Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Tim-Ole Schaffeld

Organisation der Kommunikationsdaten von Testfahrten eines Prototypfahrzeuges mit Ethernet Backbone

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Tim-Ole Schaffeld

Organisation der Kommunikationsdaten von Testfahrten eines Prototypfahrzeuges mit Ethernet Backbone

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Korf
Zweitgutachter: Prof. Dr. Kossakowski

Eingereicht am: 14. Juli 2016

Tim-Ole Schaffeld

Thema der Arbeit
Organisation der Kommunikationsdaten von Testfahrten eines Prototypfahrzeuges mit Ethernet Backbone

Stichworte
Metadaten, PostgreSQL Datenbank, PHP, Echtzeit-Ethernet, Datenorganisation

Kurzzusammenfassung
Inhalt dieser Bachelorarbeit ist die Entwicklung und Umsetzung eines Konzepts zur Organisation von aufgezeichneten Kommunikationsdaten eines Echtzeit-Ethernetnetzwerks in einem Prototypenfahrzeug. Nach der Problemanalyse werden drei mögliche Ansätze zur Umsetzung entwickelt. Das Ergebnis ist eine Anwendung, die während der Testfahrten des Prototypenfahrzeugs eingesetzt werden kann, um die aufgezeichneten Daten zu katalogisieren. Zusätzlich wird die Möglichkeit geschaffen, sogenannte Metadaten festzuhalten, die die Umstände der Fahrt näher beschreiben.
Tim-Ole Schaffeld

Title of the paper
Data management of communication data from testdrives with a prototype vehicle with an ethernet backbone archticture

Keywords
Metadata, PostgreSQL, PHP, Real-time Ethernet, data management

Abstract
This bachelor thesis deals with the design and implementation of a concept to organize logged communications data of a Real-time Ethernet network in a prototype vehicle. After analyzing the problem, three design options will be developed. The outcome is a software, which can be used during testdrives to cataloguize the logged data. Additionally you can add metadata to describe the circumstances of the testdrive.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Der Einzug von Ethernet als Technologie für die Kommunikation im Automobil ist die logische Konsequenz der gewachsenen Anforderungen an Bussysteme in modernen Fahrzeugen. Die Vielzahl an Steuergeräten, Sensoren und Kameras erfordert eine skalierbare, kostengünstige und zuverlässige Technik. Die Fähigkeit, Daten mit Echtzeitanforderungen zu verarbeiten, ließ die klassischen Ethernetprotokolle für diesen Einsatzzweck lange nicht in Frage kommen. Erst das Erweitern der Ethernettechnologie um diesen Aspekt lässt es zu, diese weit verbreitete und günstige Technik im Fahrzeug einzusetzen.

Die CoRE (Communication over Realtime Ethernet) Research Group an der HAW Hamburg unter Leitung von Prof. Dr. Franz Korf und Prof. Dr. Thomas Schmidt trägt mit ihrer Forschung einen wesentlichen Teil dazu bei. Im Rahmen des RecBar (Realtime Ethernet Backbone for Cars) Projekts ist ein Prototypenfahrzeug entwickelt worden, in dem ein Echtzeit-Ethernetnetzwerk zusätzlich zur den herkömmlichen Systemen verbaut worden ist. Mit diesem Fahrzeug soll in Zukunft der Einsatz von Ethernet mit Echtzeitanforderungen unter realen Bedingungen im Straßenverkehr getestet und weiter erforscht werden. Ein besonderer Schwerpunkt liegt dabei auf der Auswertung der Daten, die während den Testfahrten anfallen. Um das in Simulationen errechnete Verhalten und den Datenfluss im Netzwerk überprüfen zu können, ist es notwendig, diese Daten auszuwerten. Ziel dieser Arbeit ist es, ein Werkzeug zu entwickeln, mit dem es in Zukunft möglich ist, die aufgezeichneten Kommunikationsdaten zu katalogisieren und zu exportieren. Hinsichtlich der Organisation der enormen Datenmengen ist es erforderlich, zusätzliche Informationen zu den Fahrten, so genannte Metadaten, zu erfassen.

Gliederung der Arbeit

Nach dieser Einleitung werden in Kapitel 2 die theoretischen Grundlagen sowie Begriffsdefinitionen, die zum Verständnis dieser Arbeit beitragen, erläutert. Dazu gehören unter anderem eine Übersicht über das KFZ-Netzwerk des Prototypenfahrzeugs und die Definition von Kommunikationsdaten im Kontext dieser Arbeit. Eine ausführliche Problemanalyse der aktuellen Situation hinsichtlich der Datenaufzeichnung folgt in Kapitel 3. Anschließend wird das Konzept zur Umsetzung entwickelt. Dadurch entsteht mit den Methoden des Software Engineerings ein

Pflichtenheft, für desssen Umsetzung drei verschiedene Designoptionen vorgestellt werden. Im fünften Kapitel wird dann die Implementation einer der Optionen beschrieben. Es folgt die Software Dokumentation, die die Bedienung des Programms erläutert. Anschließend wird im Abschnitt Qualitätssicherung auf die Testmethoden und durchgeführten Softwaretests eingegangen. Im abschließenden Fazit und Ausblick werden die wesentlichen Erkenntnisse zusammengefasst sowie weitere Optimierungsmöglichkeiten vorgestellt.

2 Grundlagen

Um einen einfachen Einstieg in das Thema zu gewähren, sollen in diesem Kapitel die Grundlagen, die zum Verständnis der Arbeit beitragen, erläutert werden. Dazu zählt eine Übersicht über das Netzwerk im Prototypenfahrzeug sowie die Definition des Begriffs Kommunikationsdaten für diese Arbeit. Es folgt eine Aufzählung wesentlicher Arbeitsmittel sowie ein kurzer Überblick über die Charakteristika von Echtzeitnetzwerken.

2.1 Übersicht KFZ-Netzwerk

Die Arbeit findet im Rahmen des RECBar Kooperationsprojekts zwischen der HAW Hamburg, IAV automotive engineering, OFFIS - Institut für Informatik (Oldenburg) und C-LAB (Paderborn) statt (CoRE Research Group) . Ziel dieses Forschungsprojekts ist es, den Einsatz von Ethernettechnologie als Kommunikationstechnologie im sicherheitskritischen Umfeld des Automobils zu entwickeln und zu testen. Für die Entwicklung und Erprobung steht ein Prototypenfahrzeug vom Typ VW Golf 7 zur Verfügung. Dieses Fahrzeug ist um eine Ethernetinfrastruktur erweitert worden, welche zusätzlich zur klassischen BUS-basierten (CAN, Flex-Ray usw.) verbaut worden ist. Somit ist ein Mischbetrieb möglich und das Fahrzeug darf weiterhin im öffentlichen Straßenverkehr bewegt werden. Das ist wichtig, da bei ausschließlicher Verwendung des neuen Netzwerkes die Straßenzulassung erlöschen würde und man für Testfahrten auf Verkehrsübungsplätze angewiesen wäre. Ziel ist es, langfristig den gesamten Datenverkehr über das Real-Time Ethernet zu transportieren.

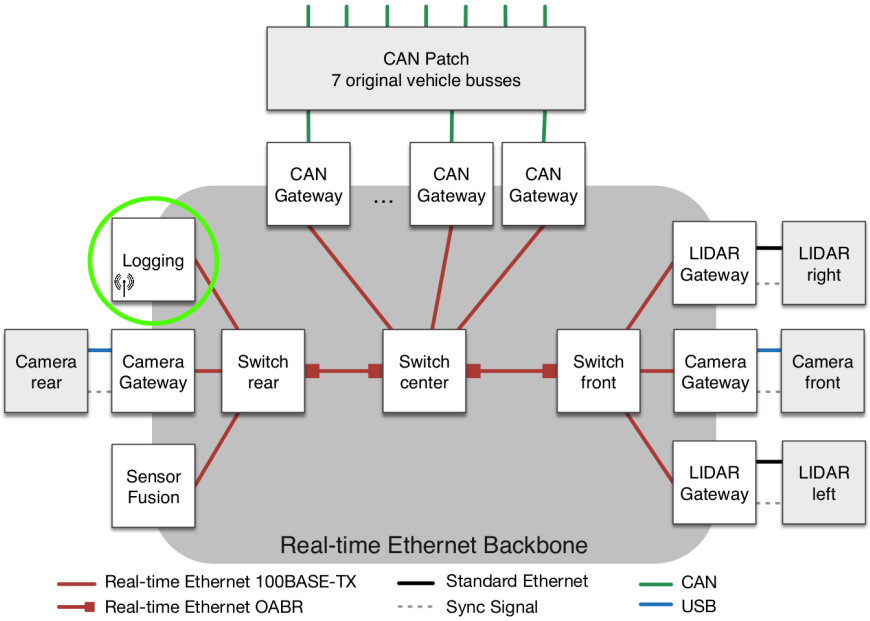

Abbildung 2.1: KFZ-Netzwerk

Innerhalb des aufgebauten Netzwerks nimmt der sogenannte Backbone eine zentrale Rolle ein. Es handelt sich hierbei um die Verbindung der drei Switche, über die sämtlicher Datenverkehr im Fahrzeug läuft (Abbildung 2.1). Um diese durch das gesamte Fahrzeug verlaufende Kabelverbindung kostengünstig zu gestalten und die Machbarkeit zu zeigen, wurde ein einfaches Single-Twisted-Pair-Kabel genutzt, wie es im Broadr-Reach-Standard[1] definiert wird (vgl. Steinbach (2014)). Durch die Verwendung dieses Kabeltyps wird deutlich, welches Potenzial in der Entwicklung von Ethernet als Technologie im Automobil steckt. Sie bietet durch ihre gute Skalierbarkeit und die Reduzierung der benötigten Kabel einen hohes Maß an Kostenersparnis. In Verbindung mit der großen Bandbreite entstehen so viele Vorteile gegenüber den bisherigen Systemen. Im Mittelpunkt dieser Arbeit wird der so genannte Logging-PC stehen (Abbildung 2.1: grüne Einkreisung). Dieser Rechner ist im Kofferraum verbaut und dort direkt an den “Switch Rear“ angeschlossen.

[1] http://www.opensig.org/

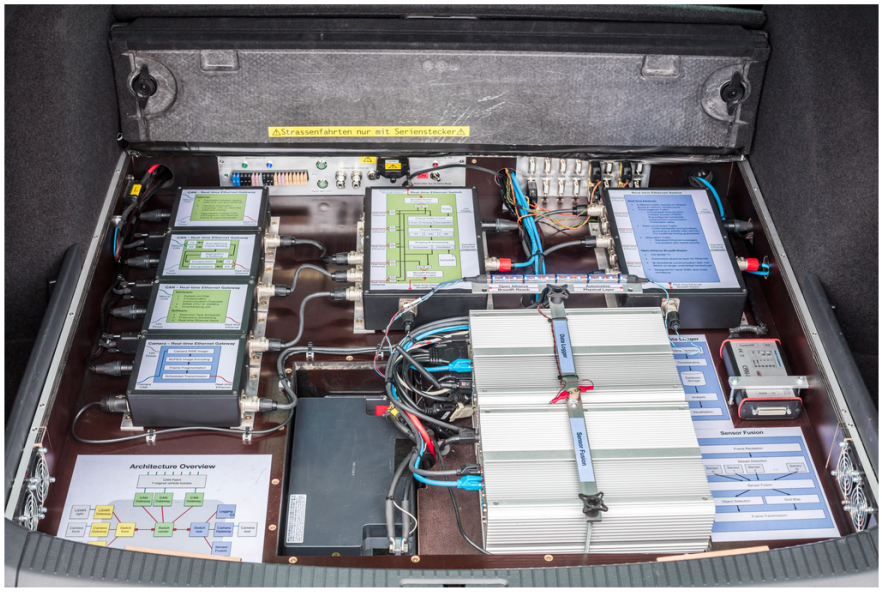

Abbildung 2.2: Blick unter die Kofferraumabdeckung

Zusätzlich hat er eine WLAN Schnittstelle und dient als Access Point eines Ad-Hoc Netzwerks.

Ebenfalls im Kofferraum untergebracht ist der “Switch center“ und die damit verbundenen CAN-Gateways. Der Switch front und daran angeschlossene Geräte befinden sich im vorderen Teil des Fahrzeugs, unter anderem im Handschuhfach.

Abbildung 2.3: Blick ins Handschuhfach

2.2 Definition Kommunikationsdaten

Der Begriff Kommunikationsdaten im Kontext der Ethernettechnologie beschreibt alle Informationen, die rund um den eigentlichen Informationsaustausch anfallen. Dem gegenüber stehen die Nutzdaten, die sich aus den tatsächlich zu übertragenden Informationen bilden.

Die Kommunikation in modernen Netzwerken geschieht paketbasiert. Das heißt, es gibt eine definierte Menge an Informationen, die pro Einheit übertragen wird. In ethernetbasierten Netzwerken ist ein sogenanntes Ethernetpaket eine solche Einheit. Der Aufbau eines Ethernetpakets ist in IEEE 802.3[2] wie folgt definiert:

Bytes	7	1	6	6	2	46-1500	4
Inhalt	Preamble	SFD	MAC DA/ *CT-Marker*	MAC SA	Lenght/ *TTEthernet -Type*	Data	Check sum
Bezeichnung			Ethernetframe				
	Ethernetpaket						

Abbildung 2.4: Ethernetpaket

Preamble Bytefolge zur Erkennung des Paketanfangs

SFD Starting Frame Delimiter

MAC-DA Zieladresse

MAC-SA Quelladresse

Lenght/Type Länge des Data-Felds beziehungsweise TTEthernettype

Data Nutzdaten

Checksum Prüfsumme für CRC-Check

In dem Prototypenfahrzeug wird TTEthernet[3] verwendet. Dies ist eine Erweiterung des Ethernetstandards, die den Standard für zeitkritische Übertragungen einsetzbar macht (siehe Abschnitt Time-Triggered Ethernet 2.3) .

[2] http://standards.ieee.org/about/get/802/802.3.html
[3] https://www.tttech.com/technologies/deterministic-ethernet/time-triggered-ethernet/

Im *Data* Feld werden die Nutzdaten transportiert. Dabei handelt es sich zum Beispiel um Daten, die zu einem IP-Paket gehören. Der Aufbau eines solchen Paketes ist wiederum eindeutig definiert. Durch weitere Verschachtelungen könnte ein tatsächlich übertragenes Ethernetpaket und desssen Inhalt so aufgebaut sein:

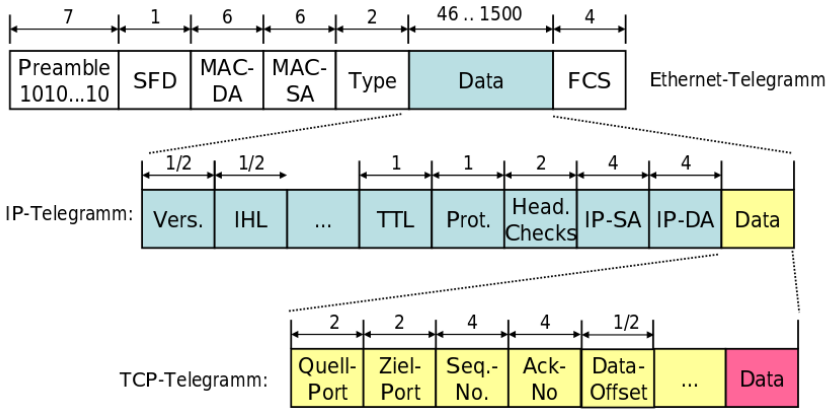

Abbildung 2.5: Verschachteltes IP-Paket, Quelle: Frank (2010)

Anhand dieser Grafik wird deutlich, dass abgesehen von den eigentlich zu übertragenden Daten (hier *DATA*) eine Menge weiterer Informationen übermittelt wird. Teile davon werden zum Routing verwendet, andere dienen zur Überprüfung der erfolgreichen Übertragung. In Verbindung mit dem Zeitpunkt, wann ein Paket empfangen beziehungsweise versendet wurde und weiteren Zeitstempeln von Vermittlungsstellen, ergibt sich die Menge der Kommunikationsdaten.

2.3 Echtzeitnetzwerke

Echtzeitsysteme zeichnen sich dadurch aus, dass Aufgaben zu einem bestimmten Zeitpunkt abgeschlossen sein müssen. In Netzwerken bedeutet das, dass Nachrichten garantiert übertragen werden müssen. Dazu gehört die eigentliche Zustellung der Nachricht, aber auch die maximale Übertragungszeit, die einen definierten Zeitraum nicht überschreiten darf. Den Zeitpunkt, zu dem eine Nachricht zugestellt sein muss, bezeichnet man als Deadline. Wird diese Vorgabe nicht eingehalten, führt das zu unerwünschtem Verhalten bis hin zum Totalausfall des Systems.

Innerhalb von Echtzeitsystemen wird zwischen Systemen mit harten Echtzeitanforderungen und Systemen mit weichen Echtzeitanforderungen unterschieden.

Harte Echtzeitsysteme: Sollte eine Deadline nicht eingehalten werden, ist die Nachricht unbrauchbar. Durch das Nicht- Erfüllen kommt es zu Störungen im System. Im schlimmsten Falle droht Gefahr für Leib und Leben. (Beispiel: Maschinensteuerung in Industrie und Kraftwerken)

Weiche Echtzeitsysteme: Sollte eine Deadline nicht eingehalten werden, ist die Nachricht zwar noch verwertbar, die Qualität leidet jedoch. (Beispiel: Video- und Audioübertragungen, bei denen es zu Aussetzern kommen kann)

Das Netzwerk im Prototypenfahrzeug des RECBar Projekts ist als hartes Echtzeitsystem einzustufen. Der Verlust von Pakten oder deren nicht fristgerechte Übertragung kann schwerwiegende Auswirkungen haben. So kann zum Beispiel ein blockierender Reifen zu einem Verkehrsunfall führen, wenn das ABS System auf Grund von Übertragungsproblemen nicht arbeiten konnte.

Time-Triggered Ethernet

Time-Triggered Ethernet oder TTEthernet geht auf ein Projekt an der TU Wien zurück (vgl. Kopetz u. a. (2005)). Es wurde von Honeywell und TTTech weiter entwickelt und ist inzwischen von der Society of Automotive Engineers unter SAE AS6802 standardisiert worden.

Es erweitert das Standard Ethernet insoweit, als dass neben normalem Best-Effort Verkehr auch Echtzeitverkehr möglich ist. Um Nachrichten möglichst verzögerungsfrei übertragen zu können, bedarf es einer Priorisierung wichtiger Nachrichten. Dazu definiert TTEthernet diese drei verschiedenen Nachrichtentypen (vgl. Steiner (2008)):

best-effort: Best-Effort Traffic entspricht Standard Ethernet Traffic. Es gibt keinerlei Zusicherung, dass Pakete zu bestimmten Zeiten übertragen und zugestellt oder überhaupt zugestellt werden. Best-Effort Nachrichten haben im TTEthernet die geringste Priorität.

rate-constraint: Rate-Constraint Traffic hat im TTEthernet eine mittlere Priorität. Es sichert einer Anwendung eine garantierte Bandbreite im Netzwerk zu.

time-triggered: Time-Triggered Traffic wird zu garantierten Zeitpunkten und mit minimaler zeitlicher Verzögerung übertragen. Um dies zu erreichen, werden die Übertragungszeitpunkte von Time-Triggered Nachrichten im Voraus festgelegt. Sie haben die höchste Priorität und unterliegen harten Echtzeitanforderungen.

Die Markierung der Nachrichten als Echtzeitverkehr findet im Feld der Zieladresse statt (vgl. 2.2). Grundlage der Echtzeitfähigkeit des Netzwerks ist eine genau Zeitsynchronisation aller

Teilnehmer. Diese ermöglicht es, Zeitfenster vorzukonfigurieren, in denen dann nur bestimmte Nachrichtentypen übertragen werden. Um in einem Netzwerk TTEthernet einzusetzen benötigt man spezielle Netzwerkgeräte. Diese sind jedoch abwärtskompatibel, sodass ein Mischbetrieb mit herkömmlicher Technik möglich ist. Mit Hilfe von TTEthernet können sich Netzwerke in Systemen wie Fahrzeugen und Flugzeugen, die wegen ihrer Echtzeitanforderungen nicht auf Standard Ethernet aufbauen können, mit Ethernet umsetzen lassen.

2.4 Arbeitsmittel

2.4.1 Postgres Datenbank

Zum Speichern der aufgezeichneten Daten kommt im Fahrzeug eine PostgreSQL Datenbank (kurz: Postgres) zum Einsatz. Postgres zeichnet sich als objektrelationales Datenbankmanagementsystem (ORDBMS) durch seine freie Verfügbarkeit und breite Unterstützung des SQL-Standards ANSI-SQL 2008 aus. Bemerkenswert ist die große aktive Community, die das System ständig weiter entwickelt. Das Datenbanksystem steht unter der PostgresSQL Lizenz, die vergleichbar mit den BSD-Lizenzen ist. Damit darf die Software frei verwendet werden und auch in kommerziellen Produkten zum Einsatz kommen. (PostgreSQL-Global-Development-Group)

2.4.2 Bootstrap

"Bootstrap ist ein freies CSS-Framework. Es enthält auf HTML und CSS basierende Gestaltungsvorlagen für Typografie, Formulare, Buttons, Tabellen, Grid-Systeme, Navigations- und andere Oberflächengestaltungselemente.“ (Wikipedia.de)

Bootstrap wird verwendet, um Webanwendungen auf unterschiedlichen Geräten einheitlich und ansprechend darzustellen. Es verfolgt dabei den mobile-first Ansatz und ist dadurch besonders für die Darstellung von Webseiten auf Tablets und Smartphones geeignet.

Das gesamte Framkework wird von den Entwicklern unter der MIT-Lizenz zur Verfügung gestellt, ist damit frei verfügbar und darf ohne Einschränkung auch in kommerziellen Projekten benutzt werden. Für die Verwendung reicht es aus, die kompillierten CSS-Dateien in das eigene Projekt einzubinden.

2.4.3 PHP

PHP ist eine serverseitig ausgeführte Skriptsprache, die zur Erstellung von dynamischen Webseiten dient. Durch die große Anzahl unterstützter Datenbanken ist sie für dieses Projekt

die erste Wahl. Erwähnenswert ist die ausgezeichnete Dokumentation von PHP und die Vielzahl an Codebeispielen, die im Netz zu finden sind. [4]

[4] http://php.net/

3 Analyse

In diesem Kapitel erfolgt eine Beschreibung der aktuellen Herausforderungen, die sich bezüglich des Datenloggins im bisherigen Versuchsaufbau des RecBar Projekts ergeben haben. Dazu erfolgt nach einer Charakterisierung des Problemfelds eine IST-Analyse, anhand derer im Anschluss mögliche Lösungsansätze entwickelt werden.

3.1 Problemfeld

Die Entwicklung des Echtzeitnetzwerks im Prototypenfahrzeug geschieht unter besonderen Sicherheitsansprüchen. Die gemachten Aussagen zur Echtzeitübermittlung von Daten, seien es Statusinformationen für Assistenzsysteme wie ABS oder ESP, aber auch größere Nutzdaten, wie zum Beispiel für Kameraanwendungen, müssen zu einhundert Prozent belegbar und belastbar sein. Dafür sind Netzwerkanalysen zur Validierung und Verifizierung der Laufzeiten unbedingt notwendig. Das Zeitverhalten muss unter allen denkbaren Szenarien getestet und dokumentiert werden. Nur so kann sichergestellt werden, dass das Netzwerk den hohen Ansprüchen bezüglich der Zeitanforderungen gerecht wird. Der Fokus liegt auf der Einhaltung von Deadlines, das heißt dem Einhalten garantierter Laufzeiten. Darüberhinaus müssen aber auch eventuelle Paketverluste und Jitter, also die Abweichungen von durchschnittlichen Übertragungszeiten, berücksichtigt werden.

Um das Fahrzeug im Straßenverkehr sicher bewegen zu können und die Straßenverkehrsordnung einzuhalten, muss bewiesen werden, dass alle sicherheitsrelevanten Daten in jeder Situation unter Einhaltung der Deadlines übertragen werden. Die durchgeführten Analysen des Netzwerks dienen nicht nur dem Nachweis von Konformitäten, sondern auch der Bestätigung und Verfeinerung von Simulationsergebnissen. So können die Simulationsmodelle mit den Informationen aus Daten echter Testfahrten angepasst und verfeinert werden. In Zukunft können damit weitere Kosten und Zeit gespart werden, da feinere Modelle genauere Aussagen über das Netzwerk zulassen.

Um all das zu ermöglichen, ist ein umfassendes Mitschreiben/ Loggen der Kommunikationsdaten im Fahrzeugnetzwerk während des tatsächlichen Betriebs nötig. Absicht ist es, nach

den Testfahrten die Daten auszuwerten, um Probleme zu verdeutlichen oder Annahmen zu bestätigen.

Testfahrten können unter unterschiedlichen Bedingungen stattfinden. So gibt es bisher keinen definierten Rahmen bezüglich der Länge, der Strecke oder der zeitlichen Dauer einer Fahrt.

3.2 IST-Analyse

Das Netzwerk im Prototypenfahrzeug ist zur Zeit so konfiguriert, dass jedes Ethernetpaket zusätzlich zu seinem eigentlichen Empfänger auch zum Logging-PC geroutet wird. Dabei handelt es sich um einen dezidierten Rechner mit folgender Ausstattung: Intel i5, HDD, 8GB DDR3 Speicher, Ubuntu 14.04 LTS. Der Logging-PC ist zusätzlich zum kabelgebundenen Ethernetnetz auch über ein ad-hoc WLAN erreichbar.

Je nach Konfiguration kommunizieren derzeit maximal drei CAN Busse, zwei Kameras mit einer Auflösung von 1280x1024 Pixeln und zwei Laserscanner über den Backbone. Dazu kommt ein beliebiger Teil sogenannter Best Effort Traffic. Dabei handelt es sich um Datenpakete ohne Echtzeitanforderung, wie zum Beispiel die Ethernetpakete zum Steuern der Datenaufzeichnung an den Logging-PC.

Geplant ist, in Zukunft weitere Geräte über die neue Infrastruktur kommunizieren zu lassen. Hierbei handelt es sich beispielsweise um Daten weiterer Kameras, Daten mobiler Internetanwendungen, Kommunikation mit anderen Fahrzeugen oder Daten des Multimediasystems. Somit wird das Datenaufkommen langfristig zunehmen.

Aktuell werden bei Testfahren pro Minute ca. 140000 (can1: 97000, can2: 25000 Pakete, can3: 18000) CAN Nachrichten bei Berücksichtungen von dei CAN Bussen verarbeitet. Dazu kommen die Daten der Kameras und der Laserscanner. Die Auslastung des Netzwerks schwankt stark und ist zudem abhängig von der Fahrsituation des Autos. Die größten Unterschiede gibt es bei der Datenmenge, die die Kameras übertragen, da die aufgenommenen Bilder sich unterschiedlich stark komprimieren lassen und so verschiedene Dateigrößen haben. Die Dateigröße steigt bei zunehmender Detaillierung der aufgenommenen Bilder.

Um sämtliche Möglichkeiten der Auswertung zu gewährleisten, werden derzeit alle Pakete inklusive der zugehörigen Nutzdaten gespeichert. Um den Loggingprozess zu entlasten und die Menge der Daten, die gespeichert werden, in Zukunft zu minimieren, könnte auf das Speichern der Nutzdaten verzichtet werden. Für das Auswerten von Timingverhalten genügt es, nur die Paketheader zu speichern, anhand derer man Übertragungszeiten und das Kommunikationsverhalten erkennen kann.

Das Empfangen und Aufzeichnen der Daten auf dem Logging-PC erfolgt mit der Software EthernetLogger, einem Programm, das innerhalb des RecBar-Projekts für eben diesen Zweck entwickelt worden ist. Die Pakete werden von diesem Programm analysiert und anschließend nach Quellen sortiert in einer Postgres Datenbank gespeichert. In dieser Datenbank existiert für jede mögliche Quelle (CAN-Bus, Kamera usw.) eine Tabelle.

Es existieren derzeit folgende Tabellen, in denen die über das Netzwerk versendeten Pakete, nachdem sie anhand ihrer ID unterschieden worden sind, seperat gespeichert werden:

can1, can2, can3: Diese drei Tabellen sind für die Pakete der drei CAN-Busse, die per Gateway über den Ethernetbackbone geleitet werden können.

gps: Die hier gespeicherten Pakete beinhalten die aktuellen GPS-Koordinaten sowie die Geschwindigkeit in km/h und zusätzlich die Orientierung als Winkelangabe. Die Werte werden beim Empfangen extrahiert und in die dementsprechende Spalte eingefügt.

kamera_front, kamera_rear: Für die Bilddaten der beiden im Fahrzeug verbauten Kameras gibt es diese zwei Tabellen.

lidar_1, lidar_2: In diesen Tabellen werden die Pakete der Laserscanner gespeichert.

Für die Daten der Laserscanner werden in extra Tabellen auch die einzelnen Ethernetframes ohne Nutzdaten, aber mit seperaten Timestamps, geloggt. Die Tabllen heißen **lidar_1_ethernetframes** und **lidar_2_ethernetframes**. Alle ankommenden Pakete, die keiner dieser Tabellen zugeordnet werden können, werden in **defaulttable** gespeichert.

Weiterhin gibt es Tabellen, die nicht vom Logger genutzt werden, sondern der Verwaltung dienen. Dazu zählen **device_config** mit Angaben zum aktuellen Softwarestand auf den verschiedenen Switches, Gateways usw. und **vlinkstable**.

p_key	date	time_utc	timestamp_a	timestamp_b	data

Abbildung 3.1: Allgemeines Tabllenschema

Abbildung 3.1 zeigt den allgemeinen Aufbau einer Tabelle, in der die aufgezeichneten Daten gespeichert werden. *Date* und *time_utc* beziehen sich immer auf den Zeitpunkt, an dem das Paket am Logging-PC eingetroffen ist. *timestamp_a* und *timestamp_b* sind Platzhalter für Zeitpunkte während der Übertragung und existieren nicht in allen Tabellen, da die Route der unterschiedlichen Pakettypen variiert und somit unterschiedlich viele timestamps festgehalten

werden können. In *data* liegen letztendlich die Nutzdaten des jeweiligen Paketes. Die Tabellen unterscheiden sich je nach Pakettyp, der in ihnen gespeichert wird.

Beispiel eines speziellen Tabellenaufbaus anhand der Tabelle für CAN-Pakete: [1]

	p_key [PK] bigint	date date	time_utc time without time zone	timestamp bigint	busid bigint	canmsgid bigint	gatewaytimestamp bigint	data character varying
1	541687	2016-03-14	13:41:54	1457962910927544072	2	1635	416788373	7828000600990000
2	541686	2016-03-14	13:41:54	1457962910927544072	2	253	416783196	25df1f8100000000
3	541685	2016-03-14	13:41:54	1457962910918942712	2	263	416781596	000000000000f2cf
4	541684	2016-03-14	13:41:54	1457962910918942712	2	987	416778369	5080000003200076

Abbildung 3.2: Tabelle für CAN Pakete

Die Datenaufzeichnung startet automatisch mit Inbetriebnahme des Fahrzeugs. Es sind keine weiteren Tätigkeiten nötig und während einer Testfahrt hat man keine Möglichkeit auf die Aufzeichnung Einfluss zu nehmen.

So kann man derzeit nicht festlegen, welche Pakete geloggt werden sollen, um so eine Minimierung der Daten zu erreichen. Auch eine Auswertung der Daten während der Fahrt ist zur Zeit nicht möglich.

Eine Auswertung nach der Fahrt wird dadurch erschwert, dass kein Export der Daten aus der Datenbank möglich ist. Nur durch ein manuelles Kopieren der gesamten Datenbank per ssh auf dem Logging-PC selbst gelangt man an den Inhalt. Es existiert keine grafische Benutzeroberfläche, weder um die Aufzeichnung zu steuern, noch um die aufgezeichneten Daten zu sichten.

Abgesehen von den oben genannten Daten werden keinerlei Informationen, die eine Testfahrt und ihre Umstände beschreiben, festgehalten. Es fehlen zum Bespiel Angaben zu dem Fahrer, den Witterungsbedingungen oder dem Zweck der Testfahrt. Zur Zeit ist es lediglich über den Zeitpunkt der Aufzeichnung möglich, die Daten einer Fahrt zuzuordnen.

[1] Eine detaillierte Aufzählung aller Tabellen inklusive der Datentypen befindet sich im Anhang der digitalen Version dieser Arbeit.

4 Konzept

Im Zuge der Problemanalyse wurde schon eine Vielzahl von Möglichkeiten der Optimierung deutlich. Im folgenden Kapitel werden diese aufgezählt und näher beschrieben. Am Ende steht eine Übersicht der Verbesserungsmöglichkeiten. Diese sollen dann im nächsten Abschnitt bewertet werden, um daraufhin eine begründete Auswahl der Aspekte, die umgesetzt werden sollen, treffen zu können.

4.1 Möglichkeiten der Optimierung

Bei der Erarbeitung des Lastenhefts wurden verschiedene Stakeholder identifiziert. Dazu zählen unter anderem der/ die Fahrer/ Beifahrer während der Testfahrt als Nutzer einer grafischen Benutzeroberfläche und Entwickler, die ebenfalls Software für das Auto entwickeln im Hinblick auf gemeinsame Schnittstellen. Ebenfalls berücksichtigt werden müssen Programme, die die aufgezeichneten Daten auswerten sollen. Hier ist es wichtig, bereits etablierte Datenformate zu unterstützen. Die vorhandene Hardware gibt den Rahmen vor, mit der sich die Software umsetzten lassen muss. Unter Berücksichtigung dieser Stakeholder und deren Anforderungen ergaben sich folgende Anforderungen an das Programm:

- **das Starten und Beenden von Testfahrten**, um einen zeitlichen Rahmen setzen zu können
- die Möglichkeit bestimmte **Zeitpunkte/ Ereignisse festzuhalten** und zu kommentieren, um diese später schnell wiederfinden zu können
- das zusätzliche Festhalten von **Metadaten**, wie zum Beispiel: Fahrer, Beifahrer und Zweck der Testfahrt
- eine Funktion, um die aufgezeichneten **Daten exportieren** zu können
- eine **Schnittstelle**, um Versionsstände der Firmware der im Fahrzeug verbauten Switches, Gateways und Controller speichern zu können

- ein Konzept zur **Minimierung der aufgezeichneten Daten**, um ein Überlaufen der vorhandenen Speichermedien zu verhindern
- ein **lokales Verwalten** der exportierten Daten
- eine **grafische Nutzeroberfläche**, um die neuen Funktionen komfortabel steuern zu können
- die **Kompatibilität** der Anwendung mit verschiedenen Endgeräten, eventuell auch mobilen wie Smartphone und Tablet
- die **Darstellung des Aufzeichnungsprozesses**, um die Funktionalität zu überwachen
- eine **Visualisierung der aufgezeichneten Daten** während der Fahrt, um erste Auswertungen just in time zu machen

4.2 Definition von Kriterien für die Umsetzung

Um aus der obigen Anforderungsliste zu einem Pflichtenheft zu kommen, bedarf es einer Bewertung der aufgelisteten Punkte. Dafür werden nun zunächst drei allgemeine Aspekte für die Bewertung festgelegt und anschließend unter Berücksichtigung der Besonderheiten des RecBar-Projekts konkretisiert.

- **Umsetzbarkeit** - Ist die Anforderung mit angemessenen Mitteln und in der vorgegebenen Zeit realisierbar?
- **Nutzen** - Profitiert das gesamte Projekt davon oder ist es nur ein minimaler Gewinn, von dem nur wenige etwas haben?
- **Priorität** - Muss dieses Feature möglichst schnell zur Verfügung stehen?

Konkretisierung:

- **Umsetzbarkeit in Bezug auf Mittel:** Kann man das Feature mit der vorgegebenen Hardware umsetzen? Reicht die Performance des gesamten Systems dafür aus oder wären Veränderungen notwendig? Wenn ja, wie teuer wäre diese Anpassung? Stehen für die Arbeit finanzielle Mittel zur Verfügung?
- **Umsetzbarkeit in Bezug auf Zeit:** Ist es möglich, die Programmteile alleine und in der vorgegebenen Zeit von 3-6 Monaten zu entwickeln, zu testen und zu dokumentieren? Gibt es innerhalb des RecBar-Projekts eventuell Erfahrungen und Vorwissen mit der in Frage kommenden Programmiersprache/ Framework/ Entwicklungsumgebung?

- **Nutzen:** Bringt die neue Software eine deutliche Verbesserung für das gesamte Projekt? Wer würde, wie stark, von welchen Verbesserungen profitieren? Wie hoch ist die Wahrscheinlichkeit, dass die Software in Zukunft auch genutzt wird?
- **Priorität:** Welche Verbesserung sollte zeitnah umgesetzt werden, da ihre Existenz die Arbeit des gesamten Projekt erleichtert, welche kann eventuell in einer Folgearbeit behandelt werden?

4.3 Auswahl von Optimierungen

Die Bewertung der möglichen Optimierungen anhand der Kriterien ergibt folgende Auflistung, die einzelnen Aspekte werden nachstehend näher erläutert:

	Umsetzbarkeit Mittel (tech. Voraussetzung)	Umsetzbarkeit Zeit (Zeitfaktor)	Nutzen	Priorität
Starten/Beenden	gegeben	-	hoch	sehr hoch
Points of Interest	gegeben	-	hoch	sehr hoch
Metadaten	gegeben	-	hoch	sehr hoch
Export	gegeben	-	hoch	hoch
GUI	gegeben	-	hoch	hoch
Endgerätekompatibilität	gegeben	gering	mittel	niedrig
Schnittstellen	gegeben	gering	-	mittel
Datenminimierung	gegeben	mittel	mittel	niedrig
Lokales Verwalten der exportierten Daten	gegeben	mittel	mittel	niedrig
Überwachung der Aufzeichnung	gegeben	hoch	mittel	niedrig
Visualisierung der Daten	gegeben	sehr hoch	hoch	mittel
Steuern per Autoradio	nicht gegeben	sehr hoch	gering	niedrig

Tabelle 4.1: Kriterienübersicht

Das **Starten und Beenden der Aufzeichnung**, um einen genauen Zeitraum für eine Aufzeichnung bzw. Testfahrt zu definieren, ist die zentrale Anforderung und soll definitiv umgesetzt werden. Das Gleiche gilt für das Setzen sogenannter **Points of Interest**, um Zeitpunkte manuell festhalten und kommentieren zu können. Dazu kommt das Festhalten sogenannter **Metadaten**. Dazu sollen Datum und Zeitraum der Fahrt, Fahrer, Beifahrer, Zweck der Testfahrt und das Wetter gehören. Zusätzlich wird der aktuelle Stand bezüglich der Firmware aller

verbauten Netzwerkgeräte festgehalten. Diese drei Punkte bilden den Kern der Anwendung und werden daher zusammenhängend betrachtet. Ihre Umsetzung hat die größte Priorität innerhalb der Arbeit. Alle weiteren Features sind als Ergänzungen zu betrachten. Überprüft man die Umsetzbarkeit dieser drei Aspekte bezüglich der zur Verfügung stehenden Mittel lässt sich feststellen, dass die technischen Voraussetzungen im Fahrzeug durch die Hardwareausstattung bei der Planung geschaffen worden sind, um diese Anforderungen zu realisieren. So ist der Logging-PC durch das Loggen der Netzwerkdaten bisher lediglich mit zehn Prozent ausgelastet. Weiterer Best-Effort Netzwerkverkehr, der beim Steuern der Anwendung oder während einer Übertragung von exportierten Daten entsteht, ist ebenfalls kein Hindernis, da dieser die Realtimeübertragung nicht beeinflusst. In Bezug auf den Nutzen der drei Aspekte, lässt sich festhalten, dass durch die zusätzlich gespeicherten Daten eine Auswertung aller Kommunikationsdaten in Zukunft deutlich einfacher wäre. Somit besteht ein großer Nutzen für die Entwickler, die sich mit der Analyse der Daten auseinandersetzen. Von einer erleichterten Auswertung würde ebenfalls der Teil der Arbeitsgruppe des RecBar Projekts, die das Verhalten des Netzwerks simuliert, profitieren. Es besteht also insgesamt ein großer Nutzen für das gesamte Projekt.

Zusätzlich zu den drei festgelegten Hauptanforderungen erscheint es sinnvoll, diese durch eine **Exportfunktion** sowie eine **grafische Nutzteroberfläche** zu ergänzen. Die Möglichkeit, die angefallenen Daten zu exportieren, ist zwingende Voraussetzung, um diese anschließend weiter zu verarbeiten und zu analysieren. Der Export sollte dafür in einem möglichst kompatiblen Format geschehen und sich damit den Anforderungen der Arbeitsgruppe anpassen.

Um die Metadaten der Testfahrt komfortabel eingeben und Points of Interest auch während der Fahrt unkompliziert erfassen zu könnnen, ist eine grafische Benutzeroberfläche notwendig. Diese soll vom Beifahrer bedient werden und möglichst einfach gestaltet sein. Hierbei liegt der Schwerpunkt deutlich auf der Funktionalität und weniger auf Designaspekten.

Grundsätzlich kann man davon ausgehen, dass der Beifahrer mit einem Laptop ausgestattet ist. Somit steht sowohl für die Oberfläche als auch für einen Export der Daten ein vollwertiges System hinsichtlich Leistung und Darstellungsmöglichkeiten zur Verfügung.

Insgesamt wird somit deutlich, dass erst durch die Kombination dieser fünf Punkte eine Leistungssteigerung und Erleichterung des bisherigen Arbeitsablaufs erreicht werden kann.

Ein Zeitraum von 3-6 Monaten sollte nach meinen bisherigen Erfahrungen reichen, um die angesprochenen Features zu realisieren.

Zusätzlich erscheint die Berücksichtigung der beiden Aspekte **Schnittstelle** und **Endgerätekompatibilität** aufgrund ihrer problemlosen Umsetzbarkeit sinnvoll, obwohl sie keine besondere Priorität haben. Durch die Verwendung der bereits installierten Datenbank eta-

bliert sich diese als zentrale **Schnittstelle** auf dem Logging-PC. So können darüber auch die Versionsstände der Firmware der im Fahrzeug verbauten Switches, Gateways und Controller gespeichert und organisiert werden. Der Forderung nach **Kompatibilität** mit möglichst vielen Endgeräten kann man einfach durch Wahl eines geeigneten Frameworks gerecht werden.

Die **Minimierung der aufgezeichneten Daten**, um ein Überlaufen der vorhandenen Speichermedien zu verhindern, muss langfristig berücksichtigt werden. Aktuell ist das Datenaufkommen bei Testfahrten für das System keine Überforderung. Die Priorität ist somit gering und dieser Punkt wird daher in dieser Arbeit nicht weiter berücksichtigt.

Die Forderung, ein und dasselbe Programm zum **lokalen Verwalten** der exportierten Daten zu nutzen, liegt auf der Hand. Absicht hier ist es jedoch, die exportierten Daten so handlich zu machen, dass diese mit herkömmlichen Bordmitteln der Betriebssysteme verwaltet werden können. Der Nutzen wäre somit gering und dieses Feature wird nicht weiter berücksichtigt.

Das Programm EthernetLogger ist für den Aufzeichnungsprozess zuständig. Eine Funktion zum **Darstellen des Aufzeichnungsprozesses** müsste also hier ansetzen. Vergangene Untersuchungen im Rahmen der Projektarbeit haben gezeigt, dass der Aufzeichnungsprozess zuverlässig abläuft. (vgl. Meyer (2014)) Die Priorität und der Nutzen sind daher niedrig. Der hohe Aufwand zur Realisierung ist also nicht gerechtfertigt, sodass diese Funktion nicht weiter berücksichtigt wird.

Ein Programm zur **Visualisierung der aufgezeichneten Daten** während der Fahrt wird parallel in einer anderen Arbeit der Arbeitsgruppe entwickelt und ist daher nicht weiter Gegenstand dieser Arbeit.

Zusammengefasst ergibt sich aus der Prüfung möglicher Optimierungen folgende Auswahl an Anforderungen an das Programm: Starten/Beenden der Aufzeichnung, Festhalten von Points of Interest, Speichern zusätzlicher Metadaten, Export der aufgezeichneten Daten, GUI, Kompatibilität zu verschiedenen Endgeräten, Schaffung einer Schnittstelle für den Datenaustausch.

Prüfung anhand allgemeiner Qualitätsanforderungen

In dem Planungsprozess wurden auch allgemeine Qualitätsanforderungen an Software, wie sie in der ISO 9126[1] definiert werden, berücksichtigt.

Die Bewertungen der einzelnen Kriterien sind in dieser Tabelle zusammengefasst und werden im weiteren Text erläutert:

Qualitätsforderung	sehr hoch	hoch	mittel	niedrig
Funktionalität		x		
Zuverlässigkeit	x			
Benutzbarkeit			x	
Effizienz			x	
Änderbarkeit				x
Portierbarkeit				x

Tabelle 4.2: Qualitätsanforderungen

Der Fokus sollte auf der Funktionalität und der Zuverlässigkeit liegen, da die Software im erweiterten sicherheitskritischen Bereich angesiedelt ist. Darüber hinaus sollte auch beim Testen besonderer Wert auf diese Aspekte gelegt werden. Da der Kreis der Nutzer eingeschränkt und grundsätzlich als versiert einzustufen ist, ist Benutzbarkeit mit “mittel“ bewertet. Die vorhandene Hardware wurde im vorherigen Kapitel als potent und ausreichend bewertet. Da die Rechenleistung in Zukunft weiter steigen wird, ist Effizienz ebenfalls mit "mittel“ bewertet. Das Programm wird für einen speziellen Einsatzzweck in einem definierten System entwickelt. Änderbarkeit und Portierbarkeit sind daher nicht irrelevant aber insgesamt als niedrig einzustufen.

[1] https://de.wikipedia.org/wiki/ISO/IEC_9126

4.4 Designoptionen

Für die Umsetzung wurden mehrere mögliche Ansätze entwickelt. Drei Optionen werden im Folgenden näher erläutert und miteinander verglichen.

4.4.1 Option A

Option A besteht aus zwei lokalen Anwendungen, einer auf dem Logging-PC und der anderen auf dem Gerät des Beifahrers. Die Anwendung des Beifahrers stellt eine grafische Nutzeroberfläche bereit und kommuniziert mit der Anwendung auf dem Logging-PC. Diese startet und beendet das Programm EthernetLogger und hat damit direkt Einfluss auf die Datenaufzeichnung. Für den Export von Daten aus der Datenbank kann sie auch direkt auf diese zugreifen. Als Programmiersprachen kämen C, C++ oder Java in Frage. Um entfernte Methodenaufrufe zu realisieren und die Kommunikation der beiden Programmteile zu ermöglichen, könnte Corba oder Java RMI zum Einsatz kommen.

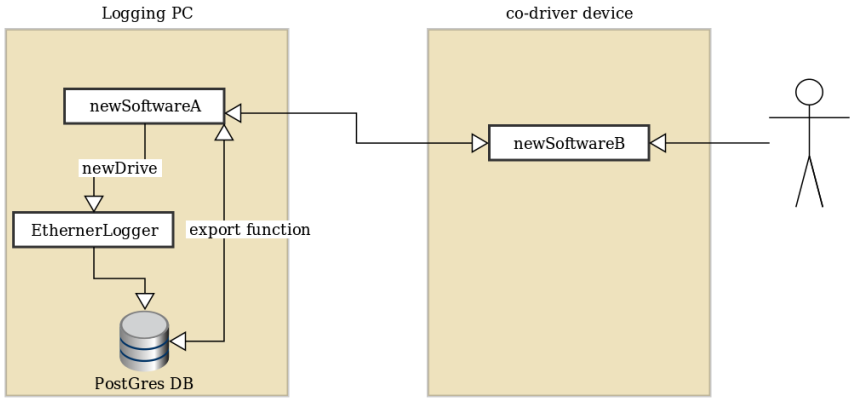

Abbildung 4.1: Komponentendiagramm Option A

4.4.2 Option B

Option B verfolgt den Ansatz einer reinen Webanwendung mit Hilfe von Webtechniken wie PHP, Javascript oder Ajax. Der Nutzer würde das Programm ausschließlich über einen Browser nutzen. Dadurch wird eine hohe Kompatibilität erreicht. Voraussetzung dafür ist ein Webserver, der die Möglichkeit bietet, PHP-Skripte auszuführen. Der bisherige Loggingvorgang wird dabei nicht verändert.

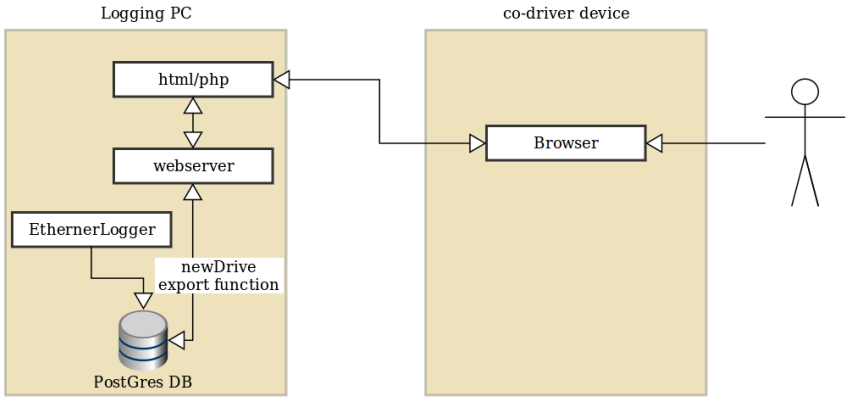

Abbildung 4.2: Komponentendiagramm Option B

4.4.3 Option C

Als dritte Option ist eine Anwendung denkbar, die lokal auf dem Gerät des Beifahrers läuft und von da aus direkt auf die Datenbank zugreift. Dieser Ansatz wurde nicht weiter entwickelt, da er der Anforderung mit vielen Endgeräten kompatibel zu sein, massiv widerspricht. Eine solche Anwendung, die auf unterschiedlichen Betriebssystemen und mobilen Geräten nutzbar ist, zu programmieren, bedeutet einen enorm hohen Aufwand und ist mit anderen Ansätzen einfacher umzusetzen.

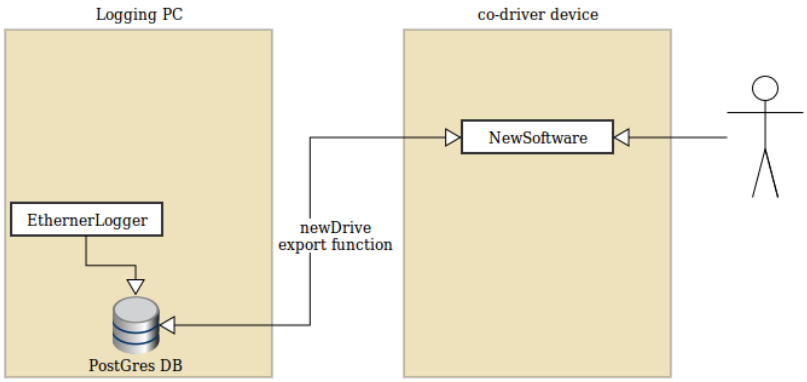

Abbildung 4.3: Komponentendiagramm Option C

4.4.4 Vergleich der Optionen

Webanwendungen bieten die Möglichkeit, unkompliziert ansprechende grafische Oberflächen für verschiedene Geräte zu gestalten. Die Darstellung im Browser verspricht eine extrem hohe Kompatibilität zu unterschiedlichsten Endgeräten. Eine Webanwendung ist zudem auch auf der Serverseite plattformunabhängig, da lediglich ein Webserver vorausgesetzt wird und diese für viele Plattformen verfügbar sind. Ein weiterer Vorteil besteht darin, dass lediglich der Code auf der Serverseite entwickelt und gewartet werden muss. Dadurch verkürzt sich die Entwicklungszeit und das Programm ist in Zukunft unkomplizierter anzupassen als eine klassische Client/Server Anwendung, bei der häufig beide Seiten angepasst werden müssen. Insgesamt ist dadurch eine langfristge Nutzung der Software möglich.

Beiden Optionen gemein ist, dass sie aus zwei Komponenten bestehen. Ein Teil, der auf dem Logging-PC läuft und einer Komponente zum Steuern der Anwendung. Das wird vor allem bei Option A deutlich. Hier müssten zwei Programme entwickelt werden, die dann miteinander kommunizieren. Das erfordert weitere Komponenten wie zum Beispiel einen Namensdienst. Sollte C als Sprache gewählt werden, wäre eine Portierung auf ein anderes System nur mit relativ viel Aufwand möglich. Java böte eine breitere Unterstützung, erzeugt durch sein Model der virtuellen Maschinen aber auch mehr CPU-Last. Diese beiden Gegenargumente würden nur durch einen besonderen Performancebedarf der Anwendung gerechtfertigt. Dieser besteht aber wie dargelegt nicht.

Option	Pro	Contra
A - lokal	- Performance - persönliche Erfahrung mit C,C++, Java - Nutzung bereits vorhandener Software	- plattformabhängig - geringer Bedienungskomfort - Anpassung von EthernetLogger nötig - zwei Programmteile nötig - hoher Wartungsaufwand
B - web	- Kompatibilität mit vielen Endgeräten - einfache Oberflächengestaltung - Webserver bereits installiert - plattformunabhängig - breite Unterstützung von Postgres durch PHP - nur ein Programmteil zu entwickeln	- wenig eigene Erfahrung mit Webentwicklung - anfällig für Inkonsistenzen

Tabelle 4.3: Designoptionen

Die aufgeführten Argumente zeigen, dass die Vorteile von Option B überwiegen und die Entwicklung einer Webanwendung der eines Client/Server Programms vorzuziehen ist. Der Unterschied in der eigenen Erfahrung mit den jeweiligen Technologien muss hingenommen

werden und ist mit zusätzlicher Einarbeitungszeit aufzufangen. Insgesamt betrachtet scheint daher Option B die passendere Alternative zu sein.

5 Implementation

Nach der Entscheidung, dass Programm als Webanwendung zu realisieren, soll dieses Kapitel die Umsetzung beschreiben. Es wird erläutert, welche Probleme aufgetaucht sind und wie damit umgegangen worden ist. Zudem wird auf die zum Einsatz kommenden Frameworks eingegangen und der Programmaufbau beschrieben.

5.1 Verwendete Skriptsprachen

Wie bereits herausgestellt wurde, spielt die Postgres Datenbank in dieser Arbeit eine zentrale Rolle. Für Datenbankabfragen und das Eintragen von neuen Datensätzen hat sich PHP im Bereich der Webentwicklung serverseitig als Quasistandard etabliert.

JavaScript (JS) ist eine Skriptsprache, die häufig in Webanwendungen verwendet wird. Im Gegensatz zu PHP wird der Programmcode jedoch nicht vom Server ausgeführt, sondern an den Browser übermittelt und dort vom Interpreter auf der Clientseite ausgeführt. JS dient so zum Beispiel dazu HTML Inhalte dynamisch nachzuladen oder Benutzereingaben zu überprüfen. In dem entwickelten Programm kommt JS bei der Darstellung einer ProgressBar und zum Anzeigen einer dynamischen, sortierbaren Tabelle zum Einsatz.

Sicherheitsaspekte der PHP Programmierung

PHP Skripte stellen durch ihre serverseitige Ausführung grundsätzlich ein Sicherheitsrisiko dar. Es gibt diverse typische Angriffsszenarien, bei denen sich der Angreifer unberechtigten Zugriff auf Daten verschafft oder sogar den Server im Ganzen komprimittiert. Zu den bekanntesten zählen SQL-Injektionen und Cross-Site-Scripting(XSS). Bei einer SQL-Injektion nutzt der Angreifer die Möglichkeit, eigenen Code in eine Datenbankabfrage einzuschleusen. Diese Lücke entsteht durch fehlende Validierung oder Maskierung der übergebenen Benutzereingaben. Erfolgreiche SQL-Injektionen ermöglichen es dem Angreifer, Daten auszuspähen, Daten zu verändern oder im schlimmsten Falle, die Kontrolle über den Server zu erlangen. Ähnlich wie bei SQL-Injektionen versucht der Angreifer beim Cross-Site-Scripting(XSS) ebenfalls,

schadhaften Code einzuschleusen und ihn im Anschluss auszuführen. Das Ziel hier ist jedoch meist der Browser des Nutzers.

Dazu kommen weitere mögliche Sicherheitslücken, die durch nachlässige Programmierung entstehen können. Stichworte hierzu sind register_globals und PHP-Sessions. Das Risiko, welches durch diese Lücken entsteht, wirkt sich stets auf die Sicherheit des gesamten Systems aus. Besonders gefährdet von solchen Angriffen sind Server, die über das Internet erreichbar sind. Da das Prototypenfahrzeug jedoch weder über das Internet erreichbar, noch ein lokaler unautorisierter Zugriff möglich ist, ist das Risiko eines solchen Angriffs auf die Webanwedung als sehr gering einzustufen.

Wenn in diesem Zusammenhang von Sicherheit gesprochen wird, bezieht sich das auf andere Aspekte. Dazu zählen Sicherheit im Bezug auf unbeabsichtigte fehlerhafte Benutzung und Sicherheit hinsichtlich der Richtigkeit von erfassten Daten. Es muss also bei der Implementation besonders darauf geachtet werden, dass eine fehlerhafte Bedienung ausgeschlossen werden kann. Dazu werden zum Beispiel Benutzereingaben stets mit default-Werten initialisiert. Durch das Maskieren von Benutzereingaben wird verhindert, dass schadhafte Zeichenketten in Datenbankabfragen eingefügt werden. Darüber hinaus werden Formulare mit dem in HTML5 eingeführten *required*-Attribut versehen, um Benutzereingaben in Freitextfeldern zu erzwingen.

5.2 Programmstruktur

Die Anwendung ist in fünf Module aufgeteilt. Diese Einteilung dient der Kapselung einzelner Programmfunktionen und erleichtert die zukünftige Wartung, da so Programmteile durch das Ersetzen dieser einzelnen Module ausgetauscht werden können. Die Aufteilung in die seperaten Dateien und ihre Bezeichnungen ergeben sich aus den Funktionen, die sie erfüllen.

db_connect.php stellt die Oberfläche mit der Loginmaske zum Verbinden mit einer Datenbank dar.

db_show.php ist das zentrale Modul der Anwendung. Bei Aufruf zeigt es die bereits eingetragenen Fahrten an, außerdem werden von hier alle weiteren Funktionen aufgerufen.

db_export.php beinhaltet die Funktionen und Aufrufe für den Datenexport aus der Datenbank. de_export.php wird ebenfalls beim Löschen von Fahrten und den dazugehörigen Daten aufgerufen.

db_insert.php stellt die Funktion zum Eintragen einer neuen Fahrt zur Verfügung.

poi.php hält bei Aufruf den aktuellen Zeitpunkt fest und lässt den Nutzer anschließend die Beschreibung des Point of Interest als Freitext eintragen.

Einen Überblick über den Zusammenhang der einzelnen PHP Dateien bietet Abbildung 5.1 :

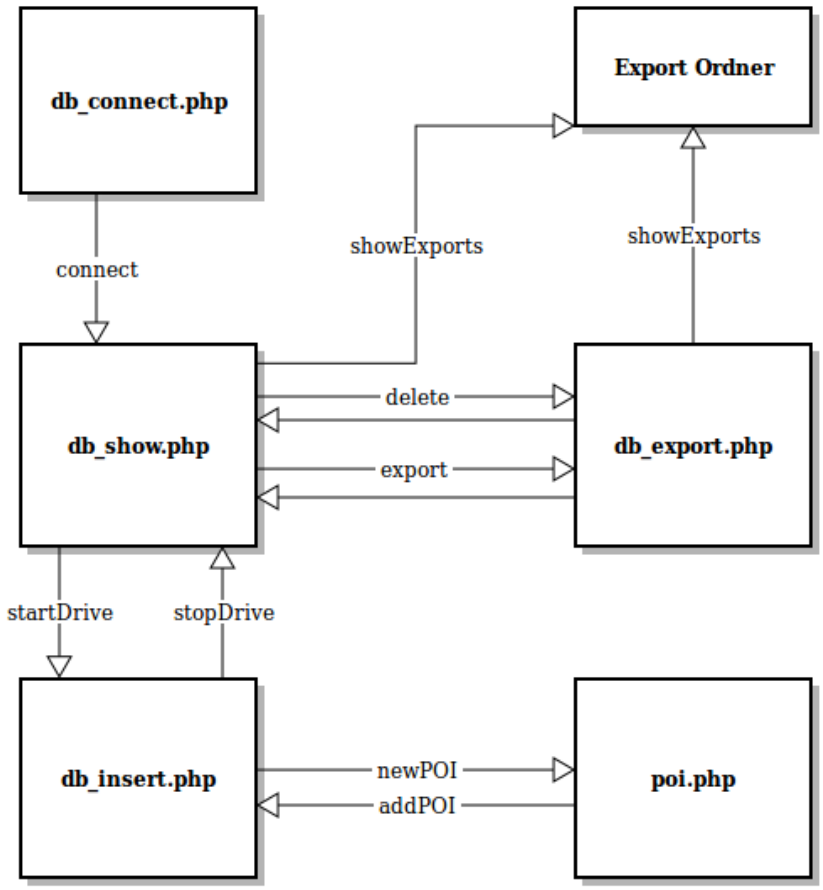

Abbildung 5.1: Programmstruktur

Das Diagramm ist ein Mischung aus Ablauf- und Komponentendiagramm. Die Kästen stehen für die einzelnen PHP-Dateien und die Pfeile sind als Pseudomethodenaufrufe zu interpretieren. Man kann das Schaubild auch als Zustandsdiagramm verstehen, um nachzuvollziehen, in welcher Beziehung die Module zueinander stehen. Ein Zustandsübergang erfolgt dabei immer beim Absenden eines HTML-Formulars, sprich beim Aufruf einer Funktion.

Export

Die Möglichkeit, die aufgezeichneten Daten zu exportieren, war eine wesentliche Anforderung an das Programm. Für den Export stehen zwei verschiedene Dateiformate zur Verfügung. Der Nutzer kann zwischen der Option "csv“ und "db-dump“ wählen. Beiden Optionen gemein ist, dass die exportierten Dateien in einem Archiv komprimiert im Export-Ordner bereitgestellt werden. In diesem Ordner werden alle Exporte vorgehalten und können vom Benutzer beliebig oft heruntergeladen werden. Der Zugriff erfolgt dabei direkt über den Browser, in dem die Dateien mit dessen Boardmitteln angezeigt werden.

Grundsätzlich können für jeden Exportvorgang eine oder mehrere Fahrten ausgewählt werden. Das Gleiche gilt für die Tabellen, die exportiert werden sollen. So kann ein Export zum Beispiel ausschließlich die aufgezeichneten CAN-Daten der letzten Fahrten enthalten.

CSV

CSV-Dateien sind hoch kompatibel und bieten sich für kleinere Auszüge aus der Datenbank an, um diese direkt weiterzuverarbeiten. Bei der Wahl dieser Option wird für jede ausgewählte Tabelle pro Fahrt eine CSV-Datei erstellt. Bei drei ausgewählten Fahrten und zwei zu exportierenden Tabellen heißt das, dass das Archiv sechs Dateien enthält. Den genauen Ablauf für den CSV-Export kann man dem Nassi-Shneidermann-Diagramm in Abbildung 5.2 entnehmen.

DB-Dump

Datenbankdumps hingegen sind geeignet, die Daten langfristig zu archivieren, um später darauf zugreifen zu können. Zudem können diese als Datensicherung dienen. Die erzeugten Datenbankdumps sind komplette Abbilder der Datenbank zum Zeitpunkt des Exports. Mit diesen Dateien lässt sich die Datenbank auf unterschiedlichen Maschinen komplett wiederherstellen. Das bezieht sich sowohl auf das Tabellenschema als auch auf die Daten in den Tabellen. Export und Import funktionieren auch auf unterschiedlichen Architekturen, zum Beispiel zwischen 32bit und 64bit Systemen. Zusätzlich sind die Dumps kompatibel mit allen zukünftigen Versionen von Postgres Datenbanken [1]. Die erstellten Abbilder sollen, genauso wie die CSV-Dateien, nur die Daten der ausgewählten Fahrten und Tabellen enthalten. Um das zu erreichen, müssen die betroffenen Daten vor dem Erstellen des Datenbankdumps erst einmal in eine temporäre Datenbank kopiert werden. Nur so kann eine Auswahl umgesetzt werden, da das Erstellen eines Abbildes keinerlei Selektion von Daten vorsieht, sondern stets die gesamte Datenbank extrahiert. Diese Exportdatenbank existiert parallel zur originalen

[1] https://www.postgresql.org/docs/9.1/static/backup-dump.html

und wird vor jedem Exportvorgang geleert. Die wesentlichen Schritte zum Erstellen eines Datenbankdumps, der nur eine bestimmte Auswahl von Daten der ursprünglichen Datenbank enthält, sind also:
Inhalt der temporären DB löschen -> Daten aus originaler DB auf das Dateisystem kopieren -> Daten von Dateisystem in temporäre DB importieren -> Datenbankdump der temporären DB anfertigen.
Den genauen Ablauf kann man dem Nassi-Shneidermann-Diagramm in Abbildung 5.2 entnehmen.

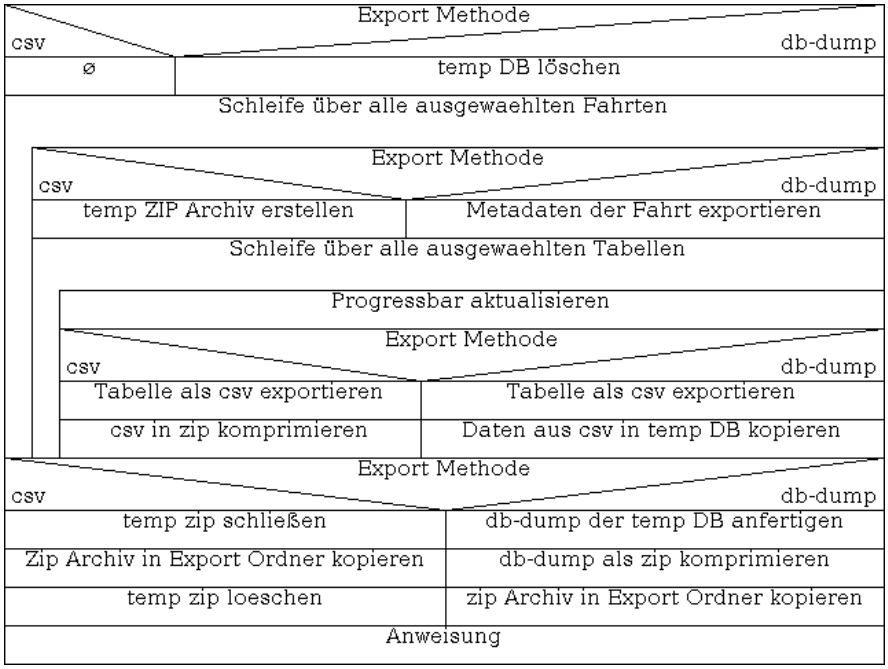

Abbildung 5.2: Nassi-Shneiderman-Diagramm der Exportfunktion

5.3 Systembefehle vs. PHP-Methoden

Die Nutzung von Systembefehlen, also das Ausführen von externen Programmen auf Betriebssystemebene, mit Hilfe von *system()* oder *exec()* sollte im PHP Kontext generell vermieden werden. Da dadurch Programme direkt auf dem Server ausgeführt werden, ergeben sich schnell unvorhersehbare Sicherheitslücken. Durch in Zukunft veraltete Programmversionen entstehen diese eventuell unbermerkt und sind somit umso schwerer zu entdecken.

Wenn möglich sollten also PHP-Methoden genutzt werden. Diese bieten in der Regel mehr Sicherheit, da ihre Funktionen gekapselt und für den jeweiligen Zweck optimiert sind. Durch

die große Anzahl an Plugins für PHP stehen für viele Aufgaben passende Funktionen zur Verfügung. Das Nutzen dieser Funktionen bietet einen weiteren großen Vorteil. So müssen bei der Verwendung keine Änderungen an dem lokalen Rechtesystem vorgenommen werden.

Die Datenbankzugriffe werden daher komplett durch PHP-Methoden durchgeführt. So geschieht eine typische Datenbankabfrage wie folgt:

```
1 $result = pg_query($con,"SELECT * FROM $table WHERE condition");
```

Listing 5.1: DB-Abfrage mit PHP

Das Ergebnis wird dabei in Form eines Array in der Variable $result gespeichert. Leider werden nicht alle benötigten Funktionen durch PHP-Methoden abgedeckt. So muss für die Funktion einen Datenbankdump zu erstellen, also einen gesamten Auszug inklusive der Tabellendefinitionen und der darin gespeicherten Daten, ein Systembefehl ausgeführt werden.

Grundsätzlich bietet die Postgres Datenbank dafür den Befehl *pg_dump* an. Der kann allerdings nicht über die PHP-Erweiterung als Datenbankabfrage ausgeführt werden, sondern muss per *system()*-Befehl aufgerufen werden. Dadurch ergeben sich einige Besonderheiten, die im folgenden Abschnitt erläutert werden. Dabei wird auch die Lösung der Probleme erklärt.

Für die vom Webserver ausgeführten Befehle und Dateioperationen existiert ein extra dafür eingerichteter lokaler Nutzer auf dem System. Auch die vom PHP-Interpreter ausgeführten Systemaufrufe werden mit den Rechten dieses Nutzers ausgeführt. Typischerweise heißt dieser Nutzer www-data. Der Nutzer dient nur diesem Zweck, eine lokale Anmeldung als www-data ist nicht möglich. Ein eigener Nutzer für den Webserver und damit auch die Skripte, die vom Webserver ausgeführt werden, bietet die Möglichkeit, unkompliziert mit Systemmitteln die Rechte einzuschränken und bringt damit einen grundsätzlichen Sicherheitsgewinn. So können vom Webserver ausgeführte Skripte und damit auch die Nutzer des Webdienstes per default nur in bestimmten Verzeichnissen Daten lesen und schreiben. Das Ausführen von Programmen auf dem Webserver ist dadurch ebenfalls eingeschränkt. Zugriffsrechte beziehen sich immer auf das Verhältnis zwischen einem Nutzer und einer Datei. Es werden grundsätzlich drei verschiedene Berechtigungen unterschieden. So kann ein Nutzer das Recht haben eine Datei zu lesen, sie zu beschreiben und/ oder sie auszuführen. [2]

Das Programm */usr/bin/pg_dump*[3] wird in der Regel von einem lokalen Nutzer des Systems, dem Datenbankadministrator, benutzt. Es erstellt ein Abbild der Datenbank, welches als Backup dienen oder auf anderen Systemen wiederhergestellt werden kann (siehe Abschnitt DB-Dump).

[2] https://wiki.ubuntuusers.de/Rechte/
[3] https://www.postgresql.org/docs/9.5/static/app-pgdump.html

Damit der Befehl erfolgreich ausgeführt werden kann und der Datenbankdump erstellt wird, muss der Nutzer entweder berechtigt werden, diesen auszuführen oder der Befehl wird von einem berechtigten Nutzer ausgeführt.

Um den ausführenden Nutzer eines Befehls einmalig anzupassen, gibt es das Werkzeug *sudo*. *sudo* (kurz für SuperUserDo) dient normalerweise dazu, Befehle als Administrator des jeweiligen Systems auszuführen. Mit der Option *-u NUTZER* ist es auch möglich, einen bestimmten Benutzer anzugeben. So könnte man per

```
1 system("sudo -u NUTZER $pg_dump_dir/pg_dump -h
2       localhost $dbname >> $file_name");
```

Listing 5.2: pg_dump: sudo

den Befehl als NUTZER ausführen.

Vorher muss der Nutzer aber erst das Recht bekommen, den Befehl *pg_dump* per sudo ausführen zu dürfen. Dafür ist eine Anpassung der Datei */etc/sudoers* nötig. In dieser Konfigurationsdatei wird bestimmt, welche Nutzer welche Befehle mit sudo ausführen dürfen. Eine Veränderung dieser Datei kann schwerwiegende Folgen haben und sollte nur mit Bedacht durchgeführt werden, da das System im schlimmsten Falle unbrauchbar werden kann oder durch unvorsichtiges Handeln Sicherheitslücken entstehen können. Insgesamt entsteht durch diese Lösung ein höherer Installationsaufwand und durch das Anpassen der Systemkonfiguration eine mögliche Sicherheitslücke.

Um das zu vermeiden, sollte alternativ der Nutzer *www-data* der Benutzergruppe *postgres*, die per default Datenbankdumps erstellen darf, hinzugefügt werden. Die Benutzergruppe postgres existiert nach der Postgres Installation und dient der Administration der Datenbank. Nach dem Ändern der Gruppenzugehörigkeit per

```
1 sudo usermod -aG postgres www-data
```

Listing 5.3: Hinzufügen des Nutzer www-data zur Gruppe postgres

kann *pg_dump* so aufgerufen werden:

```
1 putenv('PGPASSWORD=postgres');
2 system("$pg_dump_dir/pg_dump -U postgres -h
3       localhost $dbname >> $file_name");
```

Listing 5.4: Endgültiger pg_dump Aufruf

In der ersten Zeile wird das Passwort des Datenbanknutzers, das von *pg_dump* verwendet wird, gesetzt und erhält den Wert “postgres“. Der Datenbanknutzer wird per Option “-U postgres“ direkt im Aufruf übergeben.

6 Software Dokumentation

6.1 Einrichtung

6.1.1 Voraussetzungen

Die Anwendung wurde unter Ubuntu 15.04 mit der PHP Version 5.6.4-4ubuntu6.4 und Apache/2.4.10 (Ubuntu) entwickelt und getestet. Bei der Kompilierung von PHP muss die Option –enable-zip gesetzt sein. Die Postgres Datenbank lief in der Version 9.4.5. Als Browser kam Firefox in der Version 44.0 zum Einsatz.

6.1.2 Installation

Die Installation erfolgt durch das Kopieren der Verzeichnisstruktur in das root-Verzeichnis des Webservers.

```
1 scp -r QUELLVERZEICHNIS USER@HOST:ZIELVERZEICHNIS
```

Listing 6.1: Kopieren der Dateien

Der Nutzer www-data muss dabei Schreib-/Lesezugriff auf die Ordner ./exports und /tmp haben, um die Exportarchive erstellen zu können. Zusätzlich muss er der Gruppe postgres hinzugefügt werden, um Datenbankdumps erstellen zu können.

```
1 sudo usermod -aG postgres www-data
```

Listing 6.2: www-data der Gruppe postgres hinzufügen

Um die Exportarchive anzeigen zu können, muss *autoindex* in den Webservereinstellungen aktiviert sein.

6.1.3 Nutzung

Login

Die Anwendung ist je nach Konfiguration oder manuell unter SERVERIP/db_connect.php erreichbar. Db_connect ist der Startpunkt und die Anwendung muss stets so aufgerufen werden,

da sonst keine Verbindung zur Datenbank hergestellt werden kann. In der vorgesehenen Konfiguration, die Webanwendung läuft auf derselben Maschine wie die Datenbank, erfolgt die Verbindung zur Datenbank so:

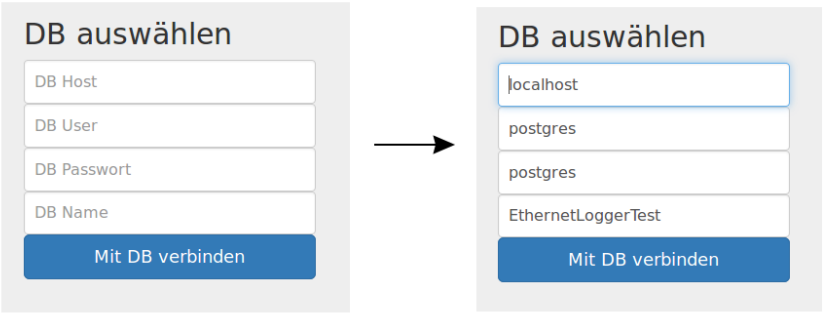

Abbildung 6.1: db_connect: Login

An dieser Stelle ist die Möglichkeit geschaffen worden, die Anwendung in Zukunft mit beliebigen Datenbanken, zum Beispiel einem bereits exportierten Datenbankdump, zu verwenden.

Nach dem Verbinden mit der Datenbank werden alle bisherigen Testfahren in der Tabelle angezeigt.

Abbildung 6.2: db_show: Tabelle der Testfahren

Neue Fahrt

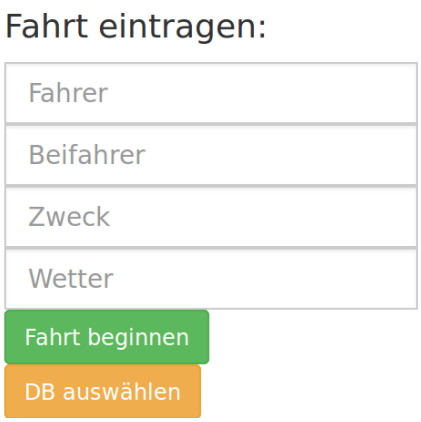

Abbildung 6.3: db_show: Eingabeformular für neue Fahrt

Soll eine neue Testfahrt begonnen werden, müssen die Formularfelder Fahrer, Beifahrer, Zweck und Wetter ausgefüllt werden. Nach Klick auf “Fahrt beginnen“ sind die Daten eingetragen. Sollte ein Feld freigelassen werden, wird dieses durch den Eintrag *default* ersetzt. Nach dem Klick auf "Fahrt beginnen“ sind der Startzeitpunkt in der Datenbank eingetragen und die Aufzeichnung gestartet. Anschließend sieht der Nutzer eine Tabelle mit den während der aktiven Fahrt erstellten Points of Interest.

Abbildung 6.4: db_insert: Darstellung während der Fahrt, ohne POIs

POI erstellen

Points of Interest können während einer laufenden Fahrt festgehalten und der Datenbank hinzugefügt werden. Nach dem Klick auf "POI erstellen“ ist der Zeitpunkt des Klicks als

Zeitstempel festgehalten und der Beifahrer hat anschließend die Möglichkeit, die Beschreibung als Freitext ohne Zeitdruck einzutragen.

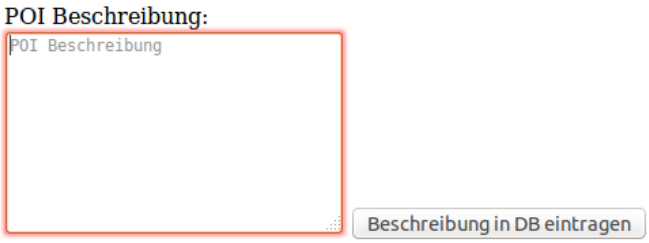

Abbildung 6.5: POI: Freitextfeld für die Beschreibung eines POIs

Nach dem Eintragen des Inhalts gelangt man wieder zur Darstellung während der Fahrt (siehe Abbildung 6.4).

Daten exportieren

Beim Exportieren der Daten einer Testfahrt kann der Nutzer eine oder mehrere Fahrten auswählen. Zusätzlich kann er auswählen, welche Tabellen exportiert werden sollen. Es ist also unter anderem möglich, in einem Schritt aus mehreren Fahrten jeweils alle Tabellen, in denen CAN-Pakete aufgezeichnet worden sind, zu exportieren. Dazu sind in der Übersicht die Fahrten auszuwählen und anschließend "Fahrten exportieren“ zu klicken. Daraufhin erscheint als Auswahlformular für die Tabellen:

Abbildung 6.6: Auswahl der Tabellen zum Export und Auswahl des Formats

Der eigentliche Exportvorgang startet nachdem “Export starten“ angeklickt worden ist. Je nach Länge der Fahrt und Menge der zu exportierenden Daten variiert die Dauer des Exportvorgangs. Eine genauere Analyse der Last, die während des Exports entsteht, findet man im Kapitel Qualitätssicherung unter dem Punkt Lasttest (7.1.2). Das Ergebnis des Exports ist stets ein Archiv pro Fahrt im Export Ordner, benannt nach dem Muster *FahrtId-Exportdatum-Exportzeit.zip*.

6.2 Anpassungen

Bei Veränderungen am Tabellenschema oder der Konfiguration des Servers muss gegebenenfalls auch dieses Programm angepasst werden.

6.2.1 Absolute Pfadangaben

Der Postgres-Befehl *COPY* erfordert einen absoluten Datenpfad als Parameter. Um eine höchstmögliche Kompatibilität zu erreichen, wird dafür das */tmp*-Verzeichnis genutzt, da dieses Verzeichnis auf allen Linuxsystemen, von allen Nutzern beschrieben werden darf. Der *COPY*-Befehl wird im Modul db_export.php in den Exportmethoden genutzt.

6.2.2 Hard-Coded-Values

Tabellennamen mussten teilweise direkt in die Datenbankabfragen codiert werden. Sollten Änderungen an dem Tabellenschema, im Speziellen an den Namen der Tabellen erfolgen oder sollten weitere Tabellen hinzugefügt werden, müssen die Datenbankabfragen in den Skripten dahingehend angepasst werden. Betroffen davon ist das Modul *db_export*.

7 Qualitätssicherung

7.1 Tests

Da innerhalb der Skripte auf Klassen und Methoden verzichtet worden ist, konnten keine automatisierten Komponententests durchgeführt werden.

7.1.1 Zeitmessung in PHP-Skripten

Um die Bearbeitungsdauer von PHP-Skripten zu messen, wird die Funktion *microtime()* genutzt. Mit dem Parameter true gibt diese einen float-Wert zurück, mit dem die Zeit auf eine Mikrosekunde genau bestimmt werden kann.

```
1 $time = -microtime(true);       //Negativen Startzeitpunkt speichern
2 foobar();                       //zumessender Codeabschnitt
3 $time += microtime(true);       //Differenz errechnen
```

Listing 7.1: Zeitmessung mit nur einer Variablen

Die Funktion ist ab PHP 5 verfügbar. Zudem muss das System dem Systemaufruf *gettimeofday()* unterstützen.

7.1.2 Lasttest

Um die Last, die das Programm beim Ausführen erzeugt, überprüfen zu können, wurden Tests durchgeführt, bei denen eine Vielzahl von Datenbankoperationen provoziert wurde. Um die Tests vergleichbar zu machen und den Aufwand gleichzeitig überschaubar zu halten, wurde eine "Testfahrt“ gewählt, die eine Minute dauert. Durch den Zeitraum von einer Minute lassen sich die Ergebnisse gut hochrechnen, während die kurze Ausführungszeit eine Vielzahl von Testdurchläufen ermöglicht. Die Testdaten, die in der Datenbank gespeichert waren, wurden während einer Fahrt am Tag bei bedecktem Himmel im öffentlichen Straßenverkehr aufgezeichnet. Das Fahrzeug war so konfiguriert, dass alle drei möglichen CAN-Busse über den Ethernetbackbone geroutet und dementsprechend aufgezeichnet wurden. Zusätzlich konnten die Daten der Laserscanner geloggt werden. Die Kameras lieferten zu dem Zeitpunkt keine

Bilder. Das Abbild dieser Daten hat eine unkomprimierte Größe von 43,7 MB und entspricht ca. 155000 Zeilen in der Datenbank.

Um die Testergebnisse einordnen zu können, wurden die gleichen Messungen auch auf dem Entwicklungssystem durchgeführt. Hierbei handelt es sich um einen Laptop mit Intel i7 Prozessor (i7-3630QM), acht GB Arbeitsspeicher und einer 128 GB SSD. Das Zielsystem ist im Leerlauf durch das Aufzeichnen von Ethernetpakten zu ca. 20 Prozent ausgelastet.

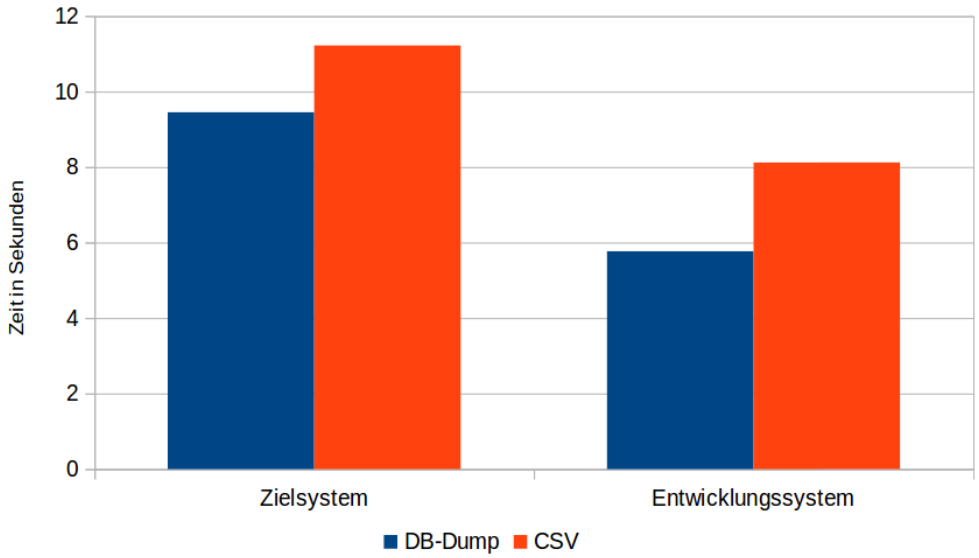

Abbildung 7.1: Zeit für den Export der Daten einer Fahrminute

Um Schwankungen durch unterschiedliche Lastsituationen auszugleichen wurden jeweils zehn Messungen durchgeführt. Die Werte in der Grafik sind die Durchschnittswerte dieser Messungen. Anhand der Grafik wird deutlich, dass der Exportvorgang auf dem Entwicklungssystem schneller abläuft. Zudem wird deutlich, dass der Export via CSV-Dateien auf beiden Systemen länger dauert als der Export als DB-Dump. Der generelle Unterschied zwischen den Systemen war zu erwarten und erklärt sich durch die unterschiedliche Leistung der Prozessoren und der schnelleren SSD im Entwicklungssystem. Dass der Export als CSV-Datei hingegen länger dauert, widerspricht der Erwartung.

Die Auflistung wesentlicher Abläufe beim CSV-Export und der Erstellung eines DB-Dumps ergibt folgende zwei Schemata:

1) Ablauf CSV-Export:

erstelle CSV-Datei per COPY -> füge CSV zip Archiv hinzu -> kopiere Archiv in Export Ordner

2) Ablauf DB-Dump:
leere ExportDB -> erstelle temp-CSV-Datei per COPY -> fülle ExportDB mit temp-csv-Datei per COPY -> erstelle Dump -> komprimiere dump -> kopiere Archiv in Export Ordner

Der Export als DB-Dump ist also wesentlich aufwendiger und erfordert mehr Schritte. Der Unterschied muss also innerhalb der Umsetzung der einzelnen Schritte liegen. Ein Blick auf die Komprimierungsfunktion erklärt den Unterschied. Die CSV-Dateien werden innerhalb des Skripts unmittelbar nach ihrer Erstellung mit der PHP-Funktion *zip->addFile()* zu einem Archiv hinzugefügt. Das tatsächliche Erstellen des Archivs und das Komprimieren der Dateien geschieht mit dem Aufruf *zip->close()*. Zeitmessungen im Ablauf des Skripts haben ergeben, dass dieser Vorgang ca. 90 Prozent der gesamten Bearbeitungszeit ausmacht.

Der Datenbankdump wird nach dem Erzeugen mit dem Programm tar zu einem Archiv hinzugefügt und anschließend per gzip komprimiert. Beide Programme werden per *system()*-Aufruf gestartet. Hier wird ein wesentlicher Performanceunterschied deutlich, der gegen die Verwendung von der in PHP integrierten ZIP-Methode spricht. In Zukunft sollten alle Komprimierungsvorgänge mit der Kombination aus tar und gzip durchgeführt werden.

7.2 Performanceanalyse

Neben der Funktionalität und Zuverlässigkeit ist der Komfort beim Benutzen einer Anwendung für den Nutzer ein wesentlicher Aspekt. Nicht nur der Nutzer profitiert von einem schnell reagierenden Programm, auch das System, auf dem es läuft, kann durch optimierte Software besser ausgenutzt werden. Bei Anwendungstests konnte die im Folgenden beschriebene Schwachstelle gefunden werden und durch eine optimierte Methode ersetzt werden.

Wenn CSV als Format für den Datenexport aus der Datenbank gewählt wird, entsteht für jede Tabelle eine CSV-Datei. Das Schreiben dieser Dateien erfolgte in der ersten Version folgendermaßen:

```
1 while ($line_array = pg_fetch_array($rs, null, PGSQL_ASSOC)) {
2         $line_to_write = implode(";", $line_array)."\r\n";
3         file_put_contents($filename, $line_to_write, FILE_APPEND);
4 }
```

Listing 7.2: Erstellen der CSV-Datei in einer while-Schleife

Dabei erfolgen in Zeile 1 und 2 zwei Arrayzugriffe innerhalb der PHP-Variablen $rs und $line_array, die aus Sicht der Performance zu vernachlässigen sind. In Zeile 3 jedoch erzeugt der Aufruf *file_put_contents()* ingesamt drei Dateizugriffe, die von dem Betriebssystem ausgeführt werden müssen. *file_put_contents()* wird durch PHP mit den Systembefehlen *fopen()*, *fwrite()*

und *fclose()* realisiert. Da es sich hierbei um Datenzugriffe außerhalb des Arbeitsspeichers handelt, entstehen erhebliche zeitliche Verzögerungen. Gerade bei den CAN-Tabellen, in denen pro Minute teilweise über 140000 Zeilen gespeichert werden, kommt es beim Exportieren so zu Wartezeiten und einer nicht hinzunehmenden Auslastung des gesamten Systems.

Der Vorgang konnte durch das Nutzen des postgres-Befehls COPY mit der Option FORMAT CSV erheblich beschleunigt werden:

```
1 $result = pg_query($connection,"COPY (SELECT * FROM $table
2           WHERE condition WITH (FORMAT CSV)");
```

Listing 7.3: Erstellen der CSV-Datei mit COPY

Nachdem die Beschleunigung der Methode zuerst subjektiv festgestellt wurde, ergaben die Messungen der Laufzeit folgende konkrete Differenzen:

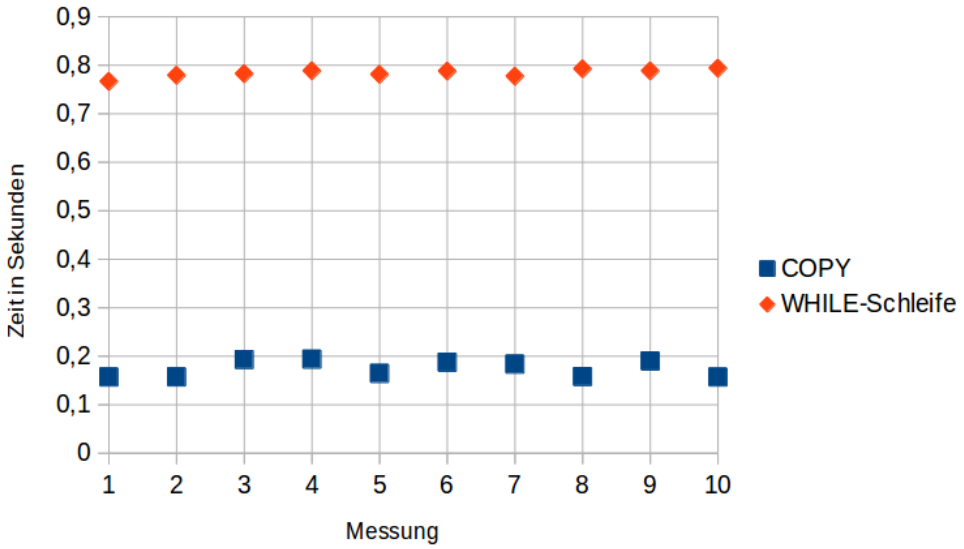

Abbildung 7.2: Vergleich der Bearbeitungszeiten

Es wurden zum vergleichenden Test 100000 Zeilen aus der Tabelle can_2 exportiert. Um Schwankungen während der Messungen auszugleichen, wurde der Test zehnmal wiederholt. Die Bearbeitungszeit per while-Schleife dauerte durchschnittlich 0.7836 Sekunden. Der gleich Export mit dem optimierten COPY-Befehl wurde in durchschnittlich 0.1742 Sekunden abgearbeitet. Der Vorgang konnte also nahezu um den Faktor 4 beschleunigt werden.

Usability

Um die Benutzerfreundlichkeit zu erhöhen, wird beim Exportieren eine progressbar angezeigt. So kann bei zeitintensiven Anfragen, zum Beispiel dem Erstellen eines größeren Datenbankdumps, der Fortschritt angezeigt werden und der Nutzer erkennt, dass das System nicht abgestürzt ist. Dadurch und durch die Verwendung von Bootstrap, um ein geräteübergreifend einheitliches Aussehen zu erzeugen, wird die Usability der Anwendung erhöht. Es soll sichergestellt werden, dass die Motivation des Nutzers zur tatsächlichen Verwendung des Programms erhalten bleibt. Die Erfahrungen, die während der Nutzung hinsichtlich des Zeitverhaltens und der Komplextität der Oberfläche gemacht werden, haben maßgeblich darauf Einfluss, ob ein Programm akzeptiert und genutzt wird. Schnelle Reaktionen und ein einheitliches Aussehen tragen hierzu bei. Sollte ein Vorgang unerwartet viel Zeit in Anspruch nehmen, muss der Nutzer darüber in Kenntniss gesetzt werden. Nur so ist zu verhindern, dass der Vorgang für gescheitert angesehen und das System unterbrochen wird.

Archive

Sämtliche Exporte werden in Archiven bereitgestellt. Bei der Exportoption "csv“ werden die einzelnen CSV-Dateien zu einer Datei zusammengefasst und sind dadurch leichter zu kopieren und herunterzuladen. Der Datenbankdump wird komprimiert, da eine Kompressionsrate von bis zu 80 Prozent erreicht wird. Hierzu wird das Programm gzip mit den Voreinstellungen verwendet. Für den Grad der Komprimierung bedeutet das Stufe fünf auf einer Skala von eins bis neun[1]. Beim Kopieren wird das Netzwerk somit deutlich weniger belastet und es ist ein deutlich schnellerer Download möglich.

[1] https://wiki.ubuntuusers.de/gzip/

8 Evaluierung

8.1 Fazit

Mit der entstandenen Software ist es in Zukunft möglich, die Daten der Testfahrten im Prototypenfahrzeug zu sortieren, sie zu archivieren und aus der Datenbank zu exportieren. Mit der grafischen Oberfläche wurde eine Benutzerschnittstelle geschaffen, die es ermöglicht, während der Fahrt Zeiträume zu definieren und bei besonderen Vorkommnissen Points of Interests zu setzen. Damit können diese im Nachhinein schnell wieder gefunden und bei der Auswertung besonders berücksichtigt werden. Die zusätzlich zu erfassenden Metadaten ermöglichen eine präzisere Kategorisierung der Messdaten. Durch das Nutzen von Webtechniken ist eine hohe Kompatibilität mit Endgeräten erreicht worden, sodass eine langfristige Nutzung mit unterschiedlichen Systemen gewährleistet ist. Bei konsequenter Benutzung kann das Programm die Auswertung der Kommunikationsdaten deutlich vereinfachen. Mit Hilfe der durchgeführten Tests konnte nachgewiesen werden, dass die Anwendung die an sie gestellten Anforderungen erfüllt.

8.2 Ausblick: Optimierungsmöglichkeiten der Anwendung

Trotz ausführlicher Analyse sind im Laufe der Entwicklung noch weitere Möglichkeiten der Verbesserung deutlich geworden, die nicht mehr berücksichtigt werden konnten. Die folgende Liste zeigt die Grenzen der Anwendung und soll als Ansatz für zukünftige Optimierungen dienen.

- Die Daten, die zurzeit keiner Fahrt zugeordnet werden, existieren in der Datenbank, können aber derzeit nicht exportiert oder gelöscht werden. Um ein Überlaufen der Datebank zu verhindern, müssen diese Daten regelmäßig manuell gelöscht werden. Ansatz hier könnte ein Mechanismus sein, der diese Daten automatisch löscht oder die Datenaufzeichnung außerhalb der Testfahrten deaktiviert. Kurzfristige Abhilfe würde die Möglichkeit schaffen, die gesamte Datenbank über das Webinterface löschen zu können.

- Das Feld “Wetter“ in der Tabelle Metadaten ist ein Freitextfeld. Um dieses Kriterium in Zukunft besser vergleichen zu können, sollte die Möglichkeit Freitext einzutragen durch eine Auswahl von Witterungen ersetzt werden. Kurzfristig sollte sich die Eingabe an einen noch zu bestimmenden Katalog halten. Denkbar wäre eine Festlegung auf folgende Möglichkeiten: bedeckt, sonnig, Nebel, Regen, Schnee, Glatteis, Nacht, Dämmerung...
- Sollte das Prototypenfahrzeug in Zukunft über das Internet erreichbar sein, muss die Anwendung hinsichtlich der Sicherheit als Webanwendung geprüft werden (siehe Abschnitt Sicherheitsaspekte der PHP Programmierung.).
- Die Exportarchive, die im Export Ordner bereitgestellt werden, müssen derzeit noch manuell gelöscht werden. Ein Lösungsansatz ist es, die Archive nur temporär zu erzeugen und nach dem Erstellen als Download anzubieten.
- Die Anwendung wurde nicht bezüglich mehrerer gleichzeitiger Zugriffe von verschiedenen Nutzern getestet, da das zur Zeit kein realistisches Anwendungsszenario ist.
- Es ließ sich nicht an allen Stellen vermeiden, Tabellennamen direkt in den Code zu schreiben. Diese hard-coded-values müssen bei Änderungen am Tabellenschema derzeit also von Hand angepasst werden. Nötig wurde das zum Beispiel beim Leeren von Tabellen, da hier aktuell Abhängigkeiten innerhalb der Tabellen berücksichtigt werden müssen. Automatisiertes Leeren aller Tabellen eines Schemas wäre möglich, wenn ON DELETE CASCADE beim Erstellen der Fremdschlüssel hinzugefügt wird. [1]

[1] https://www.postgresql.org/message-id/1104436476.26809.143.camel@linus

Literaturverzeichnis

[CoRE Research Group] CoRE Research Group: *CoRE Research Group Website*. CoRE Research Group. – URL http://core.informatik.haw-hamburg.de/en/. – Zugriffsdatum: 2016-05

[Frank 2010] Frank, Prof. Dr.-Ing. H.: *Vorlesungsmanuskript Ethernet*. 2010. – URL https://www.hs-heilbronn.de/1749557/Ethernet. – Zugriffsdatum: 2016-05

[Kopetz u. a. 2005] Kopetz, Hermann ; Ademaj, Astrit ; Grillinger, Petr ; Steinhammer, Klaus: The Time-Triggered Ethernet (TTE) Design. In: *8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC), Seattle, Washington* (2005), May.

[Meyer 2014] Meyer, Philipp: EthernetLogger - Dokumentation. (2014), September.

[PostgreSQL-Global-Development-Group] PostgreSQL-Global-Development-Group, The: *Online Dokumentation*. – URL http://www.postgres.org. – Zugriffsdatum: 2016-05

[Steinbach 2014] Steinbach, Till: Demo: Real-time Ethernet In-Car Backbones: First Insights into an Automotive Prototype. (2014). – URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=7013331. – Zugriffsdatum: 2016-05

[Steiner 2008] Steiner, Wilfried: *TTEthernet Specification*. TTTech Computertechnik AG. November 2008. – URL http://www.tttech.com

[Ubuntuusers.de] Ubuntuusers.de: *Online Dokumentation - Rechte*. – URL https://wiki.ubuntuusers.de/Rechte/. – Zugriffsdatum: 2016-05

[Wikipedia.de] Wikipedia.de: *Bootrstrap (Framework)*. – URL https://de.wikipedia.org/wiki/Bootstrap_%28Framework%29. – Zugriffsdatum: 2016-05

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 14. Juli 2016 Tim-Ole Schaffeld