräte [22].

4.3 Diagnoseschnittstelle OBDII des batterieelektrischen Fahrzeugs

Die genormte Diagnoseschnittstelle „OBD II“ wurde in allen Personenfahrzeugen ab dem Jahr 1996 in Europa eingeführt. Neben der Malfunction Indicator Light (MIL) gehört zu den Regelungen auch die Fähigkeit, über die einheitliche Diagnoseschnittstelle Daten zwischen einem Diagnosetool und dem Fahrzeug zu senden. Standardisiert befindet sich die Diagnoseschnittstelle in dem Fahrgastraum und ist vom Sitz des Fahrers aus leicht erreichbar. Die Buchse muss ohne Hilfsmittel zugängig sein, wobei eine Abdeckung vorhanden sein darf [14].

Eine Besonderheit des OBD II Steckers sind seine vorstehenden Massepins (Pin 4 und 5), wodurch die Masseverbindung des Fahrzeugs beim Verbinden des Steckers als erstes stattfindet. Die Pinbelegung für verschiedene Kommunikationsprotokolle sind paarweise platziert. Das in der Bachelorthesis relevante CAN-Bus Protokoll befindet sich auf dem Pin 6 (CAN High) und auf dem Pin 14 (CAN Low) [14]. Diese Pins werden für die Analyse der Daten des Fahrzeugs in den nächsten Kapiteln dieser Arbeit verwendet.

5 Architektonischer Entwurf nach EMSA

Zur Entwicklung eines Kommunikationssystems im Bereich der Elektromobilität, welches gleichzeitig einen Teil des Energieversorgungssystems darstellt, erfordert es eine, speziell für Smart-Grid's einheitliche Beschreibung von Systemarchitekturen und eine umfassende Methodik für das Anwendungsfall-Management. Dazu gehört der Entwurf, die Entwicklung, die Visualisierung, die Validierung und die Analyse auf Interoperabilität eines Smart-Grid-Systems. Diese Methode trägt den Namen „EMSA" [Vgl. 25, S.7ff].

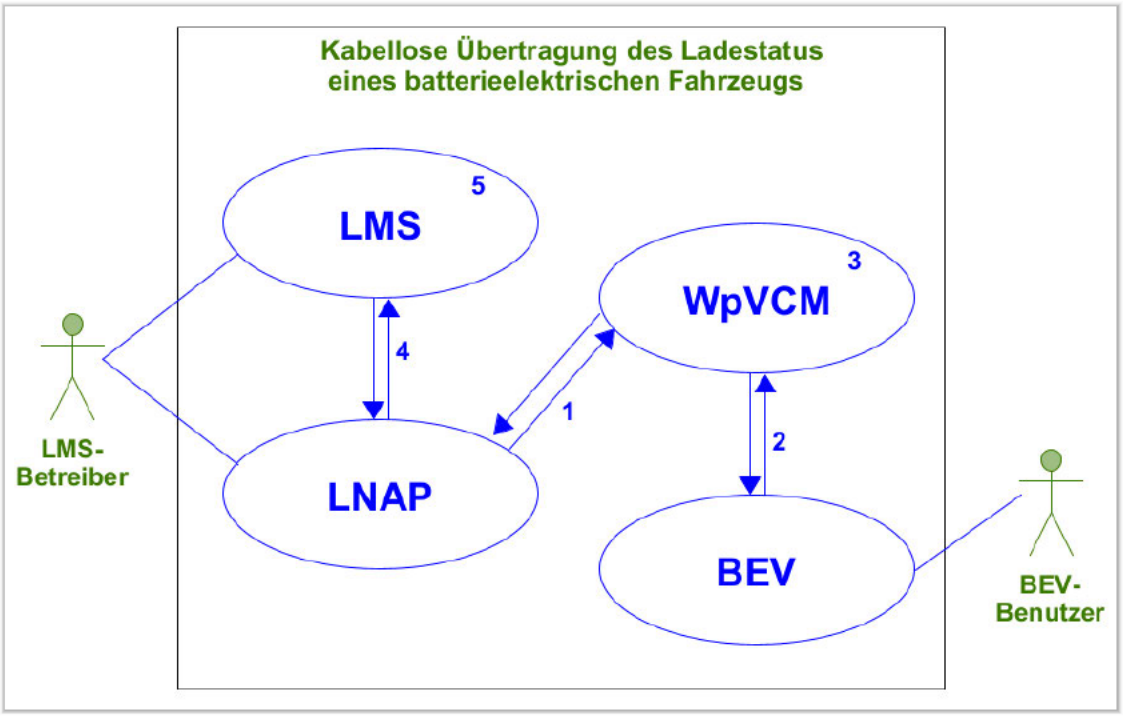


Abbildung 5.1: Use-Case-Diagramm der kabellosen Datenübertragung eines batterieelektrischen Fahrzeugs.

Am Anfang des EMSA-Modells ist das Use-Case-Diagramm (Abbildung 5.1) dafür zuständig, eine grobe Übersicht des Kommunikationssystems in Form einer einfachen Skizze darzustellen. Das System besteht aus vier untereinander kommunizierenden Einheiten, dessen Aufgabe ist es, den Ladestatus eines batterieelektrischen Fahrzeugs für das Lademanagement-System bereit zu stellen. Die Prozessschritte sind wie vom SGAM-Modell vorgesehen, einzeln in einem Sequenzdiagramm (Abbildung 5.2) nummeriert dargestellt. Das BEV stellt die erste Einheit des Systems dar. Es beinhaltet die elektrische Batterie als Speichereinheit und damit auch die Information über den aktuellen Ladestatus. Zum zweiten Element des Systems gehört das WiFi portable Vehicle Communication Module (WpVCM).

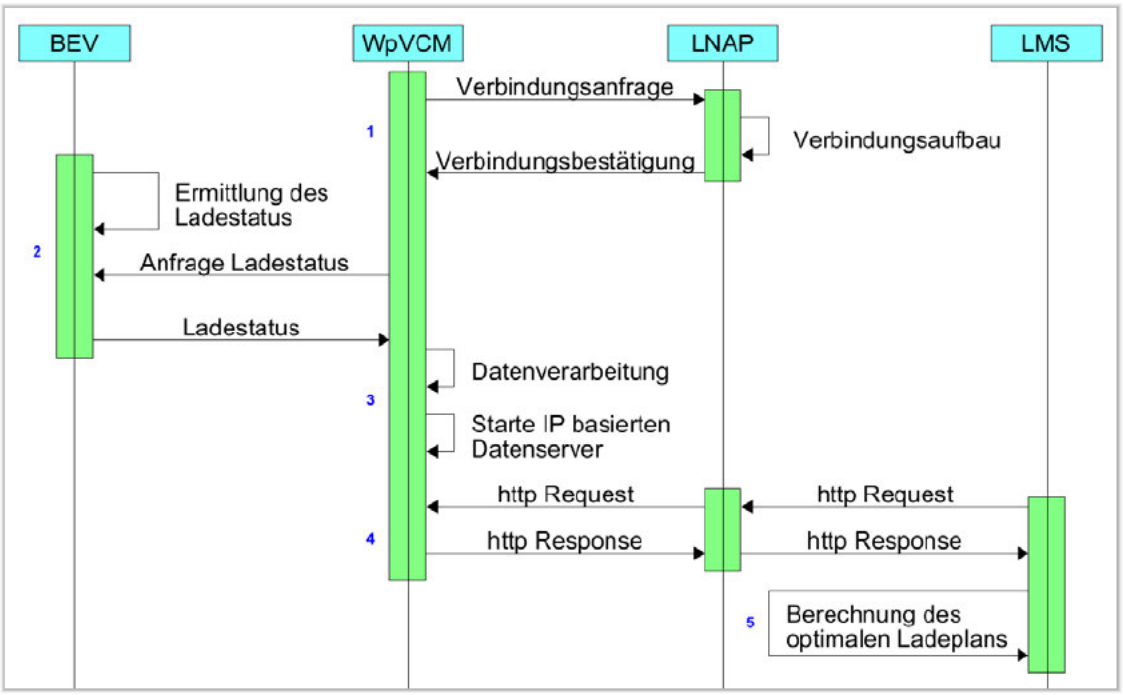


Abbildung 5.2: Sequenzdiagramm des gesamten Kommunikationssystems.

Dieses Element repräsentiert eine Hardware zur Datenverarbeitung und zur kabellosen Informationsübertragung an den Local Network Access Point (LNAP), auch bekannt als lokaler Netzwerk-Knotenpunkt, welcher im System als ein Kommunikationsmedium integriert ist und für die aktuelle Anforderung zur Kommunikation mit dem BEV und der automatischen Datenübertragung unabdingbar ist. Das Load Management System (LMS) ist eine zentrale Steuereinheit für den Aufladeprozess eines BEV und der Empfänger von Ladestatusinformationen. Zur Berechnung des Ladeplans für das aktuelle BEV, be-

nötigt das LMS dessen Ladestatusinformationen. Den signifikanten Vorteil des SGAM bildet die Dimension mit den Interoperabilitätsschichten. Die Aufteilung in Strukturen vereinfacht den Fokus auf einzelne Gebiete des Systems. Die erste Schicht „Komponentenschicht" ist in der Abbildung 5.3 dargestellt. Sie sortiert die Elemente des Systems nach fest definierten Zonen und Domänen. Dieses Beispiel repräsentiert eine Erweiterung (Komponente mit grünem Rand) eines bereits bestehenden Systems (Komponente mit schwarzem Rand).

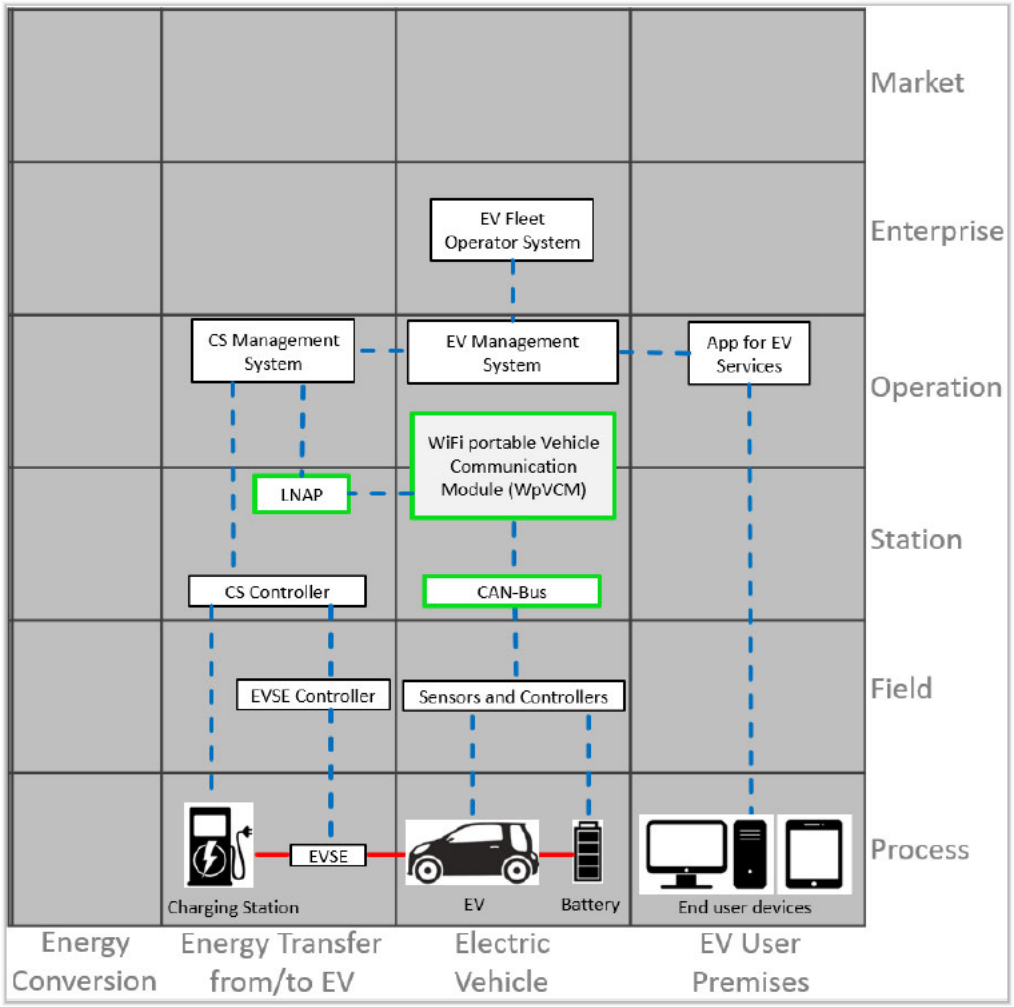


Abbildung 5.3: Die Komponentenschicht des WpVCM-Systems im EMSA-Modell (angelehnt an [25, S.17]).

Die Elemente Ladesäule, BEV und Endnutzergeräte sind in der Abbildung 5.3 in der Prozess-Zone positioniert. Sensoren und Controller, die unmittelbar an die Komponente in der Prozess-Zone zur Überwachung und Steuerung angebunden sind, werden in der Feld-Zone einsortiert. In der Station-Zone sind üblicherweise Sensorsysteme, Datenkonzentrationen oder interne Kommunikations- und Steuerungssysteme eines Fahrzeugs zu finden. Die Darstellung existierender Elemente wird zur korrekten Einordnung des neu entwickelten Systems genutzt. Dadurch werden mögliche Lücken des Entwurfs sichtbar, welche die Umsetzung der Anforderungen behindern könnten.

Aus diesem Grund sind im aktuellen Entwurf die Komponenten Ladekontroller des BEV, der lokale Netzwerkknoten LNAP, das CAN-Bus-Netzwerk des BEV und die WpVCM-Einheit, welche einen Teil des Kommunikationsnetzwerks darstellt, zu finden.

Das WpVCM-System gehört sowohl zur Station-Zone als auch zur Betrieb-Zone, weil es auch die Rolle einer Managementeinheit übernimmt und die Aufgaben der Verarbeitung und Bereitstellung der Daten erledigt. In der Betrieb-Zone sind weitere Elemente des Managements, wie das Lademanagementsystem, das BEV-Managementsystem und die Service-Applikation des BEV zu finden. In der Unternehmen-Zone sind kommerzielle und organisatorische Infrastrukturen, Dienstleistungen und Prozesse angesiedelt. Im aktuellen Beispiel ist in dieser Zone das BEV-Flottenbetreiber-System abgebildet, es übernimmt die Aufgaben der organisatorischen Prozesse zur Koordination des zukünftigen BEV-Fuhrparks.

Die funktionale Interoperabilitätsschicht „Function Layer" (links in der Abbildung 5.4) repräsentiert Funktionen zwischen den Systemen. So erstreckt sich der Prozess der Datenübertragung des neuen Systems über mehrere Komponenten, die sich auf zwei Domänenregionen aufteilen: **Energietransfer vom/zum BEV** und **Elektrofahrzeug**. Dabei finden die Interaktionen innerhalb dreier Zonen statt: *Feld*, *Station* und *Betrieb*.

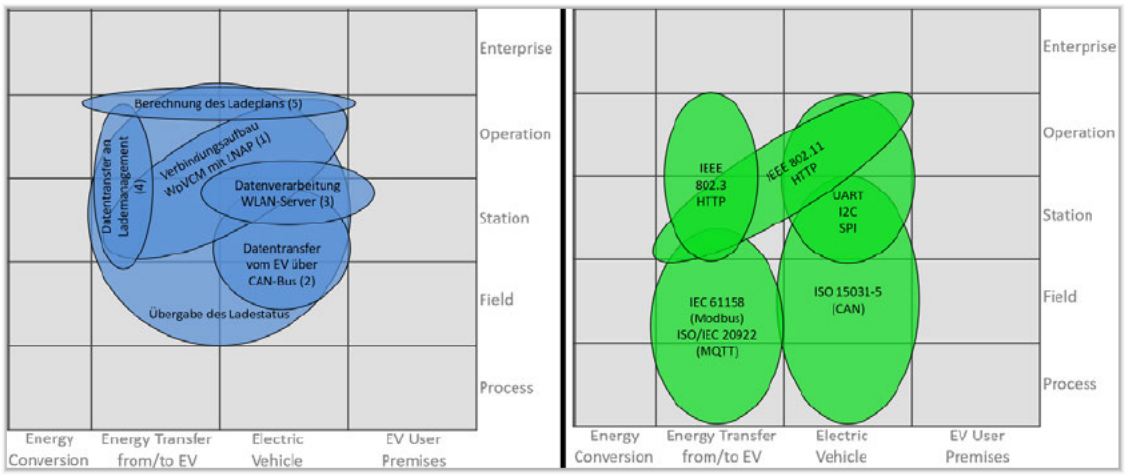


Abbildung 5.4: Links in der Abbildung ist das Function Layer und rechts ist das Communication Layer des aktuellen EMSA-Modells.

Für den Verbindungsaufbau zwischen dem WpVCM und dem LNAP, welcher die Bewältigung der Anforderung zur automatischen Kommunikation darstellt, bedarf es der Verbindung zwischen der **Elektrofahrzeug**-Dömane unter der *Betrieb*-Zone und der Domäne **Energietransfer vom/zum BEV** unter der *Station*-Zone (Funktion Nr.: 1). Die Funktion der Datenabfrage des BEV erstreckt sich in der Domäne **Elektrofahrzeug** von der *Feld*-Zone bis zur *Station*-Zone (Funktion Nr.: 2). Die Verarbeitung und Bereitstellung der Daten sind in der selben Domäne zwischen *Station*-Zone und *Betrieb*-Zone (Funktion Nr.: 3). Die Kommunikation des Lademanagement-Systems mit dem LNAP passiert auf der Domäne **Energietransfer vom/zum BEV** und teilt sich auf die Zone *Station* und *Betrieb* auf (Funktion Nr.: 4). Die Berechnung des Ladeplans (Funktion Nr.: 5) befindet sich in der *Betrieb*-Zone und verbindet die Domänen **Elektrofahrzeug** und **Energietransfer vom/zum BEV**.

Die Interoperabilitätsschicht „Information Layer" definiert die benutzten Informationsstandards des aktuellen Modells. Der erste Informationsstandard gehört zur ISO Familie (ISO 15031) und beschreibt die Informationsstruktur des CAN-Bus-Systems. Dieser Standard deckt drei Zonen in der **Elektrofahrzeug**-Domäne ab: *Feld*, *Station* und *Betrieb*. Zum zweiten Informationsstandard gehört der IEEE 802, welches die Informationsstruktur im lokalen Netzwerk definiert. Dieser Standard verbindet die Zonen *Station* und *Betrieb* der Domäne **Energietransfer vom/zum BEV** und zusätzlich die *Betrieb*-Zone der **Elektrofahrzeug**-Domäne.

Die Interoperabilitätsschicht „Communication Layer" (rechts in der Abbildung 5.4) ist für die Kommunikationsstandards zwischen den Komponenten verantwortlich. Der Kommunikationsprozess beginnt in der Domäne **Elektrofahrzeug** von der *Prozess*- bis zur *Station-Zone* mit dem CAN-Bus-Protokoll (ISO 15031-5). Die Kommunikation innerhalb des WpVCM erfolgt mit Hilfe von UART, I2C und SPI Standards. Diese sind in der selben Domäne unter den Zonen *Station* und *Betrieb* aufgeführt. Für die WLAN-Kommunikation werden die Standards IEEE 802.11 und HTTP benutzt, welche ihren Einsatz zwischen dem WpVCM und dem LNAP haben. Diese Beziehung ist zwischen dem Zonenbereich *Betrieb* in der Domäne **Elektrofahrzeug** und dem Zonenbereich *Station* in der Domäne **Energietransfer vom/zum BEV** dargestellt. Schließlich findet die Kommunikation zwischen dem LNAP und dem Lademamagement-System über ein internes TCP/IP basiertes Netzwerk (IEEE 802.3 und HTTP) statt. Dabei werden die *Station*- und *Betrieb*-Zonen in der Domäne **Energietransfer vom/zum BEV** für diesen Kommunikationsstandard verwendet. Die lückenlose und standardisierte Verbindung sorgt für die korrekte Umsetzung der Anforderungen für die Kommunikation zwischen dem BEV und dem Lademanagement-System.

In der Bachelorarbeit von Herrn Röper [Vgl. 27, S.41] wurden die Kommunikationsstandards, die bisher nicht in einem EMSA-Modell beschrieben wurden, ebenfalls erwähnt, um das Gesamtbild des Systems zu vervollständigen. Hierbei geht es um die Kommunikation zwischen der Ladesäule und dem Lademanagement-System über die Gebäudeleittechnik mit Hilfe der Standards IEC 61158 (Modbus) und ISO/IEC 20922 (MQTT) am TEC. Die Plazierung dieser Kommunikationsstandards findet in den Zonen *Prozess*, *Feld* und *Station* der **Energietransfer vom/zum BEV**-Domäne statt. Die ausführliche Systembeschreibung nach dem EMSA-Modell befindet sich im Anhang dieser Arbeit.

6 Implementierung und Umsetzung

6.1 Mikrocontroller als Kommunikationsschnittstelle

Die Mikrocontroller Arduino, Raspberry pi und ESP32 stellen einen wichtigen Beitrag zur Automatisierung der kleinen Dinge im Alltag dar. Die Besonderheit dieser Mikrocontroller ist die Möglichkeit zur Programmierung und Einstellung nach eigenen Kriterien. Die physikalische Schnittstelle der Mikrocontroller ist revolutionär wichtig, es ermöglicht die mehrfache Kommunikation mit den digitalen und den analogen Komponenten, wie andere Mikrocontroller oder Sensoren. Jedes Bord verfügt über eine oder mehrere Spannungsebenen, entweder 3,3 V oder 5,0 V. An diese Ebenen sind die meisten Endgeräte angepasst und können deshalb barrierefrei mit den Mikrocontrollern bedient werden.

6.2 Umsetzungsstrategie

Unter Berücksichtigung des begrenzten Zeitraums einer Bachelorthesis wurden einige Elemente parallel aufgebaut, um mit möglicherweise auftretenden Problemen zeitgerecht und -effizient umgehen zu können. Es wurden zwei voneinander unabhängige parallele Systeme für die CAN-Bus Kommunikation aufgebaut. Zum ersten System zählt das CAN-Bus-Shield, welches für ein Arduino Uno Board entworfen wurde. Das Arduino UNO Board wurde dementsprechend für diese Variante über die I2C-Schnittstelle implementiert. Zu dem zweiten System zählt das MCP2515 Modul, welches eine SPI-Schnittstelle besitzt und für jedes andere Bord mit SPI-Schnittstelle kompatibel ist. Die Rolle der zentralen Steuereinheit spielt das ESP32-Board, dessen wichtigste Funktion die kabellose Kommunikation mit dem LNAP darstellt. Das ESP32-Board repräsentiert zusätzlich eine Schnittstelle für beide CAN-Bus Module und ist in der Lage die CAN-Bus Daten aus beiden Quellen gleichzeitig zu empfangen.

6.3 Schaltplan

Für die Zeichnung der Steckplatinenschaltung wurde ein kostenfreies Programm namens „Fritzing" benutzt. Wie bereits erwähnt, gehören zum Experiment zwei verschiedene CAN-Bus Geräte, welche unterschiedliche Schnittstellen für die Kommunikation mit Fahrzeug-Steuergerät besitzen. Beide CAN-Bus Module sind auf einer Platine verbaut und untereinander über die CAN-Schnittstelle (grün und orange) und einem Erdschluss-Pin (schwarz) verbunden (Abbildung 6.1). Die Verfügbarkeit von zwei parallelen Systemen kann auch zur Erforschung der besseren Komponente in Hinsicht auf Einfachheit, Funktionalität und sparsamer Arbeitsweise verwendet werden.

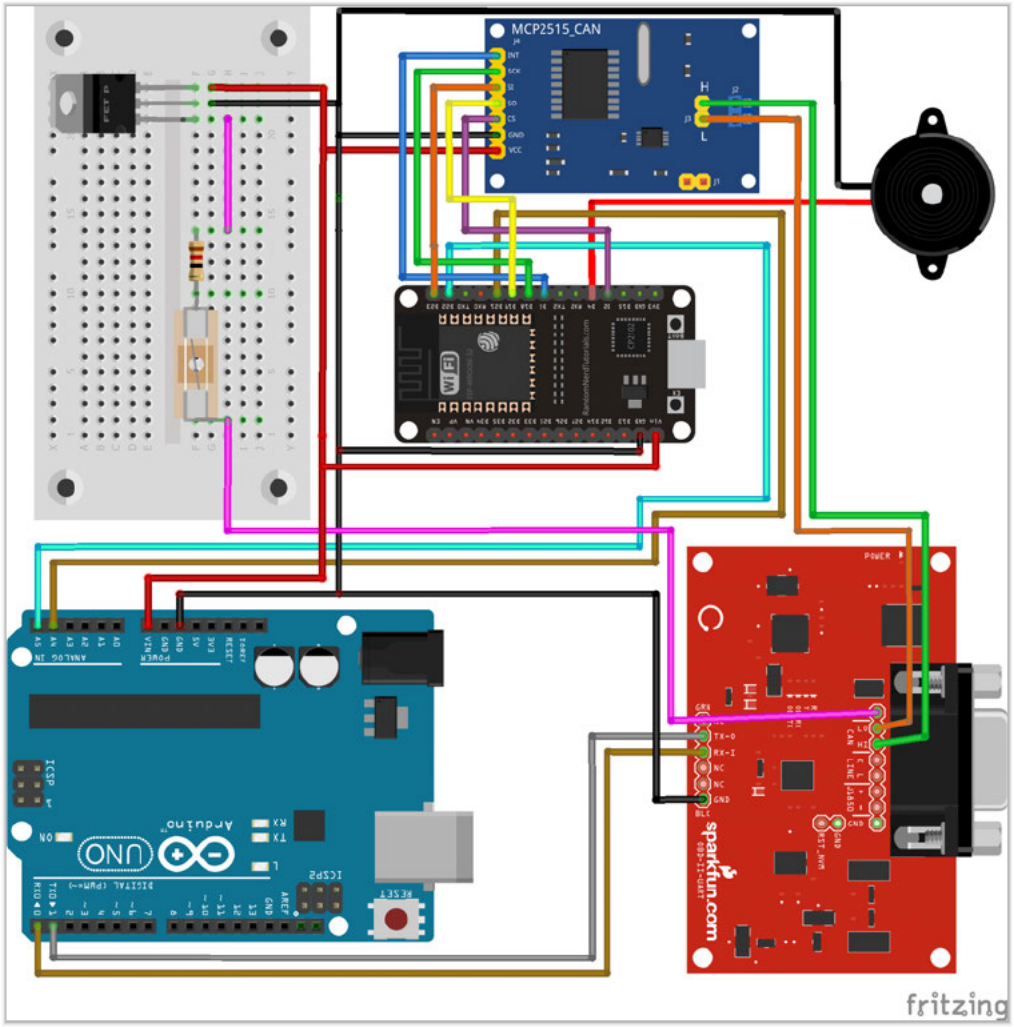


Abbildung 6.1: Steckplatinenschaltung des WpVCM-Systems.

Das CAN-Bus-Shield Modul (unten rechts in der Abbildung 6.1) besitzt zwei gesonderte Pins für den CAN-Bus Datentransfer. Allerdings hat dieses Modul eine weitere Schnittstelle „BD9", die man für die OBDII-Schnittstelle bsp. eines Kraftfahrzeugs benutzt (Abbildung 6.2). Zur Kommunikation mit dem Mikrocontroller benutzt das Modul die serielle Schnittstelle mit einem „Transmitter (TX)" (goldgelb) und einem „Receiver (RX)"-Pin (grau). Das CAN-Bus Shield ist in der Abbildung 6.1 nur symbolisch dargestellt, denn das verwendete Modell gab es bei der Zeichnungssoftware nicht zur Auswahl. Als Abhilfe wird dazu ein ähnliches Modul mit gleichen CAN-Schnittstellen verwendet.

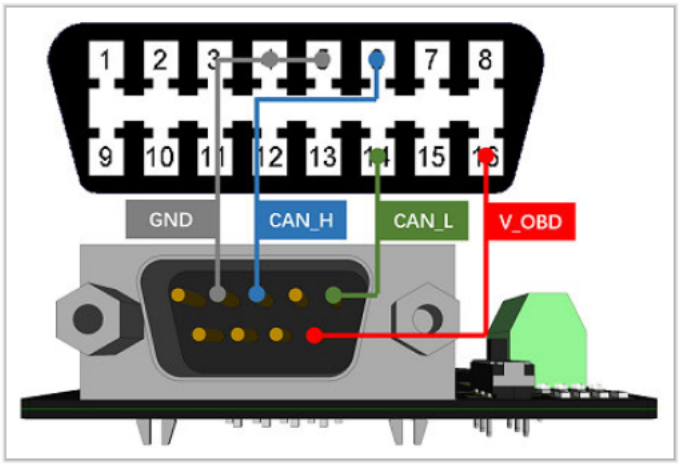


Abbildung 6.2: Verbindungsschnittstelle zwischen dem DB9-Stecker und der OBDII-Buchse [33].

Das MCP 2515 CAN Modul (oben in der Abbildung 6.1) hat nur zwei Anschlusspins für die CAN-Bus Schnittstelle „CAN-Low" und „CAN-High", die direkt an den CAN-Bus angeschlossen werden. Es hat zusätzlich einen manuell zuschaltbaren 120 Ω Widerstand, welcher bei der CAN-Bus Kommunikation zwischen Mikrocontroller Modulen standardmäßig verwendet wird. Für die Kommunikation mit Mikrocontrollern verfügt das Modul über eine SPI-Schnittstelle (blau, grün, orange, gelb, violett) und wird damit an das ESP32-Modul angeschlossen.

Die Information des CAN-Buses muss dekodiert und verarbeitet werden. Diese Funktion übernimmt Arduino UNO R3 Board (unten links in der Abbildung 6.1) für das CAN-Bus-Shield (unten rechts in der Abbildung 6.1). Arduino ist eine Open-Source-Elektronikplattform, die auf einer benutzerfreundlichen Hard- und Software basiert. Arduino-Boards sind in der Lage, Eingaben zu lesen und sie in einen Ausgang umzuwandeln [3]. Diese beiden Elemente sind durch die serielle Schnittstelle miteinander verbunden, wel-

che sich bei dem Arduino UNO R3 auf den digitalen Pins „0" und „1" befindet. Da das Arduino UNO R3 keine WLAN Schnittstelle besitzt, muss eine andere Einheit die Kommunikation mit dem AP übernehmen.

Hier kommt das ESP32-Board zum Einsatz (mittig in der Abbildung 6.1). In dem „ESP32" ist das WLAN-Modul bereits bei der Herstellung integriert und es gibt bereitgestellte Software-Bibliotheken zu deren Implementierung. Das ESP32-Board ist theoretisch in der Lage direkt mit dem CAN-Bus-Shield zu kommunizieren, jedoch würde eine Entwicklung der Software den zeitlichen Rahmen dieser Bachelorarbeit übersteigen. Aus diesem Grund werden fertige Bausteine und deren Kommunikation untereinander benutzt, um bestehende Probleme schnellstmöglich zu bewältigen. Die Kommunikation zwischen ESP32 und dem Arduino UNO ist durch die I2C Bus-Schnittstelle (magenta, goldgelb) realisiert. Hierfür werden die analogen Anschlusspins „4", „5" des Arduino UNO Boards und die Anschlusspins „21", „22" des ESP32 Boards benutzt.

Das parallel verfügbare MCP2515-CAN Modul wurde als eine weitere alternative Möglichkeit für die CAN-Bus Kommunikation vorgesehen. Dieses Modul ist über die SPI-Schnittstelle direkt mit dem ESP32-Board verbunden und wird für weitere Experimente zur Erforschung der optimalen Hardware-Konstellation bereitgestellt. Die Gegenüberstellung der beiden Systeme würde den Rahmen der dieser Arbeit überschreiten, daher wird an dieser Stelle darauf verzichtet.

Das Hardware-System besitzt keine internen Displays, somit ist die Kommunikation mit dem Benutzer, während der alltäglichen Nutzung massiv begrenzt. Es besteht dennoch Bedarf, in einigen Situationen zu wissen, in welchem Modus sich das System momentan befindet. Als Abhilfe wird dazu ein Soundmodul integriert (oben rechts in der Abbildung 6.1), um bestimmte Signale zu erzeugen und somit auf einer akustischen Ebene die Systemzustände darzustellen. Diese Erweiterung ermöglicht die Überprüfung der Funktionalität des Geräts ohne physischen Eingriff in einzelne Systeme.

Alle Boards und Module, die auf der Platine verbaut sind, brauchen eine Spannungsversorgung von 5 V. Die OBDII-Buchse des BEV stellt standardisiert eine 12 V (gemessen ca. 14 V) Spannungsversorgung zur Verfügung. Um die gewünschte 5 V Spannung zu erhalten, wird ein Spannungsregler des Models „L7805CV" eingesetzt (oben links in der Abbildung 6.1). Vor dem Spannungsregler sind eine Sicherung und ein Widerstand in Reihe geschaltet. Diese Anordnung schützt vor den Überströmen bei Fehlern, wie Kurzschlüsse. Die Grenze des maximalen Stroms eines Kraftfahrzeugs beträgt standardmäßig 1,5 A [17]. Um die Stromspitzen, während des Einschaltvorgangs zu vermeiden wurde

ein Widerstand mit 12 Ω und der Leistung 2 Watt (W) in Reihe eingebaut. Die gewählte Anordnung basiert auf den Grundlagen der Elektrotechnik und sorgt für einen stabilen Spannungs- und Strompegel für alle beteiligten Controller. Die Verdrahtung der Komponenten wird zum größten Teil mit einer Streifenrasterplatine realisiert, welche in der Abbildung 6.1 durch eine Steckplatine dargestellt ist. Die detaillierte Beschreibung der Funktionsweisen einzelner Boards und Module ist im Kapitel „Grundlagen" näher dargelegt.

6.4 Energieverbrauch

Das oberste Ziel des Projekts ist es, dem Team des CC4E zu ermöglichen, auf die Daten des batterieelektrischen Fahrzeugs zuzugreifen. Einer der Rahmenbedingungen für die Kommunikations-Hardware ist eine kontinuierlich sparsame Arbeitsweise. Dazu wird im nächsten Schritt der theoretische Energieverbrauch jedes einzelnen Elements kalkuliert. Aus den Datenblättern der Microcontroller werden folgende Daten des Stromverbrauches entnommen:

Der Stromverbrauch des ESP32 Geräts beträgt 205,6 mA bei 5 V [29]. Dagegen verbraucht der Arduino UNO im Normalbetrieb 47,6 mA und im Schlafmodus 32,6 mA [2]. Der Stromverbrauch vom CAN-Bus-Shield V2.0 der Firma SeedStudio teilt sich auf zwei verschiedene Microchips auf. Es hat einen MCP2515 Chip mit 10 mA Verbrauch im Betriebszustand und 5 µA im Schlafmodus [30]. Der zweite Microchip MCP2551 hat zwei Betriebszustände, die jeweils aktiv oder im Schlafmodus betrieben werden können. Im dominanten Modus werden 75 mA verbraucht und im rezessiven Modus nur 10 mA. Im Schlafmodus beträgt der Wert nur 365 µA [31].

Der MCP2562 Chip verbraucht im rezessiven Zustand durchschnittlich 5 mA und maximal 10 mA. Im dominanten Zustand 45 mA durchschnittlich und maximal 70 mA. Im Schlafmodus beträgt der Verbrauch durchschnittlich 5 µA und maximal 15 µA [32]. Zur Berechnung des Energieverbrauchs aller verbauten Mikrocontroller wird mit der Annahme des Zustands mit maximalem Belastungsstrom und maximaler zeitlicher Auslastung (8760 Stunden (h)) vorgegangen. Die folgenden Gleichungen dienen der Berechnung

der gesamten, maximalen Stromaufnahme und damit zusätzlich entstehenden Jahreskosten:

$$I_{ges} = I_{ESP32} + I_{ARD} + I_{CANSH} + I_{MCPM} \tag{6.1}$$

$$= 205,6\ mA + 47,6\ mA + (10 + 75)\ mA + (10 + 70)\ mA\ =\ 418,2\ mA \tag{6.2}$$

$$P_{ges} = I_{ges} * U_{MC} = 418,2\ mA * 5\ V = 2,091\ W \tag{6.3}$$

$$E_J = h_J * P_{ges} = 8760\ h * 2,091\ W = 18317\ Wh = 18,3\ Kilowattstunde(\text{kWh}) \tag{6.4}$$

Als Nächstes folgt die Gegenüberstellung des Verbrauchs des Geräts über eine bestimmte Zeit. Daraus wird die Einwirkung auf die Batterie berechnet und damit die Zweckmäßigkeit überprüft. Laut dem Bundesministerium für Wirtschaft und Energie beträgt der durchschnittliche Strompreis in Deutschland ca. 30 ct/kWh für Haushalte [6]. Dementsprechend wird das gesamte System, unter Vollastbedingungen, jährlich 18,3 kWh an Strom verbrauchen und dem Technologiezentrum ca. 5,50 € kosten.

$$K_G = E_J * K_{kWh} = 18,3\ kWh * 30\ ct/kWh = 5,49\ €\tag{6.5}$$

Das Ergebnis des theoretischen Energieverbrauchs liefert für den unterbrechungsfreien Betrieb plausible Werte und erfüllt damit die Anforderung des kosteneffizienten Betriebs des entwickelten Systems.

6.5 Zusammenbau und Test der Hardware

Die Hardware wurde, wie in der Abbildung 6.1 dargestellt, vor und nach dem Zusammenbau auf einzelne Funktionalitäten überprüft. Die fertige Hardware mit der Bezeichnung der einzelnen Komponenten ist in den Abbildungen 6.3 bis 6.7 dargestellt. Alle Komponenten wurden auf einer Lochrasterplatine zusammengelötet und aus Sicherheitsgründen in einer Plastikbox platziert. Die Konfigurationsschnittstellen vom Arduino und ESP32 Board sind nach außen geführt, um die Programmierung der Mikrocontroller zu vereinfachen, die für die Spannungsversorgung im Betriebsmodus verwendet werden. Über die offene BD9/OBDII-Schnittstelle wird die Kommunikation zu einem Fahrzeug, mit Hilfe eines speziellen Kabels (rechts in der Abbildung 6.4) aufgebaut.

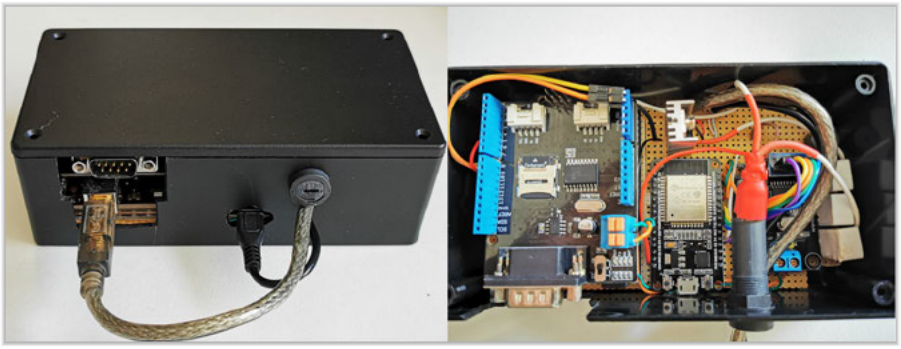


Abbildung 6.3: Links ist die Schutzbox der Hardware und rechts ist die Übersicht der Innenseite.

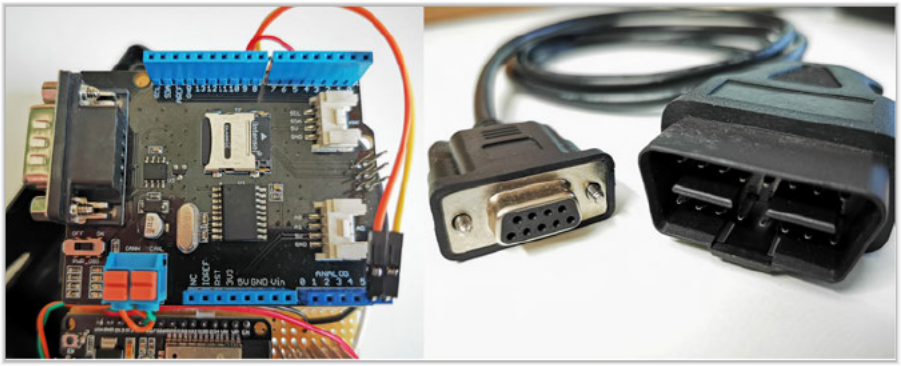


Abbildung 6.4: Links ist das CAN-Shield-Modul inkl. Micro-SD Speicherkarte. Rechts ist das BD9/OBDII Kabel abgebildet.

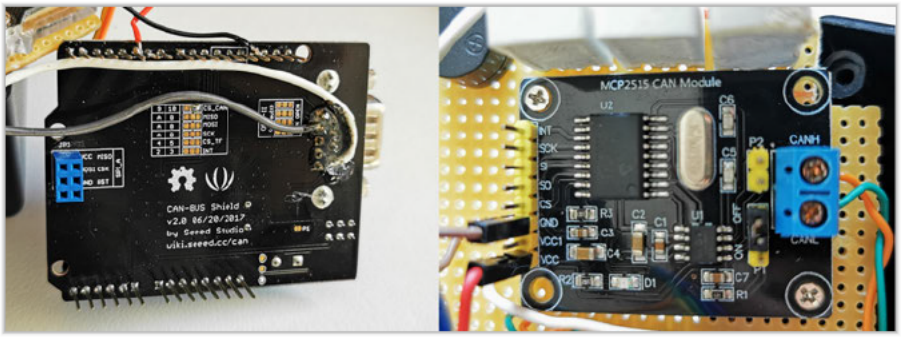


Abbildung 6.5: Links ist die Rückseite des CAN-Bus-Shields. Schwarze und rote Kabel dienen der 5 V Spannungsversorgung, das weiße und graue Kabel sind für die 12 V Spannungsversorgung des OBDII-Steckers verantwortlich. Rechts ist das MCP2515-Modul mit einer CAN-Bus-Schnittstelle.

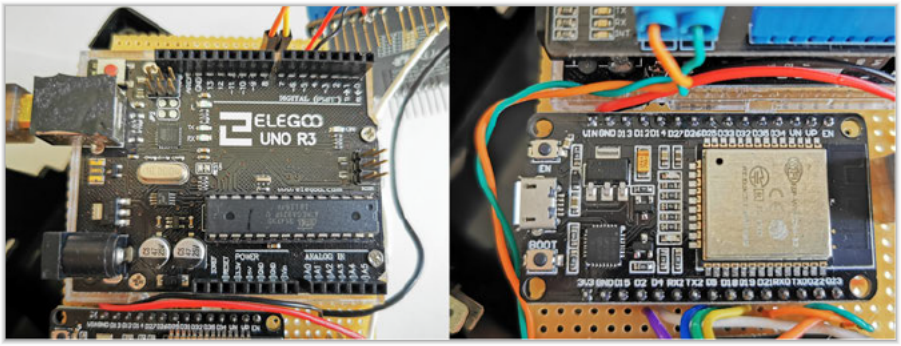


Abbildung 6.6: Links ist das Arduino UNO R3 Board. Rechts ist das ESP32 Board, welches als wichtigstes Kontrollelement in dem System agiert.

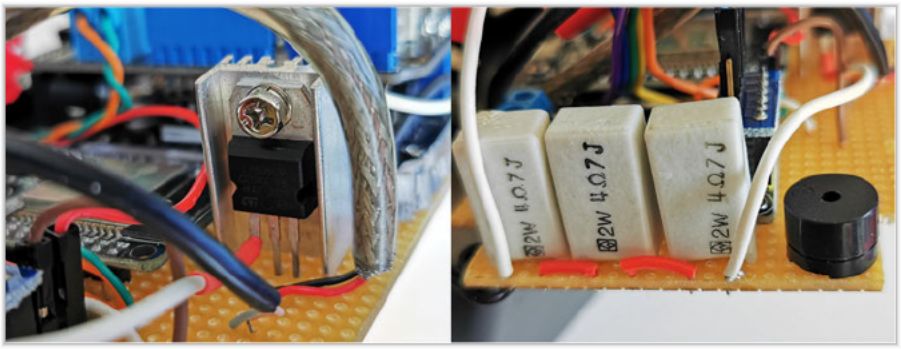


Abbildung 6.7: Links ist ein Spannungsregler L7805CV von 12 V auf 5 V. Rechts sind die drei Stabilisatorwiderstände in Reihe verschaltet und daneben befindet sich ein elektronischer Alarmsummer.

6.6 Software

In diesem Unterkapitel wird die Architektur der Software der drei verschiedenen Systeme dargestellt. Es werden hier nur die Zeilen des Codes präsentiert, die den maßgeblichen Teil zur Funktionalität und zum Verständnis beitragen. Der komplette Programmcode für die Systeme befindet sich im Kapitel „Anhang" dieser Bachelorthesis.

6.6.1 Arduino UNO

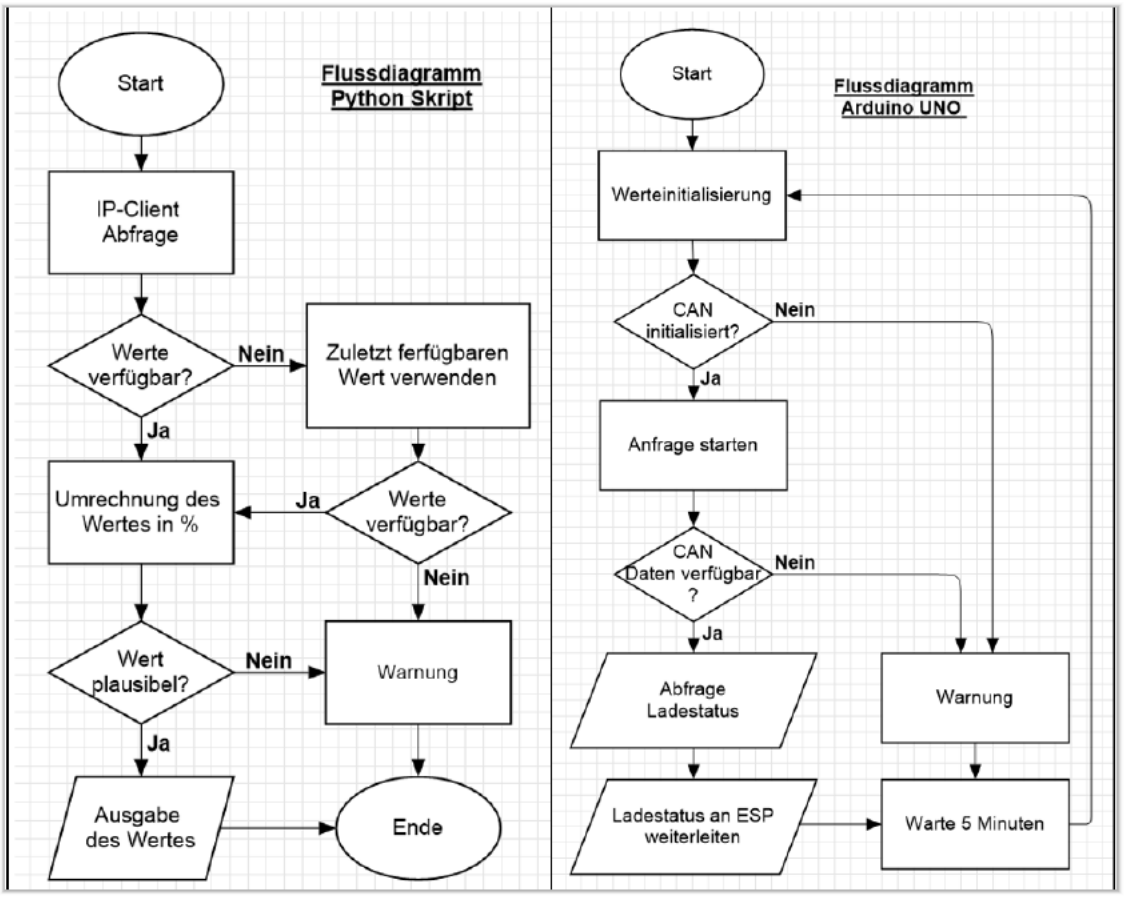


Abbildung 6.8: Das linke Flussdiagramm ist ein in Python-Programmsprache entwickeltes Abfrage-Skript und rechts ist das Flussdiagramm vom Arduino UNO für den Auslesevorgang und die Weitergabe der CAN-Bus Daten eines BEV.

Die Arduino UNO Hardware ist direkt mit dem Fahrzeug verbunden und dadurch für die Kommunikation über das CAN-Bus verantwortlich. Das Übertragen der CAN-Bus-Daten mit I2C an das ESP32 Board ist eine wichtige Funktion des Arduino UNO Boards. Der Aufbau der Software in C-Code für das Arduino UNO Board und das ESP32 Board basieren auf den Beispielen der Arduino Integrated Development Environment (IDE) [3]. Das Flussdiagramm für das Arduino UNO ist rechts in der Abbildung 6.8 dargestellt. Die wichtigsten Schritte zur Initialisierung des Arduino-Boards für die CAN-Bus Abfrage sehen wie folgt aus:

```
        // init can bus : baudrate = 500k
        while (CAN_OK != CAN.begin(CAN_500KBPS, MCP_16MHz)) {
                SERIAL_PORT_MONITOR.println("CAN init fail, retry.")
                delay(100);
        }

        while (CAN_MSGAVAIL == CAN.checkReceive()) {
                    // read data
                    CAN.readMsgBuf(&len, buf);

                                // Buffer for I2C Data:
                                IntHexData[i] = buf[i];
        }
```

```
        void requestEvent() {
          for (int n=0;n<sizeof(IntHexData);n++){
          Wire.write(IntHexData[n]);  /*send string on request */
          //  Send data to ESP32
          }
        }
```

6.6.2 ESP32

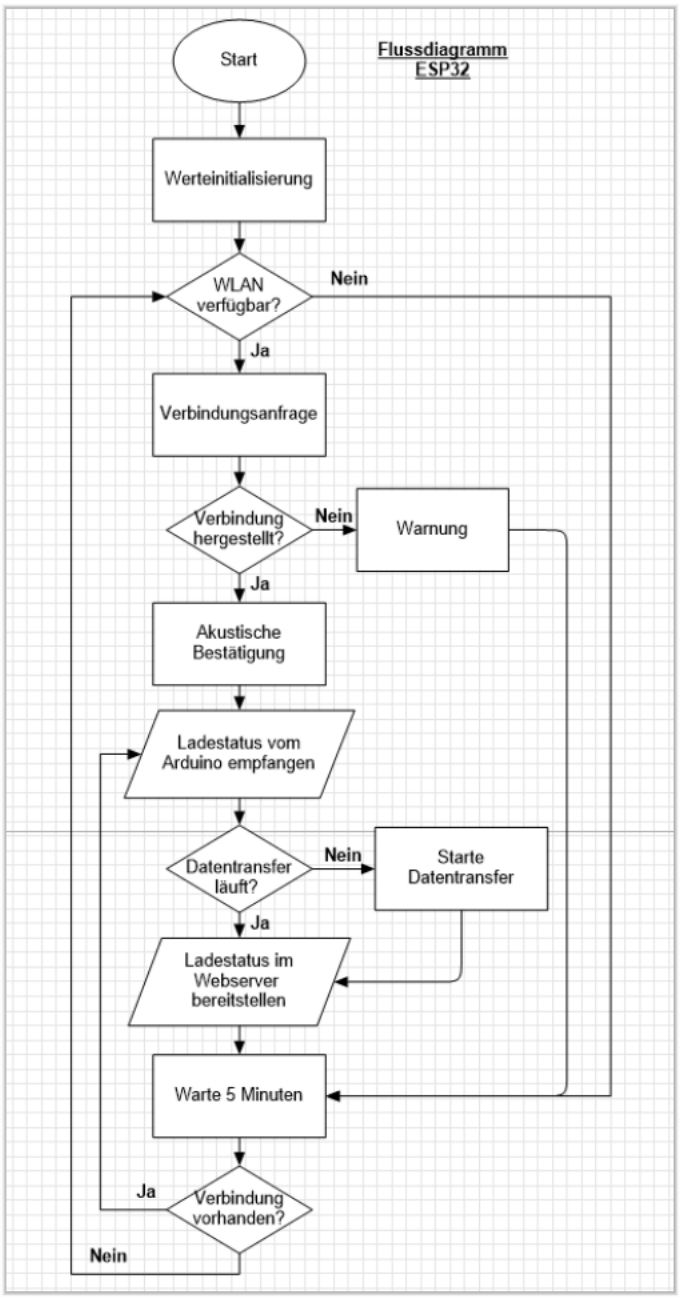


Abbildung 6.9: Flussdiagramm vom ESP32-Board.

Als Vermittler muss das ESP32 Board die Daten des Arduino Boards empfangen und in einem WLAN-Server mit festgelegter IP-Adresse bereitstellen (Flussdiagramm Abbil-

dung 6.9). Das Modul besitzt keinen Display für die Kommunikation mit dem Benutzer, daher wurde ein Buzzer-Modul implementiert, welches bei der WLAN Verbindung ein akustisches Signal erzeugt, um den Debuggen-Prozess zu vereinfachen. Dazu werden die notwendigen Konfigurationen festgelegt und folgend dargestellt:

```
        // AP network credentials:
        const char* ssid     = TEST_AP_NAME;
        const char* password = TEST_AP_PSWD;

        // Set web server port number to 80
        WiFiServer server(80);

        // Variable to store the HTTP request
        String header;

        // Set your Static IP address
        IPAddress local_IP(192, 168, 0, 184);

        // Set your Gateway IP address
        IPAddress gateway(192, 168, 0, 1);
        IPAddress subnet(255, 255, 0, 0);
        IPAddress primaryDNS(8, 8, 8, 8);

        // Configures static IP address
        WiFi.config(local_IP, primaryDNS, gateway, subnet);

        // Listen for incoming clients
        WiFiClient client = server.available();

        // Read I2C Data from Arduino UNO
            int i = 0;
            Wire.requestFrom(8, 21);  // request 21 b from slave device #8
            while (Wire.available()) {// slave may send less than requested
            char c = Wire.read();     // receive a byte as character
            Serial.print(c);          // print the character
            data[i] = c;              // Save recived Data
            i += 1;}

        if (client) {
            while (client.connected()) {  // loop while client's connected
              if (client.available()) {

            // Send Arduino data to WiFi Server
            for (int n=0;n<sizeof(data);n++){
```

```
40              //client.print("Data: ");
41              client.print(data[n], HEX);     // Send data to WiFi-Server
42              }
43
44          // BUZZER____________________
45          int Buzzer = 4; // GPIO Pin
46          pinMode (Buzzer, OUTPUT);
47
48          void connect_wlan ()
49          {
50             for (int i=1;i<delay_short_loop;i++){
51              digitalWrite (Buzzer, HIGH); //turn buzzer on
52              delay(1);
53              digitalWrite (Buzzer, LOW);  //turn buzzer off
54              delay(1);}
55
56            delay(delay_short_loop);
57
58             for (int i=1;i<delay_short_loop;i++){
59              digitalWrite (Buzzer, HIGH); //turn buzzer on
60              delay(1);
61              digitalWrite (Buzzer, LOW);  //turn buzzer off
62              delay(1);}}
63
64          // function call
65          connect_wlan(); // Buzzer connect sound
66
67          }
```

6.6.3 Datenanfrage im Intranet mit Python-Code

Die Ladestatusinformationen werden mit Hilfe eines Web-Servers im Intranet bereitgestellt. Die einfachste Variante der Datenabfrage im LNAP kann mit einem beliebigen Internetbrowser durch Eingabe der bekannten IP-Adresse erfolgen. Dabei wird allerdings eine 11-stellige Hexadezimalzahl zurückgeliefert, welche manuell übersetzt werden muss. Das Flussdiagramm des Python-Skripts ist in der Abbildung 6.8 zu finden. Eine Implementierung der Datenabfrage im Lademanagement-System kann mit diesem Python-Skript durchgeführt werden:

```
# IP-Adresse des Kommunikationgeraets eines BEV
Vehicle_IP = 'http://192.168.0.184'

# Serverdaten werden empfangen
rueckgabewert = request.urlopen(Vehicle_IP)
datensatz = rueckgabewert.read()

# Die PID-Nummer wird in den ersten 3 Stellen kodiert
pid_start = 0
pid_end = 3
pid_number = datensatz[pid_start:pid_end]

# Die Anzahl der Datenbits werden im ersten Byte nach der PID-Nummer
    mitgeliefert
size_start = 4
size_end = 5
data_length= datensatz[size_start:size_end]

# Hier werden nur die Datenbytes uebergeben, um Rohdaten zu erhalten
hexdata = datensatz[size_end:endbyte_of_data]
rohdaten = int(hexdata,16)

# Hier werden nur die Datenbytes uebergeben um Rohdaten zu erhalten
hexdata = datensatz[size_end:endbyte_of_data]
rohdaten = int(hexdata,16)

# Umrechnung in %
ladestatus = round(float(100/(hex_max))*rohdaten,nachkommastellen)

# Ausgabe
print("Der Ladestatus betraegt: ", ladestatus, " %.")
```

7 Test und Auswertung

7.1 WLAN Empfang

Für die Wahl eines geeigneten WLAN-Netzwerks, für einen LNAP, wurden die Signalstärken aller verfügbaren Netzwerke am Ladeplatz des Elektrofahrzeugs gemessen und anhand der bekannten Richtwerte (Abbildung 7.1) sortiert. Mit Hilfe eines fertigen Softwareprogramms, des Arduino Softwarepackets, wurden die Netzwerke in der unmittelbaren Umgebung gescannt. In der Abbildung 7.2 sind die am Ladeplaz verfügbaren Netzwerke dargestellt. Die geeigneten Werte für die Signalstärke eines AP werden in Dezibel Milliwatt (dBm) angegeben und liegen zwischen -70 und -30 (vgl. Abbildung 7.1). Je höher der Wert ist, desto stärker ist das Signal.

Signal Strength	TL;DR		Required for
-30 dBm	Amazing	Max achievable signal strength. The client can only be a few feet from the AP to achieve this. Not typical or desirable in the real world.	N/A
-67 dBm	Very Good	Minimum signal strength for applications that require very reliable, timely delivery of data packets.	VoIP/VoWiFi, streaming video
-70 dBm	Okay	Minimum signal strength for reliable packet delivery.	Email, web
-80 dBm	Not Good	Minimum signal strength for basic connectivity. Packet delivery may be unreliable.	N/A
-90 dBm	Unusable	Approaching or drowning in the noise floor. Any functionality is highly unlikely.	N/A

Abbildung 7.1: Tabelle mit den Richtwerten der zulässigen Signalstärke eines WLAN-Netzwerks [37].

```
-> scan start
-> scan done
-> 14 networks found
-> 1: C4DSI (-55)*
-> 2: eduroam (-66)*
-> 3: HAW.guest (-66)*
-> 4: HAW.onboarding (-67)
-> 5: eduroam (-68)*
-> 6: HAW.guest (-68)*
-> 7: HAW.onboarding (-68)
-> 8: BESS-Box (-76)*
-> 9: Wi-Fi-TEC (-86)*
-> 10: eduroam (-86)*
-> 11: HAW.guest (-87)*
-> 12: HAW.onboarding (-87)
-> 13: SPHT (-90)*
-> 14: SPHT-WUEST (-90)*
```

Abbildung 7.2: Die verfügbaren WLAN-Netzwerke am Ladeplatz des BMW I3, die als LNAP genutzt werden können.

Zu den geeigneten Netzwerken zählen daher „eduroam" (-66dBm), „HAW.guest" (-66dBm), „HAW.onboarding" (-67dBm) und „C4DSI" (-55dBm). Das „C4DSI" Netzwerk besitzt das stärkste Empfangssignal und ist für die Kommunikationsinfrastruktur am besten geeignet. Die Signalstärke des „C4DSI" Netzwerks ist deshalb so stark, weil es vom Unified Services Router erzeugt wird, welcher in unmittelbarer Nähe zum Ladeplatz des BEV installiert ist.

7.2 CAN-Bus Kommunikation

Als Nächstes wird eine Kommunikation zwischen dem BEV und dem CAN-Bus-Shield getestet. Das Testscript hört die globalen CAN-Bus Telegramme und gibt diese mit den PID-Nummern wieder. Als Antwort liefert das Arduino UNO Programm (Abbildung 7.3) im hexadezimalen Zahlenformat die ID 130, den Data Length Code (DLC) 5 und die Daten 05 F0 FC FF FF. Um die CAN-Bus Kommunikation vor Überlastung zu schützen werden nur wichtige Informationen an alle Geräte, in definierten Zeitabständen gesendet. Bei dem PID 130 handelt es sich um den Status der Zündung des BMW I3 Fahrzeugs [11]. Eine erfolgreiche Kommunikation mit dem CAN-Bus des BEV erfüllt einen

Teil der gestellten Anforderung dieser Bachelorarbeit und zwar die Möglichkeit der direkten Kommunikation mit dem BEV und dessen ECU's. Die abgelesenen Daten werden an das ESP32-Board weitergeleitet.

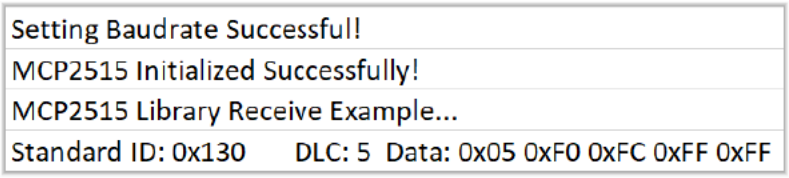


Abbildung 7.3: CAN-Bus Daten des BMW I3.

7.3 Daten des ESP32

Die Weiterleitung der Daten an das ESP32 wird mithilfe des I2C Bus realisiert. Die empfangenen Daten werden zu Testzwecken auf dem seriellen Monitor des Arduino Programms ausgegeben (Abbildung 7.4). Das aktuelle Ergebnis zeigt nur die Daten des jeweiligen PID an. In dieser Abbildung sind drei unterschiedliche Daten dargestellt, die bei verschiedenen Zuständen des Fahrzeugs abgefragt wurden. Da es sich um die Daten des Zündungsstatus handelt, sind es folgende drei Zustände: Fahrzeug verriegelt, Fahrzeug entriegelt und Fahrzeug fahrbereit. Das ESP32 besitzt die Möglichkeit die Daten direkt an den WLAN-Server weiterzuleiten, somit wurde eine Basis für die Kommunikation über WLAN erreicht. Als Nächstes folgt der Funktionstest des WLAN-Servers.

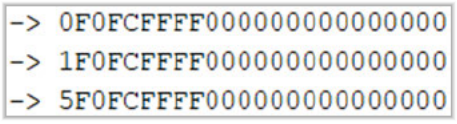


Abbildung 7.4: Transferierte Daten des ESP32 Boards.

7.4 Webserver Daten

Mit Hilfe des eingebauten akustischen Signalgebers, lässt sich schnell feststellen zu welchem Zeitpunkt das ESP32 sich mit dem lokalen Netzwerk verbunden hat. Die Rolle des AP hat im Testlauf ein privates Smartphone übernommen. Das Gerät, welches auf den WLAN-Server mit einer IP-Adresse zugreifen möchte (Client), muss sich im gleichen WLAN-Netzwerk befinden. Als Client wurde ein gewöhnlicher Web-Browser eines Computers benutzt, welcher zuvor mit demselben Netzwerk verbunden wurde. Unter Eingabe der definierten IP-Adresse werden als Rückgabe die CAN-Bus Werte im hexadezimalen Format, die zuvor an das ESP32 weitergeleitet wurden, zurückgegeben (Abbildung 7.5).

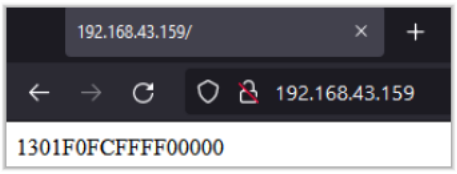


Abbildung 7.5: Datenrückgabe eines Web-Browsers.

Die Abfrage über den Web-Browser wird bei dem ESP32 registriert und somit die Kommunikation von Seiten des Clients bestätigt (Abbildung 7.6). Es werden unter anderem der Typ der Anfrage, die IP-Adresse des Host's, die Details des benutzten Web-Browsers, akzeptierte Sprachen und Verbindungsstatusinformationen des Client's mitgeteilt. Die IP-Adresse des Servers ist manuell einstellbar und kann je nach Einschränkungen des CC4E beliebig konfiguriert werden. Dieser Schritt vervollständigt den Datentransfer vom CAN-Bus des BEV bis hin zum LNAP und ermöglicht den Zugriff eines Lademanagement-Systems auf die zur Verfügung stehenden Daten. Die Kommunikationsinfrastruktur ist damit vollständig aufgebaut und getestet. Um nun die Abfragen, wie z.B. Ladestatus des BEV zu machen, muss die Software für die gezielte Abfrage in einer bestimmten Form konfiguriert werden.

Nach dem erfolgreichen Erhalt der Fahrzeugdaten über den Web-Browser wurde der letzte Schritt zur Realisierung des Systems erledigt. Damit wird die Anforderung zur Umsetzung des entwickelten Systems ebenfalls vollständig erfüllt.

```
-> New Client.
-> GET / HTTP/1.1
-> Host: 192.168.43.159
-> Connection: keep-alive
-> Upgrade-Insecure-Requests: 1
-> User-Agent: Mozilla/5.0 (Linux; Android 10; VOG-L29) AppleWebKit/537.36 (KHTML, like Gecko
-> Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
-> Accept-Encoding: gzip, deflate
-> Accept-Language: de-DE,de;q=0.9,ru-DE;q=0.8,ru;q=0.7,en-US;q=0.6,en;q=0.5
->
-> Client disconnected.
```

Abbildung 7.6: Bestätigung des Client-Requests.

7.5 Python Skript

Um die Kommunikation zwischen dem Lademanagement-System und dem LNAP zu realisieren, wird ein Softwareprogramm in Python-Sprache benutzt. Die Abfrage des Ladestatus kann gleichzeitig, sowohl vom Lademanagement-System, zwecks Ladeplanberechnung als auch vom Programmierwerkzeug namens „Node-Red" für die graphische Darstellung, erfolgen. Das Ergebnis der Abfrage des Python-Skripts ist in der Abbildung 7.7 dargestellt. Aufgrund der gleichen Datenstruktur und der noch fehlenden Abfrage des Ladestatus, werden die Daten der Zündung als Ladestatusinformation für den Testlauf verwendet. Das Programm teilt die Rohdaten auf, rechnet diese in Prozent um und gibt das Ergebnis in der Konsole aus. Die Anpassung des Skripts kann für jede beliebige Schnittstelle durchgeführt werden. Mit dem Erfolg des letzten Tests wurde die vollständige Datenübertragung zwischen dem BEV und dem Lademanagement-System abgeschlossen.

```
λ  python .\WebAbfrage.py
PID:
b'130'
raw data:
b'F0FCFFFF'
Der Ladestatus betraegt:  94.1  %
```

Abbildung 7.7: Ergebnis des Python-Scripts.

7.6 Begrenzter Handlungsspielraum

Durch den Aufbau und die erfolgreiche Funktion des Systems wurde bewiesen, dass es sowohl theoretisch als auch praktisch möglich ist, die Werte des Fahrzeugs über den CAN-Bus an der OBDII-Schnittstelle abzufragen und kabellos zu übertragen. Nach dem ersten Erfolg der funktionierenden Kommunikationsinfrastruktur fehlt allerdings die spezifische Abfrage der Ladestatusinformationen des BEV. Es handelt sich hierbei um sensible Informationen, die von dem Hersteller für Forschungszwecke nicht veröffentlicht werden. Nach Angaben der Fachabteilung des Herstellers, liegen zu dem Thema keine geeigneten, veröffentlichungsfähigen Informationsmaterialien vor. Aus diesem Grund muss die gezielte Abfrage des Wertes manuell herausgefunden werden. Es müsste ein Programm entwickelt werden, welches aus Tausenden von Sensordaten den richtigen Wert ermittelt. Sobald die exakte Abfrage des SoC Wertes bekannt ist, kann dieser, aufgrund der fertigen Kommunikationsinfrastruktur, dem Lademanagement problemlos zur Verfügung gestellt werden.

8 Zusammenfassung und Ausblick auf Forschungsdesiderate

In dieser Bachelorarbeit sollte eine Möglichkeit zur Datenübertragung eines batterieelektrischen Fahrzeugs an das Lademanagement des CC4E entwickelt werden, wobei der Schwerpunkt auf der Entwicklung eines Smart-Grid's nach EMSA-Vorgaben lag. Mit Hilfe mehrerer Mikrocontroller sollte ein Hardware-System als Kommunikationsinfrastruktur kostengünstig entwickelt und realisiert werden. Die abschließenden Tests haben bewiesen, dass es möglich ist die Daten eines batterieelektrischen Fahrzeugs automatisiert und kabellos an das Lademanagement am CC4E zu übertragen.

Zu diesem Zweck erfolgte am Anfang der Arbeit eine Literaturrecherche, anhand derer deutlich wurde, dass es mehrere Alternativen gibt an die Fahrzeugdaten zu gelangen. Die Anforderungen, die sich aus den örtlichen Gegebenheiten des CC4E in Bergedorf ergeben, beschränken diese Möglichkeiten jedoch sehr. Eine der Optionen die Daten über die OBDII-Schnittstelle des Fahrzeugs zu bekommen, ergab sich als machbar. Die Wahl der Schnittstellen zur Umsetzung der Kommunikation ist zwar durchführbar, allerdings anhand der einzigartigen Bedingungen des CC4E sehr spezifisch.

Nach Analyse der technischen Ausstattung des CC4E und nach der ausgiebigen Literaturrecherche wird ein System zur Umsetzung der Kommunikationsinfrastruktur entwickelt und vorgestellt. Dabei wird aufgezeigt, dass keine Lücken im System nach dem EMSA-Modell existieren und die Kommunikationsstruktur deshalb architektonisch als vollständig gilt. Zum besseren Verständnis und zur Einordnung des Systems wurde bei der Entwicklung der Architektur des aktuellen Systems auf die bereits existierenden Elemente hingewiesen. Dadurch steigt allerdings die Komplexität, welche eine Ablenkung von wichtigen Details des aktuellen Systems darstellen könnte.

Bei der Umsetzung des Systems wurde zur Entwicklung einer Hardware, ein Verbund aus mehreren Mikrocontrollern, als wichtiges Systemelement mit Hilfe einer frei zur Verfügung stehenden Software modelliert. Aus den theoretischen Entwürfen wurde anschlie-

ßend Schritt für Schritt die Hardware aufgebaut und getestet. Aufgrund der guten Dokumentation der einzelnen Mikrocontroller, ist die Verbindungsherstellung zwischen diesen schnell gelungen. Während der Verschaltung der vielen elektrischen Komponenten, auf einer einzigen Lochrasterplatine, besteht die Gefahr bei kleinen Fehlern einen Kurzschluss zu erzeugen und damit Elemente zu beschädigen. Dieser Teil der Umsetzung muss mit höchster Achtsamkeit und Präzision durchgeführt werden, um zeitaufwendige Fehler zu vermeiden.

Zur Berechnung des Energieverbrauchs des neuen Systems wurden die Datenblätter der einzelnen Mikrocontroller benutzt. Das Ergebnis des Energieverbrauchs bei der Berechnung in voll belasteten Betriebszustand, ergab einen relativ geringen Verbrauch von 18 kWh im Jahr. Es fehlt allerdings die Betrachtung des Energieverbrauchs im Stand-by und im normalen Arbeitsmodus, um den tatsächlichen Wert des Energieverbrauchs abschätzen zu können.

Die Entwicklung der Software für alle beteiligten Systemelemente wurde hauptsächlich mit Unterstützung der fertigen Software-Beispiele des Programms Arduino IDE umgesetzt. Bei der Entwicklung der eigenen Softwarekomponente ist die steigende Komplexität mit jeder implementierten Komponente weiter gewachsen. Es bestand eine große Herausforderung darin die Schnittstellen jeder einzelnen Hardware exakt aneinander anzupassen.

In der Testphase des Kommunikationsgeräts wurden zu Überprüfungszwecken die Testdaten an jedem einzelnen Element ausgegeben. Die Benutzung des Web-Browsers ist allerdings nur für Überprüfungszwecke durchgeführt worden, weil die Rohdaten im Hexadezimalformat als Antwort empfangen werden. Die Rückgabe des Webservers ist nach dem spezifischen Standard für den CAN-Bus Datenverkehr zu entziffern. Jeder, der die IP-Adresse des Kommunikationsgeräts kennt und sich in dem selben Netzwerk befindet, kann die Datenabfrage durchführen, ohne dabei Einfluss auf die Datenübertragung zu haben.

Durch die erfolgreiche Entwicklung eines Smart-Grid's, basierend auf einem Verbund mehrerer Mikrocontroller, wurde ein Beitrag zur Flexibilisierung des Lastenmanagements geleistet. Es stellt einen Zwischenschritt für die Übergabe der Ladestatusinformationen dar, die für die Berechnung des Ladeplans eines BEV am Energie-Campus in Bergedorf benötigt werden. Mit Hilfe dieses Systems kann die Erforschung des Einsparpotentials durch gezieltes Laden der BEV, mit Ausnahme des Herausfindens der SoC-Abfrage, fortgesetzt werden.

Daraus ergibt sich eine konkrete Aussicht für weitere Schritte, um das System vollständig ans Ziel zu bringen. Als einer der wichtigsten Schritte muss eine Möglichkeit entwickelt werden, den SoC-Wert des Fahrzeugs BMW I3 am Energie-Campus herauszufinden. Zur Verbesserung der Forschungsergebnisse kann eine Funktion zur Speicherung und zur Synchronisierung der Werte mit der Datenbank entwickelt werden. Momentan wird der Ladestatus nur ermittelt, wenn das Fahrzeug sich am Ladeplatz befindet. Des Weiteren kann die Funktion der Datenübertragung grundsätzlich für weitere Zwecke unter Verwendung weiterer Daten genutzt werden, um die Möglichkeiten der entwickelten Kommunikationsinfrastruktur maximal ausnutzen zu können. In der Bachelorarbeit wurde der Fokus der Datenabfrage, aufgrund der bereits vollständig entwickelten Software- und Hardwareelemente, auf das Arduino Uno Board gelegt. Diese können als einzelner Baustein genutzt werden und damit massive Vorteile bei dem zeitlichen Rahmen der Bachelorarbeit darstellen. Zur Verbesserung und zur Vereinfachung der Komplexität des Systems kann über die Reduzierung, der im Smart-Grid befindlichen Komponente nachgedacht werden. So kann beispielsweise auf das Arduino UNO Board komplett verzichtet werden, wenn die CAN-Bus-Kommunikation allein über das bereits parallel integrierte MCP2515-Modul implementiert wird.

Literaturverzeichnis

[1] Amarasinghe, M., Kottegoda, S., Arachchi, A. L., Muramudalige, S., Dilum Bandara, H. M. N. and Azeez, A. [2015], Cloud-based driver monitoring and vehicle diagnostic with OBD2 telematics, *in* '2015 Fifteenth International Conference on Advances in ICT for Emerging Regions (ICTer)', pp. 243–249.

[2] Arduino [2017], 'Stromverbrauch Arduino & Wemos Boards - arduino-projekte.info'.
URL: *https://arduino-projekte.info/stromverbrauch-arduino-wemos-boards/*

[3] Arduino [2021], 'Software | Arduino'.
URL: *https://www.arduino.cc/en/software*

[4] automotivedose [2019], 'Automotive Dose'.
URL: *https://automotivedose.blogspot.com/*

[5] AZ-Delivery [2020], 'Teamwork Raspberry Pi und Micro-Controller – AZ-Delivery'.
URL: *https://www.az-delivery.de/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/teamwork-raspberry-pi-und-micro-controller*

[6] BMWi [2018], 'BMWi - Alle Tarife mengengewichteter Elektrizitätspreis für Haushaltskunden'.
URL: *https://www.bmwi.de/Redaktion/DE/Infografiken/Energie/ strompreisbestandteile.html*

[7] BMWi [2019], 'Energieeffizienzstrategie 2050', p. 76.

[8] Boris Adlung [2020], 'LSS-Signale unter der Lupe', *Fachjournal Elektronik* p. 70.

[9] Bremer, W. [1998], ISO 15031 Parts 1 amp; 2, *in* 'IEE Communication Standards for European On-Board-Diagnostics Seminar (Ref. No. 1998/294)', pp. 7/1–7/7.

[10] Broy, M. and Rausch, A. [2005], 'Das neue V-Modell® XT: Ein anpassbares Modell für Software und System Engineering', *Informatik-Spektrum* **28**(3), 220–229.
URL: *http://link.springer.com/10.1007/s00287-005-0488-z*

[11] *Controlling BMW E90 instrument cluster - Using Arduino / Project Guidance* [2020].
URL: *https://forum.arduino.cc/t/controlling-bmw-e90-instrument-cluster/670728*

[12] D-Link [2014], 'Beschreibung des DSR-1000N Routers'.
URL: *https://eu.dlink.com/de/de/products/dsr-1000n-wireless-n-dual-band-unified-service-router*

[13] eaa-phev [2013], 'Escape PHEV TechInfo - My wiki'.
URL: *http://www.eaa-phev.org/wiki/Escape$_P$HEV$_T$echInfoPIDs*

[14] Florian Schäffer [2020], *Fahrzeugdiagnose mit OBD II | Florian Schäffer | Taschenbuch | ISBN 9783895763915*, Elektor Verlag GmbH, Aachen.
URL: *https://www.isbn.de/buch/9783895763915$_f$ahrzeugdiagnose − mit − obd − ii.htm*

[15] Frauenhofer ISI [2012], 'Produkt-Roadmap Lithium-Ionen-Batterien 2030'.
URL: *https://www.econstor.eu/bitstream/10419/60231/1/719953316.pdf*

[16] Gao, K. and Mo, B. [2011], Design and application of new avionics control bus upon SPI, *in* 'Engineering design and Manufacturing informatization 2011 International Conference on System science', Vol. 2, pp. 203–205.

[17] Gijsel, I. d. [2010], *CAN und EOBD in der Fahrzeugtechnik*. OCLC: 700340250.

[18] HAW-Hamburg [2021*a*], 'Blick auf den Energie-Campus in Curslack vom CC4E'.
URL: *https://www.haw-hamburg.de/detail/news/news/show/erste-direct-air-capture-anlage-geht-in-betrieb/*

[19] HAW Hamburg [2021*b*], 'HAW Hamburg: Competence Center für Erneuerbare Energien und EnergieEffizienz'.
URL: *https://www.haw-hamburg.de/cc4e/*

[20] HAW Hamburg [2021*c*], 'HAW Hamburg: Technologiezentrum'.
URL: *https://www.haw-hamburg.de/index.php?id=2152*

[21] Hosting-Technik [2019], 'So optimiert HTTP/2 das World Wide Web - IONOS'.
URL: *https://www.ionos.de/digitalguide/hosting/hosting-technik/so-optimiert-http2-das-world-wide-web/*

[22] IEEE [2019], 'Adaptive Centralized Random Access for Collision Free Wireless Local Area Networks | IEEE Journals & Magazine | IEEE Xplore'.
URL: *https://ieeexplore.ieee.org/document/8666983*

[23] IEEE [2020], '802.1X-2020 - IEEE Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control | IEEE Standard | IEEE Xplore'.
URL: *https://ieeexplore.ieee.org/document/9018454*

[24] Ken Tindell [2020], 'The canframe.py tool | Dr. Ken Tindell'.
URL: *https://kentindell.github.io/2020/01/03/canframe$_p$y$_t$ool/*

[25] Kirpes, B., Danner, P., Basmadjian, R., Meer, H. d. and Becker, C. [2019], 'E-Mobility Systems Architecture: a model-based framework for managing complexity and interoperability', *Energy Informatics* **2**(1), 15.
URL: *https://energyinformatics.springeropen.com/articles/10.1186/s42162-019-0072-4*

[26] Leens, F. [2009], 'An introduction to I2C and SPI protocols', *IEEE Instrumentation Measurement Magazine* **12**(1), 8–13. Conference Name: IEEE Instrumentation Measurement Magazine.

[27] Mathias Röper [2018], Entwicklung und Implementierung eines Systems zur Flexibilisierung der Ladevorgänge von batterieelektrischen Fahrzeugen, PhD thesis, HAW-Hamburg, Hamburg.

[28] Meyer, N. [2021], Bewertung und Optimierung eines smarten Lademanagements fu¨r ein Elektrofahrzeug am Energie-Campus Bergedorf, PhD thesis, HAW Hamburg, Hamburg.

[29] Michael Danzig [2018], Nutzung des Prozessors ESP-32 als ressourcensparende Plattform für die Aufzeichnung von WLAN-Pprobe-Requests, PhD thesis, HTW Dresden.

[30] Microchip Technology [2005], 'Stromberbrauch MCP2515 Chip'.
URL: *https://files.seeedstudio.com/wiki/CAN$_B$US$_S$hield/resource/MCP*2515*.pdf*

[31] Microchip Technology [2010], 'Stromverbrauch des MCP2551 Chips'.
URL: *https://files.seeedstudio.com/wiki/CAN$_B$US$_S$hield/resource/Mcp*2551*.pdf*

[32] Microchip Technology [2013], 'Stromverbraich Microchip MCP2562.'.
URL: *http://ww1.microchip.com/downloads/en/devicedoc/20005167c.pdf*

[33] Oxzibyte [2021], 'BD9 auf OBDII Schnittstelle'.
URL: *https://www.oxzibyte.com/index.php?main*$_{p}$*age* = *product*$_{i}$*nfoproducts*$_{i}$*d* = 898934

[34] Patrick Jochem [2021], 'Demand Side Management (DSM) • Definition | Gabler Wirtschaftslexikon'.
URL: *https://wirtschaftslexikon.gabler.de/definition/demand-side-management-dsm-53701*

[35] Piembsystech [2021], 'CAN Protocol'.
URL: *https://piembsystech.com/can-protocol/*

[36] Rattan, S. [2013], 'W-Lan Standards'.
URL: *https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.403.2956rep=rep1type=pdf*

[37] rcdesign [2020], 'rcdesign, Author at CORE Cabling'.
URL: *https://corecabling.com/author/rcdesign/*

[38] Speculatrix [2018], 'Wissen Sie, was Ihr Code macht? - Machina Speculatrix'.
URL: *https://mansfield-devine.com/speculatrix/2018/01/do-you-know-what-your-codes-doing/*

[39] Tim Berners-Lee [1991], 'Das HTTP-Protokoll, wie es in W3 implementiert ist'.
URL: *https://www.w3.org/Protocols/HTTP/AsImplemented.html*

[40] Trefke, J., Rohjans, S., Uslar, M., Lehnhoff, S., Nordström, L. and Saleem, A. [2013], Smart Grid Architecture Model use case management in a large European Smart Grid project, *in* 'IEEE PES ISGT Europe 2013', pp. 1–5. ISSN: 2165-4824.

[41] Umwelt Bundesamt [2020], 'Endenergieverbrauch 2019 nach Sektoren und Energieträgern | Umweltbundesamt'.
URL: *https://www.umweltbundesamt.de/bild/endenergieverbrauch-2019-nach-sektoren*

[42] Wallbe [n.d.], 'wallbe® PRO | Comfortable charging with up to 22kW | wallbe®'.
URL: *https://www.wallbe.de/en/wallbe-pro/*

[43] Wolff, D. and Göbel, R. [2018], *Digitalisierung: Segen oder Fluch: Wie die Digitalisierung unsere Lebens- und Arbeitswelt verändert*, Springer-Verlag. Google-Books-ID: 1i1dDwAAQBAJ.

Anhang

A.1 Standard-PID's (0-1F)

PID (hex)	Data bytes returned	Description	Min value	Max value	Units	Formula[a]
00	4	PIDs supported [01 - 20]				Bit encoded [A7..D0] == [PID $01..PID $20] See below
01	4	Monitor status since DTCs cleared. (Includes malfunction indicator lamp (MIL) status and number of DTCs.)				Bit encoded. See below
02	2	Freeze DTC				
03	2	Fuel system status				Bit encoded. See below
04	1	Calculated engine load	0	100	%	$\frac{100}{255}A$ (or $\frac{A}{2.55}$)
05	1	Engine coolant temperature	-40	215	°C	$A - 40$
06	1	Short term fuel trim—Bank 1	-100 (Reduce Fuel: Too Rich)	99.2 (Add Fuel: Too Lean)	%	$\frac{100}{128}A - 100$ (or $\frac{A}{1.28} - 100$)
07	1	Long term fuel trim—Bank 1				
08	1	Short term fuel trim—Bank 2				
09	1	Long term fuel trim—Bank 2				
0A	1	Fuel pressure (gauge pressure)	0	765	kPa	$3A$
0B	1	Intake manifold absolute pressure	0	255	kPa	A
0C	2	Engine RPM	0	16,383.75	rpm	$\frac{256A + B}{4}$
0D	1	Vehicle speed	0	255	km/h	A
0E	1	Timing advance	-64	63.5	° before TDC	$\frac{A}{2} - 64$
0F	1	Intake air temperature	-40	215	°C	$A - 40$
10	2	MAF air flow rate	0	655.35	grams/sec	$\frac{256A + B}{100}$
11	1	Throttle position	0	100	%	$\frac{100}{255}A$
12	1	Commanded secondary air status				Bit encoded. See below

Abbildung A.1: Standard PID 1

13	1	Oxygen sensors present (in 2 banks)				[A0..A3] == Bank 1, Sensors 1-4. [A4..A7] == Bank 2...
14	2	Oxygen Sensor 1 A: Voltage B: Short term fuel trim	0 -100	1.275 99.2	volts %	$\frac{A}{200}$ $\frac{100}{128}B - 100$ (if B==$FF, sensor is not used in trim calculation)
15	2	Oxygen Sensor 2 A: Voltage B: Short term fuel trim				
16	2	Oxygen Sensor 3 A: Voltage B: Short term fuel trim				
17	2	Oxygen Sensor 4 A: Voltage B: Short term fuel trim				
18	2	Oxygen Sensor 5 A: Voltage B: Short term fuel trim				
19	2	Oxygen Sensor 6 A: Voltage B: Short term fuel trim				
1A	2	Oxygen Sensor 7 A: Voltage B: Short term fuel trim				
1B	2	Oxygen Sensor 8 A: Voltage B: Short term fuel trim				
1C	1	OBD standards this vehicle conforms to				Bit encoded. See below
1D	1	Oxygen sensors present (in 4 banks)				Similar to PID 13, but [A0..A7] == [B1S1, B1S2, B2S1, B2S2, B3S1, B3S2, B4S1, B4S2]
1E	1	Auxiliary input status				A0 — Power Take Off (PTO) status (1 == active) [A1..A7] not used
1F	2	Run time since engine start	0	65,535	seconds	$256A + B$

Abbildung A.2: Standard PID 2

A.2 Arduino Uno Code

```
#include <SPI.h>
#include <Wire.h>

//Serial Communication Sender Arduino UNO
#include<SoftwareSerial.h>
SoftwareSerial s(3,1);

#include "mcp2515_can.h"
#include <avr/sleep.h>

const int SPI_CS_PIN = 10;
#define CAN_INT 2

char IntHexData[21];
unsigned long IntHexID;
char IntHexChar;

```

```
18 #define RS_TO_MCP2515 true
19
20 mcp2515_can CAN(SPI_CS_PIN);
21
22 #define KEEP_AWAKE_TIME 200
23 unsigned long lastBusActivity = millis();
24
25 unsigned char flagRecv = 0;
26 unsigned char len = 0;
27 unsigned char buf[8];
28
29 int lastLen = -1;
30 unsigned char lastBuf[8];
31 unsigned long lastMsgTime = 0;
32 #define DUPLICATE_TIMEOUT 20
33
34 char str[20];
35
36 void setup() {
37     // I2C Configuration
38     Wire.begin(8);                /* join i2c bus with address 8 */
39     Wire.onRequest(requestEvent); /* register request event */
40     s.begin(115200);              /* start serial for debug */
41     SERIAL_PORT_MONITOR.begin(115200);
42
43     while (CAN_OK != CAN.begin(CAN_500KBPS, MCP_16MHz)) {
44         SERIAL_PORT_MONITOR.println("CAN init fail, retry...");
45         delay(100);
46     }
47     SERIAL_PORT_MONITOR.println("CAN init ok!");
48
49     // attach interrupt
50     pinMode(CAN_INT, INPUT);
51     attachInterrupt(digitalPinToInterrupt(CAN_INT), MCP2515_ISR, FALLING);
52
53     CAN.setSleepWakeup(1); // tells the MCP2515 to wake up
54
55     // Pull the Rs pin of the MCP2551 transceiver low to enable it:
56     if (RS_TO_MCP2515) {
57         CAN.mcpPinMode(MCP_RX0BF, MCP_PIN_OUT);
58         CAN.mcpDigitalWrite(RS_OUTPUT, LOW);
59     } else {
60         pinMode(RS_OUTPUT, OUTPUT);
61         digitalWrite(RS_OUTPUT, LOW);
```

```
62      }
63  }
64
65  uint32_t id;
66
67  void MCP2515_ISR() {
68      flagRecv = 1;
69  }
70
71  void loop() {
72      if (flagRecv) {
73          flagRecv = 0;                   // clear flag
74          lastBusActivity = millis();
75
76          while (CAN_MSGAVAIL == CAN.checkReceive()) {
77
78              // read data,  len: data length, buf: data buf
79              CAN.readMsgBuf(&len, buf);
80              unsigned long canId = CAN.getCanId();
81
82              // CAN PID NUMBER:
83              id = CAN.getCanId();
84
85              // check if this is a duplicate message (
86              if ((len != lastLen) || (millis() > lastMsgTime +
    DUPLICATE_TIMEOUT)
87                      || (memcmp((const void*)lastBuf, (const void*)buf,
    sizeof(buf)) != 0)) {
88                  lastLen = len;
89                  memcpy(lastBuf, buf, sizeof(buf));
90                  lastMsgTime = millis();
91
92                  // print the data
93                  for (int i = 0; i < len; i++) {
94                      SERIAL_PORT_MONITOR.print(buf[i]); SERIAL_PORT_MONITOR.
    print("\t");
95
96                  // Send I2C Data:
97                  IntHexData[(i+3)] = buf[i];
98                  }
99
100             SERIAL_PORT_MONITOR.println("");
101             SERIAL_PORT_MONITOR.print("get data from ID: 0x");
102             SERIAL_PORT_MONITOR.println(canId, HEX);
```

```
103                 Wire.write((unsigned char)canId);
104
105                 for (int n=0;n<sizeof(IntHexData);n++){ // %TEST CODE
106                 //Wire.write Data / Send to Server Global Data @
107                 Wire.write(IntHexData[n]);}  /*send string on request */
108
109                 SERIAL_PORT_MONITOR.println();
110               }
111         }
112     } else if (millis() > lastBusActivity + KEEP_AWAKE_TIME) {
113         // Put MCP2515 into sleep mode
114         SERIAL_PORT_MONITOR.println(F("CAN sleep"));
115         CAN.sleep();
116
117         // Put the transceiver into standby (by pulling Rs high):
118         if (RS_TO_MCP2515) {
119             CAN.mcpDigitalWrite(RS_OUTPUT, HIGH);
120         } else {
121             digitalWrite(RS_OUTPUT, HIGH);
122         }
123
124         // Put the MCU to sleep
125         SERIAL_PORT_MONITOR.println(F("MCU sleep"));
126
127         // Clear serial buffers before sleeping
128         SERIAL_PORT_MONITOR.flush();
129
130         cli(); // Disable interrupts
131         if (!flagRecv) {
132             set_sleep_mode(SLEEP_MODE_PWR_DOWN);
133             sleep_enable();
134             sleep_bod_disable();
135             sei();
136             sleep_cpu();
137             // Now the Arduino sleeps until the next message arrives...
138             sleep_disable();
139         }
140         sei();
141
142         CAN.wake(); // LISTENONLY mode
143
144         // Wake up the transceiver:
145         if (RS_TO_MCP2515) {
146             CAN.mcpDigitalWrite(RS_OUTPUT, LOW);
```

```
147         } else {
148             digitalWrite(RS_OUTPUT, LOW);
149         }
150 
151         SERIAL_PORT_MONITOR.println(F("Woke up"));
152     }
153 }
154 
155 // I2C CONNECTION
156 // function that executes whenever data is received from master
157 void receiveEvent(int howMany) {
158  while (0 <Wire.available()) {
159     char c = Wire.read();      /* receive byte as a character */
160     Serial.print(c);           /* print the character */
161     IntHexChar = c;
162   }
163   Serial.println();             /* to newline */
164  }
165 
166 // function that executes whenever data is requested from master
167 void requestEvent() {
168 
169   for (int n=0;n<sizeof(IntHexData);n++){ // %TEST CODE
170   //Wire.write Data
171   Wire.write(IntHexData[n]);  /*send string on request */
172   }
173 }
```

A.3 ESP32 Code

```
#include <WiFi.h>
#include <Wire.h>
#include "...\AP\ap.h" // Voller Pfad

int D1 = 21;
int D2 = 22;

// Replace with your network credentials
const char* ssid     = CC4E_AP_NAME;
const char* password = CC4E_AP_PSWD;

// Slave-Adresse
const int SLAVE_ADR = 5;

// BUZZER ------------------------------------
int Buzzer = 4; //for ESP32 Microcontroller
int delay_short_loop = 50;
int delay_long_loop = 200;
//--------------------------------------------


// I2C extentions
char data[13];

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;

// Set your Static IP address
IPAddress local_IP(192, 168, 43, 159);
// Set your Gateway IP address
IPAddress gateway(192, 168, 43, 1);

IPAddress subnet(255, 255, 255, 0);
IPAddress primaryDNS(8, 8, 8, 8);
IPAddress secondaryDNS(8, 8, 4, 4);

// BUZZER ------------------------
void connect_wlan ()
{
```

```
43    for (int i=1;i<delay_short_loop;i++){
44     digitalWrite (Buzzer, HIGH); //turn buzzer on
45     delay(1);
46     digitalWrite (Buzzer, LOW);  //turn buzzer off
47     delay(1);
48     }
49
50   delay(delay_short_loop);
51
52    for (int i=1;i<delay_short_loop;i++){
53     digitalWrite (Buzzer, HIGH); //turn buzzer on
54     delay(1);
55     digitalWrite (Buzzer, LOW);  //turn buzzer off
56     delay(1);
57     }
58
59   delay(delay_short_loop);
60
61    for (int i=1;i<delay_short_loop;i++){
62     digitalWrite (Buzzer, HIGH); //turn buzzer on
63     delay(1);
64     digitalWrite (Buzzer, LOW);  //turn buzzer off
65     delay(1);
66     }
67
68    delay(delay_long_loop);
69
70    for (int i=1;i<delay_long_loop;i++){
71    digitalWrite (Buzzer, HIGH); //turn buzzer on
72    delay(1);
73    digitalWrite (Buzzer, LOW);  //turn buzzer off
74    delay(1);
75   }
76 }
77
78 void setup() {
79   Serial.begin(115200);
80
81     // I2C Config bus with SDA=D1 and SCL=D2 of NodeMCU
82     Wire.begin(D1, D2);
83
84     // Configures static IP address
85     WiFi.config(local_IP, primaryDNS, gateway, subnet);
86
```

```
87      pinMode (Buzzer, OUTPUT);
88
89      // I2C ----------------------
90      Serial.print("Wire Beginn!!");
91      Wire.begin();  // join i2c bus
92      //_____________________________
93
94    delay(10);
95    // Connect to Wi-Fi network with SSID and password
96    Serial.println();
97    Serial.println();
98    Serial.print("Connecting to ");
99    Serial.println(ssid);
100   WiFi.begin(ssid, password);
101   while (WiFi.status() != WL_CONNECTED) {
102     delay(500);
103     Serial.print(".");
104   }
105
106   // Print local IP address and start web server
107   Serial.println("");
108   Serial.println("WiFi connected.");
109   Serial.println("IP address: ");
110   Serial.println(WiFi.localIP());
111   server.begin();
112
113   connect_wlan(); // connest sound
114 }
115
116 int value = 0;
117
118 void loop(){
119
120 // Read I2C Data from Arduino UNO
121    int i = 0;
122   Wire.requestFrom(8, 13);    // request 13 bytes from slave device #8
123   while (Wire.available()) { // slave may send less than requested
124     char c = Wire.read();     // receive a byte as character
125     Serial.print(c, HEX);     // print the character
126     data[i] = c;              // Save globaly the recived Data
127     i += 1;
128     }
129   Serial.println("");
130
```

```
131 WiFiClient client = server.available(); // Listen for incoming clients
132 if (client) {                     // if you get a client,
133  Serial.println("New Client.");  // print a message out the serial port
134    String currentLine = "";      // make a String to hold incoming data
135    while (client.connected()) {  // loop while the client's connected
136      if (client.available()) {   // if there's bytes to read from the
     client,
137        char c = client.read();   // read a byte, then
138        Serial.write(c);          // print it out the serial monitor
139        if (c == '\n') {          // if the byte is a newline character
140
141          // that's the end of the client HTTP request, so send a response:
142          if (currentLine.length() == 0) {
143            client.println("HTTP/1.1 200 OK");
144            client.println("Content-type:text/html");
145            client.println();
146
147            // Send Arduino data to WiFi Server
148            for (int n=0;n<sizeof(data);n++){
149              //client.print("Data: ");
150              client.print(data[n], HEX);
151              }
152
153            client.println();
154            // break out of the while loop:
155            break;
156          } else {
157            currentLine = "";
158          }
159        } else if (c != '\r') {
160          currentLine += c;
161        }
162      }
163    }
164    // Clear the header variable
165    header = "";
166    // Close the connection
167    client.stop();
168    Serial.println("Client disconnected.");
169    Serial.println("");
170  }
171 }
```

A.4 Python Skript Abfrage

```
import urllib
from urllib import request

# IP Adresse des Kommunikationgeraets eines EV
Vehicle_IP = 'http://192.168.0.184'

# Serverdaten werden empfangen
rueckgabewert = request.urlopen(Vehicle_IP)
datensatz = rueckgabewert.read()

# Das PID Nummer wird in den ersten 3 Stellen kodiert
pid_start = 0
pid_end = 3
pid_number = datensatz[pid_start:pid_end]
print(pid_number)

# Anzahl der Datenbits wird im ersten Byte nach PID nummer mitgeliefert
size_start = 4
size_end = 5
data_length= datensatz[size_start:size_end]

# Ein Byte hat 2 hexadezimale Stellen
byte_size = 2

# Anzahl der Nachkommastellen
nachkommastellen = 1

# Anzahl der Bits in einem Byte
bit_counts = 8

# Berechnung des letzen Datenbytes
endbyte_of_data = (size_end+(int(data_length)*byte_size))

# Hier weden nur die Datenbytes uebergeben um Rohdaten zu erhalten
hexdata = datensatz[size_end:endbyte_of_data]
rohdaten = int(hexdata,16)

# Berechnung der gesamten Anzahl der Bytes (1 x data_length = 2^8)
```

```
39 data_byte_length = int(data_length)*8
40
41 # Maximalwert eines hexodezimalen Zahls
42 hex_max = (byte_size ** int(data_byte_length))
43
44 # Umrechnung in %
45 ladestatus = round(float(100/(hex_max))*rohdaten,nachkommastellen)
46
47 # Ausgabe
48 print("Der Ladestatus betraegt: ", ladestatus, " %.")
```

A.5 EMSA Dokument

1 Beschreibung des Anwendungsfalles

1.1 Name des Anwendungsfalles

Anwendungsfallbezeichnung		
ID	**Fachgebiet(e)**	**Name des Anwendungsfalles**
	Elektro- und Informationstechnik	WiFi portable Vehicle Communication Module

1.2 Versionsverwaltung

Versionsverwaltung						
Änderungen/ Version	**Datum**	**Name Autor(en) oder Komitee**	**Domänen Experte**	**Fachgebiet/ Rolle**	**Titel**	**Freigabe Status** *Entwurf, zur Kommentierung, zur Abstimmung, Final*
1.0	10.10.2021	Fuhr, Alex			Kabellose Datenübertragung eines BEV	Entwurf

1.3 Anwendungsbereich und Ziel des Anwendungsfalls

Anwendungsbereich und Ziel des Anwendungsfalls	
Verbundenes Geschäftsszenario	BEV Lademanagement Optimierung
Anwendungsbereich	Drahtlose Übertragung der Fahrzeugdaten ans Lademanagementsystem
Ziel	Zur Verfügung stehende Ladestatusinformationen eines BEV zur Erstellung des optimalen Ladeplans

1.4 Erläuterung des Anwendungsfalls

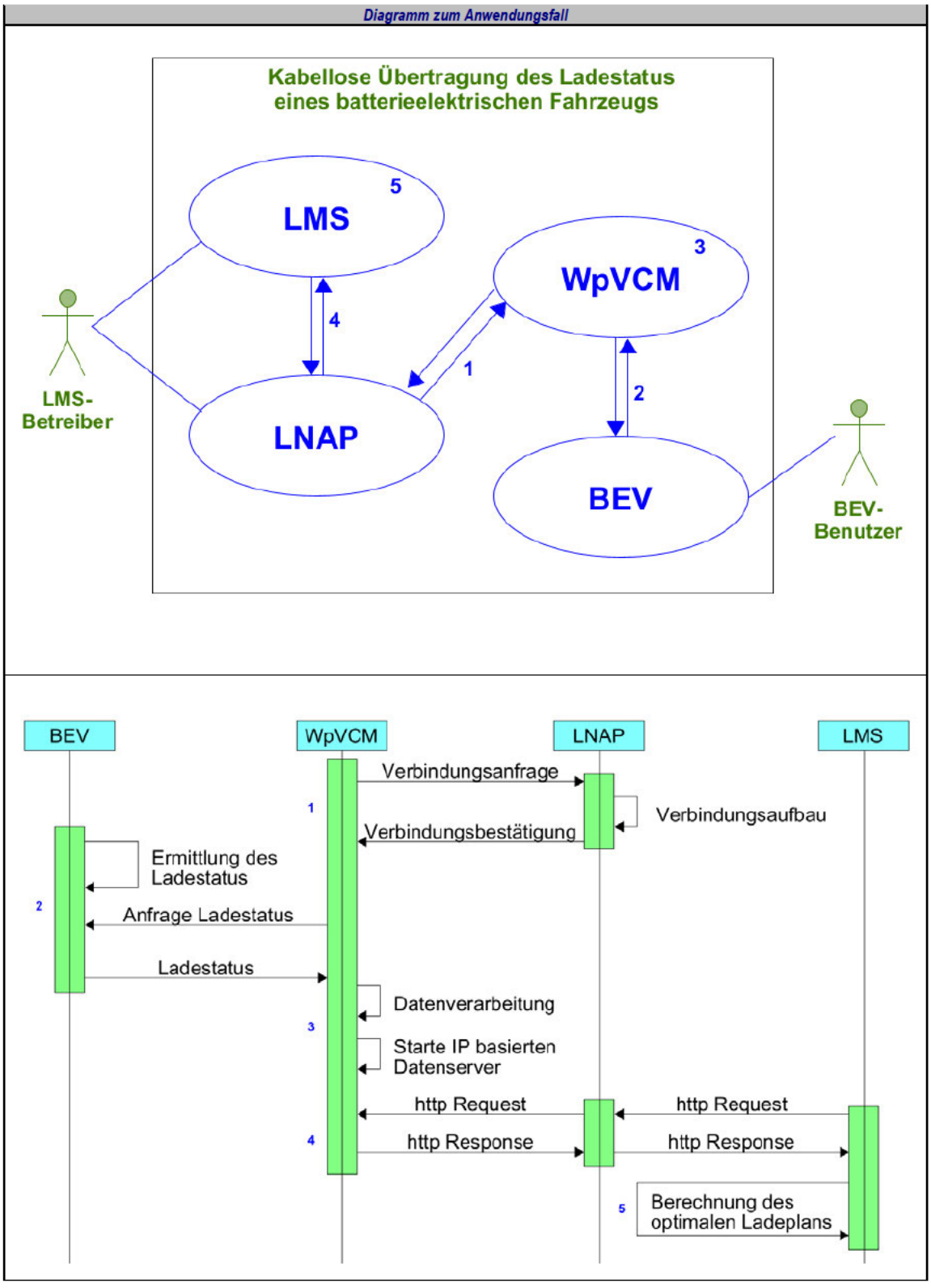
Erläuterung des Anwendungsfalls
Kurzbeschreibung – *max. 3 Sätze*
Um das batterieelektrische Fahrzeug optimal laden zu können, werden dessen Ladestatusinformationen vor dem Ladevorgang benötigt. Dazu werden die Fahrzeugdaten vor dem Ladevorgang erfasst und drahtlos übertragen. Der Datentransfer basiert auf der CAN-Bus-Schnittstelle des Fahrzeugs, welche den WLAN-Datenknoten für die drahtlose Übertragung benutzt.
Ausführliche Beschreibung
In diesem Szenario ist das Ziel des Betreibers eines Lademanagementsystems, ein oder mehrere batterieelektrische Fahrzeuge optimal zu laden. Die Fahrzeugaufladung wird über einen Lade-Plan gesteuert. Dieser wird vor jedem Laden erneut kontrolliert und angepasst. Die Daten werden am Fahrzeug mit Hilfe eines Mikrocontrollers über die integrierte OBDII-Schnittstelle am CAN-Bus abgelesen. Es werden nur die benötigten Daten durch das ausgewählte Softwareprogram abgefragt. Das CAN-Bus-Mikrocontrollersystem stellt die Informationen als WLAN-Server über den „WiFi Access Point" zur Verfügung, auf den das interne Lademanagementsystem Zugriff erhält.

1.5 Allgemeine Anmerkungen

Allgemeine Anmerkungen
Der Anwendungsfall beschreibt den Vorgang aus Sicht des Systembetreibers

Abbildung A.3: Spezifizierung des Anwendungsfalls

2 Diagramme zum Anwendungsfall



3 Technische Details

3.1 Akteure: Menschen, Systeme, Anwendungen, Datenbanken, das Energieversorgungssystem und weitere Stakeholder *(eine Tabelle pro Gruppierung)*

Akteure			
Gruppierung (Community)		**Gruppenbeschreibung**	
Mobile Elemente des Systems		Mobile Elemente des Systems, die das Transportmittel oder deren Erweiterung repräsentieren	
Akteurname siehe Akteursliste	**Akteurtyp** siehe Akteursliste	**Akteurbeschreibung** siehe Akteursliste	**Informationen in Bezug auf diesen Anwendungsfall**
EV	Device	Das „Electric Vehicle" ist ein Transportmittel und zugleich ein Batteriespeicher mit speicherregelndem System.	
WpVCM	Device	„WiFi portable Vehicle Communication Modul " ist ein System, das die Ladestatusinformationen vom EV abfragt und an den lokalen Netzwerk Access Point zur Verfügung stellt.	

Akteure			
Gruppierung (Community)		**Gruppenbeschreibung**	
Stationäre Elemente des Systems		Stationäre Bestandteile des Systems, die im Gebäude fest integriert sind	
Akteurname siehe Akteursliste	**Akteurtyp** siehe Akteursliste	**Akteurbeschreibung** siehe Akteursliste	**Informationen in Bezug auf diesen Anwendungsfall**
LMS	Device / Software	Das „Lademanagementsystem" ist eine Software basierte Kontrolleinheit des Technologiezentrums, die zum Berechnen und zum gezielten Ausführen des optimalen Ladevorgangs eines EV verantwortlich ist.	
LNAP	Device	Der „Local Network Access Point" ist ein WLAN-Netzwerk, der durch ein bestimmtes Gerät für interne Kommunikation des Technologiezentrums in Bergedorf zur Verfügung steht.	

3.2 Vorbedingungen, Annahmen, Nachbedingungen, Ereignisse

Voraussetzungen des Anwendungsfalls			
Akteur/System/ Information/Vertrag	**Auslösendes Ereignis**	**Vorbedingungen**	**Annahmen**
EV	Anfrage vom WpVCM an EV	Offener Zugang auf die Ladestatusinformationen auf dem CAN-Bus System	Stellt die Ladestatusinformationen der Akkumulatoren über die CAN-Bus-Schnittstelle bereit
WpVCM	Anfrage vom WpVCM an EV und an LNAP	Ladestatus des EV wird im 5-Minuten-Zyklus erfasst	Daten zwischen batterieelektrischem Fahrzeug und dem LNAP werden durch WpVCM ausgetauscht
LMS	Anfrage vom LMS an LNAP	Lademanagement-System hat Zugriff auf das gleiche interne Netzwerk, wie das WpVCM	Stellt alle 5 Minuten eine Ladestatusanfrage
LNAP	Anfrage vom LMS und vom WpVCM an LNAP	EV mit integriertem WpVCM befndet sich im Fuhrpark des Technologiezentrums in Bergedorf	Daten zwischen WpVCM und LMS werden mit Hilfe von LNAP ausgetauscht.

3.3 Referenzen/Einflussfaktoren

Referenzen						
Nr.	**Referenztyp**	**Referenz-bezeichnung**	**Status**	**Einfluss auf Anwendungsfall**	**Urheber/ Organisation**	**Link**
1	Standard	IEEE 802.3	IS	Kommunikation zwischen Lademanagement-System und LNAP über lokales Netzwerk (LAN)	IEEE	
2	Standard	IEEE 802.11	IS	WLAN-Kommunikation zwischen LNAP und WpVCM über lokales Netzwerk (LAN)	IEEE	
3	Standard	HTTP	IS	Kommunikation zwischen WpVCM, LNAP und Lademanagement-System	IETF	
4	Standard	ISO 15031-5	IS	Kommunikation mit dem EV über CAN-Bus, um Ladestatus und Fahrzeugidentifikationsnummer auszulesen	ISO	
5	Standard	SPI	IS	Kommunikation innerhalb des WpVCM Geräts	MOTOROLA INC	
6	Standard	UART	IS	Kommunikation innerhalb des WpVCM Geräts	National Semiconductor	
7	Standard	I2C	IS	Kommunikation innerhalb des WpVCM Geräts	Philips Semiconductors	
8	Standard	IEC 61158	IS	Modbus-Protokoll für Kommunikation zwischen Lademanagement-System und dem Ladekontroller	IEC	
9	Standard	ISO/IEC 20922	IS	MQTT-Protokoll für Kommunikation zwischen Lademanagement-System und dem Ladekontroller	OASIS	

3.4 Beziehung zu anderen Anwendungsfällen und weitere Klassifikationskriterien

Klassifikationsinformationen
Beziehung zu anderen Anwendungsfällen
Gehört zur Gruppe „Demand Side Management“
Detailgrad
Low Level
Priorisierung
Mittel
Generische, regionale oder nationale Einordnung
Fokus: Deutschland
Sichtweise
Technisch
Weitere Schlüsselwörter zur Klassifikation
Messung, Datentransfer, Lademanagement

4 Schrittweise Analyse des Anwendungsfalls

Szenario-Bedingungen					
Nr.	**Szenarioname**	**Primärer Akteur**	**Auslösendes Ereignis**	**Vorbedingung**	**Nachbedingung**
1	Normalfall	EV-Nutzer	Die Daten werden ausgelesen und gesendet	WpVCM hat Zugriff auf die Ladestatusinformationen des EV und auf den LNAP	Das WpVCM stellt die Daten im LNAP für das Lademanagementsystem zur Verfügung.

4.1 Schritte – Normaler Ablauf

Szenario								
Szenarioname:		Normaler Ablauf						
Schritt Nr.	**Ereignis**	**Name des Prozesses/Aktivität**	**Beschreibung des Prozesses/Aktivität**	**Diensttyp**	**Informationssender**	**Informationsempfänger**	**Ausgetauschte Informationen**	**Anforderungs R-ID**
1	Verbindung wird benötigt	Verbindungsanfrage	Das WpVCM prüft den Verbindungsaufbau zum LNAP	REQUEST	WpVCM	LNAP	IP-Adresse	ID01
2	Verbindung wird angefragt	Verbindungsaufbau	Das LNAP baut die Verbindung mit dem WpVCM auf und registriert dessen IP-Adresse	SET	LNAP	LNAP	IP-Adresse	ID02
3	Verbindung wird bestätigt	Verbindungsbestätigung	WpVCM empfängt die Bestätigung der Verbindung vom LNAP	RESPONSE	LNAP	WpVCM	Bestätigungsdaten	ID03
4	Ladestatusinformationen werden benötigt	Ermittlung des Ladestatus	ECU des BEV erfasst den Ladestatus und stellt ihn über CAN-Bus zur Verfügung	GET	BEV ECU	BMS	Messwerte vom Ladestatus	ID04
5	Ladestatusinformationen werden benötigt	Anfrage Ladestatus	Das WpVCM fragt Ladestatusinformationen an	REQUEST	WpVCM	BEV ECU	Messwerte vom Ladestatus	ID05
6	Ladestatusinformationen werden angefragt	Ladestatus	Die angefragte Ladestatusinformation wird zum WpVCM gesendet	RESPONSE	BEV ECU	WpVCM	Messwerte vom Ladestatus	ID06
7	Verarbeitung der empfangenen Daten	Datenverarbeitung	Das WpVCM entschlüsselt und bearbeitet die empfangenen Daten für weitere Prozesse	SET	WpVCM	WpVCM	Messwerte vom Ladestatus	ID07
8	Datenserver muss bereitgestellt werden	Starte den Datenserver	Ein IP basierter Datenserver wird vom WpVCM gestartet	SET	WpVCM	WpVCM	Serverdaten	ID08
9	Ladestatusinformationen werden benötigt	http Request	Das Lademanagementsystem fragt Ladestatusinforationen bei der bekannten IP-Adresse an	REQUEST	LMS	WpVCM	Messwerte vom Ladestatus	ID09
10	Ladestatusinformationen werden empfangen	http Response	WpVCM sendet vorhandene Ladestatusinformationen an LMS	RESPONSE	WpVCM	LMS	Messwerte vom Ladestatus	ID10
11	Optimaler Lade-Plan wird berechnet	Berechnung des optimalen Ladeplans	Das Lademanagementsystem berechnet, u.a. anhand der Ladestatusinformationen, den optimalen Ladeplan für aktuelles BEV	SET	LMS	LMS	Ladeplan Information	ID11

5 Ausgetauschte Informationen

Information Exchanged		
Bezeichnung der ausgetauschten Informationen	**Beschreibung der ausgetauschten Informationen**	**Anforderungen an die Informationsdaten R-ID**
IP-Adresse	Anfrage der Verbindung zu einem WLAN mit gewünschter IP-Adresse	ID12
Bestätigungsdaten	Information über erfolgreiche Anbindung an das WLAN mit definierter IP-Adresse, unter der der Ladestatus abgefragt werden kann	ID13
Messwerte vom Ladestatus	Der Ladestatus der Batterie des EV im hexadezimalen Format	ID14
Serverdaten	Nachdem der WLAN-Server gestartet wurde, werden detaillierte Informationen, wie IP-Adresse, DNS, Subnetz und Gateway-Adresse mitgeteilt	ID15
Ladeplan Information	Das LMS errechnet die optimalen Parameter, wie Leistungsmenge, Dauer und Zeitraum, mit dem das EV geladen werden soll. Anhand dieser Daten ist das Technologiezentrum in der Lage diesen Lade-Plan auch umzusetzen	ID16

6 Verwendete Begriffe und Definitionen

Verwendete Begriffe und Definitionen	
Begriff	**Definition**
LMS-Betreiber	Ist ein Mitarbeiter des Technologiezentrums, der für Wartung, Einstellung und Überwachung des Lademanagement-Systems sowie dazugehörigen Komponenten zuständig ist.
BEV-Nutzer	Ein Mitarbeiter des Technologiezentrums mit autorisierter Nutzung des batterieelektrischen Fahrzeugs.
Demand Side Management	Mit Demand Side Management können Stromverbraucher ihren Energieverbrauch beeinflussen. Durch diese Last-Steuerung lassen sich die schwankende Stromerzeugung aus Erneuerbare-Energien-Anlagen ausgleichen und neue Erwerbsquellen erschließen.
BEV ECU	Electronic Control Unit, Steuergerät des batterieelektrischen Fahrzeugs
BMS	Battery Management System, ist ein elektronisches Bauteil, es dient der Überwachung und Kontrolle des Ladevorgangs der Autobatterie.

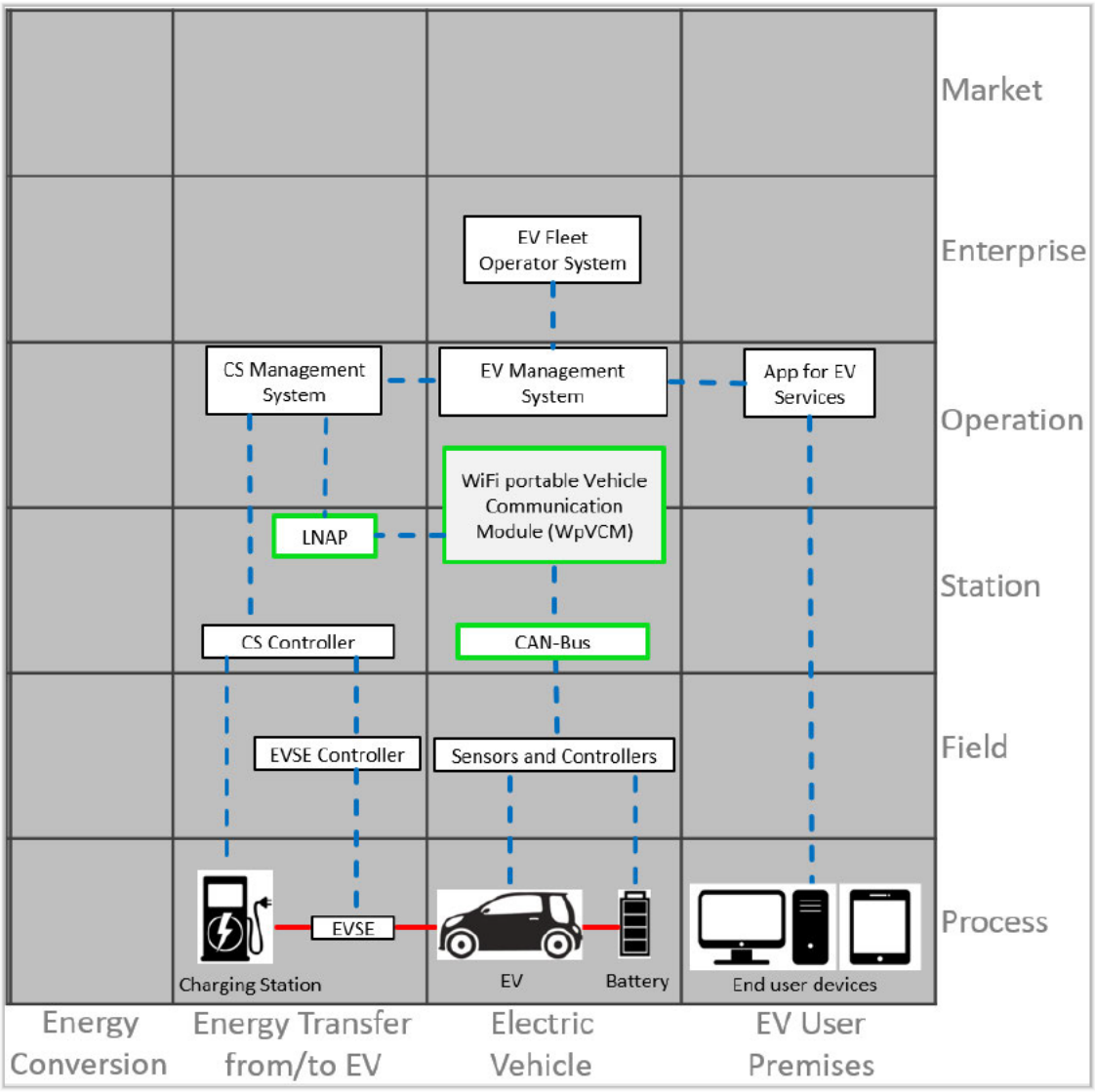


Abbildung A.4: Component Layer

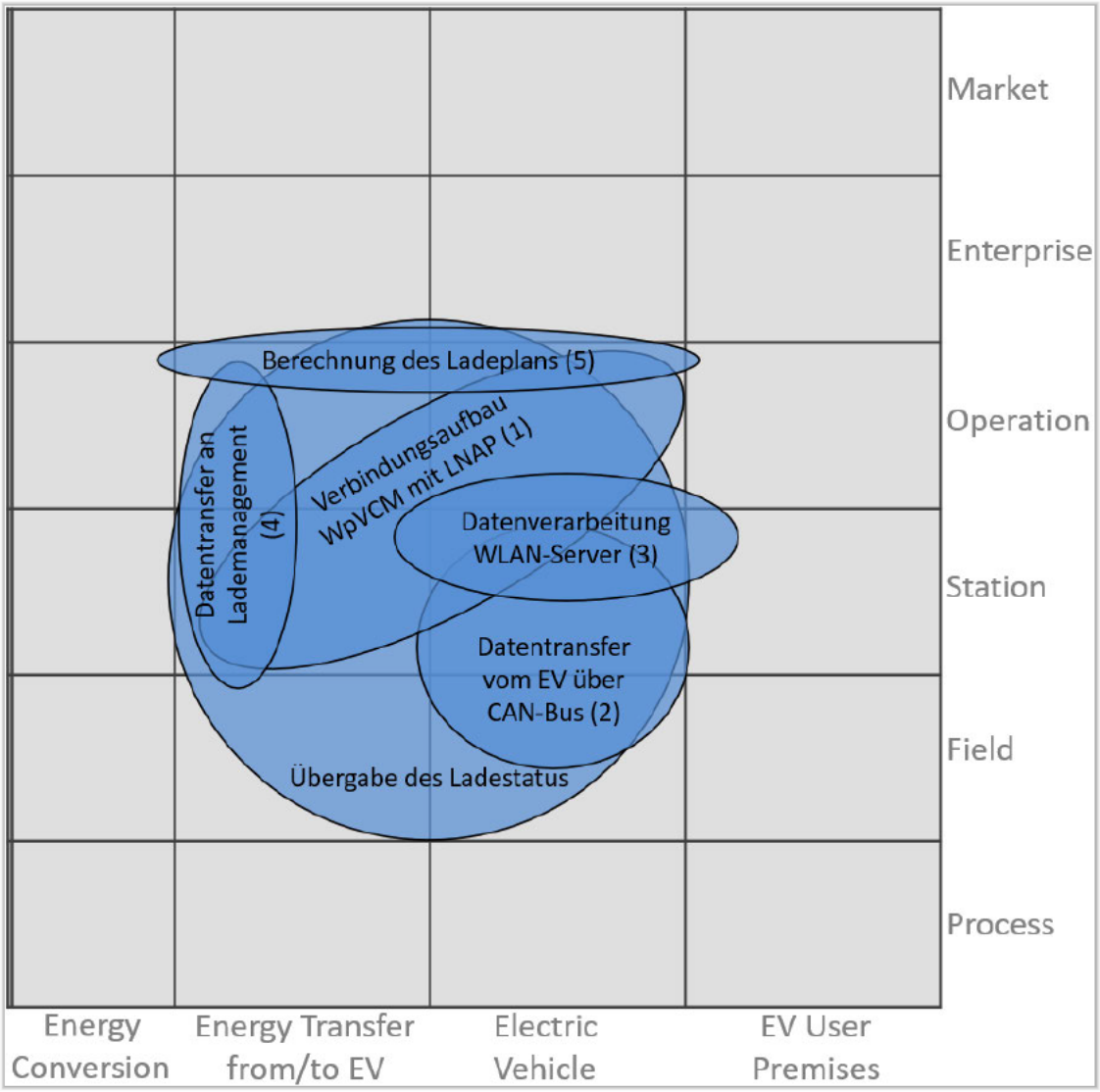


Abbildung A.5: Function Layer

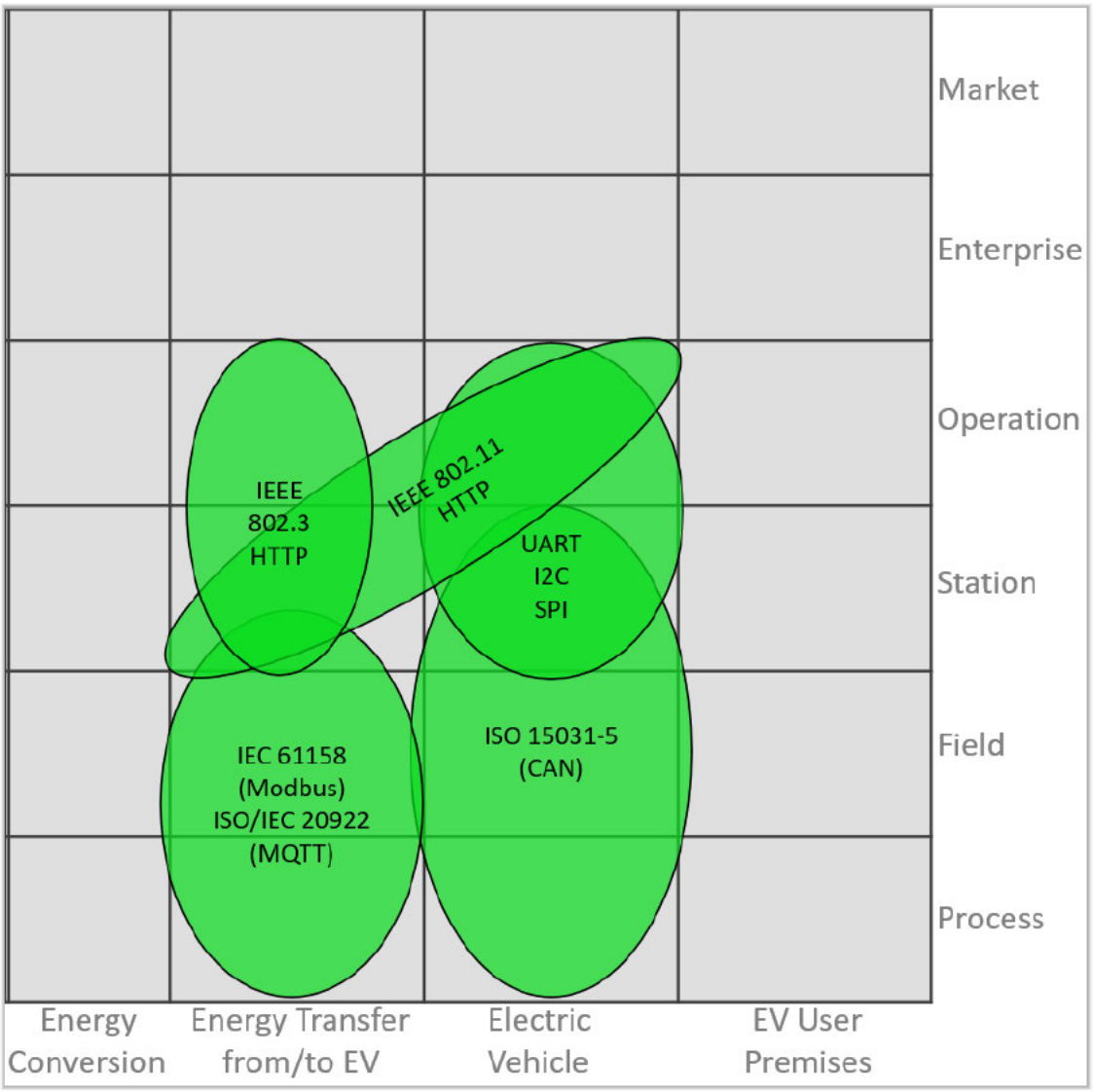


Abbildung A.6: Communication Layer

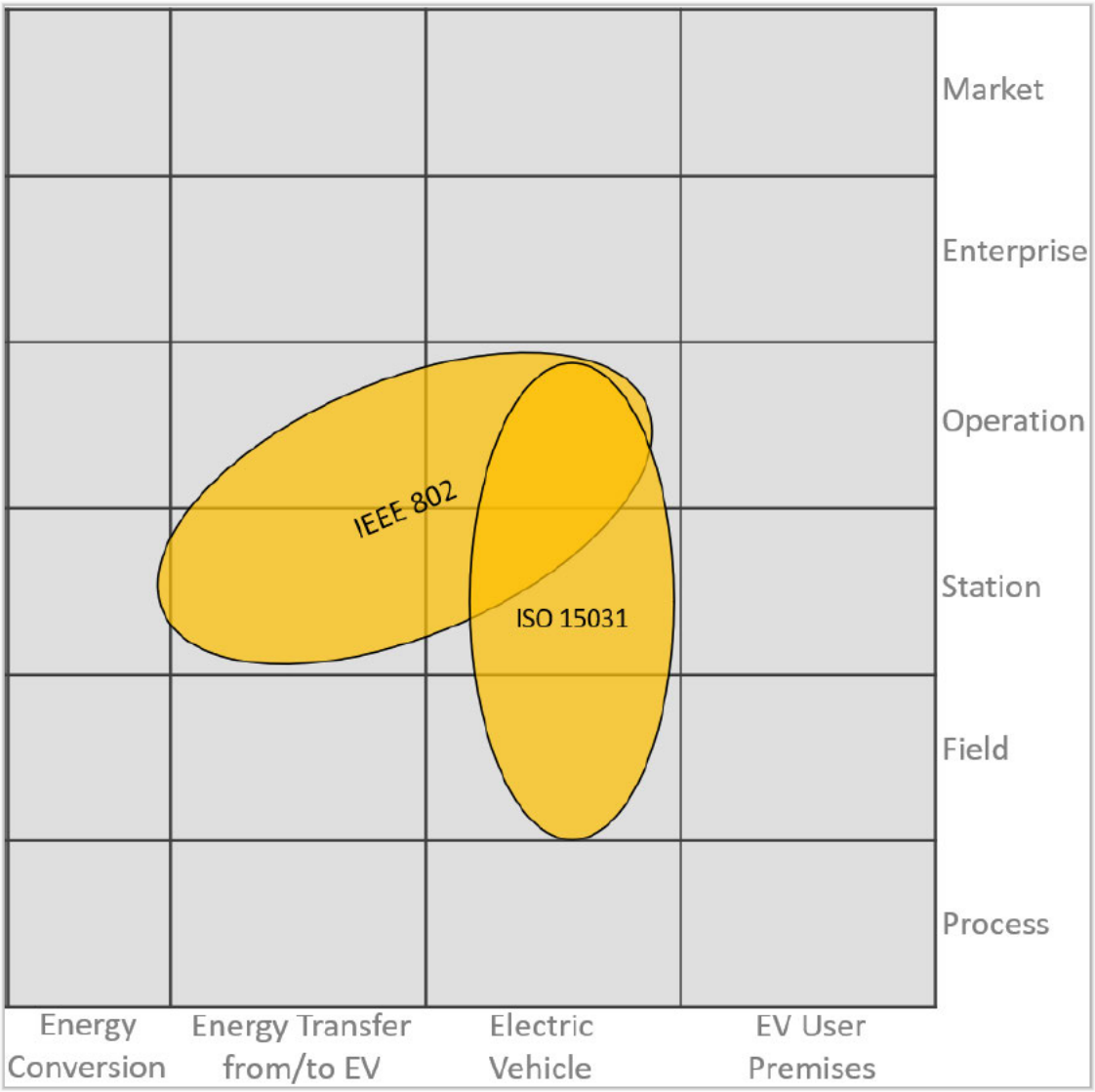


Abbildung A.7: Information Layer

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne Inanspruchnahme fremder Hilfe angefertigt habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen. Ich erkläre mich damit einverstanden, dass die Arbeit mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate überprüft wird.

Ort	Datum	Unterschrift im Original