UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Entwicklung eines Systems zur Integration und zum Monitoring von Elektrokrafträdern im Carsharing Betrieb

Developing a system for integration and monitoring of electrically powered two-wheeler in car sharing context

Masterarbeit

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
B.Sc. Martin Blankenburg

ausgegeben und betreut von
Prof. Dr. Martin Leucker

Lübeck, den 30. November 2013

„*Unsere Einstellung der Zukunft gegenüber muß sein:*
Wir sind jetzt verantwortlich für das, was in der Zukunft geschieht.“

„*Während die Philosophen noch streiten, ob die Welt überhaupt existiert,*
geht um uns herum die Natur zugrunde.“

Karl Raimund Popper (1902-94),
brit. Philosoph u. Wissenschaftslogiker östr. Herk.

Kurzfassung

Carsharing erfreut sich in Deutschland zunehmender Beliebtheit. Fast eine halbe Million Nutzer sind allein in Deutschland registriert. Durch den Einsatz von Elektrofahrzeugen im Carsharing wird langfristig zum Klimaschutz beigetragen. Die hier vorliegende Arbeit soll einen Beitrag dazu leisten, Elektrokrafträder auf einfache Art und Weise im Carsharing Betrieb nutzbar zu machen.
Zu Beginn wird der Kontext, in dem das System verwendet werden soll, und einige grundlegende verwendete Technologien beschrieben. In der darauf folgenden Analyse werden die Teilsysteme und Vorgänge, die in diesem Kontext auftauchen und durchgeführt werden müssen, analysiert. Auf der Grundlage dieser Analysen werden Soft- und Hardwarekomponenten des Systems konzipiert und Teilsysteme und deren Funktion definiert, bevor auf die Umsetzung des Konzeptes eingegangen wird. Am Ende dieser Arbeit werden die wesentlichen Punkte zusammengefasst und ein Ausblick auf mögliche Erweiterungen des Systems gegeben.

Schlüsselwörter

Elektromobilität, Carsharing, Monitoring, Webtechnologien

Abstract

Car sharing enjoys great popularity within the Federal Republic of Germany. Almost half a million citizens are currently registered to local car sharing providers. Utilizing electric vehicles for car sharing is conductive to the preservation of the environment. This thesis shall make a contribution to an easy integration of electrically powered two-wheelers into the context of car sharing.
As an introduction, a concise review of the context in which the system will be used is given, followed by detailed analysis of system components and processes involved in the car sharing context. On the basis of these reviews and analysis, soft- and hardware components of the system are defined and the implementation of the system's concept is described. An outlook on future extensions to the system is presented at the end of this thesis.

Keywords

Electromobility, car sharing, monitoring, web technology

Danksagungen

Ich danke meinem Betreuer Prof. Dr. Martin Leucker dafür, dass er mir diese Arbeit ermöglicht und viele Kontakte hergestellt hat. Außerdem war es kein Problem, die benötigte Hardware zu bekommen.

Ich danke

Dipl.-Kfm. Markus Spiekermann von Move About in Bremen für die Bereitstellung des Govecs GO! S1.2

Rafał Stankiewicz von Govecs Poland Sp. z.o.o. für seine Unterstützung bei der Reparatur des Ladegerätes

Dipl.-Inf. Thomas Tosik vom Institut für Technische Informatik für die Bereitstellung des Messgerätes und diverser Kleinteile

M.Sc. Grigori Goronzy für seine Unterstützung bei der Fehlersuche und -behebung an dem ersten CAN-Bus Adapter

Dipl.-Inf. Oliver Kleine und Dirk Frank Schmidt vom Institut für Telematik für die vielen konstruktiven Gespräche während der Zigarettenpausen

Dipl.-Inf. Marc Paul vom Institut für Multimediale und Interaktive Systeme für die Entwicklung von UsER und die Bereitstellung eines Accounts

Dr. Jork Milde vom Institut für Multimediale und Interaktive Systeme für die Bereitstellung diverser Kleinteile

M.Sc. René Schönfelder für viele hilfreiche Tipps und seinen Workshop zum Schreiben von wissenschaftlichen Arbeiten

B.Sc. Sven Schlinga für seine Beratungen zum Thema Sicherheit

M.Sc. Franziska Kühn und B.Sc. Syavoosh Khabbazzadeh für Korrekturlesen und Verbesserungsvorschläge

Inhaltsverzeichnis

1 Einleitung

Umweltschutz, erneuerbare Energien und Elektromobilität sind in den letzten Jahren immer wieder Gegenstand der Diskussion in Politik, Medien und Gesellschaft. Der zunehmende Klimawandel und die knapper werdenden Ressourcen an fossilen Brennstoffen zwingen uns zu einem Umdenken in der Art und Weise, wie wir uns in Zukunft fortbewegen wollen und Energie gewinnen und verbrauchen. Das hat auch die Bundesregierung erkannt und mit dem Nationalen Entwicklungsplan Elektromobilität vom August 2009 erstmals nationale Ziele für Elektromobilität formuliert. Dabei kann Elektromobilität nur dann einen Beitrag zum Umweltschutz leisten, wenn der verwendete Strom aus erneuerbaren Energien stammt. Effiziente Elektrofahrzeuge, moderne Energiespeichertechnologien und nicht zuletzt ein sparsamer Umgang mit dieser Energie leisten weitere wichtige Beiträge zum Schutz unserer Umwelt ([Mühlenhoff, 2010]).
Carsharing erfreut sich in Deutschland zunehmender Beliebtheit. Zur Zeit nutzen über 450.000 Teilnehmer die Angebote von etwa 110 Carsharing Betreibern und greifen dabei auf fast 12.000 Fahrzeuge zurück. Durch die hohe Nutzerzahl pro Fahrzeug wird eine hohe Auslastung des Fahrzeugs erreicht und die Kosten für Betreiber und Nutzer minimiert. Die verstärkte Nutzung von Carsharing Angeboten geht auch mit einer sinkenden Zahl an zugelassenen Fahrzeugen einher, da Carsharing Nutzer verstärkt auf eigene Kraftfahrzeuge verzichten. Auf diese Weise leistet Carsharing einen konkreten Beitrag zur Entlastung der Umwelt.
Carsharing mit Elektrofahrzeugen kombiniert die Vorteile dieser beiden Felder und kann Synergien nutzen, um langfristig zum Klimaschutz beizutragen. Die hier vorliegende Arbeit soll einen Beitrag dazu leisten, Elektrokrafträder auf einfache Art und Weise im Carsharing Betrieb nutzbar zu machen.

1.1 Zielsetzung

Das Ziel dieser Arbeit ist es, ein System zu entwerfen, welches es ermöglicht, Elektrokrafträder in den Carsharing Betrieb zu integrieren und deren Zustand zu überwachen. Das System soll dabei möglichst wenig Energie verbrauchen, nur geringe Kosten verursachen und leicht erweiterbar und wartbar sein.
Für den Prototypen des Systems soll nur Hardware verwendet werden, die für jedermann erhältlich ist. Software-Komponenten, die nicht selbst implementiert werden, sollen quelloffen und frei sein, damit das System nach Belieben verbreitet, verändert und erweitert werden kann. Die Architektur

des Systems soll dabei so konzipiert werden, dass einzelne Komponenten leicht austauschbar sind und um weitere Funktionalitäten ergänzt werden können. Zusätzliche Komponenten sollen nach Möglichkeit im laufenden Betrieb hinzugefügt werden können, ohne dabei das System zu beeinträchtigen.
In das System sollen bereits abgeschlossene Arbeiten, die am Institut für Softwaretechnik und Programmiersprachen der Universität zu Lübeck durchgeführt wurden, integriert werden. Zu diesen Arbeiten gehören eine Smartphone-App zur Buchung eines Fahrzeugs bei einem Carsharing Anbieter ([Kampsen, 2013]), die kontaktlose Verteilung virtueller Fahrzeugschlüssel ([Block, 2012]) und ein Navigationssystem für energieoptimiertes Routing (GreenNav).

1.2 Aufbau der Arbeit

Zu Beginn wird der Kontext, in dem das System verwendet werden soll, und einige grundlegende verwendete Technologien beschrieben. In der darauf folgenden Analyse werden die Teilsysteme und Vorgänge, die in diesem Kontext auftauchen und durchgeführt werden müssen, dargestellt und analysiert. Dazu werden verschiedene bestehende Systeme untersucht, die in den Ausleihvorgang involviert sind, um einen Überblick über die beteiligten Komponenten und Teilsysteme zu geben. Eine Untersuchung verschiedener Cockpits von Elektromotorrollern gibt Aufschluss über die darzustellenden Informationen während der Fahrt. Eine Benutzeranalyse mit Personas und Szenarien verschafft Einblicke in die Nutzung des Systems aus der Sicht verschiedener Stakeholder. Zum Abschluss der jeweiligen Analysen werden die Features des Systems für die anschließende Konzeption extrahiert.
Auf die Soft- und Hardwarekomponenten des Systems sowie deren Zusammenspiel wird in Kapitel 4 ausführlich eingegangen. Dabei steht zunächst die grundlegende Architektur der Software, die Datenstrukturen und die Kommunikationsschnittstellen im Vordergrund. Es werden Teilsysteme und deren Funktion innerhalb des Gesamtsystems konzipiert und auf Sicherheitsaspekte eingegangen. Anforderungen an die Hardware werden festgelegt, bevor in Abschnitt 4.4.3 die Interaktionsmöglichkeiten des Benutzers mit dem System festgelegt werden. Ein Entwurf der ersten Anzeigeinstrumente rundet die Konzeption ab.
Darauf folgt eine Beschreibung der Realisierung des Systems, wobei zunächst auf das verwendete Fahrzeug und die Hardware-Komponenten eingegangen wird. Abschnitt 5.2 beschäftigt sich mit den Software-Komponenten des Systems. Beginnend bei dem Betriebssystem, welches auf dem Prototypen zum Einsatz kommt, werden anschließend die Implementierungen des Datenservers und der Clients für den CAN-Bus Zugriff und die Anzeige vorgestellt.
Am Ende dieser Arbeit werden die wesentlichen Punkte noch einmal zusammengefasst. Es wird auf Teile des Konzeptes, die bisher nicht realisiert wurden, eingegangen und ein Ausblick auf mögliche Erweiterungen des Systems gegeben.

2 Grundlagen

Bevor mit der Analyse des Kontextes begonnen wird, sollen in diesem Kapitel einige Grundlagen diskutiert werden, um sowohl einen ersten Einblick in den Kontext zu erlangen als auch getroffene Designentscheidungen besser nachvollziehen zu können. Dazu werden zunächst einige Aspekte des Carsharing und der Elektromobilität erläutert und eine Verbindung dieser beiden Felder hergestellt. Daraufhin werden Technologien und Konzepte erklärt, die in dem hier zu entwickelnden System Verwendung finden.

2.1 Carsharing und Elektromobilität

Der Bundesverband CarSharing e.V. definiert Carsharing als „die organisierte, gemeinschaftliche Nutzung von Kraftfahrzeugen“ ([Bundesverband CarSharing e.V., 2013]). Im Gegensatz zur klassischen Autovermietung können beim Carsharing Fahrzeuge auch kurzfristig und kurzzeitig ausgeliehen werden. Bevor ein Fahrzeug genutzt werden kann, ist eine Registrierung bei dem Carsharing Betreiber und der Nachweis einer Fahrerlaubnis erforderlich. Bei der konventionellen Autovermietung ist in der Regel keine Registrierung notwendig.

Die Nutzung von Carsharing Angeboten verzeichnet in den letzten Jahren hohe Wachstumsraten, wie in Abbildung 2.1 zu sehen ist. Dies hat positive Auswirkungen vor allem auf die Umwelt und das innerstädtische Verkehrsaufkommen. In mehreren Studien konnte nachgewiesen werden, dass Carsharing Nutzer teilweise ihre eigenen Fahrzeuge abschaffen, da ihr Bedarf an Mobilität durch das Carsharing Angebot und verstärkte Nutzung des öffentlichen Personennahverkehrs gedeckt werden kann. Die Einführung von Free Floating Angeboten, bei denen die Kunden die Fahrzeuge nicht mehr an festen Entleihstationen abholen und zurückgeben müssen, sorgte für einen zusätzlichen Anstieg der Kundenzahlen. Zudem hat die Nutzung von Carsharing Angeboten für den Kunden den Vorteil, dass er sich nicht um Wartung, Pflege und sonstigen Service zu kümmern braucht, da diese Dienstleistungen von dem Carsharing Betreiber erbracht werden.

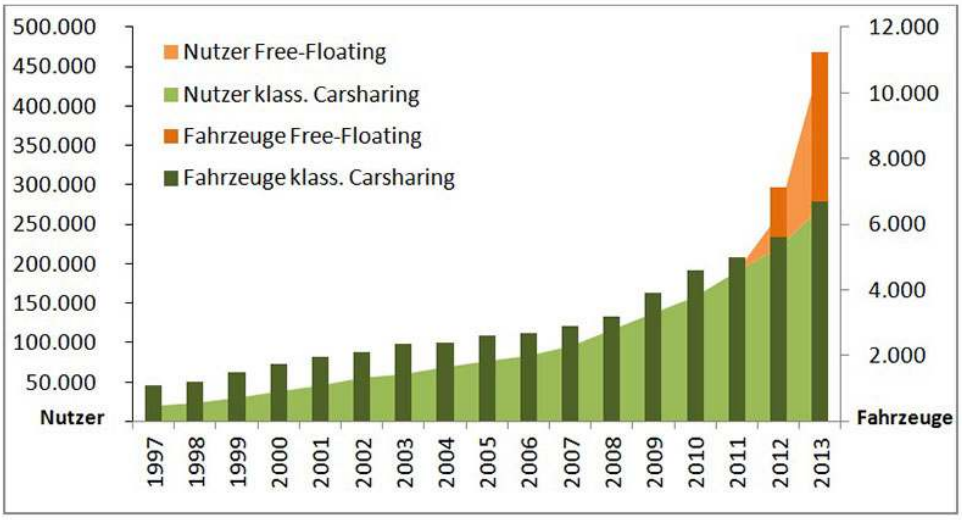

Abbildung 2.1: Entwicklung des Carsharing in Deutschland seit 1997[1]

Eine Befragung von Fuhrpark Managern deutscher Unternehmen zeigte zudem, dass die gewerbliche Nutzung, das sogenannten Corporate Carsharing, Einsparpotenzial für die Unternehmen bietet. Zu einer Nutzung des Corporate Carsharing erklärte sich jedes zweite Unternehmen bereit. Darüber hinaus stößt Carsharing in Kombination mit Elektromobilität auf großes Interesse bei den befragten Unternehmen ([Metropolregion, 2013]).

Elektromobilität ist laut Bundesministerium für Verkehr, Bau und Stadtentwicklung eine „Schlüsseltechnologie für ein nachhaltiges Verkehrssystem“ ([BMVBS, 2013a]). Durch Förderprogramme wie die *Nationale Plattform Elektromobilität* (NPE) und *Elektromobilität in Modellregionen* und einer Fördersumme von knapp zwei Milliarden Euro sollen auf Deutschlands Straßen im Jahr 2020 eine Million Elektrofahrzeuge unterwegs sein. Allerdings sind die USA und Japan dem deutschen Markt weit voraus, was die Verbreitung von Elektrofahrzeugen anbelangt (vgl. Abb. 2.2).

Aufgrund der Fördermittel, der Umweltentlastung und nicht zuletzt der Öffentlichkeitswirkung bieten sich große Potentiale für die Kombination aus Carsharing und Elektromobilität. Jedoch halten vor allem die vergleichsweise hohen Anschaffungskosten und die geringe Erfahrung im Umgang mit dieser Technologie viele Carsharing Betreiber davon ab, ihre Flotte auf Elektrofahrzeuge umzurüsten. Andere Betreiber wiederum, wie beispielsweise *Move About*[2], betreiben in ihrer Flotte ausschließlich Elektrofahrzeuge ([MoveAbout, 2013]).

[1] Quelle: http://www.vcd.org/uploads/pics/carsharing_tabelle_nutzung_2013.jpg
[2] http://www.moveabout.biz

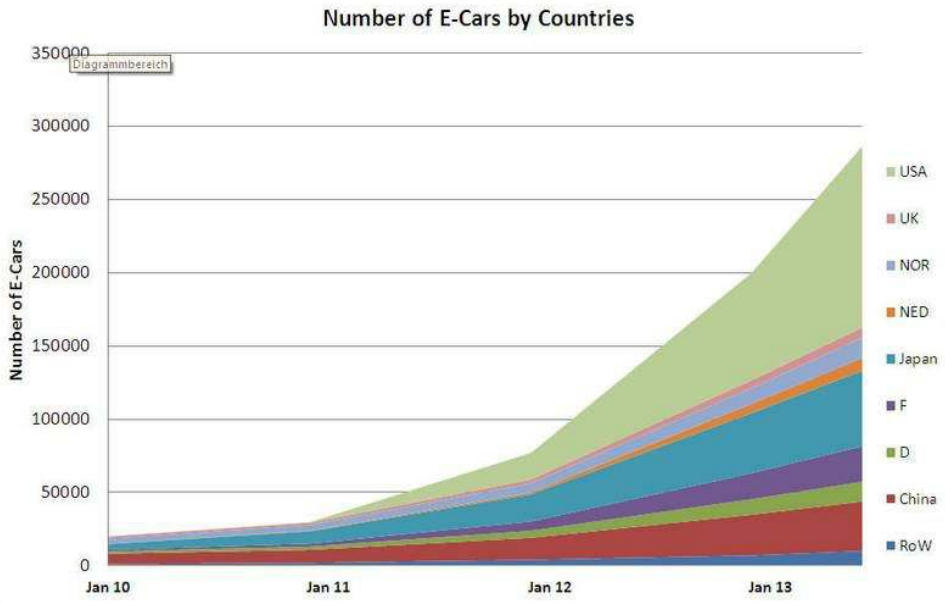

Abbildung 2.2: Anzahl der Elektrofahrzeuge international[3]

2.2 Controller Area Network (CAN)

Das Controller Area Network (CAN) ist ein serielles Kommunikationsprotokoll, welches Echtzeitkontrolle der angeschlossenen Geräte und Sensoren ermöglicht. In Fahrzeugen wird es dazu verwendet, um beispielsweise Motorkontrolleinheit, Anti-Blockier-System, Sensoren etc. miteinander zu verbinden, wobei Datenraten bis zu 1Mb/s erreicht werden ([Robert Bosch GmbH, 1991]).
Das CAN Protokoll wurde in der Version 1.0 im Jahr 1986 von Bosch und Intel vorgestellt und existiert seit 1991 in der Version 2.0. Seit 1993 existiert der ISO-Standard 11898, der mittlerweile aus sechs Teilen besteht ([ISO, 2013]). Die Spezifikation in Version 2.0 wird in Part A und Part B unterschieden. Part A beschreibt das Nachrichtenformat, wie es in Version 1.2 definiert wurde. In Part B wird dieses Nachrichtenformat um einen größeren Adressraum erweitert, damit mehr Teilnehmer an den CAN-Bus angeschlossen werden können. Es können dabei Geräte beider Spezifikationen an ein und demselben Bussystem betrieben werden, zumal die Verwendung des erweiterten Adressraumes in der Spezifikation Part B keine Pflicht ist. Im Jahr 2012 wurde die CAN 2.0 Spezifikation zu CAN FD (CAN with flexible Data-Rate) erweitert. Das CAN FD Protokoll ermöglicht höhere Datenübertragungsraten und größere Datenpakete ([Robert Bosch GmbH, 2012]). Aufgrund der Tatsache, dass in dem Fahrzeug für den Prototypen des hier zu entwickelnden Systems ein CAN-Bus nach Version 2.0B integriert ist, beschränken sich die weiteren Erläuterungen

[3] Quelle: http://www.energie-tipp.de/_storage/asset/4227787/storage/etipp_lightbox_image/file/381633820/27099179.jpg

auf ebd. Spezifikation.
Version 2.0B des Protokolls ist in drei Schichten aufgeteilt:

1. Logical Link Control (LLC) Sublayer
   - stellt Dienste für Datentransfer und -anfragen bereit
   - entscheidet, ob empfangene Nachrichten akzeptiert werden
   - Fehlermanagement und Überlastbenachrichtigungen

2. Medium Access Control (MAC) Sublayer
   - Busarbitrierung und Acknowledgement
   - Fehlerentdeckung und -signalisierung
   - Framing
   - Serialisierung und Deserialisierung

3. Physical Layer
   - Bitübertragung
   - Timing und Synchronisation
   - Kodierung

LLC und MAC Sublayer sind auf der Sicherungsschicht des OSI-Referenzmodells angesiedelt, der Physical Layer auf der Bitübertragungsschicht. Die Spezifikation des Physical Layer macht hierbei keinerlei Vorgaben zum Übertragungsmedium oder den Signalpegeln, solange diese Parameter konsistent über das gesamte Bussystem gehalten werden. Der LLC Sublayer ist in erster Linie für die Filterung von Nachrichten zuständig. Er gibt empfangene und akzeptierte Nachrichten an höhere Schichten bzw. Protokolle weiter und reicht Nachrichten dieser Schichten an den MAC Sublayer weiter, damit diese entsprechend verpackt und versendet werden können. Der MAC Sublayer bildet den Kern der CAN Spezifikation. In ihm werden die Nachrichten ver- und entpackt, Fehler detektiert und Fehlermeldungen ausgesendet, der Zugriff auf das Übertragungsmedium gesteuert und die Nachrichten vor dem Versenden serialisiert und nach dem Empfang deserialisiert.
Der CAN-Bus ist ein Multicast- und Multimaster-Bussystem. Das bedeutet, dass bei unbelegtem Bus jeder angeschlossene Teilnehmer eine Nachricht versenden kann, die von jedem anderen Teilnehmer empfangen wird. Ein Teilnehmer entscheidet dann je nach Funktion, ob und wie er auf eine solche Nachricht reagiert. Jeder Nachrichtentyp hat innerhalb des Bussystems eine eindeutige ID, welche neben der Nachrichtenidentifikation auch zur Priorisierung verwendet wird. Eine niedrigere ID bedeutet eine höhere Priorität. Bei der Arbitrierung bekommt immer diejenige Nachricht Buszugriff zugesprochen, welche die höhere Priorität besitzt. Bereits in Übertragung befindliche Nachrichten werden dabei nicht unterbrochen.
Das CAN Protokoll kennt vier unterschiedliche Nachrichtentypen. Data Frames werden von Teilnehmern dazu genutzt, um Sensorwerte oder Steuersignale zu versenden. Durch Senden eines Remote Frames können andere Teilnehmer dazu veranlasst werden, Data Frames zu senden. Dadurch

wird eine Art Request-Response-Mechanismus realisiert. Zur Signalisierung von Fehlern werden Error Frames verwendet. Diese können von jedem Teilnehmer gesendet werden, sobald dieser einen Fehler feststellt. Der letzte Nachrichtentyp ist der Overload Frame. Er wird verwendet, um eine Verzögerung zwischen zwei Data Frames bzw. einem Remote und einem Data Frame zu erzeugen, wenn der intrinsische Zustand eines Teilnehmers gerade kein Senden eines Data oder Remote Frames erlaubt, weil dieser beispielsweise noch mit der Verarbeitung der vorherigen Nachricht beschäftigt ist.
Damit ein CAN-Bus mit einem PC System verbunden werden kann, bedarf es eines CAN Controllers, der als weiterer Teilnehmer an den Bus angeschlossen wird. Die Treiber solcher Controller legen dann ein zeichenorientiertes serielles Gerät (character device) im System an und senden und empfangen Nachrichten durch dieses Gerät. Dadurch ist zum einen der Zugriff auf das Gerät abhängig von dem verwendeten Treiber, zum anderen ist das Gerät für alle Prozesse blockiert, sobald ein Prozess darauf zugreift. Um diese Schwächen zu beheben und einheitlichen parallelen Zugriff auf den CAN Controller zu gewähren, wurde das Low Level CAN Framework (LLCF), auch bekannt als SocketCAN, von der Volkswagen AG entwickelt. SocketCAN erweitert die Berkeley Socket API um die Protokollfamilie PF-CAN und schafft so eine Abstraktion von dem Treiber des Controllers. Dadurch wird auch der Zugriff auf den CAN-Bus durch mehrere Prozesse möglich ([Hartkopp und Thürmann, 2006]).

2.3 Representational State Transfer (REST)

REST ist ein von Roy Fielding entwickeltes Programmierparadigma, welches er in seiner Dissertation „Architectural Styles and the Design of Network-based Software Architectures“ im Jahr 2000 vorgestellt hat. Es definiert eine Menge von Anforderungen an die Architektur eines verteilten Systems und versucht dabei, Latenzen und Netzwerkkommunikation zu minimieren, während gleichzeitig die Unabhängigkeit und Skalierbarkeit der Implementierung der Komponenten maximiert wird ([Fielding, 2000]).
Der Fokus liegt hierbei auf den Ressourcen eines Systems und wie diese adressiert und übertragen werden können. Um eine Ressource zu identifizieren, werden *Uniform Resource Identifyer* (URI) verwendet. Diese sollen leicht verständlich sein, sodass sie von Menschen und Maschinen gelesen und produziert werden können. Daher bietet es sich an, eine URI ähnlich wie einen Verzeichnisbaum eines Dateisystems aufzubauen, z.B. http://www.myblog.org/{year}/{month}/{day}/{topic}. Die URI soll dabei statisch sein. Wenn sich der Wert einer Ressource oder die Implementierung des Services ändert, bleibt die Adresse der Ressource erhalten ([Rodriguez, 2008]).
Für die Repräsentation einer Ressource und deren Attribute können verschiedene Formate wie *Extensible Markup Language* (XML), *JavaScript Object Notation* (JSON) oder Klartext zum Einsatz kommen. Ein Client kann anhand des MIME-Types die Art der Repräsentation der Ressource wäh-

rend des Requests wählen. Der Zugriff auf die Ressourcen erfolgt über die HTTP-Methoden POST (Erstellen), GET (Anfordern), PUT (Ändern) und DELETE (Löschen), welche die grundlegenden Datenbankoperationen CRUD (Create, Read, Update, Delete) umsetzen.
Der Vorteil einer REST Architektur ist ihre Zustandslosigkeit. Ein Client sendet alle für die Anfrage nötigen Informationen bei einem Request an den Server, der diesen unabhängig von anderen Anfragen bearbeitet. Es müssen keine Mechanismen für Load-Balancing oder Clustering eingesetzt werden, um beispielsweise Sitzungsdaten zu speichern oder zu synchronisieren. Des Weiteren lassen sich REST-Server Anwendungen leichter verteilen und aufsetzen.

3 Analyse

Damit ein geeignetes System für Elektrokrafträder zur Integration in den Carsharing Betrieb konzipiert werden kann, müssen zunächst die Vorgänge verstanden werden, die in diesem Kontext auftauchen und durchgeführt werden müssen. Um dieses Verständnis zu fördern, werden in diesem Kapitel die Teilsysteme und Vorgänge dargestellt und analysiert. Anhand der Analysen lassen sich dann benötigte Features des zu entwickelnden Systems für die anschließende Konzeption extrahieren.
Zu Beginn werden verschiedene bestehende Systeme untersucht, die in den Ausleihvorgang involviert sind. Dies verschafft einen Überblick über die beteiligten Komponenten und Teilsysteme. Betrachtet werden darüber hinaus auch Teilsysteme, die das Fahrzeug selbst betreffen. Im Anschluss daran werden die primären Stakeholder des Systems identifiziert. Szenarien aus Sicht ebd. Stakeholder geben Aufschluss über die Vorgänge im Einzelnen und definieren die zu realisierenden Features.

3.1 Vergleich bestehender Systeme

In diesem Abschnitt werden Teilsysteme untersucht, die im Carsharing Kontext Verwendung finden. Es werden Gemeinsamkeiten aufgezeigt und Unterschiede in ihren Ansätzen diskutiert. Den Einstieg bilden Systeme zur Buchung eines Fahrzeugs. Betrachtet werden die Portale etablierter Carsharing Betreiber und die Vorgänge von der Reservierung bis zur Rückgabe des Fahrzeugs. Verfahren zur Realisierung digitaler Fahrzeugschlüssel werden in Abschnitt 3.1.2 vorgestellt. Im Anschluss daran werden die Cockpits aktueller Elektrokrafträder genauer inspiziert, um identifizieren zu können, welche Informationen dem Fahrer während der Fahrt in welcher Weise dargeboten werden. Navigationssysteme, die dem Benutzer eine möglichst energiesparende Route vorschlagen, bilden den Abschluss der Vergleiche.

3.1.1 Fahrzeugbuchung

Systeme zur Buchung und Reservierung von Fahrzeugen gibt es ebenso viele, wie es Carsharing Betreiber gibt. Bei den meisten Anbietern gibt es sogar mehrere Wege, ein Fahrzeug zu buchen, nämlich telefonisch, über die Webseite des Anbieters oder mit Hilfe von Smartphone-Apps. Für

diesen Vergleich wurden die Portale mehrerer Carsharing Betreiber untersucht. Zu den betrachteten Anbietern gehören:

- Move About [MoveAbout, 2013]
- stadtmobil [stadtmobil, 2013]
- StattAuto [StattAuto, 2013]
- Greenwheels [Greenwheels, 2013]
- Car2go [Car2go, 2013]
- Zipcar [Zipcar, 2013]
- City Car Club [CityCarClub, 2013]
- Autolib [Autolib, 2013]

In ihrer Essenz teilen sich die Systeme folgende Eigenschaften:

Registrierung als Neukunde

Neben persönlichen Daten wie Name, Geburtsdatum, Anschrift, Telefonnummer, Email und Bankverbindung (Konto oder Kreditkarte) wird die Vorlage eines gültigen Führerscheins verlangt. Einige Anbieter erlauben das Hochladen eines Scans des Führerscheins auf deren Webseite oder das Zusenden einer Kopie per Post, während *stadtmobil* eine persönliche Vorlage des Führerscheins in einer Niederlassung vor Ort verlangt. Bei *StattAuto* erfolgt die Anmeldung ausschließlich in der örtlichen Geschäftsstelle.

Auswahl eines Abholstandortes

Sowohl Anbieter mit festen Abhol- und Rückgabestationen als auch solche mit Free Floating Flotte bieten ihren Kunden beim Entleihen eines Fahrzeugs eine Karte im Stil von Google Maps auf ihren Portalen an, auf der die Standorte der Fahrzeuge markiert sind. Um diesen Service anbieten zu können, müssen die Fahrzeuge mit GPS oder einer anderen Form von Ortsbestimmung (z.B. Triangulation über Mobilfunkmasten oder bekannter WLANs in der Umgebung) ausgestattet sein. Der Kunde kann sich dann für einen für ihn günstig zu erreichenden Standort entscheiden. Wird die Buchung von einem mobilen Endgerät aus vorgenommen, nutzen einige Anbieter auch die aktuellen Standortdaten des Kunden, um ihm eine Anfahrt zum Fahrzeug mit öffentlichen Verkehrsmitteln vorzuschlagen.

Auswahl eines Fahrzeugs

Befinden sich an einem Standort mehrere Fahrzeuge des Anbieters, kann der Kunde sich für eines dieser Fahrzeuge entscheiden. Neben dem gewünschten Ausleihzeitraum in der Regel auch die An-

gabe der Länge der Strecke vonnöten, die der Kunde zurückzulegen beabsichtigt. Dadurch kann der Betreiber z.B. bei Elektrofahrzeugen die anschließende Ladezeit besser abschätzen. Nach Eingabe der Daten kann dem Benutzer mitgeteilt werden, ob das gewünschte Fahrzeug zu dem beabsichtigten Ausleihzeitpunkt auch über ausreichend Energie verfügt, die anvisierte Strecke zurücklegen zu können.

Zugang zum Fahrzeug

Handelt es sich bei dem zu entleihenden Fahrzeug um ein Auto, erfolgt die Freischaltung bei allen betrachteten Anbietern durch eine Mitgliedskarte mit RFID Chip. An der Windschutzscheibe des Fahrzeugs befindet sich ein entsprechendes Lesegerät, welches bei erfolgreicher Verifikation der Mitgliedskarte die Türen öffnet. Der Zündschlüssel befindet sich in einem Fach innerhalb des Fahrzeugs. Während des Mietzeitraumes kann das Fahrzeug wie gewohnt mit dem Zündschlüssel verschlossen und wieder geöffnet werden. Erst bei Abschluss des Entleihvorganges wird der Schlüssel wieder im Fahrzeug deponiert und das Fahrzeug mit der Mitgliedskarte verriegelt, wodurch der Buchungsvorgang abgeschlossen wird. Ausnahmen bilden hierbei *Move About*, bei deren Fahrzeugen die Freischaltung auch per Smartphone-App und Eingabe einer PIN erfolgen kann, und *StattAuto*, bei denen die Fahrzeugschlüssel in einem per Mitgliedskarte zu öffnenden Tresor an den Entleihstationen deponiert werden. Wird das Fahrzeug bei der Rückgabe nicht wieder an eine Stromquelle angeschlossen, ist die Zahlung einer Strafgebühr fällig.
Da Zweiräder in der Regel nicht über verriegelbare Türen verfügen, kann dieses Verfahren hier in dieser Form keine Anwendung finden. Um diesem Problem zu begegnen, werden in Abschnitt 3.1.2 Verfahren für digitale Fahrzeugschlüssel untersucht, die sich auf Zweiräder anwenden lassen.

Tabelle 3.1: Featureliste Fahrzeugbuchung

Nr.	Feature	Beschreibung
F1	Standortangabe	Das System sendet regelmäßig die aktuellen GPS-Koordinaten des Fahrzeugs an den Carsharing Betreiber, damit dieser seinen Kunden über eine App oder seine Webseite mitteilen kann, wo welches Fahrzeug ausleihbar ist, und damit er immer darüber informiert ist, wo sich welches Fahrzeug gerade befindet.
F2	Ladestandangabe	Das System sendet regelmäßig den aktuellen Ladestand des Akkus an den Carsharing Betreiber, damit er die benötigte Ladezeit abschätzen und die Vergabe der Fahrzeuge besser planen kann.
F3	Kilometerstand-angabe	Das System sendet regelmäßig den aktuellen Kilometerstand an den Carsharing Betreiber, damit er die gefahrenen Kilometer für die Abrechnung der Buchung und Erstellung von Kundenprofilen nutzen kann.

3.1.2 Digitale Fahrzeugschlüssel

Wie bereits in Abschnitt 3.1.1 erwähnt, setzen fast alle betrachteten Carsharing Betreiber auf eine Freischaltung des Fahrzeugs mittels RFID Karte. Ein entsprechendes Lesegerät an der Frontscheibe liest die Daten auf der Mitgliedskarte aus. Eine nachgeschaltete Komponente verifiziert die Daten der Karte und gibt bei Erfolg ein Signal an die Zentralverriegelung des Fahrzeugs, damit sich die Türen öffnen lassen. Gleichzeitig wird der Beginn der Buchung registriert. Während der Buchung lässt sich das Fahrzeug wie gewohnt mit dem im Fahrzeug befindlichen Zündschlüssel starten und ver- und entriegeln. Erst wenn der Kunde das Fahrzeug zurückzugeben gedenkt, verwendet er wieder seine Mitgliedskarte, um das Fahrzeug zu verriegeln, nachdem er den Zündschlüssel im Fahrzeug deponiert hat, und die Buchung wird abgeschlossen. Dieser Vorgang wird beim Betreiber registriert, der dann wiederum das Fahrzeug für den nächsten Kunden zur Miete freigeben kann. Damit dies gelingt, muss es eine Form von Kommunikation zwischen Fahrzeug und Anbieter geben. Diese muss bei einem Fahrzeug in einer Free Floating Flotte direkt zwischen Fahrzeug und Anbieter erfolgen. Bei Anbietern mit festen Abholstationen kann die Kommunikation auch über eine Relaystation vor Ort realisiert werden. Dabei verständigen sich Fahrzeug und Relaystation z.B. via Bluetooth, WLAN oder kabelgebunden. Lediglich die Relaystation unterhält dabei eine direkte Verbindung zum Anbieter.

Ein weiteres Verfahren, welches keine Verbindung zwischen Fahrzeug und Anbieter erfordert, wurde von [Block, 2012] vorgestellt. Das Verfahren wurde als Android-App implementiert und macht sich sowohl die NFC- als auch die mobile Internetverbindung des Smartphones zunutze. Nach Eingabe der Buchungsdaten, die in Abschnitt 3.1.1 beschrieben wurden, wird seitens des Anbieters ein virtueller Fahrzeugschlüssel erzeugt und über eine verschlüsselte Verbindung an das Smartphone des Kunden übertragen. Über eine Bluetooth- oder RFID-Verbindung überträgt der Kunde diesen Schlüssel dann ebenfalls verschlüsselt kontaktlos an das Fahrzeug, welches nach Verifikation des Schlüssels das Fahrzeug freischaltet.

Tabelle 3.2: Featureliste digitale Fahrzeugschlüssel

Nr.	Feature	Beschreibung
F4	Freischaltung via RFID	Das System verfügt über ein RFID-Lesegerät, welches leicht zugänglich an dem Fahrzeug angebracht ist. Dadurch kann der Kunde das Fahrzeug mit seiner Kundenkarte oder seinem Smartphone für die Benutzung freischalten.
F5	Freischaltung via Bluetooth	Das System verfügt über eine Bluetooth-Schnittstelle, damit der Kunde das Fahrzeug mit seinem Smartphone für die Benutzung freischalten kann.

3.1.3 Cockpits von Elektrokrafträdern

Seit Beginn der 1990er Jahre werden von verschiedenen staatlichen und internationalen Institutionen Richtlinien und Empfehlungen verfasst, um die Ablenkung des Fahrers durch Fahrerassistenzsysteme (FAS) und In-Vehicle-Informationssysteme (IVIS) zu begrenzen. In der Automobilindustrie finden neben diesen Richtlinien und Empfehlungen auch Normen zur Software-Ergonomie und Gebrauchstauglichkeit (z.B. ISO 9241-10) Anwendung (vgl. [Müller, 2004]). Bei einem Zweirad bildet das Cockpit mit seinen Anzeigeinstrumenten das zentrale Informationssystem. Im Folgenden werden vier Gestaltungsansätze für Cockpits von Elektrokrafträdern vorgestellt.

Govecs GO! S1.2

Das Modell *GO! S1.2* der 2009 in München gegründeten *GOVECS GmbH* stellt das Einsteigermodell der Firma dar ([Govecs, 2013]). Mit seinem 54 Nm Wechselstrommotor und 2 kWh Silizium-Akku erlaubt das Fahrzeug eine Spitzengeschwindigkeit von 45 km/h und Reichweiten bis zu 60 km. Der Elektromotorroller entspricht einem Kraftrad der 50 ccm Klasse und darf mit Führerscheinen der Klassen 1, 1a, 1b, 3, 4, B, M, AM, A, A1 und A2 gefahren werden ([BMVBS, 2013b]).

Abbildung 3.1: Cockpit Govecs GO! S1.2[1]

Die Anzeigeinstrumente des *Govecs GO! S1.2* sind, wie Abbildung 3.1 zeigt, als einfaches LC-Display und fünf Kontrollleuchten realisiert. Zentrales Element ist die 7-Segment Anzeige in der Mitte des Displays, welche die aktuelle Geschwindigkeit angibt. Sie wird zusätzlich als Kreissegmentanzeige links von dem 7-Segment dargestellt. Unterhalb der Geschwindigkeitsangabe befindet

[1] Quelle: http://cdn3.spiegel.de/images/image-366376-galleryV9-finm.jpg

sich die Anzeige des aktuellen Kilometerstandes des Fahrzeugs. Ein 8-elementiger Bargraph rechts von der Geschwindigkeitsanzeige gibt Auskunft über den Ladestand des Akkus, Die Uhrzeit lässt sich an der Zeitanzeige rechts unten auf dem Display ablesen.
Die Kontrollleuchten geben folgende Auskünfte (von links unten im Uhrzeigersinn):

- Ladestandanzeige: Gibt während des Aufladens Auskunft über den Ladestand des Akkus
- Batterielampe: Leuchtet auf, wenn der Akku einen kritischen Entladezustand erreicht hat und aufgeladen werden muss
- Blinker links
- Scheinwerfer
- Blinker rechts

Mit Hilfe der beiden Knöpfe an der rechten Seite (oben: Select; unten: Mode) lassen sich die folgenden Modifikationen an den Anzeigeelementen vornehmen ([Govecs, 2011]):

- Umschalten der Anzeige zwischen Gesamtkilometer- und Tageskilometerstand (Mode)
- Rücksetzen des Tageskilometerstandes (Select 3 Sekunden halten)
- Umschalten der Anzeige zwischen km/h und mph (Mode und Select 3 Sekunden halten)
- Einstellen der Uhrzeit (Mode 3 Sekunden halten, mit Select die Minuten einstellen, mit Mode auf Stunden wechseln und mit Select einstellen, Speichern der Uhrzeit mit Mode)

BMW Concept C evolution

Das Konzeptfahrzeug *C evolution* von *BMW* ist mit einem Elektromotor mit 11 kW Dauerleistung und 35 kW Spitzenleistung ausgestattet und transportiert seinen Fahrer mit bis zu 120 km/h. Die Lithium-Ionen-Speichermodule mit einer Kapazität von 8 kWh erlauben Fahrten mit einer Reichweite von bis zu 100 km. Hierbei handelt es sich um die gleichen Speichermodule, die auch in BMWs i3 zum Einsatz kommen ([Naumann, 2012]).

Abbildung 3.2: Cockpit BMW Concept C evolution[2]

Das TFT-Display des *C evolution*, welches konzeptionell an das Display aus dem *i3* angelehnt ist, bietet neben der Anzeige der Geschwindigkeit, des Ladezustandes des Akkus und der Angabe von Tages- und Gesamtkilometerstand noch weitere Elemente, die dem Fahrer Informationen über den Energiehaushalt des Fahrzeuges geben. So verfügt das Display über eine Angabe der Reichweite, die mit dem aktuellen Ladestand noch zurückgelegt werden kann, sowie die Spannung des Akkumulators. Die Warn- und Kontrollleuchten für Blinker, Scheinwerfer etc. sind ebenfalls in das Display integriert und dort an der oberen Displaykante zu finden.
Das Fahrzeug verfügt im Gegensatz zu dem *Govecs GO! S1.2* über ein Energierückgewinnungssystem. Ob und wie viel Energie gerade erzeugt oder verbraucht wird, bekommt der Fahrer als horizontalen Bargraphen unterhalb der Geschwindigkeit dargestellt. Solche Informationen fördern beim Fahrer eine energiebewusste und sparsame Fahrweise ([Meschtscherjakov et al., 2009]). Weitere Ansätze zur Darstellung und Unterstützung der Energieökonomie finden sich auch bei [Strömberg et al., 2011] und [Cahour et al., 2012].

[2] Quelle: http://www.auto-news.de/webcore/data/content/Auto_Article_EXT/32708_bmw_c_12_evolution_10_big.jpg

Vectrix VX-1 Li+

In dem *VX-1 Li+* der 1996 in den USA gegründeten Firma *Vectrix* sind Lithium-Eisen-Phosphat-Akkus mit einer Kapazität von 5,4 kWh verbaut. Zusammen mit dem 20 kW starken Elektromotor können Spitzengeschwindigkeiten von 110 km/h und Reichweiten bis zu 120 km erreicht werden ([ADAC, 2011]).

Abbildung 3.3: Cockpit Vectrix VX-1 Li+[3]

Die verchromten Anzeigeinstrumente und der große analoge Tachometer in der Mitte des Cockpits erinnern vom Design her an klassische Cockpits von Rollern mit Verbrennungsmotor, wie etwa einer Vespa oder Simson. Wie bei den beiden anderen bereits vorgestellten Fahrzeugcockpits ist auch hier die Geschwindigkeit die zentrale Größe. In den Tachometer ist die Anzeige für den Gesamtkilometerstand integriert, der Tageskilometerzähler ist in der linken Anzeige zu finden, ebenso wie die Angabe der Reichweite, der Uhrzeit und des Status des Fahrzeugs. Das rechte Anzeigeinstrument ist komplett dem Ladestand des Akkus gewidmet. Kontroll- und Warnleuchten befinden sich in einer leicht gewölbten Leiste oberhalb der Geschwindigkeitsanzeige.
Mit Hilfe der vier Knöpfe unterhalb des Tachometers lassen sich die Uhrzeit einstellen, der Tageskilometerstand zurücksetzen und die Angaben von Tages- und Gesamtkilometerstand zwischen Kilometern und Meilen umschalten. Obwohl der *VX-1 Li+* ebenso wie *BMWs C evolution* über ein Energierückgewinnungssystem verfügt, gibt es in dem Cockpit keine Angaben über die Energieökonomie.

[3] Quelle: http://www.adac.de/_mmm/jpg/Vectrix%2520021_664x443_156916.jpg

smart escooter

Im Jahr 2010 stellte *smart* auf dem Pariser Autosalon den *escooter* vor, der 2014 auf den Markt kommen soll ([smart, 2013]). Sein Lithium-Ionen-Akku mit einer Kapazität von 3,84 kWh speist einen 4 kW starken Motor, der das Fahrzeug auf bis zu 45 km/h beschleunigt. Es sind Reichweiten bis zu 100 km mit einer Akkuladung möglich ([Hybrid-Autos, 2013]).

Abbildung 3.4: Cockpit smart escooter[4]

smart verfolgt bei dem *escooter* im Vergleich zu den bisher betrachteten Herstellern und Modellen ein komplett anderes Konzept, was die Gestaltung des Cockpits anbelangt. Hier übernimmt ein Smartphone die Rolle der Anzeigeinstrumente und erlaubt auch außerhalb des Fahrzeugcockpits die Steuerung einiger Funktionen des Fahrzeugs. Vor der Fahrt muss das Smartphone in die dafür vorgesehene Halterung geschoben werden, wodurch sich das Smartphone mit dem Fahrzeug vernetzt. Die Wegfahrsperre und der Diebstahlschutz werden automatisch deaktiviert und das Fahrzeug kann gestartet werden. Während der Fahrt gibt das Smartphone Auskunft über die Geschwindigkeit, den Ladestand des Akkus und eine Reichweitenprognose. Darüber hinaus ist in die Anzeige ein Navigationssystem integriert, welches die zu fahrende Richtung, die Entfernung zum nächsten Knotenpunkt und den aktuellen Straßennamen anzeigt.
Befindet sich das Smartphone nicht in dem Fahrzeug, kann mit Hilfe der App die Position des Rollers bestimmt werden, damit man das Fahrzeug auffinden kann. Als Komfort-Feature lässt sich mit der App auch die Beheizung der Lenkergriffe regeln. Neben einem Energierückgewinnungssystem verfügt der *smart escooter* zur zusätzlichen Energiegewinnung über Sonnenkollektoren in seiner Frontverkleidung, die den Lithium-Ionen-Akku während der Fahrt mit Energie versorgen. Allerdings gibt das Smartphone bzw. die App keinerlei Auskunft darüber, ob und wie sehr der Akku gerade ge- oder entladen wird.

[4] Quelle: http://www.hybrid-autos.info/images/stories/Smart/escooter_2010/Display_gross.jpg

Tabelle 3.3: Featureliste Fahrzeugcockpit

Nr.	Feature	Beschreibung
F6	Geschwindigkeits-anzeige	Das System zeigt dem Benutzer während der Fahrt die aktuelle Geschwindigkeit in Echtzeit an.
F7	Kilometerstands-anzeige	Das System zeigt dem Benutzer den aktuellen Kilometerstand an. Der Benutzer kann dabei zwischen der Angabe des Gesamt- und Tageskilometerstandes wechseln. Der Tageskilometerstand lässt sich auf Wunsch vom Benutzer zurücksetzen.
F8	Uhrzeit	Das System zeigt dem Benutzer die aktuelle Uhrzeit und das Datum an.
F9	Ladestands-anzeige	Das System zeigt dem Benutzer den aktuellen Ladestand des Akkus in Prozent an.
F10	Reichweiten-anzeige	Das System zeigt dem Benutzer eine Schätzung der mit der aktuellen Akkuladung noch zurücklegbaren Streckenlänge an.
F11	Ecometer	Das System zeigt dem Benutzer während der Fahrt die Energieeffizienz seiner momentanen Fahrweise an.
F12	Kontroll-, Warn- und Fehlermeldungen	Das System zeigt dem Benutzer Kontroll- und Warnleuchten wie Blinker, Scheinwerfer, Fernlicht und Batteriewarnungen an. Fehlermeldungen, die das Fahrzeug produziert, werden sowohl dem Benutzer auf dem Display mitgeteilt als auch an den Carsharing Betreiber geschickt, damit dieser entsprechende Hilfs- oder Servicemaßnahmen ergreifen kann.

3.1.4 Navigationssysteme für Elektrofahrzeuge

Neben der noch zur Verfügung stehenden Ladung des Energiespeichers ist vor allem die Angabe einer Reichweite, die mit dieser Ladung zurückgelegt werden kann, sowohl für den Fahrer als auch für den Betreiber interessant. Der Kunde kann an ihr ablesen, ob er sein Ziel noch erreicht oder den Weg zur Abgabestation schafft, während der Betreiber diese Angabe zur Abschätzung der Ladezeit nach Rückgabe des Fahrzeugs und somit für das Management seines Fuhrparks nutzen kann.

Da die Reichweite aber über die Ladung hinaus von weiteren Faktoren wie der Steigung und der Oberflächenstruktur der Strecke, dem Gesamtgewicht, dem Typ des Energiespeichers oder dem Fahrstil abhängt, wurden Navigationssysteme entwickelt, welche diese Parameter in die Abschätzung der Reichweite einbeziehen. Im Folgenden werden die Systeme *mapZero* der *ALL4IP TECHNOLOGIES GmbH & Co. KG* und das an der Universität zu Lübeck und Technischen Universität München entwickelte *GreenNav* vorgestellt.

mapZero

Das Navigationssystem *mapZero* ist als App für Android realisiert. Der Benutzer kann sich je nach Präferenz die energie- oder zeiteffizienteste Route oder eine Mischung aus beiden berechnen lassen. Auf der Karte wird dem Benutzer ein rotes Polygon angezeigt, welches die zu erwartende Reichweite in Bezug zu den gegebenen Geodaten darstellt (vgl. Abb. 3.5). Das System ist in zwei Versionen verfügbar, einer kostenlosen Testversion und einer auf einen bestimmten Fahrzeugtypen abgestimmten lizenzierten Version für 49 Euro pro Jahr. In der kostenlosen Testversion berücksichtigt das System neben den Geodaten lediglich die vom Hersteller angegebene Reichweite, um eine Prognose zu erstellen. In der lizenzierten Version werden zusätzlich Fahrzeugprofildaten verwendet, wodurch eine wesentlich genauere Reichweitenprognose abgegeben werden kann ([ALL4IP TECHNOLOGIES, 2013]).

Damit die *mapZero*-App die Geschwindigkeit und den aktuellen Batterieladezustand in seine Berechnungen mit einbeziehen kann, ist zusätzliche Hardware vonnöten, welche besagte Daten aus dem Fahrzeug ausliest und an die App weiterleitet. Diese Hardware kann über die Webseite von *ALL4IP TECHNOLOGIES* bezogen werden. Während der Fahrt bezieht die App ihre Kartendaten sowie die Eckpunkte des Reichweitenpolygons von einem Server über die Internetverbindung des Smartphones. Die aktuelle Position des Fahrzeugs ermittelt die App mit Hilfe des in das Smartphone integrierten GPS-Sensors.

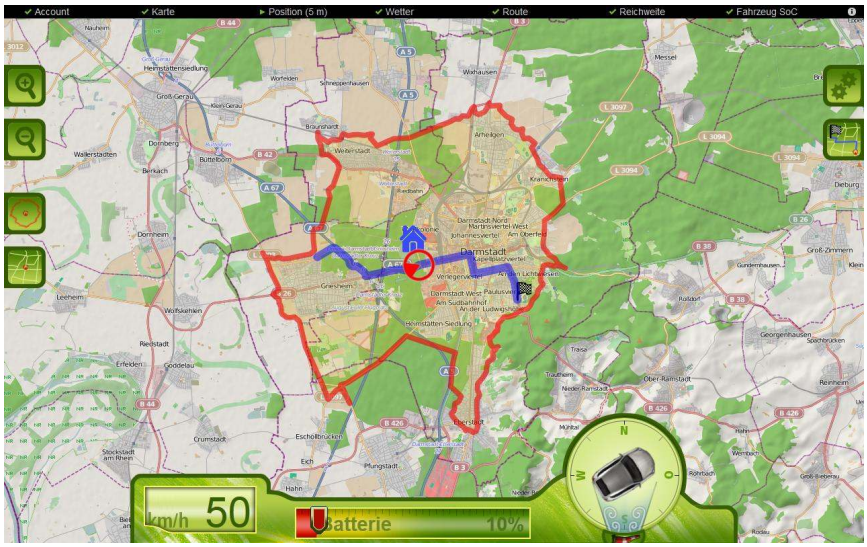

Abbildung 3.5: mapZero[5]

[5] Quelle: https://lh5.ggpht.com/6_0fOhwBeb91PgCOfE0zHhDpr5EUP2j_vLkQrFJ6eIu5V592PGKFQVxOdAoPwy1ZZfI=h900

GreenNav

Das an der Technischen Universität München und der Universität zu Lübeck entwickelte *Green Navigation*, kurz GreenNav, ist sowohl ein prototypisches System zur Berechnung von Reichweitenprognosen und energieeffizienten Routen als auch eine Forschungsplattform zur Evaluation von Algorithmen und Datenstrukturen für energieoptimiertes Routing. Für die Nutzung von GreenNav existieren verschiedene Frontends für unterschiedliche Plattformen. Die Berechnungen finden serverseitig mit Hilfe von Kartendaten von GoogleMaps, OpenStreetMap und Höhendaten der NASA aus der Shuttle Radar Topography Mission ([Kliesch, 2013]). Weitere Daten, die in die Berechnung einbezogen werden, sind unter Anderem Ladestand, Verbrauch (bei ausgewählten Fahrzeugen) und Reibungskoeffizienten der Fahrbahnbeläge.
Das System wird am Institut für Softwaretechnik und Programmiersprachen der Universität zu Lübeck durch Abschlussarbeiten und studentische Projekte stetig weiterentwickelt und erweitert. Parallel zu der hier vorliegenden Arbeit laufen Projekt- und Abschlussarbeiten mit dem Ziel, GreenNav im Carsharing Kontext nutzbar zu machen und Frontends für Navigationssysteme in Elektrofahrzeugen zu implementieren. Die Verbesserung der Reichweitenprognosen ist ebenfalls Gegenstand laufender und zukünftiger Arbeiten des Instituts.

Abbildung 3.6: GreenNav Reichweitenprognose

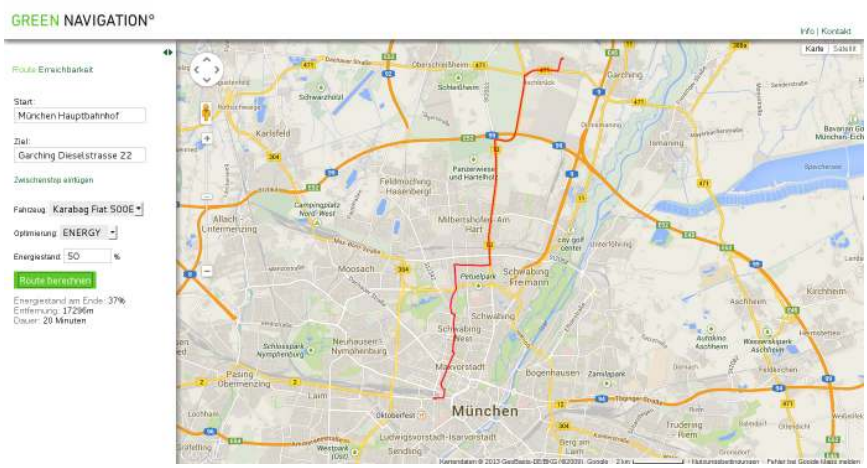

Abbildung 3.7: GreenNav Route

Tabelle 3.4: Featureliste Navigationssystem

Nr.	Feature	Beschreibung
F13	Navigations-system	Das System verfügt über ein Navigationssystem, welches dem Benutzer die schnellste, kürzeste oder sparsamste Strecke anzeigen kann.
F14*	Reichweiten-prognose	Das Navigationssystem zeigt dem Benutzer sowohl die berechnete Strecke als auch eine Reichweitenprognose auf der Karte an. Die Prognose steht in Bezug zu dem aktuellen Akkuladestand, Höhendaten, Straßenbelag, Witterungsverhältnissen etc.
F15*	POIs	Das Navigationssystem zeigt dem Fahrer auf der Karte an, wo sich Ladestationen und öffentliche Steckdosen befinden, an denen er das Fahrzeug aufladen kann. Dabei wird markiert, ob es sich um eine kostenlose oder entgeltliche Lademöglichkeit handelt.

*Dieses Feature bezieht sich auf das Navigationssystem und nicht auf das hier zu entwickelnde System

3.2 Benutzeranalyse

Nachdem im vorherigen Abschnitt geklärt worden ist, welche Teilsysteme und Vorgänge aus dem Carsharing Kontext Einfluss auf die Gestaltung des hier zu entwickelnden Systems haben, wollen wir uns nun der Frage widmen, welche Personen bzw. Personengruppen als Stakeholder dieses Projektes identifiziert werden können. Um diese Frage zu beantworten, werden im Folgenden einige direkt und indirekt betroffene Personen(gruppen) untersucht. Die so ermittelten primären Stakeholder werden durch entsprechende Persona nach [Cooper et al., 2012] manifestiert und dienen im weiteren Verlauf der Analyse als Akteure.

3.2.1 Stakeholder

Die ISO 10006 definiert Stakeholder als Personen oder Personengruppen, die ein berechtigtes Interesse am Verlauf oder Ausgang eines Projektes oder Prozesses haben oder davon in irgendeiner Weise betroffen sind. Bei einem Prozess wie dem Carsharing erstreckt sich der Kreis der Betroffenen von dem neugierigen Neukunden, der das Carsharing zum ersten mal ausprobiert, über den Betreiber der Carsharing Firma, der seine Kosten minimieren und seine Flotte bestmöglich ausnutzen will, bis hin zur gesamten Gesellschaft eines Landes, sogar der Gesamtheit aller Lebewesen auf diesem Planeten, die letztlich alle davon profitieren, dass insgesamt weniger Fahrzeuge unterwegs sind, die CO_2 ausstoßen. Abbildung 3.8 zeigt potenzielle Stakeholder, die ein Interesse am Ausgang des Projektes haben.

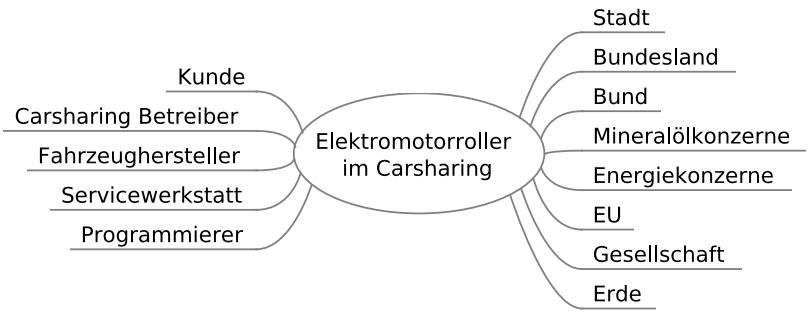

Abbildung 3.8: Potenzielle Stakeholder

Eine solch weitreichende Betrachtung sprengt den Rahmen dieser Arbeit und ist im Kontext des hier zu entwickelnden Systems auch wenig sinnvoll. Vielmehr sollen sich die weiteren Analysen auf diejenigen Personen und Personengruppen beziehen, deren Handeln direkt durch das System beeinflusst wird und die in unmittelbare Interaktion mit dem System treten. Daher beschränken sich die weiteren Analysen im Folgenden auf die Stakeholder *Kunde*, *Carsharing Betreiber* und *Service-Techniker*, der als Schnittmenge aus *Servicewerkstatt* und *Programmierer* angesehen werden kann, was die weiteren Analysen noch zeigen werden.

Der Kunde

Der Carsharing Kunde kann als primärer Stakeholder betrachtet werden. Er ist direkt von der Konzeption und Gestaltung des Systems betroffen und stellt den zentralen Akteur dar, der unmittelbar mit dem System interagiert. Als solcher erwartet er neben einer hohen Verfügbarkeit auch eine leichte Bedienbarkeit und ein modernes und ansprechendes Design der Benutzungsschnittstellen. Der Kunde möchte sich nicht mit technischen Details und Fragen der Wartung und Pflege des Fahrzeugs auseinander setzen müssen, sondern lediglich mit möglichst wenig physischem und kognitivem Aufwand ein Fahrzeug mieten, um von einem Ort zum anderen zu gelangen.

Der Carsharing Betreiber

Als Betreiber des Systems kommt dem Carsharing Anbieter besondere Bedeutung zu. Nicht nur, dass er für die Installation, Wartung und Pflege des Systems verantwortlich zeichnet, sondern er hat auch großes Interesse daran, dass seine Kunden das System akzeptieren und gerne benutzen. Dazu muss das System nicht nur eine hohe Verfügbarkeit, leichte Bedienbarkeit und ein ansprechendes Design aufweisen, sondern ihm auch die Daten liefern, die für den Betrieb des Systems in einer Carsharing Flotte vonnöten sind. Neben dem aktuellen Standort des Fahrzeugs sind auch Zustandsdaten wie Gesamtkilometerstand, Akkuladung (SoC) und -gesundheit (SoH) und gegebenenfalls Warn- und Fehlermeldungen des Fahrzeugs für den Betreiber von Interesse.
Die Wartung und Pflege des Systems kann der Betreiber entweder von eigenem Personal durchführen lassen oder an externe Firmen outsourcen. In beiden Fällen erwartet er, dass das System eine möglichst geringe Komplexität aufweist, um die Kosten für diese Tätigkeiten so gering wie möglich zu halten. Da der Carsharing Betreiber auch für die Wartung und Pflege der Fahrzeuge seines Fuhrparks zuständig ist, können Stakeholder wie der Fahrzeughersteller oder die Mitarbeiter der Servicewerkstatt dem Betreiber untergeordnet werden. Der Betreiber kann, sofern erforderlich, die benötigten Informationen an den Hersteller bzw. die Werkstatt weiterleiten.

Der Service-Techniker

Der Service-Techniker spielt insbesondere bei der Wartung und Pflege des Systems eine wichtige Rolle. Seine Arbeit wird direkt von der Architektur und dem Aufbau des Systems beeinflusst. Aus seiner Sicht sollte das System Features bieten, die von modernen Hard- und Softwaresystemen zu erwarten sind: Eine offene und flexible Architektur, wohl definierte Schnittstellen, gut dokumentierte APIs, leichte Wartbarkeit, simple Kommunikationsprotokolle, hohe Performanz und nicht zuletzt state-of-the-art Technologien. Auch die Erweiterbarkeit des Systems hat Auswirkungen auf seine Arbeit. Wenn der Kunde oder der Carsharing Betreiber ein neues Feature verlangt, wie beispielsweise die Integration von Sensoren zur Überwachung der Akkutemperatur, soll die Integration so einfach wie möglich vonstatten gehen. Auch das Deployment des Systems fällt in seine Verantwortung.

3.2.2 Persona 1: Der Kunde

Name:	Jenny Weber
Geburtsdatum:	13.07.1988
Geburtsort:	Hamburg, Deutschland
Größe:	1,76 m
Gewicht:	61 kg
Familienstand:	Ledig
Beruf:	Studentin
Benutzerklasse:	Routinebenutzer

Abbildung 3.9: Jenny Weber[6]

Als Tochter eines KFZ-Meisters mit eigener Werkstatt und einer Meteorologin entdeckte Jenny schon in jungen Jahren ihre Begeisterung für motorisierte Zweiräder. Mit 16 Jahren machte sie ihren ersten Zweirad Führerschein für Krafträder bis 80 ccm und mit 18 für Automobile und Krafträder. Nach ihrem Abitur und einem freiwilligen sozialen Jahr in einer örtlichen Kindertagesstätte begann sie ihr Studium der Geowissenschaften an der Universität Hamburg. Weil ihre finanziellen Mittel als Studentin sehr begrenzt sind, musste sich Jenny bereits im zweiten Semester von ihrer Honda CBF 125 trennen und stieg vorerst auf öffentliche Verkehrsmittel um.
Im Laufe ihres Studiums entwickelte sie ein starkes Bewusstsein für Umweltverträglichkeit und sparsamen Umgang mit Ressourcen. Da sie sich bedingt durch ihr Studium für alternative Energien und effiziente Fortbewegung interessiert, recherchierte sie nach Car- bzw. Vehiclesharing Anbietern in ihrer Umgebung und fand einen, der auch Elektroroller zum Verleih anbietet, den Carsharing Club Hamburg (CCHH). Auf der Webseite des Anbieters fand sie einen Link zu einer App für ihr Smartphone, mit der sie die Standorte der zur Verfügung stehenden Elektroroller feststellen (vgl. [MoveAbout, 2013]), den Buchungsvorgang durchführen (vgl. [Kampsen, 2013]) und das Fahrzeug für die Benutzung freischalten kann (vgl. [Block, 2012]).

[6] Quelle: http://www.pixelio.de/media/323119, ©arctwo / pixelio.de

3.2.3 Persona 2: Der Carsharing Betreiber

Name:	Tom Schneider
Geburtsdatum:	11.09.1977
Geburtsort:	Schleswig, Deutschland
Größe:	1,84 m
Gewicht:	82 kg
Familienstand:	Verheiratet
Beruf:	Unternehmer
Benutzerklasse:	Experte

Abbildung 3.10: Tom Schneider[7]

Nach seinem Abitur an der Lornsenschule in Schleswig absolvierte Tom seinen Wehrdienst bei den Funkelektronikern der Flugabwehrraketengruppe in Stadum bei Flensburg. Dort lernte er neben dem Umgang mit Multimeter und Lötkolben dank des Einsatzes seines Oberfeldwebel die Strukturen und nötigen Prozesse kennen, um einen Betrieb wie die Elektronikwerkstatt am Laufen zu halten. Begeistert von dem Zusammenspiel von Technik, Organisation und Wirtschaft begann er nach seiner einjährigen Dienstzeit ein Studium des Wirtschaftsingenieurwesens an der Hochschule für Wirtschaft und Technik in Berlin.
Im Rahmen seines Studiums musste Tom ein Praktikum in einem Betrieb absolvieren. Dabei fiel seine Wahl eher zufällig auf ein Carsharing Unternehmen. Schon nach kurzer Zeit hatte er die Strukturen der Firma verstanden und verschiedene Abläufe und Prozesse verinnerlicht. Der Chef der Firma war von Toms Auffassungsgabe und Talent so überzeugt, dass er ihm einen Studentenjob auf 400 Euro Basis anbot, den Tom mit Freude annahm.
Nach seinem Masterabschluss bewarb er sich bei einem Carsharing Unternehmen in Hamburg und arbeitete dort für ein paar Jahre, um sich dann mit einem eigenen Unternehmen selbständig zu machen. So gründete er 2010 den Carsharing Club Hamburg (CCHH), der ausschließlich auf elektrisch betriebene Fahrzeuge setzt. Um seinen Fuhrpark zu erweitern und die Angebote für seine Kunden attraktiver zu gestalten, möchte Tom seine Flotte um aktuelle Elektrokrafträder erweitern. Zudem beabsichtigt er, das Batteriemonitoring und -management für seinen Fuhrpark zu verbessern, um zuverlässigere Angaben über den Zustand und die zu erwartende Lebensdauer der Batterien machen zu können.

[7] Quelle: http://www.pixelio.de/media/489987, ©Konstantin Gastmann / pixelio.de

3.2.4 Persona 3: Der Service-Techniker

Name:	Simon Bern
Geburtsdatum:	17.05.1982
Geburtsort:	Lübeck, Deutschland
Größe:	1,78 m
Gewicht:	76 kg
Familienstand:	Ledig
Beruf:	Informatiker
Benutzerklasse:	Experte

Abbildung 3.11: Simon Bern[8]

Bereits im Alter von 12 Jahren fing Simon an, seine ersten kleinen Programme auf dem PC seines Vaters in Turbo Pascal zu schreiben. Im Informatikunterricht auf dem Gymnasium lernte er dann HTML und JavaScript kennen und war begeistert von den Möglichkeiten, die ihm diese Werkzeuge boten. Nach seinem Abitur und dem Zivildienst in einem Altenpflegeheim entschied er sich für das Studium der Informatik an der Universität zu Lübeck, das er erfolgreich als Master of Science abschloss. Seitdem verdient er sich seinen Lebensunterhalt als freier Informatiker und arbeitet für verschiedene Firmen in Schleswig-Holstein und Hamburg.
Durch seine häufig wechselnden Arbeitgeber und Projekte hat er sich die Fähigkeit angeeignet, sich schnell in fremden Programmcode einzuarbeiten, Strukturen und Architekturen zu verstehen und Mängel oder Fehler in diesen zu erkennen und zu beheben. Trotz dieser Fertigkeiten hat er es bevorzugt mit Systemen zu tun, die gut dokumentiert und strukturiert sind.
Seit einem halben Jahr arbeitet Simon für den Carsharing Club Hamburg (CCHH). Dort hat er eine Smartphone-App mitentwickelt, mit deren Hilfe der Kunde sich die Standorte der verfügbaren Fahrzeuge anzeigen lassen, ein Fahrzeug buchen und mittels digitalem Schlüssel kontaktlos freischalten bzw. öffnen kann. Zur Zeit ist er für die Wartung und Erweiterung der Systeme zuständig, die für die Nutzung im Carsharing Betrieb in die Fahrzeuge eingebaut wurden.

[8] Quelle: http://www.pixelio.de/media/647333, ©Tim Reckmann / pixelio.de

3.3 Szenarien

Die folgenden Szenarien in Form von Stories im Sinne von [Alexander und Maiden, 2005] beschreiben das zu entwickelnde System und den Umgang mit diesem jeweils aus der Perspektive eines der Stakeholder, die in Abschnitt 3.2 identifiziert wurden. Die beschriebenen Persona aus ebd. Abschnitt bilden die Akteure der Szenarien. Anhand der in den Szenarien dargestellten Vorgänge werden Funktionalitäten und Features des zu entwickelnden Systems extrahiert ([Carroll, 1995]), die am Ende der Szenarien aufgelistet und in die Konzeption in Kapitel 4 einbezogen werden.
Das Kunden-Szenario beschreibt das System aus Sicht eines Carsharing Teilnehmers. Es umfasst alle Vorgänge vom Entschluss, einen Elektroroller zu mieten, bis hin zur Rückgabe des Fahrzeugs, damit es von weiteren Kunden gemietet werden kann. Das Betreiber-Szenario schildert diese Vorgänge vom Standpunkt des Carsharing Betreibers aus und gibt vor allem Aufschluss über die Daten, die der Betreiber für einen reibungslosen Ablauf des Ausleihvorganges benötigt. Zum Abschluss wird das System aus Sicht des Service-Technikers beschrieben, um Features ableiten zu können, die eine leichte Wartbarkeit, Erweiterbarkeit und einfaches Deployment ermöglichen.

3.3.1 Szenario 1: Der Kunde

Es ist Freitag Vormittag. Jenny Weber sitzt mit ihren Kommilitonen in der Vorlesung *Sedimentgeologie der Karbonate*. Sie hat geplant, nach ihren Vorlesungen über das Wochenende mit dem Zug zu einer Freundin nach Berlin zu fahren. Ihr Gepäck hat sie bereits dabei, sodass sie nach ihrer letzten Vorlesung direkt zum Bahnhof Dammtor gehen kann. Zwischen dem Ende ihrer Vorlesung und der Abfahrt des Zuges hat Jenny noch ausreichend Zeit, um am Bahnhof in aller Ruhe etwas zu essen und einen Kaffee zu trinken.
Nach der Sedimentgeologie-Vorlesung fällt ihr ein, dass sie ihren frisch ausgefüllten BAföG-Antrag für das kommende Semester zu Hause vergessen hat. Da sich die Frist für die Abgabe des Antrags ihrem Ende nähert, beschließt sie, vor ihrer Abfahrt noch einmal nach Hause zu fahren und den Antrag zu holen. Nach einer kurzen Recherche nach den Fahrzeiten der öffentlichen Verkehrsmittel stellt sie fest, dass die verfügbare Zeit nicht ausreichen wird, um den Antrag zu holen und den Zug nach Berlin noch zu erreichen. Deshalb entschließt sie sich, für die Fahrt einen Elektromotorroller auszuleihen. Sie nimmt ihr Smartphone zur Hand, startet die App des CCHH, bei dem sie sich bereits letztes Semester registriert hat. Für die Verbindung ihres Smartphones zum Internet verwendet Jenny das WLAN-Netz ihrer Universität. Nach Eingabe ihres Benutzernamens und Passwortes schaut sie nach, ob am Dammtor noch Elektromotorroller zum Verleih bereit stehen. Glücklicherweise befinden sich an dem Bahnhof noch zwei Modelle mit unterschiedlicher Ausstattung.
Das erste Modell ist ein modifizierter *Govecs GO! S1.2*, dessen LC-Display durch einen TFT-Bildschirm ersetzt wurde und der zusätzlich mit einem kleinen Joystick, wie man ihn z.B. von Controllern für Spielekonsolen kennt, zur Interaktion mit dem System ausgestattet ist. In der Be-

schreibung des Fahrzeuges entdeckt sie, dass verschiedene Designs für die auf dem Display dargestellten Anzeigeinstrumente zur Verfügung stehen, aus denen der Kunde wählen kann. Die unterschiedlichen Ausgestaltungen geben allesamt Auskünfte über Geschwindigkeit, Kilometerstände, Uhrzeit und Akkuladestand und verfügen darüber hinaus über ein Ecometer zur Angabe der Energieeffizienz der momentanen Fahrweise. Ein Navigationssystem, das sowohl die zeit- als auch die energieeffizienteste Route berechnen kann, gehört ebenfalls zur Ausstattung des Rollers.

Bei dem zweiten Modell handelt es sich um einen *smart escooter*, der dahingehend modifiziert worden ist, dass sich Smartphones verschiedener Hersteller mit dem *escooter* verwenden lassen. Die Anzeigeinstrumente sowie das Navigationssystem werden durch die App des CCHH bereitgestellt. Auch hier kann der Kunde zwischen mehreren Fahrzeugcockpitdesigns wählen.

Nach kurzer Überlegung entscheidet sich Jenny für den *Govecs* Roller, da sie dieses Modell bereits mehrmals ausgeliehen hat und mit dessen Fahrverhalten und Bedienung vertraut ist, und ein schlichtes modernes Design der Anzeigeinstrumente. In die App trägt sie noch den Beginn und das Ende der Ausleihzeit sowie eine grobe Schätzung der Streckenlänge, die sie zu fahren beabsichtigt, ein und bestätigt ihre Reservierung. Nach ein paar Sekunden meldet die App den Erfolg der Buchung. Kurz darauf bekommt Jenny die Mitteilung, dass ihr digitaler Fahrzeugschlüssel für den Roller erfolgreich generiert und sowohl Schlüssel als auch die Daten für die Anzeigeinstrumente auf ihr Smartphone übertragen worden sind. Sie bestätigt die Meldung und geht in ihre nächste Vorlesung.

Nach Ende ihrer Vorlesung macht Jenny sich auf den Weg zum Dammtor Bahnhof, wo sie ihren reservierten Roller abholen will. Das Fahrzeug ist hinter dem Bahnhof auf dem Parkplatz abgestellt und das Ladekabel steckt in einer Ladesäule vor dem Roller. Sie nimmt ihr Smartphone zur Hand und startet erneut die CCHH-App. Die App erkennt, dass sich bereits ein gültiger Fahrzeugschlüssel auf dem Smartphone befindet und fragt Jenny, ob sie den Schlüssel verwenden, ihre aktuellen Buchungsdaten ändern oder eine weitere Buchung vornehmen möchte. Sie wählt die Schlüssel-Option und bekommt daraufhin die Anweisung, das Smartphone an das RFID-Lesegerät des Fahrzeuges zu halten. Jenny folgt dieser Anweisung. Das System registriert das Smartphone an dem Lesegerät und die App initiiert die Übertragung des digitalen Fahrzeugschlüssels. Nach einem kurzen Augenblick erscheint das Logo des CCHH auf dem Display des Rollers und die App meldet, dass der Schlüssel erfolgreich an das Fahrzeug übertragen wurde und nun verwendet werden kann. Gleichzeitig sendet das System Uhrzeit, Kilometerstand und Schlüssel-ID zu Abrechnungszwecken an den Betreiber.

An der Seite des Rollers knapp unterhalb der Sitzbank befindet sich ein Druckknopf, mit dem die Sitzbank entriegelt werden kann, sobald das Fahrzeug freigeschaltet wurde. Jenny zieht das Ladekabel aus der Ladesäule, entriegelt die Sitzbank und verstaut das Kabel in dem dafür vorgesehenen Staufach. Ein Helm und ein paar Handschuhe befinden sich in einer Transportbox am Heck des Rollers, welches sich ebenfalls mittels eines Druckknopfes öffnen lässt, sobald das Fahrzeug aktiv ist. Sie packt ihr Smartphone in ihre Hosentasche, zieht sich Helm und Handschuhe an und setzt

sich auf den Roller.

Bei einem Blick auf das Display stellt sie fest, dass das System bereits vollständig hochgefahren und ihr zuvor ausgewähltes Design der Anzeigeinstrumente zu sehen ist. Mit dem *Select*-Knopf schaltet Jenny in den Navigationsmodus um. Dort sieht sie neben Geschwindigkeits-, Ladestands- und Reichweitenanzeige eine Karte, auf der ihre aktuelle Position eingetragen ist. Ein rotes Polygon markiert auf der Karte die zu erwartende Reichweite bei aktuellem Ladestand in Bezug auf die Gegebenheiten der Straßen innerhalb des Polygons. Sie benutzt den kleinen Joystick am Gasgriff, um das Ziel ihrer Fahrt einzugeben[9] und entscheidet sich für die schnellste Route, damit sie in möglichst kurzer Zeit wieder hier am Dammtor ankommt und ihren Zug nach Berlin noch bekommt. Nach wenigen Sekunden stellt ihr das Navigationssystem die entsprechende Route auf der Karte dar und Jenny macht sich auf den Weg. Während der Fahrt lässt sie sich das Navigationssystem anzeigen. Auf der Karte wird ihre Position und Fahrtrichtung dabei ständig aktualisiert. Auch das Reichweitenpolygon und die Reichweitenanzeige passen sich ihrer Position und Fahrweise an.

Als Jenny nach einer knappen viertel Stunde zu Hause ankommt (mit den öffentlichen Verkehrsmitteln hätte sie fast doppelt so lange gebraucht), verstaut sie Helm und Handschuhe wieder in der dafür vorgesehenen Transportbox, nimmt ihr Smartphone aus der Hosentasche und startet die CCHH-App. Die App fragt Jenny wieder, ob sie den Schlüssel verwenden, ihre aktuellen Buchungsdaten ändern oder eine weitere Buchung vornehmen möchte. Sie wählt die Schlüssel-Option und bekommt die Anweisung, das Smartphone wieder an das RFID-Lesegerät des Fahrzeuges zu halten. Das System registriert das Smartphone an dem Lesegerät, die App überträgt den digitalen Fahrzeugschlüssel und meldet eine erfolgreiche Übertragung. Auf dem Display des Rollers erscheint ein Dialogfeld, in dem Jenny gefragt wird, ob sie das Fahrzeug nur parken oder die Buchung abschließen will. Mit dem Joystick wählt sie die Parken-Option. Daraufhin verdunkelt sich das Display und der Roller schaltet sich ab. Sie geht in ihre Wohnung, um den BAföG-Antrag zu holen.

Schon nach drei Minuten kommt Jenny mit dem in einem frankierten und adressierten Briefumschlag enthaltenen Antrag wieder aus ihrer Wohnung, startet den Roller mit ihrem Smartphone und zieht sich Helm und Handschuhe an. Da sie sich die Strecke, die sie auf dem Hinweg gefahren ist, gut gemerkt hat, verzichtet sie während der Rückfahrt auf das Navigationssystem und lässt sich statt dessen die Anzeigeinstrumente darstellen. Für den Rückweg zum Dammtor benötigt sie aufgrund erhöhten Verkehrsaufkommens etwa 18 Minuten, womit sie die komplette Tour 35 Minuten gekostet hat. Mit öffentlichen Verkehrsmitteln wäre sie inklusive aller Fußmärsche eine knappe Stunde unterwegs gewesen.

Am Dammtor angekommen, stellt sie den Roller wieder auf dem Parkplatz ab, an den sie ihre Fahrt aufgenommen hat und verstaut Handschuhe und Helm in der Transportbox. Sie nimmt ihr Smartphone in die Hand, startet die CCHH-App, wählt die Schlüssel-Option und hält das Smartphone an das RFID-Lesegerät. Mit dem Joystick wählt sie in dem Dialogfeld auf dem Display die Option

[9] Eine genaue Beschreibung der Interaktionsmöglichkeiten mit dem System erfolgt in Abschnitt 4.4.3

Buchung abschließen, aber der Roller schaltet sich nicht ab. Statt dessen erscheint eine weitere Meldung auf dem Display, in der Jenny aufgefordert wird, erst das Fahrzeug wieder an die Ladesäule anzuschließen und aufzuladen und dann das Buchungsende erneut zu bestätigen. Sie öffnet die Sitzbank, entnimmt das Ladekabel und schließt es an die Ladesäule an. Mit dem Joystick bestätigt sie auf dem Display das Ende der Buchung und das System überträgt wieder Uhrzeit, Kilometerstand und Schlüssel-ID an den Betreiber. Zudem markiert das System den für diese Buchung verwendeten Schlüssel als ungültig, damit das Fahrzeug nicht mehr mit diesem Schlüssel freigeschaltet werden kann. Danach schaltet das System den Roller ab und verriegelt die Staufächer. Kurz darauf erhält Jenny eine Email von dem Carsharing Betreiber, in der die Abrechnung der Buchung als PDF angehängt und der Hinweis enthalten ist, dass der von ihr verwendete Schlüssel nun nicht mehr gültig ist und somit nicht für weitere Fahrten genutzt werden kann.
Froh darüber, die Fahrt in so kurzer Zeit erledigt zu haben, begibt sich Jenny in den Bahnhof und wirft den Briefumschlag mit dem BAföG-Antrag in einen Briefkasten ein. Sie hat sogar noch ausreichend Zeit, um sich einen Kaffee und etwas zu Essen zu holen, bevor ihr Zug Richtung Berlin abfährt.

3.3.2 Szenario 2: Der Carsharing Betreiber

Tom Schneider, Gründer und Eigentümer der Carsharing Firma *Carsharing Club Hamburg (CCHH)*, kommt morgens in sein Büro und schaltet wie jeden Morgen erst einmal seinen Rechner an und holt sich während der Rechner hochfährt eine Tasse Kaffee aus der Küche. Er setzt sich an seinen Schreibtisch und gibt seinen Benutzernamen und sein Passwort ein, um sich einzuloggen. Danach sieht er in sein Email-Postfach und findet dort eine neue Nachricht von einem Ingenieur der Firma Govecs, mit denen er seit längerem kooperiert. Tom hatte zuvor um einen Termin für ein Telefonat gebeten und nun eine Bestätigung für ein Gespräch in einer knappen halben Stunde bekommen, genug Zeit, um einen Blick in das Service-Postfach zu werfen, an das seine Kunden Anregungen, Wünsche oder Beschwerden senden können. Es sind dort aber seit gestern Abend keine neuen Nachrichten eingegangen.
Damit er über den Zustand seiner Fahrzeugflotte informiert ist, schaut Tom in seinem Flottenmanagement-System (FMS) zunächst nach, ob auch alle Fahrzeuge dort sind, wo sie sein sollten. Die Software bietet ihm dazu eine Karte an, in der die Standorte der einzelnen Fahrzeuge markiert sind. Da dies der Fall ist, überprüft er zusätzlich, ob alle zur Verfügung stehenden Fahrzeuge aufgeladen sind und keine Warn- oder Fehlermeldungen gesendet haben. Mit Freude stellt er fest, dass seine gesamte Flotte vollständig einsatzbereit ist. Dies nimmt er zum Anlass, sich noch schnell die bereits getätigten Buchungen seiner Kunden anzusehen, bevor er das Telefonat mit Govecs tätigt. Dort findet er eine Buchung von Jenny Weber vor, die einen Govecs GO! S1.2 am Bahnhof Dammtor am Nachmittag für eine Stunde auszuleihen beabsichtigt. Vor dem Telefonat ruft er noch Simon Bern, einen seiner Service-Techniker, zu sich ins Büro, damit dieser dem Gespräch beiwohnt. Kurz

Tabelle 3.5: Featureliste Szenario 1

Nr.	Feature	Beschreibung
F16	Display	Das System verwendet ein TFT-Display, um sowohl die Anzeigeinstrumente als auch das Navigationssystem darzustellen.
F17	Designauswahl	Der Kunde kann aus einer Anzahl von Cockpitdesigns dasjenige auswählen, welches ihm am besten gefällt.
F18	Joystick	Für die Interaktion mit dem Navigationssystem befindet sich an dem Gasgriff ein kleiner Joystick, wie er z.B. in Controllern von Spielekonsolen zum Einsatz kommt. Der Joystick wird mit dem Daumen der rechten Hand bedient. Dadurch hat der Benutzer auch während der Interaktion beide Hände am Lenker.
F19	Buchungsbeginn	Nach erfolgreicher Verifikation des digitalen Fahrzeugschlüssels sendet das System Uhrzeit, Gesamtkilometerstand und die Schlüssel-ID an den Betreiber, damit er diese Daten für die spätere Abrechnung verwenden kann. Gleichzeitig startet das System das Fahrzeug.
F20	Startbildschirm	Während das System hochfährt, wird das Logo des Carsharing Betreibers und/oder des Fahrzeugherstellers angezeigt. Sobald der Bootvorgang beendet ist, werden die Anzeigeinstrumente dargestellt.
F21	Zugang zu Staufächern	Das System gibt den Zugang zu Staufächern frei, sobald das Fahrzeug freigeschaltet worden ist. Die Staufächer lassen sich dann per Knopfdruck öffnen und es wird kein Schlüssel mehr benötigt. Wenn das Fahrzeug ausgeschaltet wird, verriegelt das System die Fächer wieder.
F22	Darstellungsauswahl	Mit dem *Select*-Knopf des Fahrzeugs kann der Benutzer zwischen der Darstellung der Anzeigeinstrumente und der Navigationsansicht wechseln.
F23	Parken	Wenn das Fahrzeug freigeschaltet ist und der Schlüssel erneut übertragen wird, fragt das System den Benutzer, ob er das Fahrzeug parken oder den Buchungsvorgang abschließen will. Beim Parken wird das Fahrzeug ausgeschaltet und die Staufächer werden verriegelt.
F24	Buchung abschließen	Beim Abschluss der Buchung überprüft das System, ob das Fahrzeug zum Laden an eine Stromquelle angeschlossen wurde. Ist dies nicht der Fall, wird auf dem Display ein Dialogfeld angezeigt, in dem der Benutzer aufgefordert wird, das Fahrzeug an eine Steckdose anzuschließen und das Ende der Buchung erneut zu bestätigen. Ist das Fahrzeug bei Abschluss der Buchung mit einer Stromquelle verbunden, sendet das System Uhrzeit, Gesamtkilometerstand und Schlüssel-ID an den Betreiber, markiert den verwendeten Schlüssel als ungültig, verriegelt die Staufächer und schaltet das Fahrzeug aus.

darauf trifft Simon ein und Tom ruft den Ingenieur bei Govecs an.

In dem Telefonat geht es um die Verbesserung des Batteriemonitorings und -managements in den Elektrokrafträdern von Govecs. Dadurch sollen nicht nur die Reichweitenprognosen genauer erstellt, sondern auch Fehlfunktionen und Ableben der Akkumulatoren frühzeitig erkannt werden. Diese Informationen sollen dann genutzt werden, um Profile für die Akkus zu erstellen, damit gegebenenfalls besser geeignete Energiespeicher für die einzelnen Fahrzeugmodelle gefunden werden können. Während des Gesprächs diskutieren sie verschiedene Möglichkeiten, wie man mit den vorhandenen Informationen eine zuverlässigen Wert für den *State of Health (SoH)* berechnen kann. Schließlich einigen sie sich darauf, für die Berechnung die Anzahl der Ladezyklen und den Ladestand in Bezug auf die zurückgelegte Strecke zu verwenden.

Der Vorschlag des Govecs-Ingenieurs, auch den Entladestrom, dessen Wert bereits auf dem CAN-Bus der Fahrzeuge ausgegeben wird, in die Berechnung einfließen zu lassen, wird von Tom und Simon akzeptiert. Kurz vor Ende des Telefonats wirft Simon ein, das auch das Temperaturverhalten des Energiespeichers Einfluss auf dessen Lebenszeit hat. Er schlägt vor, mit Toms Erlaubnis einen Prototypen einer solchen Messeinrichtung zu konstruieren und das System, welches bereits in den Govecs-Rollern verbaut ist, entsprechend zu erweitern. Tom stimmt dem zu und auch der Govecs-Ingenieur ist von der Idee überzeugt. Da auch Govecs selbst ein großes Interesse an den Daten und den Ergebnissen der Analysen der Akkumulatoren hat, erstattet Govecs die Entwicklungs- und Materialkosten für die ersten zehn Prototypen. Zufrieden mit den Ergebnissen verabschieden sich Tom und Simon von dem Govecs-Ingenieur und beenden das Telefonat. Simon macht sich daraufhin an die Recherchearbeiten und auch Tom geht wieder seinen Geschäften nach.

Tabelle 3.6: Featureliste Szenario 2

Nr.	**Feature**	**Beschreibung**
F25	Warn- und Fehlermeldungen angeben	Das System sendet auftretende Warn- und Fehlermeldungen des Fahrzeugs an den Betreiber.
F26	Ladezyklusangabe	Das System meldet dem Betreiber, wenn ein neuer Ladezyklus initiiert wird.
F27	Entladestrom-angabe	Das System überträgt den aktuellen Entladestrom an den Betreiber.
F28	Temperatur des Akkus	Das System verfügt über einen Sensor zur Messung der Akkutemperatur
F29	Angabe der Temperatur des Akkus	Das System misst in regelmäßigen Abständen die Temperatur des Akkus und überträgt die Daten an den Betreiber.

3.3.3 Szenario 3: Der Service-Techniker

Kurz nachdem Simon Bern, Service-Techniker beim *Carsharing Club Hamburg (CCHH)*, am Morgen sein Büro betritt, klingelt das Telefon. Sein Chef, Tom Schneider, bittet ihn für eine Telefonkonferenz mit einem Ingenieur des Elektromotorrollerherstellers Govecs in sein Büro. In dem Gespräch geht es um die Verbesserung des Batteriemonitorings und -managements in den Govecs-Fahrzeugen des CCHH.
Im Anschluss an das Gespräch bekommt Simon die Aufgabe, das in den Govecs-Rollern verbaute System für die Integration der Fahrzeuge in den Carsharing Betrieb um Temperatursensoren für die Akkumulatoren zu erweitern und die Software des Systems um entsprechende Komponenten zu erweitern. Dazu gehört ein Modul zum Einlesen und Verarbeiten der Sensordaten, eine neue Warnmeldung auf dem Display des Fahrzeugs bei zu hohen Temperaturen und die Erweiterung des Moduls, welches Fahrzeugdaten an den Betreiber sendet, um ebd. Temperaturdaten. Zusätzlich soll auch der aktuelle Entladestrom dem Betreiber mitgeteilt werden. Die Änderung des Sendemoduls erfordert außerdem eine entsprechende Modifikation an dem Proxy-Server, der die versendeten Daten des Systems beim Betreiber entgegen nimmt und für das FMS aufbereitet, bevor sie in die Datenbank eingepflegt werden.
Simon beginnt sogleich mit den Recherchearbeiten. In der Dokumentation des CAN-Busses der Govecs-Roller entdeckt er, dass der Wert des Entladestroms bereits verfügbar ist. Dieser Wert wird auch schon für die Anzeige eines Ecometers auf dem Display des Systems verwendet. Er nimmt sich ein Testsystem zur Hand, welches dem aktuell in den Govecs-Rollern eingebauten System entspricht, und modifiziert das Sendemodul dahingehend, dass es obendrein den Entladestromwert versendet. Anschließend modifiziert er eine Kopie des Proxy-Servers, damit dieser auch den zusätzlichen Wert verarbeiten kann.
Nach weiteren Recherchen findet er Schaltpläne, Anleitungen und Bibliotheken, um Temperatursensoren via One-Wire-Bus in ein bestehendes Hardwaresystem zu integrieren. Gemäß den Schaltplänen stellt er eine Liste der benötigten Teile für zehn Prototypen auf und ermittelt deren Einkaufspreis. Nach Rücksprache mit seinem Chef bestellt er die Teile, die zwei Tage später eintreffen sollen. In der Zwischenzeit arbeitet Simon weiter an der Integration der neuen Sensoren. Er beginnt mit der Definition der Datenstruktur, welche die Sensordaten beinhalten soll, in einer XML-Datei, die von dem System eingelesen und verarbeitet werden kann. Nach Festlegung der Datenstruktur kann er nun auch das Sendemodul und den Proxy-Server entsprechend erweitern.
Am nächsten morgen berichtet Simon zunächst seinen Chef von den Fortschritten des Projektes. Danach implementiert er ein Programm, das Daten von dem One-Wire-Bus gemäß der Spezifikation der Temperatursensoren ausliest, aufbereitet und in die am Vortag definierte Datenstruktur schreibt. Um die Temperatursensoren zu simulieren, nimmt er seinen Arduino Uno zur Hand, den er von Zuhause mitgebracht hat. Er verbindet einen Digitalausgang des Arduino mit dem One-Wire-Bus des Systems und schreibt ein Programm, welches Daten in demselben Format und mit

derselben Frequenz wie die bestellten Temperatursensoren produziert und an dem Digitalausgang ausgeben kann. Mit diesem Setup kann er die Temperatursensoren simulieren. Noch vor Feierabend hat er auch das Anzeigeinstrument für die Temperaturwarnung fertig implementiert.
Als am nächsten Tag die bestellten Teile eintreffen, macht sich Simon gleich die Konstruktion des ersten Prototypen, den er erfolgreich in sein Testsystem integriert. Die Daten werden von den Sensoren über den One-Wire-Bus an das System gesendet, welches die Daten in die vordefinierte Struktur schreibt und so den anderen Komponenten zur Verfügung stellt. Ab einer in dem Anzeigeinstrument festgelegten Temperatur wird eine Warnung auf dem Display dargestellt. Der Proxy-Server empfängt sowohl die Sensorwerte als auch die Warnmeldung und den Entladestromwert.

Tabelle 3.7: Featureliste Szenario 3

Nr.	**Feature**	**Beschreibung**
F30	Erweiterbarkeit der Software	Das System ist so konstruiert, dass Komponenten zum Einlesen, Verarbeiten und Darstellen von Sensordaten leicht hinzugefügt werden können, ohne Einfluss auf andere Komponenten zu nehmen.
F31	Proxy-Server	Das System sendet gesammelte Daten an einen Proxy-Server, welcher die Daten entgegen nimmt und für das FMS des Betreibers aufbereitet, bevor sie in die Datenbank eingepflegt werden.
F32	Erweiterbarkeit der Hardware	Das System verfügt über verschiedene Bussysteme (z.B. USB, One-Wire, I2C), an die zusätzliche Sensoren und Aktoren angeschlossen werden können.
F33	XML-basierte Definition der Datenstrukturen	Datenstrukturen von Sensordaten lassen sich in einer XML-Datei definieren, die das System einlesen und verarbeiten kann. Das System erzeugt diese Datenstrukturen und erlaubt so anderen Komponenten des Systems den Zugriff auf die Sensordaten.

4 Konzeption

Diese Kapitel widmet sich dem soft- und hardwaretechnischen Konzept des Systems sowie der Modellierung der Interaktionsmöglichkeiten und der Gestaltung der Benutzungsschnittstellen.
Zu Beginn wird anhand der in Kapitel 3 identifizierten Features die Softwarearchitektur und deren einzelne Komponenten konzipiert. Darauf aufbauend wird die Hardware des Systems und die benötigten Schnittstellen und Kommunikationskanäle beschrieben. Auf die Benutzungsschnittstellen zur Interaktion mit dem System wird in Abschnitt 4.4.3 genauer eingegangen. Die Gestaltung eines Cockpits für Elektromotorroller bildet den Abschluss dieses Kapitels.

4.1 Featurekonsolidierung

In Kapitel 3 wurden aus den einzelnen Analysen Features extrahiert, welche nun zusammengefasst, kategorisiert und erweitert werden sollen. Mittels der konsolidierten Features werden Teilsysteme identifiziert, die als Grundlage für weitere Abstraktionen und die Ableitung einer geeigneten Architektur dienen.

Tabelle 4.1: Featurekategorien

Kategorie	**Feature**	**Beschreibung**
Versenden	F1, F2, F3, F25, F26, F27, F29	Daten, die an den Betreiber gesendet werden
Hardware	F4, F5, F16, F18, F22, F28 GPS-Sensor, CAN-Bus-Modul, UMTS-Modul	Hardware-Komponenten des Systems
Anzeige	F6, F7, F8, F9, F10, F11, F12, F13, F14, F15, F17, F20	Daten, die dem Fahrer auf dem Display angezeigt werden
Schlüssel	F19, F21, F23, F24	Funktionen des digitalen Fahrzeugschlüssels
Architektur	F30, F31, F32, F33	Eigenschaften und Komponenten der Architektur

Die Features legen eine Einteilung in Teilsysteme nahe. Welche Teilsysteme dies sind, wie sie miteinander interagieren und wie sie sich geeignet abstrahieren lassen, wird in den folgenden Abschnitten genauer erläutert. Das Komponentendiagramm in Abbildung 4.1 gibt bereits einen Überblick über die Teilsysteme und deren Zusammenwirken.

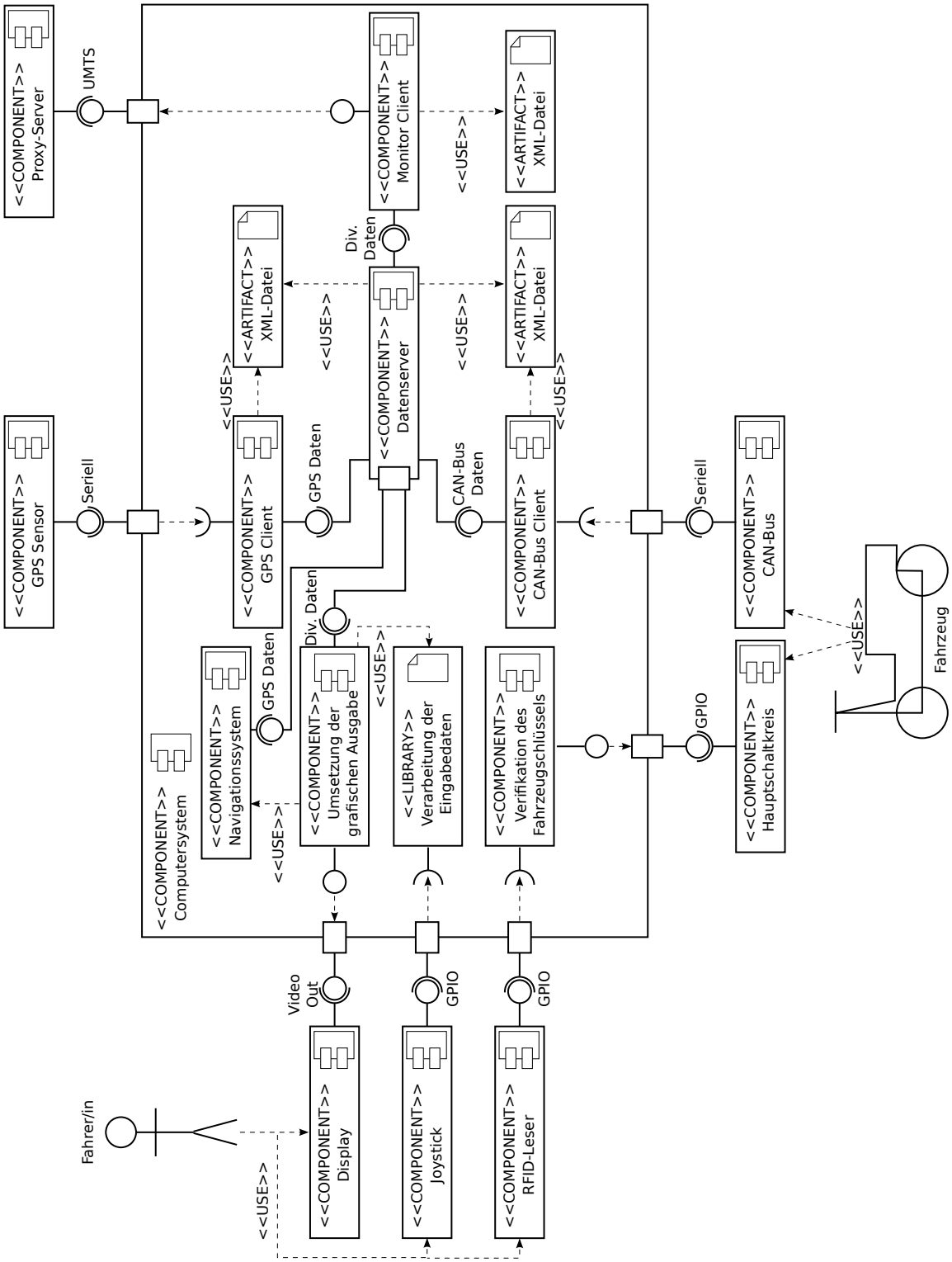

Abbildung 4.1: Komponentendiagramm

4.2 Softwarearchitektur

Um die anfallenden Daten innerhalb des Systems in geeigneter Weise verwalten zu können, bietet sich eine Client-Server-Architektur an. Ein Datenserver, der CAN-Bus- und Sensordaten entgegen nimmt und den Clients zur Verfügung stellt, bildet dabei die zentrale Komponente. Diejenigen Komponenten, welche Daten erzeugen, werden im Folgenden als *Datenproduzenten* bezeichnet. Solche, die Daten zur Anzeige, zum Versenden oder zur Steuerung des Systems verwenden, werden als *Datenkonsumenten* bezeichnet. Der Datenaustausch mit dem Server erfolgt über eine REST-Schnittstelle, um eine hohe Flexibilität zu ermöglichen. Um Polling seitens der Datenkonsumenten zu umgehen, können sich diese via Sockets über Änderungen der Daten vom Server informieren lassen. Abbildung 4.2 zeigt schematisch das Schichtenmodell mit Datenfluss und Schnittstellen.

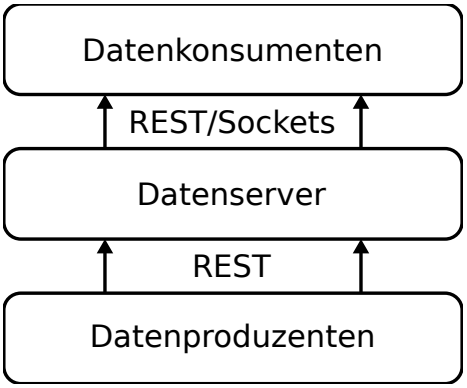

Abbildung 4.2: Schichtenmodell

4.2.1 Datenserver

Der Datenserver bildet die zentrale Komponente des Systems. Er speichert und verwaltet die anfallenden Daten und stellt Schnittstellen zum Speichern und Abrufen dieser Daten bereit. Die Schnittstellen werden durch eine REST-API realisiert, welche durch XML-Dokumente definiert wird. Die XML-Dokumente werden nach dem Schema in Listing 4.1 aufgebaut. Ein *module* stellt hierbei eine Datenquelle oder eine Gruppe von Datenquellen und ein *item* ein gemessenes Datum dar, wobei ein Modul beliebig viele Items erzeugen kann. Ein Item wird wiederum beschrieben durch einen Namen, den Datentyp des Wertes, seine Einheit und einen Initialwert.
Der Server liest die XML-Dateien beim Start des Systems oder durch eine von extern getriggerte Funktion ein, baut die entsprechende Datenstruktur auf und stellt die REST-Schnittstelle für das Modul bereit. Wird dem System beispielsweise ein Temperatursensor für den Energiespeicher hinzugefügt, könnte eine Schnittstellendefinition wie in Listing 4.2 dargestellt aussehen. Der Temperaturwert kann von dem Datenproduzenten mittels POST-Request an die URL *http://localhost/akkutemp/temp/value&value=42.0* auf den Wert 42,0 geändert und von Konsumenten über einen GET-Request an die URL *http://localhost/akkutemp/temp/value* abgefragt werden.

Listing 4.1: XML-Schema Schnittstelle

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="module">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="name"/>
        <xs:element name="items">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="item" maxOccurs="unbounded" minOccurs="1">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="name"/>
                    <xs:element type="xs:string" name="type"/>
                    <xs:element type="xs:string" name="unit"/>
                    <xs:element type="xs:string" name="value"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 4.2: Schnittstellenbeschreibung Temperatursensor

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<module>
  <name>akkutemp</name>
  <items>
    <item>
      <name>temp</name>
      <type>float</type>
      <unit>°C</unit>
      <value>0.0</value>
    </item>
  </items>
</module>
```

Damit ein Datenkonsument immer den aktuellsten Wert einer Ressource verarbeiten kann, bedarf es eines Mechanismus, durch den der Server den Datenkonsumenten über Wertänderungen der Ressource informieren kann. Durch einen solchen Mechanismus wird verhindert, dass der Konsument ständig beim Server anfragen muss, ob sich der Wert der Ressource geändert hat, wodurch sich der Kommunikationsaufwand zwischen Server und Konsument auf ein Minimum reduziert. Dies wird mit Hilfe von Websockets erreicht. Hier wird eine beständige HTTP-Verbindung zwischen Client und Server aufgebaut, über die Nachrichten zwischen den beiden Parteien ausgetauscht werden können. Der Datenserver stellt hierfür die Ressource *http://localhost/websocket* bereit, über die sich die Datenkonsumenten mit dem Server verbinden können.
Der Datenserver führt pro Ressource eine Liste mit den Websockets derjenigen Datenkonsumenten, die über Änderungen ebd. Ressource informiert werden wollen. Um in die Liste aufgenommen zu werden, müssen sich die Datenkonsumenten mit ihren Websockets für eine Ressource registrieren. Dafür wird ein einfaches Protokoll eingeführt, durch das die Registrierung vorgenommen und die Konsumenten über Wertänderungen informiert werden können. Die Konsumenten- und Server-Nachrichten können den Tabellen 4.2 und 4.3 entnommen werden. Der Ablauf einer Registrierung ist in Abbildung 4.3 zu sehen.

Tabelle 4.2: Konsumenten-Nachrichten

Typ	**Beschreibung**	**Beispielnachricht**
register	Der Client meldet sich für Benachrichtigungen bei Änderung einer bestimmten Ressource an.	register:akkutemp/temp/value
unregister	Der Client meldet sich von Benachrichtigungen bei Änderung einer bestimmten Ressource ab.	unregister:akkutemp/temp/value
status	Der Client kann beim Server anfragen, für welche Ressourcen er registriert ist.	status

Tabelle 4.3: Server-Nachrichten

Typ	**Beschreibung**	**Beispielnachricht**
info	Der Server sendet eine Informationsnachricht an den Client.	info:Your socket 0xDEADBEEF is now registered to resource list: akkutemp/temp/value
error	Bei der Verarbeitung der Nachricht ist ein Fehler aufgetreten.	error:Wrong message format
update	Der Wert einer Ressource hat sich geändert.	update:akkutemp/temp/value:47.11

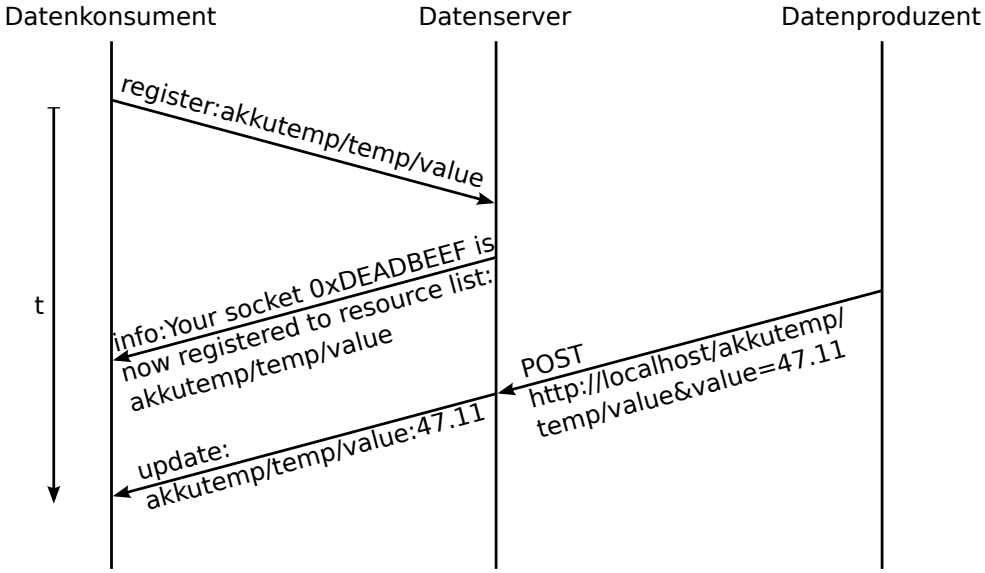

Abbildung 4.3: Ablauf der Registrierung und der Aktualisierung

4.2.2 Datenproduzenten

Datenproduzenten sind diejenigen Komponenten, welche Daten von einer Quelle auslesen, aufbereiten und an den Datenserver schicken. Aufgrund der REST-Architektur des Systems können solche Programme unabhängig vom Rest des Systems in einem eigenen Prozess ausgeführt werden und ihre Daten an den Server übertragen. Welche Daten dabei übertragen werden, wird in einer XML-Datei definiert, deren Schema bereits in Listing 4.1 erläutert wurde. Dabei ist keine Registrierung des Produzenten beim Server notwendig. Es muss lediglich die XML-Beschreibung der Daten vom Server eingelesen werden, damit dieser die entsprechenden Schnittstellen bereitstellen kann.

Zunächst werden die Werte von der Datenquelle ausgelesen und gegebenenfalls dekodiert. Die Datenquelle kann beispielsweise der CAN-Bus, ein GPS- oder Temperatursensor sein. Anschließend werden die gemäß der XML-Beschreibung aufbereitet. In einem letzten Schritt sendet der Datenproduzent die aufbereiteten Daten per HTTP POST-Request an die entsprechende Ressource des Datenservers.

4.2.3 Datenkonsumenten

Datenkonsumenten sind jene Teilsysteme, welche die anfallenden Daten in irgendeiner Form verwerten. Dabei kann es sich beispielsweise um einen Display-Client handelt, der für die Darstellung des Cockpits zuständig ist, oder eine Monitoring-Komponente, welche Daten sammelt und in regelmäßigen Intervallen an den Betreiber sendet. Ein Datenkonsument muss sich gemäß dem in

Abschnitt 4.2.1 beschriebenen Protokoll beim Server für eine oder mehrere Ressourcen registrieren, um über deren Änderungen informiert zu werden.
Beim Start des Datenkonsumenten wird eine Websocket-Verbindung zu dem Datenserver über die Ressource *http://localhost/websocket* hergestellt. Sobald die Verbindung steht, sendet der Konsument eine *register*-Nachricht mit einer Liste von Ressourcen, deren Wert es zu verarbeiten gilt, an den Server (vgl. Abb. 4.3). Sobald sich der Wert einer Ressource ändert, wird der Datenkonsument über die Änderung informiert und reagiert darauf entsprechend seiner Funktion.

4.3 Clients

Nachdem nun die zugrunde liegende Architektur feststeht, sollen im Folgenden die in der Feature-Liste identifizierten Clients konzipiert werden. Den Einstieg bildet der Display-Client, der die Darstellung der Anzeigeinstrumente und des Navigationssystems übernehmen soll. Im Anschluss daran wird auf das Monitoring des Fahrzeugs und die Verarbeitung der Daten auf Betreiberseite näher eingegangen. Sicherheitsaspekte bilden den Abschluss der Betrachtungen.

4.3.1 Display-Client

Für die Darstellung des Fahrzeugcockpits ist der Display-Client zuständig. Der Feature-Liste zufolge soll neben den obligatorischen Daten, die während der Fahrt benötigt werden, wie Geschwindigkeit, Kilometer- und Akkustand etc., auch das Navigationssystem und der sogenannte Bootsplash oder Bootscreen, der beim Starten des Computersystems angezeigt wird, dargestellt werden.
Der Bootscreen, der beispielsweise für Werbezwecke des Betreibers oder dessen Kooperationspartner verwendet werden kann, wird beim Start des Systems auf Betriebssystemebene abgehandelt und ist somit nicht Teil des Display-Clients. Das Computersystem kann allerdings nicht vollständig heruntergefahren werden, wenn das Fahrzeug ausgeschaltet wird, da sowohl das RFID-Lesegerät zur Freischaltung des Fahrzeugs als auch das Monitoring zur Überwachung der Fahrzeugposition weiterhin aktiv sein müssen. An dieser Stelle übernimmt der Display-Client die Anzeige des Bootscreens, während das Fahrzeug freigeschaltet und hochgefahren wird.
Sobald das Fahrzeug fahrbereit ist, wird das Cockpit mit den entsprechenden Anzeigeinstrumenten dargestellt. Wie in dem Kunden-Szenario in Abschnitt 3.3.1 beschrieben wurde, soll der Kunde die Möglichkeit haben, aus verschiedenen Cockpit-Designs auswählen zu können. Dies erfordert eine flexible Architektur seitens des Display-Clients. Eine solche Flexibilität bieten Web-Applikationen mit HTML5, CSS und JavaScript. HTML5 bietet mit seinem Canvas-Element eine Fläche zum Zeichnen von Objekten mittels JavaScript. Ein Anzeigeinstrument wie beispielsweise die Geschwindigkeitsanzeige wird als JavaScript-Widget realisiert, dessen Position und Aussehen durch eine entsprechende CSS-Datei beschrieben wird. Auf diese Weise können verschiedene Anzeigeinstrumente in unterschiedlichen Ausprägungen zu Cockpit-Designs kombiniert und dem

Kunden zur Auswahl angeboten werden. Die hierfür notwendigen Dateien können während der Buchung eines Fahrzeugs auf das Smartphone des Kunden übertragen und bei der Freischaltung des Fahrzeugs via NFC an das System übertragen werden.
Damit die Widgets auch die Werte erhalten, zu deren Anzeige sie vorgesehen sind, fungiert der Display-Client als Datendispatcher für die Widgets. Der Display-Client stellt über einen Websocket die Verbindung zum Datenserver her und registriert für jedes Widget die entsprechende Ressource. Bei einer Wertänderung ruft der Display-Client eine Callback-Funktion des Widgets auf und übergibt dieser den aktualisierten Wert. Das Widget übernimmt dann die Darstellung des Wertes.
Das Navigationssystem kann in diesem Kontext über ein HTML-Element in die Anzeige eingebunden werden. Je nach Implementierung des Navigationssystems kann dies ein weiteres HTML5-Canvas, ein *div*-Tag oder ein *iframe* sein. Zur Zeit werden am Institut für Softwaretechnik und Programmiersprachen der Universität zu Lübeck Anstrengungen unternommen, einen GreenNav-Client zu entwerfen, der plattformunabhängig auf Webtechnologien aufbaut und somit in das System integriert werden kann. Wie genau diese Integration vonstatten gehen kann, werden die weiteren Konzeptionen und die Umsetzung des neuen GreenNav-Clients ergeben. Prinzipiell erlaubt aber die bisher beschriebene Architektur eine Integration des GreenNav-Clients in das System.

4.3.2 Digitaler Fahrzeugschlüssel

Um eine kontaktlose Übertragung eines digitalen Schlüssels zur Freischaltung des Fahrzeugs und Öffnung der Staufächer zu ermöglichen, wird das von [Block, 2012] entwickelte System in das Konzept integriert. Dazu muss der dort verwendete Fahrzeug-Server, der für das Auslesen und die Verifikation des Schlüssels zuständig ist, so angepasst werden, dass er zum einen die Daten nicht selbst ausliest, sondern vom Datenserver erhält, und zum anderen das Ergebnis der Verifikation an den Datenserver zurück schickt. Der Datenserver signalisiert dann einer weiteren Komponente, welche die eigentliche Freischaltung übernimmt, dass der Schlüssel akzeptiert wurde.
Der Schlüsselserver des Block'schen Systems bedarf keiner Änderungen, da er unabhängig von dem System in dem Fahrzeug funktioniert und nur mit der Smartphone-App kommuniziert. Er kann sowohl als eigenständiger Server beim Betreiber als auch als Teil des Proxy-Servers realisiert werden, der in Abschnitt 4.3.4 beschrieben wird. Je nach Implementierung des Proxy-Servers erfordert letzteres eine Portierung des Schlüsselservers von Java in die Sprache des Proxys.

4.3.3 Monitoring

Laut Analyse und Feature-Liste sollen verschiedene Daten zu Abrechnungs- und Fahrzeugüberwachungszwecken an den Betreiber gesendet werden können. Dazu muss das System über eine Verbindung zum Carsharing Betreiber verfügen, doch dazu mehr in den Abschnitten 4.3.4 und 4.4.1. Zunächst soll es darum gehen, wie die Daten, die übertragen werden sollen, gesammelt und

aufbereitet werden und wie die Übertragung vonstatten gehen soll.
Es ist davon auszugehen, dass einige der zu versendenden Daten mit einer Frequenz von ein bis etwa zehn mal pro Sekunde aktualisiert werden. Würde bei jeder Wertänderung eine Nachricht versendet werden, erzeugte dies ein recht hohes Datenaufkommen und einen großen Kommunikationsoverhead. Um dies zu umgehen, werden die Daten nach Eintreffen beim Monitoring-Client zunächst mit einem Zeitstempel versehen und vorerst zwischengespeichert. Erst nach einem vorgegebenen Zeitintervall überträgt der Monitoring-Client die gesammelten Daten in einer einzigen Nachricht an den Betreiber. Die Länge des Zeitintervalls kann auf einen bestimmten Wert festgelegt oder davon abhängig gemacht werden, ob sich das Fahrzeug bewegt oder nicht. Bei einem festen Intervall erscheint ein Wert von zwei bis drei Sekunden sinnvoll. In dieser Zeit legt ein Fahrzeug bei einer Geschwindigkeit von 50 km/h eine Strecke von etwa 28 bis 42 Metern zurück, was für eine Überwachung der Fahrzeugposition hinreichend genau sein sollte. In der Zwischenzeit anfallende Messwerte werden in einer Nachricht mit übertragen, sodass beim Betreiber die komplette Messreihe rekonstruiert werden kann.
Ereignisse, die Warn- und Fehlermeldungen des Fahrzeugs und des Systems herbeiführen, können beim Auftreten eine gesonderte Nachricht an den Betreiber auslösen. Dazu zählen insbesondere solche Ereignisse, welche die Funktion und damit die Verfügbarkeit des Fahrzeugs beeinträchtigen, wie zum Beispiel eine Überhitzung des Energiespeichers. In diesem Fall kann der Monitoring-Client auch dafür genutzt werden, das Fahrzeug aus Sicherheitsgründen abzuschalten und den Betreiber zu kontaktieren, damit dieser seinen Service-Techniker über die Fehlfunktion des Fahrzeugs informieren kann.

4.3.4 Proxy-Server

Der Proxy-Server dient als Schnittstelle zwischen dem Monitoring-Client des Fahrzeugs und dem Flottenmanagementsystem respektive der Datenbank des Betreibers. Zum einen wird dadurch verhindert, dass eine direkte Verbindung zwischen Fahrzeug und Datenbank bzw. FMS des Betreibers besteht, zum anderen dient er dazu, die empfangenen Daten für die Verwaltung und Speicherung beim Betreiber den dortigen Systemeigenschaften anzupassen. Er kann also als Middleware zwischen Fahrzeug und Betreiber verstanden werden.
Je nach verwendetem System seitens des Betreibers kann der Proxy-Server als anwendungs-, kommunikations- oder nachrichtenorientierte Middleware realisiert werden. Dies ist abhängig von der Offenheit des beim Betreiber verwendeten Systems. Im einfachsten Fall besitzt das System des Betreibers eine REST- oder RPC-Schnittstelle, für welche die empfangenen Nachrichten übersetzt werden müssen. Im Worst Case ist das System vollständig proprietär und die Nachrichten, die eine entsprechende Funktion zur Speicherung oder Anzeige der Daten auslösen, müssen komplett durch Reverse Engineering rekonstruiert werden.
Soll der Proxy auch die Funktion des Schlüsselservers wie in Abschnitt 4.3.2 beschrieben überneh-

men, muss der von [Block, 2012] entwickelte Schlüsselserver als Komponente des Proxys realisiert und entsprechend in die Sprache des Proxys portiert werden. Die öffentlichen Schlüssel der Fahrzeuge können in der Datenbank des Betreibers abgelegt werden, und der Schlüsselserver bedient sich dieser Schlüssel zum Erzeugen der eigentlichen Fahrzeugschlüssel für die Kunden. Auch hier ist die Umsetzung abhängig von dem verwendeten System des Betreibers.

4.3.5 Sicherheitsaspekte

Kein System ist 100%ig sicher, so auch dieses hier nicht. Es können aber Maßnahmen ergriffen werden, um es möglichst gut gegen Angriffe von außen zu schützen. Die Schutzmaßnahmen können auf unterschiedlichen Ebenen des ISO-OSI-Referenzmodells greifen. Je tiefer dabei die Verschlüsselung erfolgt, desto transparenter ist sie für die darüber liegende Applikation. Daher ist es sinnvoll, einen Sicherheitsmechanismus auf den unteren Schichten des Referenzmodells zu integrieren.
Auf der Vermittlungsschicht (Layer 3) wird eine *Stateful-Inspection*-Firewall wie *iptables* eingesetzt, um nicht-autorisierte Verbindungen mit dem System zu verhindern. Die Firewall wird so konfiguriert, dass sie zunächst sämtliche eingehenden Verbindungen auf allen Ports blockiert. Möchte der Betreiber eine Verbindung zu dem System in dem Fahrzeug herstellen, um beispielsweise Softwareupdates über eine SSH-Verbindung zu installieren, muss er zunächst Verbindungsversuche an eine zuvor festgelegte Sequenz von Ports unternehmen. Erst wenn an all diesen Ports in der richtigen Reihenfolge „angeklopft“ wurde, öffnet die Firewall für eine kurze Zeit den Port für die SSH-Verbindung und der Betreiber kann sich mit dem Fahrzeug verbinden. Dieses Verfahren wird *Port-Knocking* genannt und kann mit Hilfe von iptables-Regeln implementiert werden ([Shaw, 2013]). Die Port-Sequenz kann prinzipiell beliebig lang sein und lässt sich für jedes Fahrzeug individuell konfigurieren.
Als weitere Schutzmaßnahme wird das gesamte System in ein verschlüsseltes *Virtual Private Network* (VPN) eingebettet. Ein VPN dient dazu, Teilnehmer aus einem in sich abgeschlossenen Netzwerk mit einem anderen Netzwerk zu verbinden. Für den Aufbau des verschlüsselten VPNs kommt eine IPsec-Implementierung zum Einsatz, die ebenfalls auf der Vermittlungsschicht des ISO-OSI-Referenzmodells angesiedelt ist. IPsec (Kurzform von Internet Protocol Security) ist eine Sammlung von Protokollen, die eine sichere Verbindung über potentiell unsichere IP-Netzwerke wie dem Internet erlauben. Der Zugang zu dem VPN erfolgt über Zertifikate, die der Betreiber für jedes Fahrzeug erstellt und signiert. Anhand des Zertifikates können sich die VPN-Teilnehmer gegenseitig identifizieren und den Verbindungsaufbau zulassen. Ohne vom Betreiber signiertes und gültiges Zertifikat ist keine Verbindung zu dem VPN möglich.
Da das System eine Verbindung zu dem CAN-Bus des Fahrzeugs aufbaut, ist prinzipiell eine Manipulation des Fahrzeugs über diese Schnittstelle möglich. Um solche Eingriffe zu verhindern, müssen Änderungen an den Zugriffsmechanismen auf den CAN-Bus vorgenommen werden, damit ein Schreiben von Daten auf den Bus nicht mehr möglich ist. Zum einen können die Treiber des CAN-

Bus-Controllers dahingehend modifiziert werden, dass Nachrichten lediglich von dem Bus ausgelesen, aber nicht auf den Bus geschrieben werden können. Zum anderen können diese Änderung an den Abstraktionsschichten zum Zugriff auf den CAN-Bus, wie beispielsweise SocketCAN, welche den CAN-Bus als Netzwerkschnittstelle abstrahiert, vorgenommen werden. Auf diese Weise wird verhindert, dass beispielsweise die Geschwindigkeitsbegrenzung des Fahrzeugs ausgehebelt wird und das Fahrzeug schneller fahren kann als die zugelassene Höchstgeschwindigkeit.
Trotz dieser Maßnahmen bleiben einige Angriffspunkte in dem System, die nicht ohne Weiteres abgesichert werden können. Dies sind insbesondere die Hardware-Schnittstellen und deren Protokollimplementierungen. Es ist denkbar, dass es einem Angreifer gelingt, über die RFID-Schnittstelle des Systems Schadcode einzuschleusen, der ihm Zugriff auf das System gewährt. Ebenso ist ein Angriff auf die UMTS-Protokollschichten vorstellbar. Verschafft sich ein Angreifer Zugriff auf die Hardware des Systems, können keine weiteren Maßnahmen mehr getroffen werden, um einen Angriff zu verhindern.

4.4 Hardware

Der folgende Abschnitt beschäftigt sich mit den Eigenschaften der Hardware des hier zu entwickelnden Systems. Neben dem zentralen Computersystem werden Ein- und Ausgabemöglichkeiten zur Interaktion mit dem Benutzer antizipiert. Im weiteren Verlauf werden die benötigten Schnittstellen und Kommunikationskanäle beschrieben.

4.4.1 Computersystem

Der verwendete Computer soll in der Lage sein, ein unix-ähnliches Betriebssystem mit Linux-Kernel zu betreiben. Dadurch bietet er eine modulare und flexible Plattform für das System. Da das System in Elektromotorrollern zum Einsatz kommt, sollte der Computer möglichst wenig Energie verbrauchen und trotzdem ausreichend Leistung bieten, die bisher beschriebenen Aufgaben zu bewältigen. Es müssen Schnittstellen vorhanden sein, um zusätzlich benötigte Hardware in das System integrieren zu können, wie ein CAN-Bus Lesegerät, RFID- und UMTS-Modul. Der Rechner sollte zudem über GPIOs für die Anbindung von Sensoren oder weiteren Bussystemen verfügen.
In den letzten Jahren haben sich Scheckkarten große Single-Board Computer erfolgreich auf dem Markt etabliert und erfreuen sich großer Beliebtheit ([heise Open, 2013]). Sie bieten eine Vielzahl an Schnittstellen und eignen sich aufgrund ihrer Größe und Konnektivität zum Bau effizienter eingebetteter Systeme und Prototypen. Die Kosten für einen solchen Computer belaufen sich auf ca. 30 bis 100 Euro, je nach Modell und Ausstattung. Einer der bekanntesten Vertreter dieser Sorte Computer ist der Raspberry Pi. Der BeagleBone Black und das Parallella Board sind erst vor Kurzem auf dem Markt erschienen und bieten im Vergleich zum Raspberry Pi mehr GPIO-Pins, mehr Arbeitsspeicher und einen höheren CPU-Takt.

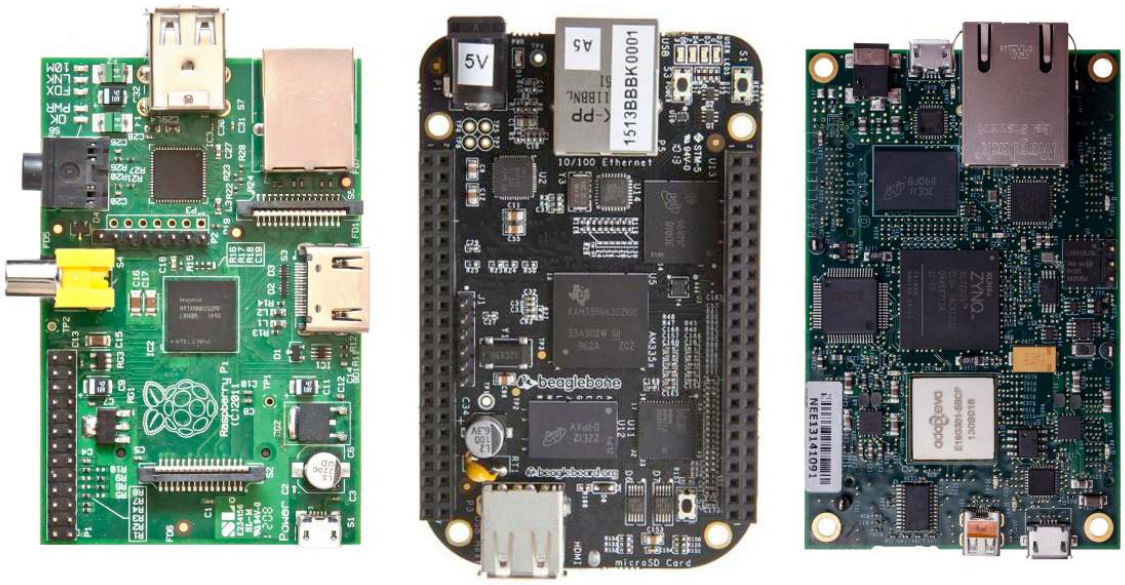

Abbildung 4.4: Raspberry Pi[1], BeagleBone Black[2] und Parallella Board[3]

Viele dieser kleinen Single-Board Computer verfügen bereits über HDMI-, Ethernet- und USB-Schnittstellen sowie I2C-Bus, JTAG- und SPI-Interface und mehrere GPIO-Pins. Nicht vorhandene, aber benötigte Schnittstellen und Sensoren wie GPS, RFID und UMTS können über die USB-Schnittstelle nachgerüstet werden. Da die Single-Board Computer in der Regel nur zwei USB-Anschlüsse besitzen, muss bei entsprechender Anzahl an USB-Peripherie zusätzlich ein USB-Hub an das System angeschlossen werden. CAN-Bus Lesegeräte werden je nach Ausführung an die USB- oder SPI-Schnittstelle angeschlossen. Der BeagleBone Black und das Parallella Board haben bereits einen CAN-Transceiver mit an Bord, der Signale über GPIO-Pins einlesen und ausgeben kann.

4.4.2 Spannungsversorgung

Die Single-Board Computer werden über ein USB-Kabel mit 5V Betriebsspannung versorgt ziehen unter Last je nach Modell zwischen 0,46A und 1A Strom. Spannungsquellen in Fahrzeugen liefern in der Regel 12V, also bedarf es zum Betrieb eines solchen Single-Board Computers einen Spannungswandler von 12V auf 5V. Ein solcher Wandler ist in Form eines Netzteils für Zigarettenanzünder mit USB-Ausgang in fast jedem größeren Supermarkt mit Elektroabteilung für wenig Geld zu finden. Da aber die Spannungsquellen in einem Fahrzeug mit Abschalten des Fahrzeugs

[1]Quelle: https://upload.wikimedia.org/wikipedia/commons/9/90/Front_of_Raspberry_Pi.jpg
[2]Quelle: http://www.ti.com/ww/en/beagleboard/product_detail_black_lg.jpg
[3]Quelle: http://2013.oshwa.org/files/2013/08/parallella_product_photo3_2000x1175.jpg

ebenfalls abgeschaltet werden, wird der Computer und die Peripherie dann auch nicht mehr mit Spannung versorgt.
Hier schafft ein Puffer-Akku Abhilfe. Der Puffer-Akku wird zwischen die Spannungsquelle und den Single-Board Computer geschaltet, um den Rechner und die Peripherie auch nach Abschalten des Fahrzeugs mit Spannung zu versorgen. Für diesen Zweck können sogenannte Power-Packs eingesetzt werden, die dazu gedacht sind, mobile Geräte wie Smartphones oder Tablets aufzuladen, wenn keine Steckdose oder kein Netzteil vorhanden sind. Die Power-Packs haben Kapazitäten von 15Ah und mehr bei einer Ausgangsspannung von 5V, womit beispielsweise ein Raspberry Pi über 20 Stunden betrieben werden kann. Ist das Fahrzeug eingeschaltet, wird der Puffer-Akku geladen und der Computer weiterhin von diesem mit Spannung versorgt. Um den Energieverbrauch zu reduzieren, kann bei ausgeschaltetem Fahrzeug nicht benötigte Hardware von dem System abgeschaltet werden.
Als Alternative zu der Lösung mit dem Puffer-Akku kann in dem Fahrzeug auch eine eigene Spannungsversorgung für den Single-Board Computer installiert werden, die ihren Strom direkt von dem Fahrzeugakku bezieht. In dem Fall wäre der Computer und die Peripherie auch bei ausgeschaltetem Fahrzeug mit Spannung versorgt. Dies erfordert allerdings größere Eingriffe in die Fahrzeugelektrik.

4.4.3 Ein- und Ausgabe

Für die Interaktion mit dem Benutzer bedarf es einiger Peripherie, die im Folgenden näher beschrieben wird. Dabei handelt es sich um ein Display zur Informationsausgabe und einem Joystick zur Informationseingabe durch den Benutzer.

Display

Für die Darstellung des Navigationssystems und der Anzeigeinstrumente während der Fahrt muss das System über ein Display verfügen, welches zur Ausgabe dieser Elemente verwendet werden kann. Damit die Anzeigen gut sicht- und lesbar dargestellt werden können, sollte das Display eine Diagonale von mindestens 5,6 Zoll aufweisen, dabei aber aus Platzgründen in der Fahrzeugkonsole eine Größe von 7 Zoll nicht überschreiten.
Die Auflösung des Displays soll heutigen Standards entsprechen und 800x600 Pixel nicht unterschreiten, um eine ansprechende Darstellungsqualität liefern zu können. Ein Display mit einem Seitenverhältnis von 16:9 oder 16:10 ist gegenüber einem 4:3-Display aufgrund des menschlichen Sichtfeldes ([Goldstein, 2002]) und der größeren Darstellungsfläche vorzuziehen.
Da die meisten Single-Board Computer über eine HDMI-Schnittstelle verfügen, sollte das Display eine entsprechende Anschlussmöglichkeit bieten. Die meisten Displays zum Einbau sind allerdings nur mit einer LVDS-Schnittstelle ausgestattet. In diesem Fall muss ein entsprechendes Controller-

Board die Übersetzung des HDMI-Signals in ein LVDS-Signal übernehmen. Solche Boards gibt es in verschiedenen Ausführungen zu unterschiedlichen Preisen. Viele Anbieter verkaufen auch Displays, bei denen ein passendes Board gleich mitgeliefert wird.
Aus Gründen der Sicherheit des Fahrers sollte auf ein Display mit Touchscreen-Funktionalität verzichtet werden. Für die Bedienung eines Touchscreens müsste der Fahrer eine Hand vom Lenker lösen, um auf dem Bildschirm Aktionen auszuführen. In dem Fall könnte bereits ein Kieselstein auf der Straße ausreichen, um einen Ruck am Lenker zu verursachen und den Fahrer samt Fahrzeug zu Fall zu bringen. Für Informationseingaben in das System seitens des Benutzers ist ein eigenes Eingabegerät vorgesehen.

Joystick

Damit der Benutzer Eingaben in das System tätigen kann, wird das Fahrzeug mit einem kleinen Joystick versehen, wie er beispielsweise in Controllern von Spielkonsolen vorkommt. Er wird direkt neben dem Gasgriff angebracht, damit er mit dem Daumen der rechten Hand leicht zu bedienen ist. Der Joystick kann in vier Richtungen bewegt werden und beinhaltet zusätzlich einen Drucktaster, der durch Druck auf den Steuerknüppel betätigt werden kann. Darüber hinaus befinden sich unterhalb des Joysticks zwei Drucktaster, mit denen der Benutzer weitere Funktionen auslösen kann. Diese Funktionen sowie die Bedienung der Systemelemente mit dem Joystick werden im Folgenden beschrieben.

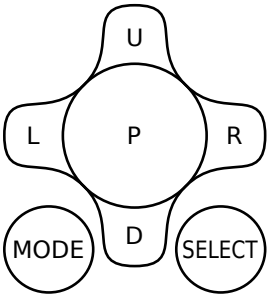

Abbildung 4.5: Schematische Darstellung der Bedienelemente

Der Mode- und Select-Taster sind dem Cockpit des Govecs GO! S1.2 entnommen, deren Funktion wird aber dem hier zu entwickelnden System angepasst. Mit dem Mode-Taster kann der Benutzer die Anzeige des Kilometerstandes zwischen Tages- und Gesamtkilometer umschalten. Sollte ein Anzeigeinstrument auf dem Display diese beiden Größen gleichzeitig darstellen, ist dieser Taster ohne Funktion. Wird der Mode-Taster länger als drei Sekunden gedrückt gehalten, wird der Tageskilometerzähler zurückgesetzt. Mit Hilfe des Select-Tasters wechselt der Benutzer zwischen der Anzeige der Messinstrumente und des Navigationssystems. Wird der Select-Taster länger als drei Sekunden gedrückt gehalten, erfolgt eine Umschaltung der Anzeigen von km/h in mph, bzw. von km in mi. Bei dem Govecs GO! S1.2 können Mode- und Select-Taster auch gleichzeit län-

ger als drei Sekunden gehalten werden, um die Uhrzeit auf dem Display des Rollers einzustellen ([Govecs, 2011]). Diese Funktion wird hier nicht benötigt, da die Uhrzeit über das GPS-Signal eingestellt werden kann.

Werden die Messinstrumente auf dem Display angezeigt, ist der Joystick ohne Funktion. Erst bei der Anzeige des Navigationssystems kann der Joystick für Eingaben genutzt werden. Wird der Joystick nach links (l), rechts (r), oben (u) oder unten (d) bewegt, verschiebt sich der angezeigte Kartenausschnitt in die entsprechende Richtung. Bei Druck auf den Joystick (p) und einer Bewegung nach oben oder unten, während der Joystick gedrückt bleibt, wird der Kartenausschnitt verkleinert (Zoom in) bzw. vergrößert (Zoom out). Eine Bewegung nach links oder rechts hat hierbei keine Funktion. Ein einfacher Druck auf den Joystick setzt die Verschiebung und den Zoom wieder auf ihre Grundeinstellungen zurück, sodass die aktuelle Position in der Mitte der Karte zu sehen ist.

Wird der Joystick länger als drei Sekunden gedrückt gehalten, öffnet sich ein Dialogfeld, in das der Benutzer die Eingabe seines Reisezieles tätigen kann. Dieses Dialogfeld kann jederzeit durch erneutes Gedrückthalten geschlossen werden, ohne Änderungen an der zuletzt getätigten Eingabe vorzunehmen. Da keine Tastatur zur alphanumerischen Eingabe vorhanden ist, müssen andere Verfahren zur Anwendung kommen. Hier bieten sich Patricia-Tries an, durch die der Benutzer mit Hilfe des Joysticks navigieren kann.

In dem Dialogfeld kann der Benutzer zunächst die Stadt eingeben, in der sich sein Reiseziel befindet. Der Name der Stadt, in der er sich gerade befindet, ist dabei bereits in das Feld eingetragen und ein Cursor befindet sich unterhalb des ersten Buchstabens des Städtenamens. Durch Bewegen des Joysticks nach oben oder unten wählt der Benutzer den alphanumerisch kleineren bzw. größeren Buchstaben, bewegt sich also innerhalb des Patricia-Tries auf einer Ebene nach links oder rechts. Bewegt er den Joystick nach rechts oder links, wählt er in dem Städtenamen den nächsten bzw. vorherigen Buchstaben aus, den er ändern möchte. Bestätigen kann er den Städtenamen durch Drücken des Joysticks.

Als Nächstes wählt der Benutzer, ob er einen Straßennamen oder einen POI als Ziel eingeben möchte. Dazu werden ihm zwei Buttons angezeigt, zwischen denen er mit einer Links- oder Rechtsbewegung des Joysticks wechseln kann. Betätigt wird die Auswahl durch Drücken des Joysticks.

Je nach vorheriger Auswahl kann der Benutzer nun den Namen der Straße bzw. des POIs eingeben. Die Eingabe erfolgt analog zu der des Städtenamens. Wurde die Eingabe eines Straßennamens gewählt, so kann der Benutzer nach Bestätigung des Straßennamens die Hausnummer auf dieselbe Weise Eingeben.

In einem letzten Schritt wählt der Benutzer aus, ob ihm die schnellste, kürzeste oder sparsamste Route angegeben werden soll. Hier werden ihm wie auch bei der Wahl zwischen Straßennamen und POI Buttons mit den entsprechenden Aufschriften angezeigt, die er durch Links- und Rechtsbewegung des Joysticks selektieren und durch Drücken des Joysticks bestätigen kann. Nach der Bestätigung wird die entsprechende Route auf der Karte des Navigationssystems angezeigt.

4.5 Interface Design

Für den Prototypen des Systems wurden Anzeigeinstrumente und ein Navigationssystem entworfen. Die Gestaltung der Anzeigeinstrumente ist teilweise an das Cockpitdesign des Govecs GO! S1.2 angelehnt. In Abbildung 4.6 sind die in Tabelle 3.3 gelisteten Elemente zu sehen. Die Geschwindigkeitsangabe bildet hier das zentrale und größte Element, rechts daneben befinden sich die Angaben zum Ladestand und eine Reichweitenprognose. Unterhalb der Geschwindigkeit wird der Kilometerstand angezeigt, die mittels des Mode-Tasters zwischen Tages- (Trip) und Gesamtkilometerstand (Odo) umgeschaltet werden kann. Das Ecometer wurde hier als Energieverbrauchsanzeige am unteren Displayrand realisiert, die von grün über gelb nach rot verläuft. Je mehr Energie verbraucht wird, desto weiter schlägt die Anzeige aus und desto mehr gleiten die Farben ins Rote.

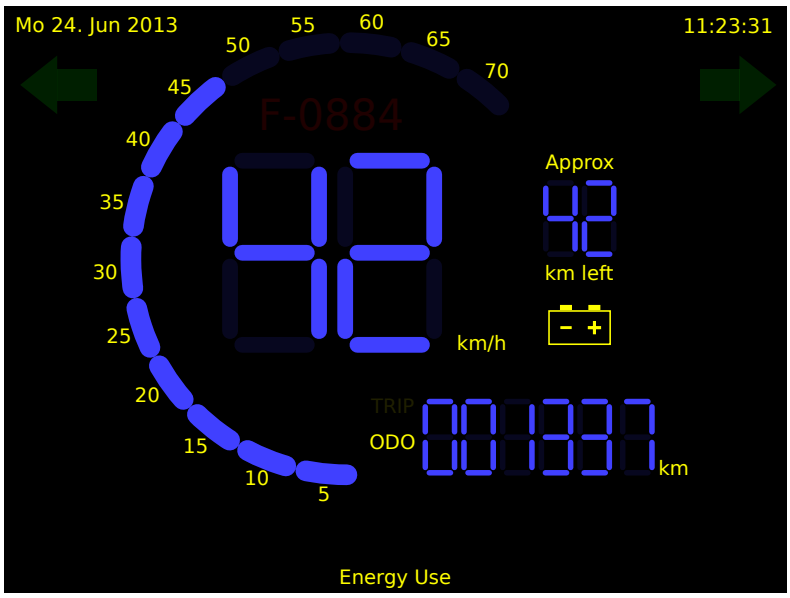

Abbildung 4.6: Entwurf der Anzeigeinstrumente 4:3

In der Navigationsansicht in Abbildung 4.7 ist neben der Karte mit der Route noch die Geschwindigkeit und der Akkustand zu sehen. Der Kartenausschnitt kann wie in Abschnitt 4.4.3 beschrieben manipuliert werden, damit der Fahrer sich die Karte so einrichten kann, wie er es für angemessen hält. Unterhalb der Karte befinden sich Richtungs- und Längenangaben, die dem Benutzer mitteilen, in welche Richtung er aktuell und an dem nächsten Wegpunkt fahren muss, wie weit es bis zum nächsten Wegpunkt ist und welche Strecke er noch bis zu seinem Ziel zurücklegen muss.

Abbildung 4.7: Entwurf des Navigationssystems 16:9

5 Realisierung

In diesem Kapitel wird die Umsetzung des Konzeptes aus dem vorherigen Kapitel beschrieben. Dazu wird zunächst auf die verwendete Hardware und das Fahrzeug für den Prototypen eingegangen. Anschließend werden die benötigten und implementierten Softwarekomponenten erläutert. Auch die während der Umsetzung aufgetretenen Probleme und retardierenden Faktoren werden in diesem Kapitel diskutiert.

5.1 Hardware

Die Hardware des Testsystems besteht im Wesentlichen aus zwei Komponenten: Einem Govecs GO! S1.2 Elektromotorroller (Abb. 5.1), der freundlicherweise von der Firma *Move About*[1] zur Verfügung gestellt wurde, und einem Raspberry Pi Model B. Im Folgenden werden diese Komponenten näher beschrieben.

Abbildung 5.1: Govecs GO! S1.2[2]

[1] http://www.moveabout.biz
[2] Quelle: http://www.voelkner.de/products/267783/100-xl.jpg

5.1.1 Elektromotorroller

Der Govecs GO! S1.2 ist ein Elektromotorroller der 50 ccm Klasse. Seine technischen Daten können Tabelle 5.1 entnommen werden ([Golla, 2012]). Das Fahrzeug verfügt über einen CAN-Bus, der über einen 4-poligen MATE-N-LOK^{TM} Steckverbinder nach außen zugänglich gemacht ist (vgl. Abb. 5.2). Die Pinbelegung des Steckverbinders (zu sehen in Abb. 5.4) wurde mit einem Agilent 54615B Oszilloskop gemessen, das freundlicherweise vom Institut für Technische Informatik der Universität zu Lübeck bereitgestellt wurde.
Über die Daten, die über den CAN-Bus gesendet werden, sowie deren Kodierung war nach Anfrage bei Govecs aus Vertraulichkeitsgründen keine Auskunft zu bekommen. Daher wurde das CAN-Bus Signal analysiert und die Kodierung der Daten durch Reverse Engineering herausgefunden. In Abschnitt 5.2.3 wird ausgiebig darüber berichtet und die Daten aufgeschlüsselt.

Tabelle 5.1: Technische Daten des Govecs GO! S1.2

Leistung	Höchstgeschwindigkeit	45 km/h
	Reichweite	50 - 60 km
Komponenten	Bremsen	Vorne und hinten hydraulische Scheibenbremsen
	Reifen	130/60-R13 vorne und hinten
	Rahmen	Stahlrohrgitterrahmen
Batterie und Elektronik	Typ	Silizium
	Kapazität	Ca. 2 kWh
	Batteriegewicht	55 kg
	Nominale Spannung	96 Volt
	Ladegerät	96V/4A, integriertes Ladegerät, 110 - 240V (50/60 Hz)
	Ladezeit	4 - 5h, ca. 2h bis 85% Ladekapazität
	Batterie (erwartete Lebensdauer)	20.000 km
Motor	Motor	Bürstenloser Wechselstrommotor mit Riemenantrieb
	Drehmoment	54 Nm
Abmessungen	Gewicht	ca. 135 kg (inkl. Batterien)
	Sitzhöhe	790 mm
	Stauraum	10 Liter
	Tragkraft	2 Passagiere / 150 kg
Preis	inkl. MwSt.	3.498 EUR

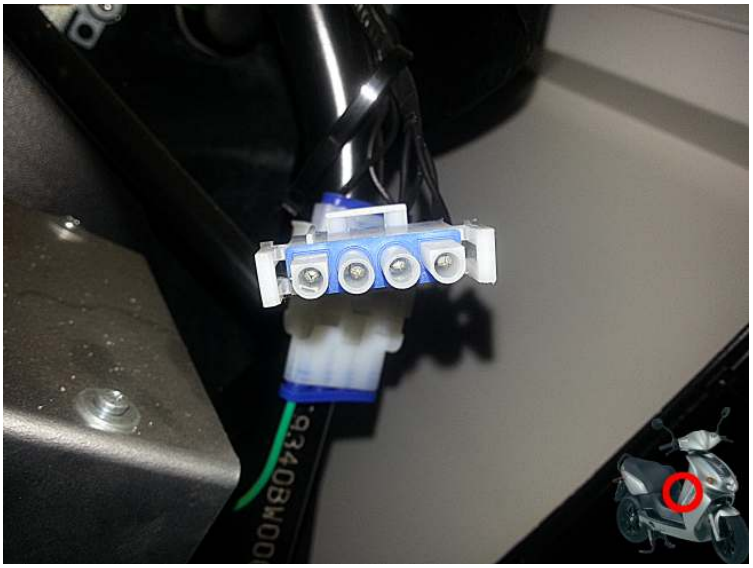

Abbildung 5.2: CAN-Bus Anschluss des Govecs GO! S1.2

Probleme mit dem Ladegerät

Während der Entwicklung des Systems versagte das integrierte Ladegerät des Rollers aus bisher ungeklärter Ursache seinen Dienst. Von einen Mitarbeiter von *Move About* wurden Kontaktdaten eines Field Support Technikers übermittelt, von dem sich aber herausstellte, dass er gar nicht mehr bei Govecs arbeitet. Über die Field Support Managerin von *Govecs Poland Sp. z.o.o.* wurde schließlich Kontakt zu dem Field Support Techniker Rafał Stankiewicz in Polen hergestellt, der sich des Problems umgehend annahm und ein neues Ladegerät in den Versand gab. Das Ladegerät in dem Roller wurde nach Eintreffen des Ersatzgerätes unverzüglich ausgetauscht und das Fahrzeug war wieder voll funktionsfähig. Vom Auftreten bis zur Behebung des Problems gingen drei Wochen ins Land, in denen nicht an dem System gearbeitet werden konnte.

5.1.2 Computersystem

Als zentraler Fahrzeugrechner kommt ein Raspberry Pi Model B zum Einsatz, einerseits wegen seiner vielfältigen Anschlussmöglichkeiten und dem großen Angebot an Peripherie, andererseits aufgrund des niedrigen Preises. Der Raspberry Pi ist mit einem 700 MHz Low Power ARM1176JZ-F RISC-Prozessor (ARMv6 Architektur), 512 MB SDRAM, 8 GPIO-Pins, I2C- und SPI-Bus, sowie zwei USB-Anschlüssen, einer HDMI- und einer 10/100 Ethernet-Schnittstelle ausgestattet. Die CPU kann bis auf 1 GHz getaktet werden. Über die 26-Pin Leiste kann der Raspberry Pi mit sogenannten *Shields* erweitert werden.

Für das Auslesen der CAN-Bus Daten sorgt ein PICAN CAN-Bus Board der Firma SK Pang Electronics Ltd. Das Board ist mit einem MCP2515 CAN Controller und einem MCP2551 CAN Transceiver bestückt. Es wird direkt auf die 26-Pin Leiste des Raspberry Pi aufgesteckt und lässt sich über einen 9-poligen Sub-D Stecker oder über eine 3-polige Schraubbuchse mit einem CAN-Bus verbinden. Das Board ist CAN v2.0B kompatibel und kann Daten mit einer Rate von bis zu 1 Mb/s auslesen und übertragen. Für die Kommunikation mit dem Raspberry Pi verwendet es den SPI-Bus bei einer Taktung von 10 MHz ([SK Pang Electronics Ltd., 2013]). Auf der Webseite des Herstellers finden sich zudem fertig kompilierte Kernel-Module und CAN-Utilities, sowie eine Installations- und Testanleitung.

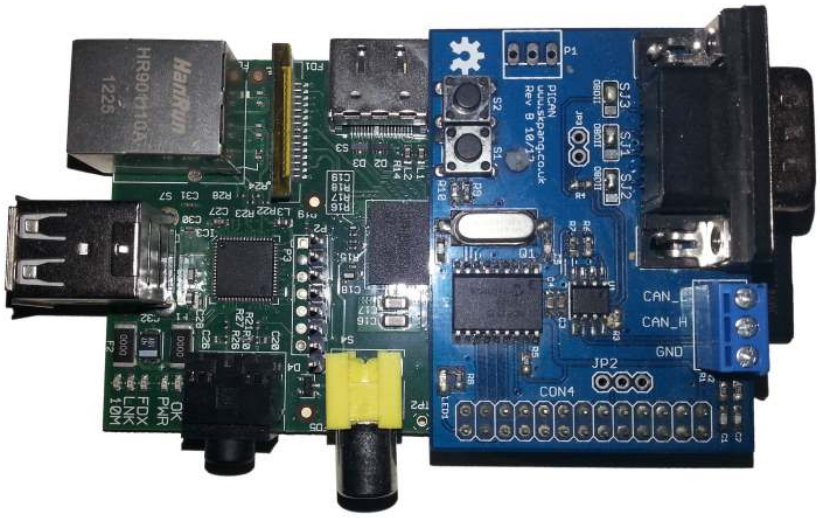

Abbildung 5.3: Raspberry Pi mit PICAN CAN-Bus Board

5.1.3 CAN-Bus Anbindung

Für die Verbindung zwischen dem CAN-Bus Board und dem CAN-Bus Ausgang des Rollers wurde ein entsprechendes Kabel konstruiert. Die Liste der dafür bei Reichelt[3] bestellten Teile findet sich in Tabelle 5.2.

Tabelle 5.2: Teileliste CAN-Bus Kabel

Artikelnr.	Bezeichnung	Anzahl
AK 231	D-SUB Kabel, 1:1, 9-pol., BU/BU, 1,8m	1
MNL 4AG	4-pol. Mate-N-Lok Aufnahmegehäuse, 1-reihig	1
MNL BK1	Mate-N-Lok Buchsenkontakte für 0,2-0,8mm^2	4

[3] http://www.reichelt.de/

Das Kabel wurde in der Mitte durchtrennt, um an dem offenen Ende die Buchsenkontakte zu befestigen. Die Belegung des 9-poligen Sub-D Steckers des CAN-Bus Boards wurde dem Schaltplan des Boards entnommen, der auf der Webseite des Herstellers verfügbar ist ([SK Pang Electronics Ltd., 2013]). Die einzelnen Adern des durchtrennten Kabels sind farbig ummantelt. Die Verschaltung der Adern kann Abbildung 5.4 entnommen werden. Entsprechend der Ein- und Ausgangsbelegung des CAN-Bus Boards bzw. des CAN-Bus Ausgangs des Rollers wurden die Buchsenkontakte gesteckt. Abbildung 5.5 zeigt das fertige Kabel. Darüber hinaus ist in der Abbildung ein USB-Kabel und ein Spannungswandler von 12V auf 5V für Zigarettenanzünder zu sehen, die ebenfalls bei Reichelt bestellt worden sind und für die Spannungsversorgung des Raspberry Pi genutzt werden.

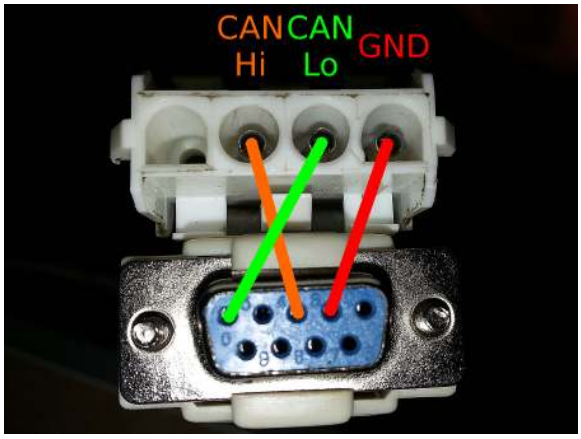

Abbildung 5.4: Anschlüsse CAN-Bus Kabel

Abbildung 5.5: CAN-Bus Kabel und Spannungsversorgung

Probleme mit dem ersten CAN-Bus Shield

Bevor das PICAN CAN-Bus Board der Firma SK Pang Electronics Ltd. in dem Prototypen zum Einsatz kam, wurde ein Versuchsaufbau mit dem CAN-Bus Shield und der Raspberry Pi to Arduino shields connection bridge der Firma Cooking Hacks[4] unternommen. Nach einer Lieferzeit von fast einem Monat stellte sich heraus, dass der Raspberry Pi aus bisher ungeklärter Ursache nicht mit dem bestellten CAN-Bus Shield kommunizieren kann. Recherchen in diversen Arduino- und Raspberry Pi-Foren haben ergeben, dass anscheinend niemand zuvor dieses Problem gehabt hat. Es muss allerdings an dieser Stelle bemerkt werden, dass die Raspberry Pi to Arduino shields connection bridge erst seit etwa einem Jahr auf dem Markt ist ([Cooking_Hacks, 2012]) und die Verbindung aus Raspberry Pi, connection bridge und ebd. CAN-Bus Shield verhältnismäßig selten sein dürfte. Aber auch intensive Lötmaßnahmen mit professioneller Unterstützung, um die Spannungsversorgung des MCP2515 CAN Controllers zu stabilisieren, ergaben nicht den erhofften Erfolg. Erst nach diesen gescheiterten Versuchen, dem Shield zur Kommunikation mit dem Raspberry Pi zu verhelfen, wurde das zuvor beschriebene PICAN CAN-Bus Board bestellt, welches sich nach Installation der Kernel-Module anstandslos betreiben lies.

5.2 Software

In diesem Kapitel werden die verwendeten Softwarekomponenten des Systems vorgestellt. Beginnend bei dem Betriebssystem, welches auf dem Prototypen zum Einsatz kommt, werden anschließend die Implementierung des Datenservers, des CAN-Bus Clients als Vertreter der Datenproduzenten und schließlich die Umsetzung eines Anzeigeinstrumentes respektive eines Tachometers als Datenkonsument erläutert.
Neben den verwendeten Frameworks, Laufzeitumgebungen und Bibliotheken finden auch Analysen Erwähnung, die zur Findung oder Konstruktion einer geeigneten Komponente beigetragen haben. Dazu zählen insbesondere ein Vergleich verschiedener Betriebssysteme, Reverse Engineering des CAN-Bus Signals des Govecs GO! S1.2 und Tests einiger Lösungen zur Darstellung der Anzeigeinstrumente. Bei der Auswahl der Komponenten und Frameworks wurde besonderer Wert auf Leichtgewichtigkeit und Performanz gelegt, um den Rechenaufwand und damit den Energieverbrauch des Systems so gering wie möglich zu halten.

[4] http://www.cooking-hacks.com/

5.2.1 Betriebssystem

Für die ARM-Architektur und insbesondere für den Raspberry Pi existieren verschiedene Linux-Distributionen, die auf diese Plattform portiert worden sind. Zu den bekanntesten gehören hier *Raspbian* (Debian-Derivat), *Pidora* (Fedora Remix), *Arch Linux ARM* und *RaspBMC* (Media Center auf Basis von XBMC) ([Raspberry Pi Foundation, 2013]). Letzteres wurde nicht in die engere Auswahl einbezogen, da RaspBMC eine Multimedia-Distribution ist, die auf das Management und das Abspielen von multimedialen Inhalten und dementsprechende Funktionalitäten hin optimiert wurde. Zwar wäre eine Umsetzung des Systems auf Basis von RaspBMC prinzipiell möglich, erforderte aber größere Eingriffe in das Betriebssystem, um vor allem nicht benötigte Komponenten zu entfernen. Die anderen Betriebssysteme wurden in Hinblick auf folgende Kriterien untersucht:

Startup Zeit	Zeitspanne vom Einschalten des Systems bis zum Login Prompt
Aktualität	Verfügbarkeit von Updates hinsichtlich Performanz, Stabilität und Funktionalität
Erweiterbarkeit	Verfügbarkeit von Bibliotheken, Laufzeitumgebungen, Treibern und weiteren Software Paketen
Stabilität	Verfügbarkeit von als stabil geltenden Software Paketen
Performanz	CPU-, Speicher- und Datenträgerauslastung während des Betriebs
Maintainability	Wartbarkeit und Administrationsfreundlichkeit des Systems

Startup Zeit

Die getesteten Betriebssysteme Raspbian und Pidora verwenden Init-Skripte, um während des Bootvorganges den Kernel und dessen Module zu laden und das System zu konfigurieren. Arch Linux hingegen setzt auf *systemd* als System- und Servicemanager. Durch Parallelisierung und nachrichtengesteuertes Starten von Diensten und Modulen werden mit systemd Startup Zeiten von 15 Sekunden und weniger erreicht. Raspbian und Pidora haben dagegen zwischen 50 und 80 Sekunden auf einen Login Promt warten lassen.

Aktualität

Arch Linux ist ein *Rolling Release* System. Das bedeutet, dass Änderungen an Software Paketen direkt in das System integriert werden. Dies hat zwar den Vorteil, dass das System stehts auf dem aktuellsten Stand ist, kann aber zu Inkompatibilitäten zwischen Bibliotheken führen, wenn nicht genügend Sorgfalt bei der Integration der neuen Pakete geboten ist. Raspbian und Pidora hingegen sind festen Release Zyklen unterworfen. Während eines Zyklus werden die in das System zu inte-

grierenden Pakete „eingefroren“ und die neue Version des Betriebssystem erst nach Behebung aller Fehler als stabil veröffentlicht.

Erweiterbarkeit

Aufgrund der dem Linux System immanenten Modularität lassen sich alle drei Betriebssysteme durch Kompilieren von nicht vorhandenen Komponenten erweitern, gesetzt dem Fall, dass die dafür benötigten Bibliotheken vorhanden sind. Allerdings lies sich unter Pidora die *SocketCAN* Bibliothek, eine Abstraktion des CAN-Bus als Netzwerkschnittstelle, nicht installieren. Diese Abstraktionsschicht ist zwar für den Betrieb mit einem CAN-Bus Adapter wie dem hier verwendeten PICAN CAN-Bus Board nicht zwingend erforderlich, erleichtert aber den Zugriff auf die Daten des Busses. Zu erwähnen ist hier noch das Arch User Repository (AUR), in dem von der Community erstellte Pakete verfügbar gemacht werden.

Stabilität

Für den Vergleich der Betriebssysteme wurden die jeweils aktuellen Releases verwendet. Während des Betriebs traten keine Abstürze, die durch das Betriebssystem bedingt wurden, auf, sodass alle drei als stabil angesehen werden können. Auch nach Recherchen auf Portalen wie DistroWatch[5], dem Raspberry Pi Forum[6] und diversen betriebssystemspezifischen Foren konnte keines der hier betrachteten Betriebssysteme eindeutig als das stabilste identifiziert werden.

Performanz

Die Betriebssysteme kommen mit unterschiedlicher Basisausstattung daher, was sich unter anderem in der Größe des Installationsimages niederschlägt. Das kleinste Image (178 MB ZIP-Datei) weist hierbei Arch Linux auf, was darauf zurückzuführen ist, dass Arch nur diejenigen Komponenten mitliefert, die für den Betrieb eines minimalen Systems notwendig sind. Dies schlägt sich vor allem beim Hochfahren des Systems, bei der Suche in der Paketdatenbank und dem benötigten Speicherplatz nach der Installation und der Größe der Auslagerungsdatei nach dem Start positiv nieder. Bezüglich CPU- und Speicherauslastung konnten keine gravierenden Unterschiede festgestellt werden.

Maintainability

In Bezug auf Wartbarkeit und Administration der Systeme konnten keine wesentlichen Unterschiede festgestellt werden. Die Wahl einer Distribution ist nicht zuletzt auch eine Frage des persönlichen Geschmacks. Ob Pakete mit *apt-get install* (Raspbian), *yum install* (Pidora) oder *pacman -S*

[5] http://distrowatch.com/index.php?language=DE
[6] http://www.raspberrypi.org/phpBB3/

(Arch Linux) installiert werden, spielt für den Betrieb des Systems dabei keine Rolle. Unter allen drei getesteten Distributionen ließen sich Pakete und Kernel Module installieren, Bibliotheken und Programme kompilieren und angeschlossene Hardware betreiben.

Wahl der Distribution

Die Wahl des Betriebssystems für den Prototypen fiel aus den eben genannten Gründen auf Arch Linux ARM. Vor allem der schnelle Bootprozess, die Aktualität, Erweiterbarkeit und das Handling des OS schaffen ideale Voraussetzungen für ein System mit den hier beschriebenen und konzipierten Funktionalitäten.

Für ein einfaches und schnelles Deployment des Systems wurden Bash Skripte geschrieben. Ein Installationsskript übernimmt das Kopieren des Betriebssystemimages und der für das CAN-Bus Shield benötigten Kernel Module und CAN Utilities auf eine SD-Karte. Ein Konfigurationsskript nimmt die anschließende Konfiguration des Betriebssystems und die Installation der für den Datenserver verwendeten Python Laufzeitumgebung und Module vor. In Listing 5.1 ist der Teil aus dem Installationsskript zu sehen, der das Image auf die SD-Karte kopiert und diese anschließend in das Dateisystem einbindet, damit weitere Dateien kopiert werden können. Listing 5.2 zeigt den Teil des Konfigurationsskriptes, der für das automatische Laden der CAN-Bus Module und die anschließende Konfiguration der SocketCAN-Schnittstelle verantwortlich zeichnet.

Listing 5.1: Auszug aus dem Installationsskript

```
ARCHLINUX='archlinux-hf-2013-06-15'
SDCARD=$(cat /proc/partitions | egrep mmcblk[0-9]$ | awk '{print $4}')

# Copy ArchLinux image to SD card
echo '===== Copying' $ARCHLINUX 'to /dev/'$SDCARD '... ====='
dd if=$ARCHLINUX.img bs=1M of=/dev/$SDCARD

# Mount SD card
echo '===== Mounting SD card ... ====='
MNT=/mnt/rpi
mkdir -p $MNT
SDPART=( $(cat /proc/partitions | grep $SDCARD'p' | awk '{print $3 " " $4}' |\
    sort -nr | head -n 2 | awk '{print $2}') )

mount '/dev/'${SDPART[0]} $MNT
mount '/dev/'${SDPART[1]} $MNT/boot
```

Listing 5.2: Auszug aus dem Konfigurationsskript

```
# Activate CAN socket interface on boot
echo "===== Activating CAN socket interface on boot ====="
CANSTART=/etc/systemd/system/multi-user.target.wants/canbus.service
echo -e "\
[Unit]\n\
Description=CANbus initialization\n\
\n\
[Service]\n\
Type=oneshot\n\
RemainAfterExit=yes\n\
\n\
ExecStart=-/usr/bin/insmod /usr/lib/modules/can/spi-bcm2708.ko\n\
ExecStart=-/usr/bin/insmod /usr/lib/modules/can/can.ko\n\
ExecStart=-/usr/bin/insmod /usr/lib/modules/can/can-dev.ko\n\
ExecStart=-/usr/bin/insmod /usr/lib/modules/can/can-raw.ko\n\
ExecStart=-/usr/bin/insmod /usr/lib/modules/can/can-bcm.ko\n\
ExecStart=-/usr/bin/insmod /usr/lib/modules/can/mcp251x.ko\n\
\n\
ExecStart=/usr/bin/ip link set can0 type can bitrate 500000\n\
ExecStart=/usr/bin/ip link set can0 up\n\
\n\
ExecStop=/usr/bin/ip link set can0 down\n\
\n\
[Install]\n\
WantedBy=multi-user.target\n\
" > $CANSTART
systemctl enable canbus
```

5.2.2 Datenserver

Der Prototyp des Datenservers wurde in der für viele Plattformen erhältlichen Programmiersprache Python in der Version 2.7 implementiert. Die Wahl fiel aus mehreren Gründen auf Python. Es existiert zum einen eine schier unerschöpfliche Anzahl an Modulen für diese Programmiersprache, insbesondere Frameworks zum Erstellen von Webservern und -applikationen. Zum anderen ist Python aufgrund seiner hohen Verbreitung und Kompatibilität und nicht zuletzt wegen der guten Lesbarkeit des Quellcodes und der leichten Erlernbarkeit für das System geeignet. Die Version 2.7 wurde deshalb gewählt, weil für Python 3.x (noch) keine Websocket Implementierung existiert.
Als Web Framework kommt *Bottle*[7], ein schnelles und leichtgewichtiges Web Server Gateway Interface (WSGI) Framework, zum Einsatz. Bottle bietet neben einem integrierten HTTP Development Server eine Template Engine zum dynamischen Erstellen von Webseiten, Zugriffsmethoden

[7] http://bottlepy.org/docs/dev/index.html

auf HTTP Metadaten wie Header, Cookies oder Formdaten und eine Routing Funktion, um Requests an Funktionsaufrufe zu binden.
Der eigentliche Webserver wird von dem Python Modul *gevent*[8] bereitgestellt. *gevent* ist eine leichtgewichtige auf Koroutinen basierende Netzwerkbibliothek, die auf der Bibliothek *libev*, einer Implementierung einer Event Loop, aufsetzt. Das Python Modul *gevent-websocket*[9] ist eine Websocket Implementierung für *gevent* und erweitert dieses um Websocket Funktionalitäten.
Der Datenserver besteht im Wesentlichen aus drei Komponenten: ModuleHandler, Router und Server. Der *ModuleHandler* ist für den Aufbau der in Abschnitt 4.2.1 beschriebenen Datenstrukturen zuständig, die durch XML-Dateien definiert werden. Die Datenstruktur wird durch die Klassen *Module* und *ModuleItem* entsprechend implementiert. Der Klasse *Module* wird dabei der Pfad zu der XML-Datei übergeben. Der *ModuleHandler* speichert die Daten und stellt Methoden zum Zugriff auf diese bereit.
Der *Router* übernimmt die Zuordnung von Werten in der Datenstruktur des *ModuleHandlers* zu URLs und bildet somit die REST-Schnittstelle des Systems. Er bietet Funktionen zum Auslesen und Setzen von Werten mittels GET- und POST-Requests. Bei Änderung eines Wertes ruft er eine Callback-Funktion des Servers auf, damit dieser wiederum die angemeldeten Clients über die Wertänderung informieren kann. Eine URL für einen Request an den Datenserver hat die Form *http://localhost:49999/<module>/<item>/<property>*. Hierbei bezeichnet *<module>* den Namen des Moduls, welches den Wert bereitstellt (z.B. „can“), *<item>* den Namen des Wertes (z.B. „geschwindigkeit“) und *<property>* den Namen des Attributs. *<property>* kann die Werte „name“ (Name des Wertes) , „type“ (Datentyp), „unit“ (physikalische Einheit des Wertes) und „value“ (der eigentliche Wert) annehmen. Für das *type* Attribut wurde in der Klasse *ModuleItem* eine Enumeration implementiert. Enumerations sind in Python nicht vorhanden, weswegen hier auf eine eigene Implementierung zurückgegriffen wurde. Die Implementierung der Enumeration ist in Listing 5.3 zu sehen. Darüber hinaus stellt er die Ressource */display* zur Verfügung, über die auf die Anzeigeinstrumente zugegriffen werden kann.

Listing 5.3: Implementierung einer Enumeration in der Klasse ModuleItem

```
@staticmethod
def enum(*sequential, **named):
    enums = dict(zip(sequential, range(len(sequential))), **named)
    reverse = dict((value, key) for key, value in enums.items())
    parallel = dict((key, key) for key, value in enums.items())
    enums['reverse_mapping'] = reverse
    enums['parallel_mapping'] = parallel
    return type('Enum', (), enums)

Types = enum.__func__('UINT', 'INT', 'UFLOAT', 'FLOAT', 'STRING', 'JSON')
```

[8] http://www.gevent.org/
[9] https://bitbucket.org/Jeffrey/gevent-websocket/

```
def __init__(self, name='speed', itemType='UINT', unit='mph', value=None):
    self._name = name
    self._type = VdsModuleItem.Types.parallel_mapping[\
                                        itemType.decode('utf-8').upper()]
    self._unit = unit
    self._value = value
```

Der *Server* nimmt sämtliche Anfragen entgegen und fungiert zunächst als Dispatcher. Er nimmt sich aller Anfragen, die an die Ressource */websocket* gehen, an und leitet alle übrigen Anfragen an den *Router* weiter. Die nächste Funktion, die von ihm übernommen wird, ist die Verarbeitung der Client-Nachrichten und das Erzeugen der Server-Nachrichten, die in den Tabellen 4.2 und 4.3 in Abschnitt 4.2.1 beschrieben wurden. Zusätzlich verwaltet er die Websockets der angemeldeten Datenkonsumenten und leitet Wertänderungen an sie weiter. Abbildung 5.6 zeigt das Klassendiagramm des gesamten Datenservers.

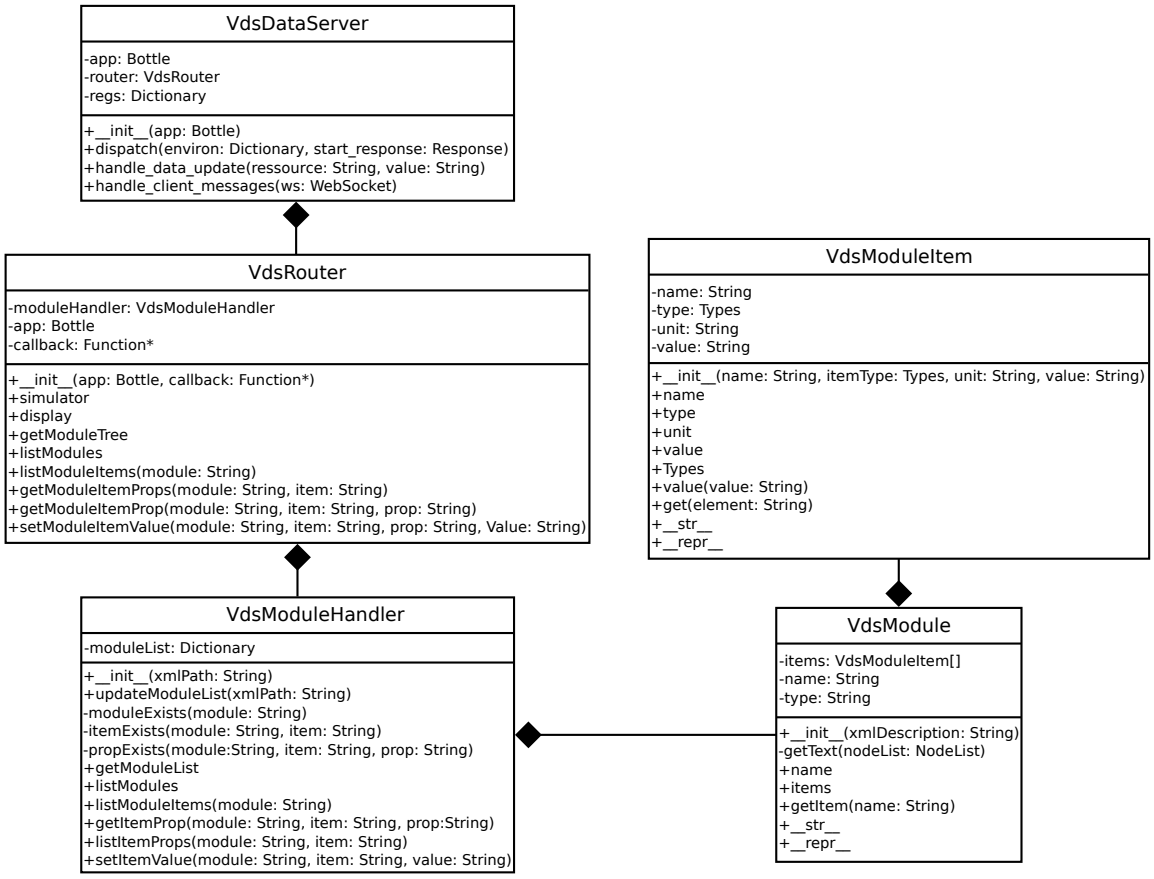

Abbildung 5.6: Klassendiagramm Datenserver

5.2.3 CAN-Bus Client

Für die Implementierung des CAN-Bus Clients musste zunächst analysiert werden, welche Informationen in welcher Nachricht über den CAN-Bus des Fahrzeugs übermittelt werden. Aus Gründen der Vertraulichkeit war von Govecs keine Spezifikation der CAN-Bus Nachrichten zu erhalten. Dies ist in soweit verständlich, als dass über den Bus unter anderem die Motorsteuerung angesprochen und dazu veranlasst werden kann, die elektronische Geschwindigkeitsbegrenzung außer Kraft zu setzen, was der Zulassung des Fahrzeugs widerspricht. In dem hier vorliegenden System wird aber lediglich lesend auf den CAN-Bus zugegriffen, um die anfallenden Daten für die Anzeige und das Monitoring zu nutzen. Darüber hinaus ist es nicht erforderlich, dass das System Nachrichten über den Bus sendet.

Analyse der CAN-Bus Nachrichten

Um dennoch an die benötigten Informationen zu gelangen, wurden CAN Frames aufgezeichnet und mit den Statusänderungen, die an dem Fahrzeug vorgenommen wurden, verglichen. Dabei kam das Programm *candump*, ein Tool aus den *can-utils*[10] zum Einsatz, welches die an der CAN-Bus Schnittstelle ankommenden Frames auf dem Terminal ausgibt. Ein Ausschnitt einer solchen Ausgabe ist in Listing 5.4 zu sehen. Jede Zeile der Ausgabe entspricht dem Format „*(Zeitstempel) CAN-Schnittstelle ID [Payload-Größe] Payload*“.

Listing 5.4: Auszug eines CAN-Dumps (1s)

```
(029.519720)  can0  200   [8] 00 00 00 00 00 01 00 00
(029.519888)  can0  280   [8] 47 01 00 00 68 32 00 00
(029.520107)  can0  290   [8] CB 00 00 00 4F 1F 00 00
(029.520404)  can0  300   [8] 2D 03 00 00 00 01 F6 FF
(029.619396)  can0  701   [1] 05
(029.719414)  can0  701   [1] 05
(029.819395)  can0  701   [1] 05
(029.919393)  can0  701   [1] 05
(030.019374)  can0   80   [0]
(030.019585)  can0  701   [1] 05
(030.019770)  can0  200   [8] 00 00 08 00 00 01 04 00
(030.119402)  can0  701   [1] 05
(030.219400)  can0  701   [1] 05
(030.319363)  can0  701   [1] 05
(030.419410)  can0  701   [1] 05
(030.519359)  can0   80   [0]
```

[10] https://gitorious.org/linux-can/can-utils/

Durch systematische Änderungen der Geschwindigkeit und den damit einhergehenden Änderungen der Kilometerstände und des Akkustandes konnten die in Tabelle 5.3 gelisteten Daten identifiziert werden. Dabei stellte sich heraus, dass die anliegenden Werte lediglich hexadezimal kodiert auf dem CAN-Bus ausgegeben werden. Um den Wert für die Entladung zu identifizieren, wurde er zusammen mit der Geschwindigkeit auf einer Zeitachse abgetragen. Der Wert wurde dabei normalisiert, um eine relative Angabe in Prozent machen zu können. Ob dieser Wert tatsächlich eine Form der Entladung angibt, konnte nicht verifiziert werden. Dafür spricht allerdings, dass der Wert sehr stark mit der Änderung der Geschwindigkeit korreliert, wie in Abbildung 5.7 zu sehen ist.

Die Nachricht, welche die Status- und Fehlermeldungen enthält, konnte zwar als solche identifiziert, die Status- und Fehlercodes an sich in Ermangelung an auftretenden Fehlern aber nicht dekodiert werden. Es lies sich während der Analysen nur ein einziger Fehler in der Einschaltsequenz des Rollers erzeugen, wodurch die Fehlernachricht erkannt werden konnte. IDs, die zwar aufgetreten sind, aber nicht identifiziert werden konnten, sind in Tabelle 5.4 gelistet.

Tabelle 5.3: Identifizierte CAN-Bus Pakete

ID	**Intervall**	**Byte-Nr.**	**Bedeutung**
81	-	1-8	Statusmeldungen und Fehlercodes
200	500 ms	3	Geschwindigkeit in km/h
		7	Geschwindigkeit in mph
280	1 s	1-4	Tageskilometer * 10 in km
		5-8	Gesamtkilometer * 10 in km
290	1 s	1-4	Tageskilometer * 10 in mi
		5-8	Gesamtkilometer * 10 in mi
300	1 s	1	Akkustand in %
		3	Entladung

Tabelle 5.4: Nicht identifizierte CAN-Bus Pakete

ID	**Bemerkungen**
80	kein Payload (Länge 0) etwa alle 500 ms
400	einmalig nach etwa 1,5 Sekunden Payload: 00 00 00 00 00 00 00 00
701	erstes Paket nach Start des Fahrzeugs, Payload: 00 erneutes Auftreten nach ca. 20 ms, Payload: 00 danach etwa alle 100 ms, Payload: 05

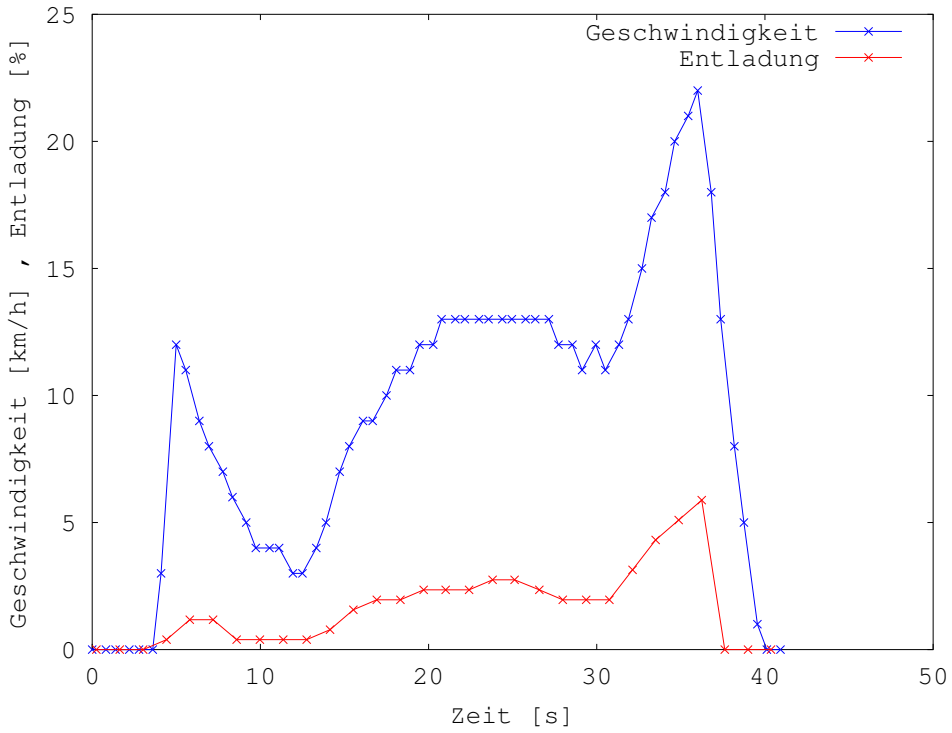

Abbildung 5.7: Geschwindigkeit vs Entladung

Implementierung

Die Implementierung des CAN-Bus Clients erfolgte ebenfalls in Python, allerdings in der Version 3.3, da erst ab dieser Version die SocketCAN Unterstützung vorhanden ist. SocketCAN ist eine Sammlung Open Source Netzwerktreibern für CAN-Controller und wird vom Linux Kernel ab Version 2.6.25 unterstützt ([BerliOS Developer, 2013]). Die Koexistenz zweier unterschiedlicher Python Versionen auf ein und demselben Betriebssystem verursachte dabei keine Probleme, da beide Versionen ihre Bibliotheken und Module in unterschiedlichen Verzeichnissen ablegen.

Die Implementierung basiert auf Beispielcode aus dem SocketCAN Patch v5 für Python 3.3[11]. Aus dem Beispielcode wurde die Funktion zum Senden von Nachrichten über den CAN-Bus entfernt und das Senden von POST-Requests an die entsprechenden Ressourcen des Datenservers hinzugefügt. Die XML-Datei, welche die Datenstruktur für den CAN-Bus Client definiert, wurde ebenfalls erzeugt und von dem Datenserver anstandslos verarbeitet.

[11] http://bugs.python.org/issue10141

5.2.4 Anzeigeinstrumente

Wie bereits in der Konzeption in Abschnitt 4.3.1 erwähnt, wurde die Darstellung der Anzeigeinstrumente mit HTML5 und JavaScript realisiert. Dafür musste auf dem Betriebssystem zunächst ein Display Manager, in diesem Fall der X.Org-Server, installiert werden, um grafische Anzeigen zu ermöglichen. Die Installation des X.Org-Servers verlief schnell und problemlos. Auf weitere Komponenten des X Window Systems wie Login- oder Window-Manager wurde bewusst verzichtet, um das System schlank zu halten. Der X.Org-Server alleine reicht bereits aus, um Programme mit grafischer Benutzerschnittstelle anzuzeigen.
Damit Webseiten, die HTML5 und JavaScript nutzen, dargestellt werden können, bedarf es eines entsprechenden Webbrowsers, der in der Lage ist, diese Sprachen zu interpretieren und die Skripte auszuführen. Eine Rendering-Engine, die genau das ermöglicht, ist WebKit[12]. Sie bildet unter anderem die Grundlage für Apples Safari, Google Chrome und Midori und findet auch in dem Smartphone Betriebssystem Android Verwendung. Die Wahl des Browsers fiel hier auf *Suckless Surf*[13], einem einfachen, leichtgewichtigen, quelloffenen Webbrowser auf WebKit Basis. Für die Darstellung der Anzeigeinstrumente und des Navigationssystems ist es notwendig, dass der Browser den Seiteninhalt bildschirmfüllend anzeigt. *Surf* lässt sich zwar in einen solchen Modus versetzen, aber nicht in diesem starten. Deshalb wurde der Quellcode des Browsers modifiziert und um eine Option zum Starten im Vollbildmodus erweitert. Zudem wurde die Datei */etc/X11/xinit/xinitrc*, die beim Starten des X.Org-Servers ausgeführt wird, dahingehend bearbeitet, dass weder Login- noch Window-Manager gestartet werden, aber dafür *Surf* im Vollbildmodus. Der Browser ruft beim Starten die Ressource */display* des Datenservers auf, unter der die Anzeigeinstrumente zur Verfügung stehen.
Für das Anzeigen des Bildschirminhaltes zeichnet ein HTML5 Canvas Element verantwortlich. Das Canvas Element verfügt über einen sogenannten 2D-Context, welcher Funktionen und Methoden zum Zeichnen von geometrischen Formen, Linien oder Bildern anbietet. Mittels JavaScript kann auf diese Methoden zugegriffen werden, um statische, dynamische oder interaktive Grafiken zu erstellen. Das Canvas Element wurde so konfiguriert, dass es stets das komplette Browserfenster ausfüllt.
Anzeigeinstrumente werden als JavaScript Widgets realisiert, die den 2D-Context des Canvas Elementes als Zeichenfläche nutzen. Für den Prototyp des Systems wurde ein parametrisiertes Tachometer Widget implementiert, welches in Abbildung 5.8 zu sehen ist. In Anlehnung an das Design des Cockpits des Govecs GO! S1.2 besteht es aus einem runden Bargraphen und einer zweistelligen 7-Segment Anzeige, um den Wert numerisch darzustellen. Der Bargraph und die 7-Segment Anzeige sind dabei jeweils als eigene Objekte implementiert, die auch für weitere Widgets genutzt werden können. Das Aussehen der Widgets kann über entsprechende Parameter gesteuert werden.

[12] https://www.webkit.org/
[13] http://surf.suckless.org/

Dazu verwenden die Widgets eigene *Options-Objekte*, in denen die Parameter gesetzt werden können. Beim Erzeugen des Widgets wird das Options-Objekt dem Konstruktor übergeben. Das Widget liest dann die Parameter für die Darstellung aus dem Options-Objekt aus. Listing 5.5 demonstriert, wie die Parameter des Tachometers gesetzt werden können.

Listing 5.5: Parameter des Tachometers

```
// configure speedo
var speedoOpts = new RoundBarWidgetOptions();
var drawingDiameter = Math.min(cnvsWidth, cnvsHeight);
speedoOpts.ctx = ctx;

speedoOpts.x = (cnvsWidth-drawingDiameter)/2;
speedoOpts.y = (cnvsHeight-drawingDiameter)/2;
speedoOpts.w = drawingDiameter;
speedoOpts.h = drawingDiameter;

speedoOpts.background.transparent = true;

speedoOpts.bar.colors = ["#000027"];
speedoOpts.bar.highlightColors = ["#4040ff"];
speedoOpts.bar.segments = [14];
speedoOpts.bar.width = drawingDiameter/13;
speedoOpts.bar.highlightWidthOffset = 0;
speedoOpts.bar.gap = 0.03;
speedoOpts.bar.radius = drawingDiameter/2.71;
speedoOpts.bar.start = 0.5 * Math.PI;
speedoOpts.bar.end = 1.75 * Math.PI;
speedoOpts.bar.counterclockwise = false;

speedoOpts.labels.font = "" + Math.round(drawingDiameter/14) + "px Arial";
speedoOpts.labels.color = "#ffff00";
speedoOpts.labels.offset = drawingDiameter/11.5;
speedoOpts.labels.title = "SPEED";
speedoOpts.labels.subtitle = "km/h";
speedoOpts.labels.minValue = 0;
speedoOpts.labels.maxValue = 70;
speedoOpts.labels.segments = 7;
speedoOpts.labels.outside = true;

// create speedo
var speedo = new RoundBarWidget(speedoOpts);
```

Abbildung 5.8: Tachometer Widget

Für die Kommunikation mit dem Datenserver kommen Websockets zum Einsatz. Das Websocket Protokoll wird von auf WebKit basierenden Browsern sowie Opera ab Version 10.70, Mozilla Firefox ab Version 4.0 und Internet Explorer ab Version 10.0 unterstützt. Die Initialisierung des Websockets sowie das Senden der Nachricht zur Registrierung für eine Ressource beim Datenserver kann Listing 5.6 entnommen werden.

Listing 5.6: Initialisierung eines Websockets

```
wsocket = new WebSocket("ws://" + window.location.host + "/websocket");
wsocket.onopen = function() {
        //console.log("WebSocket opened");
        wsocket.send("register:can/geschwindigkeit/value");
};
```

6 Zusammenfassung und Ausblick

In diesem Kapitel werden die wesentlichen Punkte dieser Arbeit zusammengefasst. Teile des Konzeptes, die nicht in die Realisierung mit eingeflossen sind, werden in Abschnitt 6.2 aufgeführt. Zum Abschluss der Arbeit wird ein Ausblick auf mögliche Erweiterungen und weitere Anwendungsfälle des Systems gegeben.

6.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde ein System entwickelt, um Elektrokrafträder in den Carsharing Betrieb zu integrieren, deren Status zu überwachen und mit einem Navigationssystem für energieoptimiertes Routing zu versehen.
Zu Beginn der Arbeit wurden Grundlagen erörtert, um im weiteren Verlauf der Arbeit einige Architektur- und Designentscheidungen besser nachvollziehen zu können. Dabei wurde neben Carsharing, Elektromobilität und dem Zusammenwirken dieser beiden auch Technologien betrachtet, die bei der Konzeption und Realisierung des Systems zum Einsatz kamen. Dazu gehörten das Controller Area Network (CAN) als weit verbreitetes Kommunikationssystem in Fahrzeugen und Representational State Transfer (REST) als Programmierparadigma für verteilte Anwendungen.
In der anschließenden Analyse wurden aktuell im Carsharing Kontext verwendete Systeme zur Fahrzeugbuchung sowie die Online-Portale verschiedener Carsharing Anbieter untersucht, um zu ermitteln, welche Daten an den Anbieter übermittelt werden müssen, damit dieser immer über den aktuellen Status des Fahrzeuges informiert ist. Betrachtet wurden dabei die Vorgänge von der Reservierung bis zur Rückgabe eines Fahrzeugs, insbesondere Verfahren zur Realisierung digitaler Fahrzeugschlüssel. Untersuchungen verschiedener Cockpits von Elektromotorrollern und Navigationssysteme für energieoptimiertes Routing gaben Aufschluss darüber, welche Daten für den Benutzer eines solchen Fahrzeuges relevant sind und während der Fahrt angezeigt werden sollten. Eine Benutzeranalyse mit Persona nach [Cooper et al., 2012] mit entsprechenden Szenarien nach [Alexander und Maiden, 2005] wurden dazu verwendet, um die Benutzung des Systems aus Sicht unterschiedlicher Stakeholder zu analysieren. Dadurch konnten weitere Systemfeatures, Komponenten und Architektureigenschaften identifiziert werden, welche in die anschließende Konzeption und Realisierung des Systems eingeflossen sind.
In der Konzeption des Systems wurden zunächst die aus den Analysen extrahierten Features zu-

sammengefasst. Die so konsolidierten Features fanden Einzug in das Konzept der Hard- und Softwarekomponenten des Systems. Es stellte sich heraus, dass eine Client-Server-Architektur sich sehr gut eignet, um die Anforderungen an das System zu erfüllen und die Features umzusetzen. Ein Datenserver zur Verwaltung der anfallenden Daten als Schnittstelle zwischen Datenproduzenten und Datenkonsumenten bildete die zentrale Komponente. Insbesondere wurden eine REST-Schnittstelle zwischen Server und Produzenten und eine Websocket-Schnittstelle zwischen Server und Konsumenten mit entsprechenden Datenstrukturen und Protokollen definiert, um den Datenfluss zwischen Produzenten und Konsumenten sicherzustellen.

Im Anschluss an die Kommunikationsinfrastruktur wurden Clients für verschiedene Aufgaben innerhalb des Systems entworfen. Für die Darstellung der Anzeigeinstrumente und des Navigationssystems zeichnet ein Display-Client verantwortlich. Die Freischaltung des Fahrzeugs mittels digitalen Fahrzeugschlüssel übernimmt ein Client, der aus der Arbeit von [Block, 2012] übernommen und in das Konzept des Systems integriert wurde. Damit der Carsharing Betreiber über die Position und den Status des Elektromotorrollers in Kenntnis gesetzt werden kann, wurde ein Monitoring-Client konzipiert, der die in der Analyse identifizierten Daten an einen beim Betreiber aufgestellten Proxy-Server überträgt. Der Proxy-Server bildet hierbei die Schnittstelle zwischen dem Fahrzeug und dem vom Betreiber verwendeten Flottenmanagementsystem bzw. dessen Datenbank. Die Betrachtung einiger Sicherheitsaspekte des Systems bildete das Ende der Konzeption der Softwarekomponenten.

Den Kern des hardwareseitigen Konzeptes bildeten Scheckkarten große Single-Board Computer, die sich aufgrund ihres geringen Preises und Energiebedarfs, dafür aber hohen Konnektivität und Erweiterbarkeit besonders gut als Basis für das hier vorliegende System eignen. Eine durchgehende Spannungsversorgung wurde mit Hilfe von Puffer-Akkus sichergestellt, um auch bei ausgeschaltetem Fahrzeug das Freischalten und die Datenübertragung zum Betreiber sicherzustellen. Im Anschluss wurden die Benutzungsschnittstellen des Systems definiert. Für die Informationsausgabe wurden Eigenschaften eines Displays festgelegt, welches in dem System die Darstellung der Anzeigeinstrumente und des Navigationssystems übernimmt. Für die Informationseingabe durch den Benutzer sieht das Konzept einen kleinen Joystick mit zwei zusätzlichen Knöpfen vor. Dabei wurde die Interaktion mit dem System so gestaltet, dass sie leicht verständlich und auch leicht erlernbar ist. Zum Abschluss der Konzeption wurde ein Design für die Anzeigeinstrumente und das Navigationssystem entworfen, welche die in der Analyse identifizierten Daten dem Benutzer während der Fahrt präsentieren. Dabei wurden Teile der analysierten Rollercockpits zur Orientierung herangezogen.

Für die Konstruktion des Prototypen kamen ein Govecs GO! S1.2 Elektromotorroller, ein Raspberry Pi und ein PICAN CAN-Bus Board der Firma SK Pang Electronics Ltd. zum Einsatz. Das CAN-Bus Board wurde über die 26-Pin Leiste mit dem Raspberry Pi und über ein selbst konstruiertes Kabel mit dem CAN-Bus des Rollers verbunden. Die Spannungsversorgung des Computers wurde

mit einem Spannungswandler von 12V auf 5V mit bis zu 2A Ausgangsleistung realisiert. Die Wahl des Betriebssystems für den Raspberry Pi fiel nach eingehenden Vergleichen mehrerer Betriebssysteme auf Arch Linux Arm. Vor allem der schnelle Bootprozess, die Aktualität, Erweiterbarkeit und das Handling des OS wussten zu überzeugen. Der in Python 2.7 implementierte Datenserver verfügt über eine REST-Schnittstelle, über die Daten ausgelesen und eingegeben werden können. Dies konnte mit Hilfe des leichtgewichtigen Web Frameworks *Bottle* realisiert werden. Die Struktur der Daten, die ein Datenproduzent an den Server schickt, wird durch XML-Dateien definiert, die der Server verarbeitet und anschließend die entsprechenden Ressourcen bereitstellt. Die Websocket-Schnittstelle wurde mit dem Python Modul *gevent-websocket* implementiert und erlaubt es dem Server, Clients über Wertänderungen einer Ressource zu informieren. Ein in Python 3.3 implementierter CAN-Bus Client greift die Daten vom CAN-Bus ab und leitet sie an den Datenserver weiter. Dafür mussten zuvor die Pakete, die über den CAN-Bus des Fahrzeugs gesendet werden, analysiert und dekodiert werden, um festzustellen, welche Informationen in welcher Form an dem Bus anliegen. Die Darstellung der Anzeigeinstrumente wurde mittels eines Browsers, eines HTML5 Canvas Elementes und JavaScript Widgets umgesetzt. Implementiert wurde ein Tachometer mit Bargraph und 7-Segment Anzeige in Anlehnung an das Tachometer des Govecs Rollers.

6.2 Offene Punkte

Für den Prototypen des Systems wurden hauptsächlich die grundlegenden Softwarekomponenten realisiert, hardwareseitig wurde lediglich die CAN-Bus Anbindung und eine rudimentäre Spannungsversorgung des Bordcomputers umgesetzt. Um den Prototypen zu vervollständigen, ist zusätzliche Peripherie vonnöten, und es müssen Modifikationen an dem Elektromotorroller vorgenommen werden. Die zusätzliche Peripherie umfasst einen Puffer-Akku sowie einen aktiven USB-Hub und ein passendes Gehäuse für den Raspberry Pi, ein RFID-Lesegerät, einen GPS-Empfänger und einen UMTS-Stick. Für die Interaktion mit dem Benutzer werden ein Bildschirm und ein Joystick benötigt.

Der Bildschirm und der Joystick sowie die Peripherie müssen in den Roller eingebaut werden. Dazu ist es erforderlich, die Abdeckung des Fahrzeugkonsole entweder zu modifizieren oder komplett durch eine Eigenanfertigung auszutauschen. Hierzu könnten bereits vorhandene Ressourcen der Universität zu Lübeck genutzt werden. Das Institut für Multimediale und Interaktive Systeme verfügt über einen 3D-Scanner und einen 3D-Drucker. Mit diesen Geräten kann ein Scan der vorhandenen Abdeckung vorgenommen werden. Das 3D-Modell wird anschließend entsprechend der Abmessungen des zu integrierenden Displays bearbeitet und mit dem 3D-Drucker ausgedruckt. Damit der Bildschirm vor Witterungseinflüssen geschützt ist, sollte er mit einer (Plexi-)Glasplatte abgedeckt werden. Der Raspberry Pi, der Puffer-Akku, der USB-Hub sowie der GPS-Empfänger und der UMTS-Stick können in der Frontsäule des Rollers oberhalb des Ladegerätes (in Abbildung

5.2 links im Bild) verstaut und befestigt werden. Das RFID-Lesegerät wird an der Abdeckung der Frontsäule angebracht, damit es für den Kunden leicht zugänglich ist.

Die Ansteuerung der Kontrollleuchten für Blinker, Licht, Akkustandswarnung und Ladeindikator ist nicht auf dem CAN-Bus ausgeführt. Um deren Funktion in die Anzeigen des Systems einzubinden, können die Leitungen für die Kontrollleuchten über einen Spannungswandler an die GPIO-Pins des Raspberry Pi angeschlossen werden. Auf diese Weise wird dem System signalisiert, wann welche Kontrollleuchte aktiv ist. Ein dann noch zu implementierender Client greift die Signale an den GPIO-Pins ab und leitet den Status der einzelnen Kontrollleuchten an den Datenserver weiter. Von dort aus kann ein Widget diese Daten verarbeiten und die Funktion der Kontrollleuchten übernehmen. Der Mode- und der Select-Knopf, die sich bereits an dem Fahrzeug befinden, müssen auf ähnliche Weise mit dem System verbunden und ihre Funktionalität entsprechend implementiert werden.

Die Umsetzung der Ver- und Entriegelung erfordert ebenfalls größere Eingriffe in die Struktur des Fahrzeugs. Zum einen muss das Ein- und Ausschalten mit Hilfe des digitalen Fahrzeugschlüssels umgesetzt werden. Zum anderen muss dafür Sorge getragen werden, dass sowohl das Lenkradschloss als auch die Verriegelungen der Staufächer durch den Schlüssel gesteuert werden können. Hierfür könnten Servomotoren eingesetzt werden, wie sie auch bei Nachrüstsätzen für Zentralverriegelungen in Automobilen zum Einsatz kommen. Die Ansteuerung der Servos kann wieder über die GPIO-Pins des Bordcomputers erfolgen.

Das verwendete Arch Linux ARM liegt in einer Version vom 15.06.2013 vor und wurde mit fertig kompilierten CAN-Bus Treibern und einem entsprechenden Kernel erweitert. Dadurch wurde die Funktionalität anderer Kernel Module wie beispielsweise der Firewall iptables in Mitleidenschaft gezogen. Für einen voll funktionsfähigen Prototypen des Systems muss also ein neuer Kernel kompiliert werden, der idealerweise auf aktuellem Quellcode basiert.

Bei der Umsetzung der grafischen Benutzungsschnittstelle ergaben sich Probleme mit der Performanz des Systems. Dabei waren weniger die implementierten Komponenten oder der Webbrowser verantwortlich für die Systemauslastung, sondern der X.Org-Server, der teilweise über 60% der CPU-Zeit in Anspruch nahm. Durch die hohe Last, die der X.Org-Server erzeugt, standen den anderen Komponenten nicht mehr genügend Ressourcen zur Verfügung, die Anzeige der Geschwindigkeit in Echtzeit durchzuführen. Wird allerdings auf die Anzeige durch den Raspberry Pi verzichtet und beispielsweise ein Browser auf einem Smartphone oder einem Desktop-PC verwendet, liegt die Auslastung der CPU des Raspberry Pi gerade einmal bei 10 bis 20%. In dem Embedded Linux Wiki wurde ein Eintrag gefunden, der einen Xorg-Treiber speziell für den Raspberry Pi beschreibt. Dieser Treiber nutzt unter anderem speziell an den Raspberry Pi angepasste DMA Techniken und dessen System-on-Chip Architektur besser aus als der Standard-Treiber, um vor allem die 2D-Beschleunigung zu verbessern ([eLinux.org, 2013]). Eine weitere Option wäre, den X.Org-Server

durch den Anzeige-Server Wayland[1] zu ersetzen und entsprechende Performance-Tests durchzuführen. Auch die Verwendung eines zweiten Single-Board Computers, der ausschließlich für die grafische Ausgabe zuständig wäre, ist denkbar und mit dem hier entwickelten System umzusetzen.

Die Implementierung weiterer Anzeigeinstrumente steht gleichermaßen noch aus. Bisher wurde ein Widget für die Geschwindigkeitsanzeige und ein 7-Segment Widget implementiert. Die Anzeigeinstrumente für Kilometerstände, Akkustandsanzeige, Ecometer, Reichweitenangabe und nicht zuletzt Datum und Uhrzeit müssen noch realisiert werden. In diesem Zuge können auch die nötigen Mechanismen umgesetzt werden, die dem Kunden eine Auswahl des Cockpit-Designs ermöglichen. Die Entwicklung eines GreenNav-Clients als plattformunabhängige Webanwendung, die in das hier vorliegende System integriert werden kann, wird demnächst am Institut für Softwaretechnik und Programmiersprachen in Form eines Studentenprojektes durchgeführt.

Der Monitoring-Client und der Proxy-Server, der dessen Daten auf Betreiberseite entgegen nimmt, müssen ebenso noch implementiert werden. Dazu werden der GPS-Empfänger und der UMTS-Stick benötigt, um die Daten für den Client bereitzustellen. Darüber hinaus muss für ein Testsystem das Flottenmanagementsystem bzw. die Datenbank beim Betreiber analysiert werden, um die Kommunikation zwischen Proxy und Datenbank bzw. FMS zu ermöglichen und die Daten entsprechend aufbereiten zu können.

Ferner sollte die Umsetzung der in Abschnitt 4.3.5 beschriebenen Sicherheitsaspekte erfolgen. Dazu zählt insbesondere das Aufsetzen eines VPNs mittels IPsec und das Einrichten der Firewall und des Port-Knockings. Um schreibenden Zugriff auf den CAN-Bus zu verhindern, kann die SocketCAN Implementierung oder direkt das Kernel Modul für den CAN-Bus Zugriff dahingehend modifiziert werden, dass kein schreibender Zugriff mehr möglich ist.

Als letzter offener Punkt soll hier die Smartphone-App, die in der Benutzeranalyse in Abschnitt 3.2 beschrieben wurde, erwähnt werden. Zwar ist eine solche App nicht Teil dieser Arbeit gewesen, wäre aber für einen vollständig funktionierenden Prototypen wünschenswert. Die App sollte die Eigenschaften der Applikationen von [Kampsen, 2013] und [Block, 2012] in sich vereinen und zusätzlich die Möglichkeit bieten, die Standorte der ausleihbaren Fahrzeuge beim Buchungsvorgang festzustellen.

[1] http://wayland.freedesktop.org/

6.3 Ausblick

Aufgrund seines modularen Aufbaus und der leicht zugänglichen Dateninfrastruktur bietet das System Erweiterungsmöglichkeiten in vielerlei Hinsicht. Es sind dabei verschiedene Szenarien denkbar, die mit dem System umgesetzt werden können. Auf einige dieser Möglichkeiten soll im Folgenden eingegangen werden.
Das Projekt „Zentralisiertes Lithium-Speicher-Monitoring“ (ZeLiM), welches an dem Institut für Softwaretechnik und Programmiersprachen in Zusammenarbeit mit der ecc Repenning GmbH durchgeführt wird, hat zum Ziel, ein dezentrales Netz aus Energiespeichern, vornehmlich Lithium-Eisenphosphat-Speicher, zentral zu überwachen und zu steuern. Die gesammelten Daten aus dem ZeLiM-Projekt sollen unter anderem dazu genutzt werden, den Zustand sowie das Auf- und Entladeverhalten der Energiespeicher zu überwachen, um Aussagen über deren Haltbarkeit treffen zu können und die Energiespeichertechnologie fortwährend zu verbessern. In diesem Kontext kann das hier vorliegende System dazu genutzt werden, Daten des Fahrzeugakkus in die ZeLiM-Datenbank einzupflegen. Die Daten können dann ebenfalls für derartige Analysen herangezogen werden, um auch mobile Energiespeichertechnologie bewerten und verbessern zu können.
In diesem Zusammenhang sind auch die Stichworte *Vehicle to Grid* (V2G) und *Smart Grid* zu nennen. V2G beschreibt eine Technologie, mit der Elektrofahrzeuge als Energiequellen genutzt werden können. Dabei besteht eine bidirektionale Verbindung zum bestehenden Stromnetz, um einerseits die Batterien zu laden und andererseits Energie wieder in das Netz abzugeben, wenn dies aus Überlastgründen erforderlich ist. Der Begriff Smart Grid bezeichnet die Überwachung, den Schutz und die automatisierte Optimierung der angeschlossenen Komponenten eines Stromnetzes ([Dollen, 2009]). Dies erstreckt sich von der Erzeugung über die Speicherung bis zur Verteilung der elektrischen Energie. Das System kann in diesem Kontext für die Steuerung der Energiezu- und abfuhr und zur Kommunikation mit weiteren Smart Grid Komponenten genutzt werden. Dazu muss allerdings zuvor die bidirektionale Verbindung zum Stromnetz realisiert werden.
Als letzter Punkt soll hier kurz eine Nutzung des Systems außerhalb des Carsharing Kontextes beschrieben werden. Es ist vorstellbar, das System auch als Privatanwender in dem eigenen Elektromotorroller einzusetzen. Die dafür notwendigen Umbauten an dem Fahrzeug können von einer Fachwerkstatt durchgeführt werden. Statt die vom Monitoring-Client gesammelten Daten an einen Carsharing Betreiber zu senden, wird hier das Smartphone des Benutzers zur Speicherung und Auswertung der Daten genutzt. Die Daten können auch nach der Fahrt vom Smartphone auf einen PC übertragen und dort ausgewertet und visuell aufbereitet werden. Die Verbindung zum Smartphone kann über Bluetooth hergestellt werden. Mit Hilfe der Daten kann eine Smartphone-App oder ein PC-Programm Fahrerprofile erstellen oder die gesammelten Akkudaten dem ZeLiM-Projekt in anonymisierter Form zur Verfügung stellen. In die Smartphone-App würde auch die Funktion des digitalen Fahrzeugschlüssels integriert.

A Verzeichnisse

A.1 Abbildungen

A.2 Tabellen

A.3 Quellcode

A.4 Abkürzungen

API	Application Programming Interface
ARM	Advanced RISC Machines
BAföG	Bundesausbildungsförderungsgesetz
Bash	Bourne-again shell
BMVBS	Bundesministerium für Verkehr, Bau und Stadtentwicklung
CAN	Controller Area Network
CCHH	Carsharing Club Hamburg
CRUD	Create, Read, Update, Delete
DVD	Digital Versatile Disc
FAS	Fahrerassistenzsystem
FMS	Flottenmanagement-System
GPIO	General-purpose input/output
GPS	Global Positioning System
HDMI	High Definition Multimedia Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IPsec	Internet Protocol Security
ISO	International Organization for Standardization
IVIS	In-Vehicle Informationssystem
JSON	JavaScript Object Notation
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LLC	Logical Link Control
LLCF	Low Level CAN Framework
LVDS	Low Voltage Differential Signal
MAC	Medium Access Control
MIME	Multipurpose Internet Mail Extensions
NFC	Near Field Communication
OS	Operating System
OSI	Open Systems Interconnection
Patricia	Practical Algorithm to Retrieve Information Coded in Alphanumeric
PIN	Persönliche Identifikationsnummer
POI	Point of Interest
REST	Representational State Transfer
RFID	Radio Frequency Identification

RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
RPC	Remote Procedure Call
SDRAM	Synchronous Dynamic Random Access Memory
SoC	State of Charge
SoH	State of Health
SPI	Serial Peripheral Interface
SSH	Secure Shell
TFT	Thin-Film Transistor
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifyer
URL	Uniform Resource Locator
VPN	Virtual Private Network
WLAN	Wireless Local Area Network
WSGI	Web Server Gateway Interface
XML	Extensible Markup Language

A.5 Literatur

[ADAC, 2011] ADAC (2011). Motorrad Fahrberichte - Vectrix VX-1 Li+. http://www.adac.de/infotestrat/motorrad-roller/fahrberichte/berichte/vextrix-vx-1.aspx. online; letzter Zugriff: 09.11.13.

[Alexander und Maiden, 2005] Alexander, I. und Maiden, N. (2005). *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*. John Wiley & Sons.

[ALL4IP TECHNOLOGIES, 2013] ALL4IP TECHNOLOGIES (2013). mapZero. http://www.mapzero.de/. online; letzter Zugriff: 12.11.13.

[Autolib, 2013] Autolib (2013). Autolib' Paris Website Autolib' Paris Website | Autolib'. https://www.autolib.eu/en/. online; letzter Zugriff: 05.11.13.

[BerliOS Developer, 2013] BerliOS Developer (2013). BerliOS Developer: Project Summary - Socket-CAN. http://developer.berlios.de/projects/socketcan/. online; letzter Zugriff: 27.11.2013.

[Block, 2012] Block, H. (2012). Kontaktlose Verteilung virtueller Fahrzeugschlüssel. Diplomarbeit, Universität zu Lübeck.

[BMVBS, 2013a] BMVBS (2013a). BMVBS - Elektromobilität. http://www.bmvbs.de/SharedDocs/DE/Artikel/IR/EHP/effizienzhaus-plus-elektromobilitaet.html. online; letzter Zugriff: 28.11.13.

[BMVBS, 2013b] BMVBS (2013b). BMVBS - Führerschein-Übersicht über die Fahrerlaubnisklassen. http://www.bmvbs.de/SharedDocs/DE/Artikel/LA/fahrerlaubnisklassen-uebersicht.html. online; letzter Zugriff: 09.11.13.

[Bundesverband CarSharing e.V., 2013] Bundesverband CarSharing e.V. (2013). bcs Bundesverband CarSharing e.V. http://www.carsharing.de/alles-ueber-carsharing/faq. online; letzter Zugriff: 28.11.2013.

[Cahour et al., 2012] Cahour, B., Nguyen, C., Forzy, J.-F., und Licoppe, C. (2012). Using an electric car: a situated, instrumented and emotional activity. In *Proceedings of the 30th European Conference on Cognitive Ergonomics*, ECCE '12, pp. 22–28, New York, NY, USA. ACM.

[Car2go, 2013] Car2go (2013). Hallo Berlin. https://www.car2go.com/de/berlin/. online; letzter Zugriff: 05.11.13.

[Carroll, 1995] Carroll, J. (1995). *Scenario-Based Design: Envisioning Work and Technology in System Development*. Wiley.

[CityCarClub, 2013] CityCarClub (2013). Car Hire Club | UK Leading Car Sharing Club | City Car Club. `http://www.citycarclub.co.uk/`. online; letzter Zugriff: 05.11.13.

[Cooking_Hacks, 2012] Cooking_Hacks (2012). Raspberry Pi to Arduino shields connection bridge. `http://www.raspberrypi.org/phpBB3/viewtopic.php?f=59&t=19724`. online; letzter Zugriff: 25.11.2013.

[Cooper et al., 2012] Cooper, A., Reimann, R., und Cronin, D. (2012). *About Face 3: The Essentials of Interaction Design*. Wiley.

[Dollen, 2009] Dollen, D. V. (2009). Report to NIST on the Smart Grid Interoperability Standards Roadmap. `http://www.nist.gov/smartgrid/upload/InterimSmartGridRoadmapNISTRestructure.pdf`. online; letzter Zugriff: 29.11.2013.

[eLinux.org, 2013] eLinux.org (2013). RPi Xorg rpi Driver. `http://elinux.org/RPi_Xorg_rpi_Driver`. online; letzter Zugriff: 29.11.2013.

[Fielding, 2000] Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California. `http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf`; online; letzter Zugriff: 28.11.2013.

[Goldstein, 2002] Goldstein, E. B. (2002). *Wahrnehmungspsychologie (2. deutsche Auflage)*. Berlin: Spektrum Akademischer Verlag.

[Golla, 2012] Golla, M. (2012). *Govecs GO! S1.2 Konfiguration*. Govecs. Konfiguration_S12_D_6_12_P.pdf.

[Govecs, 2011] Govecs (2011). *Govecs GO! S1.2 User's manual and service book*. Govecs. `http://www.govecs.com/sites/default/files/mediapool/dateien/downloads/gos_manual_07_11_0.pdf`; online; letzter Zugriff: 09.11.13.

[Govecs, 2013] Govecs (2013). Home - GOVECS - TRUE ELECTRIC MOBILITY. `http://www.govecs.de/`. online; letzter Zugriff: 08.11.13.

[Greenwheels, 2013] Greenwheels (2013). Greenwheels Home. `https://www.greenwheels.com/de/Home/Privatkunden/Home`. online; letzter Zugriff: 05.11.13.

[Hartkopp und Thürmann, 2006] Hartkopp, O. und Thürmann, D. U. (2006). Low Level CAN Framework - Application Programmers Interface. `http://elk.informatik.fh-augsburg.de/pub/gnublin-lpc3131/work_eplpc3131/Dokumente/llcf-api.pdf`. online; letzter Zugriff: 28.11.2013.

[heise Open, 2013] heise Open (2013). Zwei Milionen Raspberry Pi ausgeliefert. http://www.heise.de/newsticker/meldung/Zwei-Milionen-Raspberry-Pi-ausgeliefert-2049904.html. online; letzter Zugriff: 24.11.13.

[Hybrid-Autos, 2013] Hybrid-Autos (2013). smart escooter 2010 | Smart | Elektro-Zweiräder. http://www.hybrid-autos.info/Elektro-Zweiraeder/Smart/smart-escooter-2010.html. online; letzter Zugriff: 09.11.13.

[ISO, 2013] ISO (2013). ISO - International Organization for Standardization. http://www.iso.org/iso/home.htm. online; letzter Zugriff: 28.11.2013.

[Kampsen, 2013] Kampsen, A. (2013). Entwicklung einer iPhone Applikation für den Car-Sharing Betrieb. Bachelorarbeit, Universität zu Lübeck.

[Kliesch, 2013] Kliesch, P. (2013). Entwicklung und Implementierung von energieoptimalem Flottenrouting im Kontext des Green Navigation Projekts. Masterarbeit, Universität zu Lübeck.

[Meschtscherjakov et al., 2009] Meschtscherjakov, A., Wilfinger, D., Scherndl, T., und Tscheligi, M. (2009). Acceptance of future persuasive in-car interfaces towards a more economic driving behaviour. In *Proceedings of the 1st International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '09, pp. 81–88, New York, NY, USA. ACM.

[Metropolregion, 2013] Metropolregion (2013). Elektromobilität kann durch Carsharing wesentliche Impulse erhalten - Metropolregion. http://www.metropolregion.de/pages/themen/schaufenster_emobilitaet/news/subpages/elektromobilitaet_kann_durch_carsharing_wesentliche_impulse_erhalten/. online; letzter Zugriff: 28.11.2013.

[MoveAbout, 2013] MoveAbout (2013). Home / www.move-about.de - Move About. http://www.move-about.de/. online; letzter Zugriff: 05.11.13.

[Mühlenhoff, 2010] Mühlenhoff, J. (2010). Erneuerbare Elektromobilität. *Renews Spezial*, (30). http://www.unendlich-viel-energie.de/uploads/media/30_Renews_Spezial%_Erneuerbare_Elektromobilitaet_april10_online.pdf; online; letzter Zugriff: 30.11.2013.

[Müller, 2004] Müller, M. (2004). Evaluationswerkzeuge für Bedienkonzepte von Fahrzeug-Cockpits. Doktorarbeit, Universität der Bundeswehr München. http://d-nb.info/971973334/34; online; letzter Zugriff: 08.11.13.

[Naumann, 2012] Naumann, M. (2012). BMW C evolution: Premiere für seriennahen City-Elektroroller. http://www.auto-news.de/auto/news/anzeige_BMW-C-evolution-Premiere-fuer-seriennahen-City-Elektroroller_id_32708. online; letzter Zugriff: 09.11.13.

[Raspberry Pi Foundation, 2013] Raspberry Pi Foundation (2013). Downloads | Raspberry Pi. http://www.raspberrypi.org/downloads. online; letzter Zugriff: 26.11.2013.

[Robert Bosch GmbH, 1991] Robert Bosch GmbH (1991). CAN Specification Version 2.0. http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can2spec.pdf. online; letzter Zugriff: 28.11.2013.

[Robert Bosch GmbH, 2012] Robert Bosch GmbH (2012). CAN FD - CAN with Flexible Data-Rate Specification Version 1.0. http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can_fd_spec.pdf. online; letzter Zugriff: 28.11.2013.

[Rodriguez, 2008] Rodriguez, A. (2008). RESTfuk Web Services: The basics. http://www.ibm.com/developerworks/webservices/library/ws-restful/ws-restful-pdf.pdf. online; letzter Zugriff: 28.11.2013.

[Shaw, 2013] Shaw, G. (2013). microHOWTO: Implement port knocking using iptables. http://www.microhowto.info/howto/implement_port_knocking_using_iptables.html. online; letzter Zugriff: 24.11.13.

[SK Pang Electronics Ltd., 2013] SK Pang Electronics Ltd. (2013). PICAN CAN-Bus Board for Raspberry Pi [RSP-PICAN] - £25.90 : SK Pang Electronics, Arduino, Sparkfun, GPS, GSM. http://www.skpang.co.uk/catalog/pican-canbus-board-for-raspberry-pi-p-1196.html. online; letzter Zugriff: 25.11.2013.

[smart, 2013] smart (2013). smart escooter. http://www.smartcenter.de/smart-escooter/0_580.html. online; letzter Zugriff: 09.11.13.

[stadtmobil, 2013] stadtmobil (2013). Privatkunden : stadtmobil.de. http://www.stadtmobil.de/. online; letzter Zugriff: 05.11.13.

[StattAuto, 2013] StattAuto (2013). StattAuto eG - CarSharing in Lübeck und Kiel. http://www.stattauto-hl.de/. online; letzter Zugriff: 05.11.13.

[Strömberg et al., 2011] Strömberg, H., Andersson, P., Almgren, S., Ericsson, J., Karlsson, M., und Nåbo, A. (2011). Driver interfaces for electric vehicles. In *Proceedings of the 3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '11, pp. 177–184, New York, NY, USA. ACM.

[Zipcar, 2013] Zipcar (2013). Car Sharing, an alternative to car rental and car ownership - Zipcar. http://www.zipcar.com/. online; letzter Zugriff: 05.11.13.

B Anhang

B.1 Beiliegende DVD-ROM

Erklärung

Ich versichere, die vorliegende Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

_____

Lübeck, 30. November 2013