DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

Konzeption und Entwicklung eines Moduls für die Generierung von V2X Nachrichten aus OpenSCENARIO Daten als Teil einer HiL Laborumgebung

Bachelorarbeit

für die Prüfung zum

Bachelor of Science

des Studiengangs Informationstechnik

der Dualen Hochschule Baden-Württemberg Mannheim

von

Katharina Hartmann

Abgabe am 14.09.2020

Bearbeitungszeitraum:	22.06.2020 - 13.09.2020
Matrikelnummer, Kurs:	7132570, TINF17ITIN
Abteilung:	Testen
Ausbildungsfirma:	Deutsches Zentrum für Luft- und Raumfahrt e.V.
Betreuer der Ausbildungsfirma:	Dipl. Ing. Lennart Asbach
Gutachter der Dualen Hochschule:	Prof. Dr. Heinz Jürgen Müller

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem

Thema

Konzeption und Entwicklung eines Moduls für die Generierung von V2X Nachrichten aus OpenSCENARIO Daten als Teil einer HiL Laborumgebung

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.*

* falls beide Fassungen gefordert sind

Braunschweig, den 11. September 2020

Zusammenfassung

Hochautomatisierte und vernetzte Fahrfunktionen sind ein wesentlicher Bestandteil der Mobilität der Zukunft. Die umfangreiche Erprobung der zugrundeliegenden Advanced Driver Assistance Systems (ADAS) nimmt im Zulassungsprozess solcher „intelligenten“ Fahrzeuge eine tragende Rolle ein und wird aus Risiko- und Kostengründen zunehmend in Simulationsumgebungen statt im realen Fahrbetrieb durchgeführt. Das Dateiformat OpenSCENARIO in Kombination mit dem OpenDRIVE Standard hat sich in den letzten Jahren als Quasi-Standard bei der Simulation von komplexen, dynamischen Verkehrsszenarien etabliert. Die OpenSCENARIO Definition eines Verkehrsszenarios umfasst die Beschreibung der Infrastruktur und der Fahrzeugbewegungen enthaltener Akteure, sodass diese innerhalb einer Simulationsumgebung abgebildet werden können. Die Simulation von Vehicle to Everything (V2X) Nachrichten, die Fahrzeuge mit vernetzten Fahrfunktionen versenden und empfangen können, aus OpenSCENARIO Daten ist bislang lediglich theoretisch betrachtet worden. Inhalt dieser Arbeit ist die Generierung von V2X Nachrichten aus in OpenSCENARIO definierten Verkehrsszenarien. Dies erfolgt durch die Integration eines OpenSCENARIO Parsers und einer Cohda MK5 OBU, ausgestattet mit dem Vanetza Protokoll Stack, in die Hardware in the Loop (HiL) Laborumgebung des Instituts für Verkehrssystemtechnik des DLR in Braunschweig. Die Werkzeugkette, die das Einlesen eines Verkehrsszenarios, die Ermittlung dynamischer Fahrzeuginformationen und die Aussendung von CAMs und DENMs für sämtliche simulierten Fahrzeuge umfasst, wurde mit mehreren Verkehrsszenarien erfolgreich getestet. Die generierten V2X Nachrichten bilden die im Verkehrsszenario definierte Fahrweise ab und konnten mit einem Tablet zur Validierung von V2X Nachrichten (WaveBEE touch) empfangen werden.

Abstract

Highly automated and networked driving functions are an essential component of future mobility. Extensive testing of the underlying Advanced Driver Assistance Systems (ADAS) plays a major role in the process of approval of such „intelligent“ vehicles and is increasingly being carried out in simulation environments instead of real driving, both for risk and cost reasons. In recent years, the file format OpenSCENARIO in combination with the OpenDRIVE standard has established itself as a quasi-standard in the simulation of complex, dynamic traffic scenarios. The OpenSCENARIO definition of a traffic scenario includes the description of the infrastructure and the vehicle movements of the actors involved, so that they can be represented within a simulation environment. The simulation of Vehicle to Everything (V2X) messages that vehicles with networked driving functions can send and receive from OpenSCENARIO data have so far only been considered theoretically. Subject of this thesis is the generation of V2X Messages from traffic scenarios defined in OpenSCENARIO. This is done by integrating an OpenSCENARIO parser and a Cohda MK5 OBU, equipped with the Vanetza protocol stack, into the Hardware in the Loop (HiL) laboratory environment of the Institute of Transportation Systems of the DLR in Braunschweig. The tool chain, which includes reading a traffic scenario, determining dynamic vehicle information and sending CAMs and DENMs for all simulated vehicles, was successfully tested with several traffic scenarios. The generated V2X messages represent the driving style defined in the traffic scenario and could be received with a mobile V2X validation system (WaveBEE touch).

Inhaltsverzeichnis

Abbildungsverzeichnis vii

Tabellenverzeichnis ix

Listings x

Anhang xi

Abkürzungsverzeichnis xii

1. Einleitung 1

2. Grundlagen der V2X Kommunikation 4

2.1. Überblick 5

2.1.1. Standardisierung der V2X Kommunikation 5

2.1.2. V2X Protokoll Stack 7

2.1.3. Nachrichtenformate 9

2.1.4. Bestehende Implementierungen 11

2.2. Themenbezogene Arbeiten zur V2X Kommunikation 13

3. Grundlagen der szenariobasierten Verkehrssimulation 18

3.1. Definition Szenario 18

3.2. Datengrundlage 21

3.2.1. OpenDRIVE 21

3.2.2. OpenCRG 21

3.2.3. OpenSCENARIO 22

3.3. Themenbezogene Arbeiten zum OpenSCENARIO Standard 26

4. Konzeption zur Generierung von V2X Nachrichten aus OpenSCENARIO Daten 29
4.1. HiL Laborumgebung des Instituts für Verkehrssystemtechnik des DLR 29
4.2. Existierende Ansätze zur Kombination des OpenSCENARIO Standards und der V2X Kommunikation 33
4.3. Anforderungen 35
4.4. Architektur 37

5. Entwicklung des OpenSCENARIO Parsers 43
5.1. Implementierung des OpenSCENARIO Parsers 43
5.2. Integration des OpenSCENARIO Parsers in die HiL Laborumgebung 50

6. Entwicklung des V2X Moduls 52
6.1. Eingesetzte Soft- und Hardwarekomponenten für das V2X Modul . . 52
6.2. Integration des V2X Moduls in die HiL Laborumgebung 54
6.3. Anwendung zur Generierung von V2X Nachrichten 56

7. Validierung 58
7.1. OpenSCENARIO Parser 59
7.2. Generierung der V2X Nachrichten 63
7.3. Bewertung der gesamten Werkzeugkette 66

8. Schlussbetrachtung 68
8.1. Fazit 68
8.2. Ausblick 69

Literatur 71

Abbildungsverzeichnis

2.1. Protokoll Stack einer ITS Station nach [21] 7

3.1. Definierte Elemente innerhalb eines Verkehrsszenarios 19

4.1. SEFEV - Schematische Darstellung 31
4.2. Konzeptionierte Architektur - Schematische Darstellung 37
4.3. Ablauf innerhalb des OpenSCENARIO Parsers 38

5.1. Grafische Oberfläche des OpenSCENARIO Parsers 44
5.2. Visualisierung der Berechnung einer Position auf einer Trajektorie . . 50
5.3. Darstellung einer CAM in einer Testsequenz 51
5.4. Darstellung einer DENM in einer Testsequenz 51

6.1. Aufbau des V2X Moduls 54

7.1. Benötigte Zeit zur Erstellung der Testsequenz in Abhängigkeit von der Szenariodauer 61
7.2. Simulationszeit in Abhängigkeit von übersprungenen Simulationsschritten (Zeitschritt: 0.1) 62

A.1. Struktur des Init-Elements in OpenSCENARIO 77
A.2. Struktur des Story-Elements in OpenSCENARIO 78
A.3. Popup Fenster zur Festlegung der initialen globalen Position 79
A.4. Popup Fenster zur Festlegung des zeitlichen Abstands zweier Simulationsschritte 80
A.5. Format der Tabelle *SpeedProfile* 80
A.6. Format der Tabelle *TC_Step* 81

A.7. Cohda MK5 OBU [28] 82
A.8. Empfangene V2X Nachrichten in der WaveBee touch 83

B.1. Grundlegende Klassenstruktur innerhalb des OpenSCENARIO Parsers 84
B.2. Klassendiagramm der `PrivateAction` 85
B.3. Klassendiagramm der `Condition` 86
B.4. Klassendiagramm der `Position` 87

C.1. Parameter eines BTP-`DataRequests` [22] 88

D.1. Übertragene DENM vom Vanetza Stack zum ITS G5 Modem der Cohda MK5 OBU - analysiert mit Wireshark 111
D.2. Übertragene CAM vom Vanetza Stack zum ITS G5 Modem der Cohda MK5 OBU - analysiert mit Wireshark 112
D.3. Mit der WaveBee touch empfangene DENM - analysiert mit Wireshark 113
D.4. Mit der WaveBee touch empfangene CAM - analysiert mit Wireshark 114

Tabellenverzeichnis

4.1. Konzipierte Informationsquellen der Simulation im Vergleich zu IPG CarMaker [39] 40

Listings

5.1. OpenSCENARIO `UserDefinedAction` mit enthaltener DENM 47

Anhang

Anhang A: Bilder 76

Anhang B: Klassendiagramme 84

Anhang C: ETSI Referenzen 88

Anhang D: Testdurchführung 93

D.1. Anleitung 93
D.1.1. SEFEV 93
D.1.2. OpenSCENARIO Parser 94
D.1.3. V2X Modul in Cohda MK5 OBU 94
D.1.4. Start der Simulation 94
D.2. OpenSCENARIO Testszenarien 95
D.3. Validierungsergebnisse 110

Abkürzungsverzeichnis

ADAS	**A**dvanced **D**river **A**ssistance **S**ystems
ASN.1	**A**bstract **S**yntax **N**otation **One**
BTM	**B**alise **T**ransmission **M**odule
BTP	**B**asic **T**ransport **P**rotocol
C2C-CC	**C**ar **2** **C**ar **C**ommunication **C**onsortium
CAM	**C**ooperative **A**wareness **M**essage
CAN	**C**ontroller **A**rea **N**etwork
DENM	**D**ecentralized **e**nvironmental **N**otification **M**essage
DLR	**D**eutsches Zentrum für **L**uft- und **R**aumfahrt e.V.
DSRC	**D**edicated **S**hort **R**ange **C**ommunication
ETSI	**E**uropean **T**elecommunication **S**tandards **I**nstitute
ERTMS	**E**uropean **R**ail **T**raffic **M**anagement **S**ystem
GNSS	**G**lobales **N**avigations**s**atellite**n**system
GUI	**G**raphical **U**ser **I**nterface
HiL	**H**ardware **i**n the **L**oop
HMI	**H**uman **M**achine **I**nterface
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
ITS	**I**ntelligent **T**ransportation **S**ystem

IVIM	**I**nfrastructure to **V**ehicle **I**nformation
MAPEM	**MAP** Data
OBU	**O**n-**B**oard-**U**nit
pcap	**p**acket **cap**ture
RailSiTe	**Rail**way **Si**mulation and **Te**sting
RSU	**R**oad**S**ide **U**nit
RTM	**R**adio **T**ransmission **M**odule
SEFEV	**S**imulation **E**nvironment **f**or **E**RTMS **V**erification
SPATEM	**S**ignal **P**hase **a**nd **T**iming
SREM	**S**ignal **R**equest **M**essage
SSEM	**S**ignal **S**tatus **M**essage
SUT	**S**ystem **u**nder **T**est
TCP	**T**ransmission **C**ontrol **P**rotocol
TDL	**T**rain **D**ynamics and **L**ogic
TEP	**T**est **E**vent **P**layer
V2X	**V**ehicle **to Everything**
WAVE	**W**ireless **A**ccess in **V**ehicular **E**nvironments
XML	**Ex**tensible **M**arkup **L**anguage

1. Einleitung

Automatisierung und Vernetzung stellen heutzutage einen zunehmend relevanter werdenden Bestandteil des Verkehrsgeschehens dar. Grund für die schnelle Ausbreitung automatisierter und vernetzter Fahrfunktionen ist die Erwartung, dass mit vorausschauenden, intelligenten Fahrzeugen die Sicherheit und Effizienz im Straßenverkehr erhöht wird.[35]

Innerhalb der Entwicklung hochautomatisierter und vernetzter Fahrzeuge liegt ein besonderer Fokus auf dem Zulassungsprozess, der durch leistungsfähige Testmethoden und ganzheitliche Werkzeugketten beschleunigt werden soll. Ein wesentlicher Aspekt der Zulassung hochautomatisierter und vernetzter Fahrzeuge ist demnach das umfangreiche und gleichzeitig effiziente Testen der Fahrfunktionen auf Zuverlässigkeit und Sicherheit. Fahrzeugfunktionen, die eine Hochautomatisierung oder Kommunikation ermöglichen, genannt **A**dvanced **D**river **A**ssistance **S**ystems (ADAS), werden zunehmend in Simulations- oder Laborumgebungen anstelle von Realfahrten getestet. Das simulationsbasierte Testen, beispielsweise als **M**odel **i**n the **L**oop (MiL) oder **S**oftware **i**n the **L**oop (SiL) bis hin zu **H**ardware **i**n the **L**oop (HiL), der Erprobung auf Komponentenebene des Systems, bietet eine schnelle und kostengünstigere Möglichkeit der Erprobung, die zudem mit einem geringeren Risiko für Fahrer und Fahrzeug verbunden ist. Darüber hinaus ermöglichen Simulationsumgebungen eine exakte Reproduzierbarkeit der durchgeführten Tests und das spezifische Testen auf Fehler, beispielsweise durch vorsätzlich falsche Eingaben an Schnittstellen.

Die Methode simulationsbasierte Erprobungen als Teil einer Werkzeugkette, vorgeschaltet vor Realfahrten auf Teststrecken und im öffentlichen Raum, zu begreifen, wird bereits in mehreren Arbeiten aus Industrie [39] und Forschung [23, 45, 50] behandelt. Zielsetzung ist es, verstärkt Aspekte des realen Verkehrsgeschehens in

die Simulation mit einzubeziehen, um die Erprobung mit möglichst realitätsnahen Szenarien durchführen zu können. Innerhalb der szenariobasierten Verkehrssimulation hat sich das 2016 entwickelte OpenSCENARIO Format im Zusammenspiel mit OpenDRIVE zur Beschreibung von Verkehrsszenarien als Quasi-Standard bei der Simulation von realitätsnahen, parametrisierbaren Verkehrssituationen etabliert.

Die Entwicklung und Erprobung hochautomatisierter Fahrfunktionen, als Teilbereich von ADAS, mit Szenarien im OpenSCENARIO Format erfolgt bereits in mehreren Simulationsumgebungen [4, 9, 11, 44]. Szenarien, die im OpenSCENARIO Format definiert sind, enthalten Verkehrsinformationen, die genutzt werden können, um die Sensorik eines Fahrzeugs, die beispielsweise den Abstand zu einem vorausfahrenden Fahrzeug erkennt, zu simulieren. Mittels simulierter Sensorinformationen können Assistenzfunktionen, hier z. B. der Notbremsassistent, validiert werden. Das Testen vernetzter Fahrfunktionen, vor allem der **V**ehicle **to Everything** (V2X) Kommunikation, bei der Fahrzeuge untereinander und mit der Infrastruktur kommunizieren können, mit einem in OpenSCENARIO definierten Verkehrsszenario wird bislang lediglich theoretisch [50] behandelt oder in Ansätzen [39] umgesetzt. Dies ist darin begründet, dass der OpenSCENARIO Standard in der aktuellen Version (1.0.0) keine Informationen über V2X Kommunikation respektive gesendete und empfangene Nachrichten von Fahrzeugen oder der Infrastruktur beinhaltet.[10]

Da die Kommunikation zwischen Fahrzeugen im zukünftigen Straßenverkehr eine zunehmend stärkere Rolle einnehmen wird, ist die Integration der V2X Kommunikation in Simulationsumgebungen für das Testen von ADAS unabdingbar. Durch die Kombination eines realitätsnahen, detaillierten Verkehrsszenarios, wie es durch den OpenSCENARIO Standard beschrieben werden kann, mit der Simulation der V2X Kommunikation der enthaltenen Fahrzeuge, kann eine umfassende Simulationsumgebung für die Erprobung von ADAS geschaffen werden.

Ziel dieser Bachelorarbeit ist daher die Generierung von V2X Nachrichten aus Verkehrsszenarien, die im OpenSCENARIO Format definiert sind. Zur Erreichung dieses Ziels sollen zwei Module in die Software **S**imulation **E**nvironment **f**or **E**RTMS **V**erification (SEFEV)[1] innerhalb des **Rail**way **Si**mulation and **Te**sting (RailSiTe)

[1]Das ERTMS ist ein System zur Steuerung des innereuropäischen Eisenbahnverkehrs.

Labors des Instituts für Verkehrssystemtechnik des **D**eutsches Zentrum für **L**uft- und **R**aumfahrt e.V. (DLR) integriert werden. Das erste Modul, genannt OpenSCENARIO Parser, dient dem Einlesen und Parsen einer OpenSCENARIO Datei und der Ermittlung benötigter Informationen aus dem in der Datei definierten Verkehrsszenario. Das zweite Modul, genannt V2X Modul, fungiert als Generator zur Erstellung von V2X Nachrichten aus den ermittelten Informationen des Verkehrsszenarios. Überdies soll das V2X Modul die generierten Nachrichten über das für die V2X Kommunikation standardisierte Protokoll **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers (IEEE) 802.11p [16] versenden, sodass Fahrzeuge, die mit vernetzten Funktionalitäten ausgestattet sind, die Nachrichten empfangen können.

Im Rahmen dieser Arbeit steht die Konzeptionierung der zu entwickelnden Module und die Entwicklung einer geeigneten Architektur für die Erweiterung der Software SEFEV im Vordergrund. Daher wird die Implementierung lediglich prototypisch für ausgewählte Szenarien und Funktionalitäten des OpenSCENARIO Standards umgesetzt. Diese Arbeit dient als Grundstein für die Entwicklung und Erprobung vernetzter Fahrfunktionen des Bereichs Automotive im RailSiTe des DLR.

Der Aufbau der Arbeit unterteilt sich in die Kapitel Grundlagen, themenbezogene Arbeiten, Konzeption, Umsetzung und Validierung. Zunächst werden die Grundlagen der V2X Kommunikation und der szenariobasierten Verkehrssimulation, mit besonderem Fokus auf das Datenformat OpenSCENARIO erläutert. Im Zuge dessen wird ein Überblick über einige ausgewählte Forschungsarbeiten, bezogen auf die Generierung von V2X Nachrichten und das Erproben von ADAS mit dem Datenformat OpenSCENARIO, gegeben. Anschließend folgt die Vorstellung bereits existierender Ansätze zur Kombination des OpenSCENARIO Standard und der V2X Kommunikation. Daraufhin wird die Konzeptionierung der Architektur der zu entwickelnden Module anhand der Vorgehensweise bzw. der Resultate der betrachteten Literatur dargelegt und nachfolgend die prototypische Umsetzung erläutert. Abschließend wird die Testdurchführung der Module beschrieben und die Entwicklung validiert.

2. Grundlagen der V2X Kommunikation

Die Mobilität der Zukunft zeichnet sich durch zunehmende Vernetzung und Automatisierung im Verkehrsgeschehen aus. Ein wesentlicher Bestandteil des vernetzten Fahrens ist die Möglichkeit des Fahrzeugs mit anderen Verkehrsteilnehmern oder der Infrastruktur zu kommunizieren (V2X Kommunikation). V2X Kommunikation ist ein Oberbegriff für verschiedene Kommunikationsmöglichkeiten innerhalb der vernetzten Mobilität. Teilaspekte der V2X Kommunikation sind die Kommunikation zwischen mehreren Fahrzeugen (**V**ehicle-**to**-**V**ehicle, V2V) und die Kommunikation zwischen einem Fahrzeug und der Infrastruktur (**V**ehicle-**to**-**I**nfrastructure, V2I). Zielsetzung der V2X Kommunikation ist die Erhöhung der allgemeinen Sicherheit und Effizienz im Straßenverkehr, die durch eine Echtzeitkommunikation der Verkehrsteilnehmer gewährleistet werden soll [35, 38]. Dadurch können beispielsweise Unfälle durch frühzeitige Gefahrenmeldungen vermieden werden. Die Effizienz des Verkehrs kann durch Kooperation der Fahrzeuge gesteigert werden, sodass z. B. dem Fahrer Umleitungen vorgeschlagen werden, wenn entlang seiner ursprünglichen Strecke ein erhöhtes Verkehrsaufkommen festgestellt wird. Die V2X Kommunikation stellt zudem einen relevanten Aspekt für die zunehmende Automatisierung der Fahrzeuge dar, da das hoch automatisierte Fahrzeug, zusätzlich zu der verbauten Sensorik, Informationen anderer Verkehrsteilnehmer erhält und so ein vorausschauendes Handeln ermöglicht wird.

In diesem Kapitel werden Grundlagen zur V2X Kommunikation erläutert und ausgewählte Arbeiten, die sich mit der Erprobung von ADAS durch die Simulation von V2X Kommunikation befassen, vorgestellt.

2.1. Überblick

Im Folgenden werden die Grundlagen der V2X Kommunikation vorgestellt. Der erste Abschnitt thematisiert die verschiedenen Ansätze der Standardisierung, die in der **E**uropäischen **U**nion (EU), den USA und Japan existieren. Anschließend werden auf Basis des europäischen Standards der V2X Protokoll Stack und die unterschiedlichen V2X Nachrichtenformate erläutert. Zuletzt folgt die Vorstellung bestehender Implementierungen des V2X Protokoll Stacks bzw. vollständiger V2X Simulationsframeworks, die im Rahmen dieser Arbeit für die Entwicklung des V2X Moduls in Betracht gezogen werden.

2.1.1. Standardisierung der V2X Kommunikation

Die Standardisierung der V2X Kommunikation basiert ursprünglich auf dem 2010 veröffentlichten Standard IEEE 802.11p [16], der eine Erweiterung der WLAN Standards innerhalb der IEEE 802.11 Norm darstellt. Der IEEE 802.11p Standard ist der Grundstein für die **D**edicated **S**hort **R**ange **C**ommunication (DSRC), welche die Kommunikation von Fahrzeugen ohne ein Mitwirken des Fahrers beschreibt und einen Aspekt der V2X Kommunikation darstellt. Neben dem entwickelten 802.11p Standard existiert der **C**ellular-**V2X** (C-V2X) Standard, der 2016 innerhalb des **3rd G**eneration **P**artnership **P**rojects[1] (3GPP) entwickelt wurde und auf Mobilfunk basiert [42]. Im Gegensatz zu anderen Standards der 802.11 Familie ermöglicht der 802.11p Standard eine Ad-Hoc Kommunikation, bei der die Kommunikationsteilnehmer eine Kommunikation ohne Beteiligung einer zentralen Steuerungseinheit aufbauen können [2, 21]. Da das DLR vorrangig die V2X Kommunikation mit dem IEEE 802.11p Standard erforscht, fokussiert sich diese Arbeit auf die V2X Kommunikation im Bereich der DSRC.

Die Entwicklungen der Erforschung und Standardisierung der V2X Technologie im Rahmen der DSRC werden hauptsächlich in der EU, Japan und den USA vorangetrieben [38].

[1] Eine Kooperation von Gremien für die Standardisierung im Mobilfunk, beispielsweise 5G.

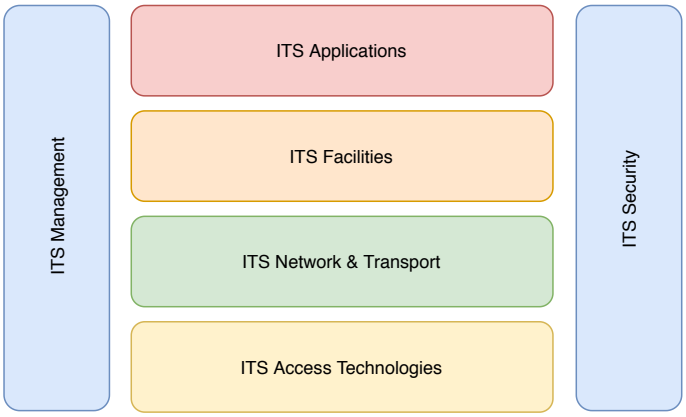
In den USA sind Automobilhersteller und das Verkehrsministerium die Hauptakteure innerhalb der V2X Kommunikation. Darüber hinaus existieren Konsortien der Industrie, welche in einzelnen Projekten mitwirken. Für die V2X Kommunikation in den USA wurden bereits 1999 75 MHz im 5,9 GHz Frequenzband reserviert. Das Übertragungsprotokoll basiert auf dem **W**ireless **A**ccess in **V**ehicular **E**nvironments (WAVE) Standard, einer Kombination aus dem IEEE 802.11p und der IEEE 1609 Standard [17]. In Japan ist die V2X Technologie aus dem bestehenden Mautsystem, welches auf ARIB STD-T55 [7] basiert, hervorgegangen. Die benötigte Infrastruktur ist bereits vorhanden, wodurch V2I Kommunikation in weiten Teilen schon unterstützt wird. Die treibende Kraft der Entwicklung und Standardisierung stellen in Japan ebenfalls die zuständigen Ministerien dar. Es existiert ein modifizierter WAVE Standard für Kommunikation im 5,8 GHz Bereich. Überdies ist ein neuer Standard im 700 MHz Bereich in Planung. Ein weiterer Standard ist das **V**ehicle **I**nformation and **C**ommunication **S**ystem (VICS), das in Japan bereits umfangreich im Straßenverkehr eingesetzt wird.[38] In Europa sind das European Commitee for Standardization (CEN)[2] und das **E**uropean **T**elecommunication **S**tandards **I**nstitute (ETSI)[3] für die Entwicklung und den Fortschritt der V2X Technologie verantwortlich. Die Standardisierung wird vom **C**ar **2** **C**ar **C**ommunication **C**onsortium (C2C-CC)[4], einem Zusammenschluss von Autoherstellern und Forschungsinstituten, begleitet. Das zugrundeliegende System der V2X Kommunikation wird als **I**ntelligent **T**ransportation **S**ystem (ITS) bezeichnet. Der ETSI ITS G5 Standard bezeichnet die Umsetzung des 802.11p Protokolls für den europäischen Raum. Für die V2X Kommunikation ist eine Bandbreite von bis zu 70 MHz mit einer Mittelfrequenz von 5,9 GHz reserviert.[2, 21]

Im Rahmen dieser Arbeit wird für die V2X Kommunikation der ITS G5 Standard der ETSI betrachtet, da das Institut für Verkehrssystemtechnik des DLR hauptsächlich innereuropäisch mit der V2X Technologie arbeitet.

Ein vollständiges V2X Kommunikationssystem nach ETSI besteht zum einen aus Fahrzeugen, die mit einer sogenannten **O**n-**B**oard-**U**nit (OBU) zur V2X Kommunikation befähigt werden. Durch die OBUs wird das Senden und Empfangen von V2X

[2] `standards.cen.eu`
[3] `etsi.org`
[4] `car-2-car.org`

Abbildung 2.1.: Protokoll Stack einer ITS Station nach [21]



Nachrichten ermöglicht. Ergänzend zu OBUs existieren **R**oad**S**ide **U**nits (RSUs), auch **R**oad**S**ide **E**quipment (RSE) genannt. RSUs können Nachrichten von Fahrzeugen erhalten und diese weiter verteilen. Überdies können RSUs Informationen an Fahrzeuge, beispielsweise Geschwindigkeitsbegrenzungen oder eine nahende Baustelle, übermitteln. Eine beispielhafte RSU ist eine Lichtsignalanlage, die den aktuellen Signalzustand übermitteln kann.

2.1.2. V2X Protokoll Stack

Die ITS Referenzarchitektur bzw. der V2X Protokoll Stack orientiert sich an dem ISO/OSI-Referenzmodell[5] und ist in vier horizontale Protokollschichten und zwei vertikale Einheiten unterteilt. In Abbildung 2.1 ist der für ETSI ITS Stationen definierte Protokoll Stack abgebildet. Die horizontalen Schichten sind:

ITS Access Technologies Diese Schicht beschreibt die Zugangstechnologien für die Kommunikation zwischen mehreren ITS Stationen auf physischer Ebene. Die Beschreibung umfasst sowohl für interne als auch für externe Kommunikation enthaltene Kommunikationsmedien und die dazugehörigen Übertragungsprotokolle. Die Zugangsschicht ist nicht auf eine Kommunikationstechnologie

[5]Das ISO/OSI Referenzmodell ist ein Modell für Netzwerkprotokolle in Schichtenarchitektur.

festgelegt, sondern kann mit verschiedenen, i.d.R. drahtlosen, Kommunikationstechnologien umgehen. Die relevanteste Zugangstechnologie ist ETSI ITS G5. Die Zugangsschicht teilt sich auf in die physische Schicht (physical layer) und die Datenübertragungsschicht (data link layer).[2]

ITS Network & Transport Innerhalb der Netzwerkschicht (network layer) wird die Kommunikation zwischen mehreren ITS Stationen bzw. die Verbreitung von Nachrichten gesteuert. Die Netzwerkschicht umfasst drei Schemata für die Verbreitungsarten von Nachrichten, die als Broadcast gesendet werden. Mit dem (i) eventbasierten geografischen Broadcast werden Nachrichten an alle ITS Stationen innerhalb eines bestimmten geografischen Gebiets übertragen. Das Gebiet kann sowohl direkt um den Sender herum als auch weiter entfernt gelegen sein. Die Verbreitung der Nachricht erfolgt effizient und zuverlässig, sodass sichergestellt ist, dass sämtliche Stationen im definierten Gebiet die Nachricht erhalten. Die Effizienz der Verbreitung wird dadurch erreicht, dass Nachrichten zunächst an eine Station innerhalb des definierten Gebiets übermittelt und von dort aus weiter verbreitet werden. Die Transportschicht (transport layer) umfasst verschiedene Dienste, die der Übertragung und der Sicherheit von Daten dienen. Darüber hinaus können in der Transportschicht Daten der Applikationsschicht gesammelt, verpackt und an die Applikationsschicht eines Empfängers übertragen werden.[2]

Die Netzwerk- und Transportschicht umfassen u. a. das Basic Transport Protocol, **I**nternet **P**rotocol **v**ersion **6** (IPv6) und das GeoNetworking[6] Protocol.

ITS Facilities Diese Ebene dient der Unterstützung und Verwaltung von Applikationen und bietet Datenstrukturen für die Speicherung und Aggregation von unterschiedlichen Daten. Daten, die innerhalb dieser Schicht verwaltet werden können, sind beispielsweise Sensordaten des Fahrzeugs oder Kommunikationsdaten. Überdies werden in dieser Ebene externe und interne Softwaremodule verwaltet. Dies beinhaltet das Suchen und Herunterladen von Diensten und die Integration dieser in die ITS Station.[21]

[6]Das GeoNetworking dient der geografischen Adressierung und dem Routing bei der Kommunikation zwischen benachbarten ITS Stationen.

ITS Applications Die Applikationsschicht dient der Verwaltung und Prozessierung von Anwendungsdaten. Ein Aspekt der Verwaltung ist das Steuern des Sendens bzw. Empfangens von Nachrichten und die Weiterleitung dieser an Applikationen. Applikationen umfassen sowohl Anwendungen zur Interaktion mit dem Fahrer (**H**uman **M**achine **I**nterface) als auch Fahrzeugsensorik, die typischerweise über ein Bussystem angesprochen wird.[2] Anwendungen können der Sicherheit oder Effizienz im Verkehr sowie dem Infotainment dienen [21].

2.1.3. Nachrichtenformate

Innerhalb der V2X Kommunikation sind von der ETSI für verschiedene Anwendungsfälle Nachrichtenformate definiert, die im Folgenden vorgestellt werden. Sämtliche Nachrichtenformate werden mit **A**bstract **S**yntax **N**otation **One** (ASN.1) codiert.

Cooperative Awareness Message (CAM) Eine CAM beinhaltet allgemeine Fahrzeuginformationen des sendenden Verkehrsteilnehmers, die über Broadcast mit weiteren Fahrzeugen in der näheren Umgebung geteilt werden können. Als nähere Umgebung, respektive als Kommunikationsbereich der CAM, wird ein Umkreis von 1 km angegeben. Dies entspricht der maximalen Reichweite der V2X Kommunikation im 5,9 GHz Bereich [19]. Eine CAM enthält beispielsweise Informationen über Fahrzeugposition, -typ, -status und -geschwindigkeit und wird von ITS jede $0{,}1 - 1$ s gesendet. Die Rate der Aussendung ist abhängig von der Fahrzeugdynamik.[2] Der Empfänger kann durch die erhaltenen Informationen ein Modell seiner Umgebung erzeugen, das beispielsweise für die Bewertung der Wahrscheinlichkeit einer Kollision mit einem anderen Fahrzeug verwendet werden kann. Die ASN.1 Spezifikation einer CAM ist in Anhang C [19] dargestellt. Das erste Byte der Nachricht ist für den `stationType` reserviert, der beschreibt welchem Typ Verkehrsteilnehmer der Sender entspricht. Beispielhafte Arten von Verkehrsteilnehmern sind (1) Fußgänger, (5) Auto, (6) Bus oder (11) Tram. Im Anschluss daran wird die aktuelle Position des Verkehrsteilnehmers und die `generationDeltaTime`, die den Zeitpunkt der Nachrichtengenerierung enthält, angegeben. Weitere Parameter, die eine CAM

beinhaltet, sind u. a. die Fahrrichtung des Fahrzeugs, die Geschwindigkeit und Beschleunigung sowie die Länge und Breite des Fahrzeugs.[19]

Decentralized environmental Notification Message (DENM) Eine DENM wird hauptsächlich für das Warnen anderer Verkehrsteilnehmer vor Ereignissen im Straßenverkehr, wie einer Unfall- oder Baustelle, verwendet. Entdeckt beispielsweise eine „Cooperative Road Hazard“ (RHW) Applikation innerhalb einer ITS Station ein Ereignis, das eine Warnung auslösen sollte, wird zunächst innerhalb der Applikation nach einem vordefinierten, dem Ereignis ähnlichen Use-Case gesucht. Auf Basis des ermittelten Use-Cases wird eine DENM mit spezifischen Informationen generiert und versendet. Die DENM wird anschließend an sämtliche Verkehrsteilnehmer in einer vom Use-Case abhängigen Umgebung weitergeleitet. Auf der Empfängerseite wird die erhaltene DENM verarbeitet und, wenn nötig, dem Fahrer über das **H**uman **M**achine **I**nterface (HMI) als Warnung angezeigt. Das Versenden und Verteilen der DENM wird solange wiederholt bis entweder das Ereignis endet oder eine spezifische DENM erhalten wird, die das Ende des Ereignisses verkündet.

Der Aufbau einer DENM teilt sich in die Management-, die Situations- und die Orts (location) Sektion auf (siehe ASN.1 Spezifikation in Anhang C [20]). Die Management Sektion beinhaltet Informationen über die Wiederholungsfrequenz des Versendens, die Version der Daten, welche der Aktualisierung eines Ereignisses dient und die Verlässlichkeit eines Ereignisses. Die Verlässlichkeit gibt die Wahrscheinlichkeit an, ob eine Information wahr ist und demnach das Ereignis existiert. Die zweite Sektion der Nachricht enthält Informationen, die das entdeckte Ereignis betreffen. Dies umfasst beispielsweise den Grund für das Auftreten des Ereignisses (*CauseCode*, *SubCauseCode*) und die direkten Auswirkungen auf den Straßenverkehr. In der Orts Sektion wird die Position des Events und der Bereich im Straßenverkehr, in dem die DENM verteilt werden soll, beschrieben.[20]

Neben den vorgestellten Nachrichtenformaten CAM und DENM existieren weitere Nachrichtenformate, die jedoch im Rahmen dieser Arbeit nicht verwendet werden.

Dies umfasst die **S**ignal **P**hase **a**nd **T**iming (SPATEM)[7], in der Informationen über den Status und Signalseiten von Lichtsignalanlagen beschrieben sind. In der **MAP** Data (MAPEM)[7] sind Informationen über die Topologie einer Straße oder Kreuzung und in einer **I**nfrastructure to **V**ehicle **I**nformation (IVIM)[7] Straßensignale, wie beispielsweise Straßenschilder mit Geschwindigkeitsbegrenzungen, enthalten. Die Nachrichtenformate **S**ignal **R**equest **M**essage (SREM)[7] und **S**ignal **S**tatus **M**essage (SSEM)[7] beschreiben Anfragen über Signaländerungen und den Status der getätigten Anfrage. SSEMs und SREMs werden beispielsweise im Zusammenhang mit der Bevorrechtigung von Einsatzfahrzeugen verwendet.

Zusätzlich zur tatsächlichen Nachricht, wie einer CAM oder DENM, werden Informationen der Geonetworking- und Facilitiesebene mit versendet.

2.1.4. Bestehende Implementierungen

Es existieren bereits verschiedene softwareseitige Implementierungen des V2X Stack, von denen ausgewählte im Folgenden vorgestellt werden:

Commsignia V2X Software Stack basiert auf der vorgestellten Spezifikation der ETSI. Der Commsignia Stack bietet ein simples Framework, welches für Linux-basierte und RTOS[8] Betriebssysteme zur Verfügung steht.[6]

Cohda Wireless stellt einen V2X Stack zur Verfügung, welcher in der Industrie weit verbreitet ist. Der Stack unterstützt den IEEE 802.11p Standard (WAVE und ITS G5) und wird in den verfügbaren Produkten MK5 OBU [28] und MK5 RSU mitgeliefert.[5] Innerhalb des DLR werden Technologien von Cohda Wireless in mehreren Projekten[9] eingesetzt.

Nordsys waveBEE ist eine Produktfamilie, die von eingebetteten V2X Systemen bis zu gesamten V2X Simulationsplattformen reicht. Innerhalb der waveBEE Reihe

[7] Der Anhang -EM / -M steht für *extended message*. Dies beschreibt die Ergänzung der in den USA standardisierten Nachrichtenformate um einen zusätzlichen Header.[18]
[8] RTOS ist ein Echtzeitbetriebssystem in der elektronischen Datenverarbeitung.
[9] DLR MAVEN: `dlr.de/content/de/video/2019/video-projekt-maven.html`
Testfeld Niedersachsen: `verkehrsforschung.dlr.de/de/projekte/testfeld-niedersachsen-fuer-automatisierte-und-vernetzte-mobilitaet`

ist der V2X Software Stack enthalten. Dieser wird in sämtlichen Produkten der waveBEE Familie verwendet.[29]

CCS Labs OpenC2X Stack ist eine frei verfügbare Entwicklungs- und Prototypingplattform für Anwendungen im V2X Bereich. Der Software Stack bietet eine weitestgehende Unterstützung des ETSI ITS-G5 Standards und ist für Linux Betriebssysteme verfügbar. Die Zugangstechnologie (access layer) wird mit Hilfe eines modifizierten Linux 3.18 Kernels und einer Ath9k[10]-basierten WLAN Karte umgesetzt, sodass der 802.11p Standard unterstützt wird. Der OpenC2X Stack ist innerhalb der einzelnen Ebenen der V2X Stacks modular aufgebaut, sodass beispielsweise für die Funktionalität DENM und CAM einzelne Module existieren. Die Module kommunizieren über ZeroMQ[11] und können unabhängig voneinander gestartet bzw. gestoppt werden.[24]

Vanetza ist eine frei verfügbare Implementierung der Protokolle Geonetworking und **B**asic **T**ransport **P**rotocol (BTP) des V2X Protokoll Stacks und ermöglicht die Kommunikation mit IEEE 802.11p fähigen Hardwaremodulen. Das Framework unterstützt sowohl die Netzwerk- und Transportebene als auch die Facilities Ebene und die Module Security und Management der ITS Referenzarchitektur. Vanetza ist in C++ geschrieben und ist nicht als alleinstehende Anwendung zu betrachten, sondern kann im Zusammenhang mit einem zusätzlichen Simulationsframework oder zugrundeliegenden Hardware Komponenten verwendet werden. [43]

Neben den vorgestellten Implementierungen des V2X Stacks existieren vollständige V2X Simulations Frameworks, die nachfolgend vorgestellt werden.

Veins ist ein Open Source Framework für die Simulation von Fahrzeugnetzwerken. Das Framework basiert auf OMNeT++[12], einer komponentenbasierten C++ Simulationsbibliothek, die speziell für eventbasierte Netzwerksimulationen ent-

[10] Ath9k ist ein Open Source Treiber, der Atheros 802.11n PCI/PCI-E (WLAN) Chips unterstützt.
[11] ZeroMQ ist eine Bibliothek für asynchronen Nachrichtenaustausch, die speziell für die gleichzeitige Ausführung von Funktionen in (verteilten) Systemen entwickelt wurde. (`zeromq.org`)
[12] `omnetpp.org`

wickelt wurde, und der mikroskopischen[13] Verkehrssimulationssoftware SUMO [26]. Neben SUMO und OMNeT++ sind die Software MiXiM[14], für die Modellierung der physischen Ebene von Kommunikationsstacks, und die Bibliothek TraCI [46] als Schnittstelle zu SUMO integriert. Veins bietet detaillierte Simulationen der V2X Kommunikation über den 802.11p Standard. Es wird sowohl der WAVE Standard als auch die DSRC unterstützt. Da SUMO auf Karten des Open Source Projekts OpenStreetMap[15] für die Verkehrssimulation zurückgreift, sind die in Veins enthaltenen Verkehrsszenarien ebenfalls ein Produkt von OpenStreetMap. [40]

dSPACE ist ein deutsches Unternehmen, das eine kommerzielle V2X Lösung anbietet. Die Bereitstellung der V2X Kommunikation erfolgt über die Integration eines bestehenden V2X Stacks, i.d.R. Cohda MK5 OBU, mittels eines V2X Hardware Adapters. Die V2X Lösung ermöglicht die Visualisierung und den vereinfachten Zugriff auf Inhalte der V2X Kommunikation durch ein bereit gestelltes Matlab Simulink Toolset[16] und eine Visualisierungsoberfläche. Überdies sind ein ASN.1 Encoder und Decoder für die V2X Nachrichten integriert. Die V2X Lösung kann in Verbindung mit dem HiL Testsystem dSPACE SCALEXIO für den Test von V2X Komponenten eingesetzt werden. Das Erstellen eines Testszenarios für die Erprobung von V2X Komponenten bzw. ADAS wird mit einem in der Software enthaltenen Szenario Editor umgesetzt. [12]

2.2. Themenbezogene Arbeiten zur V2X Kommunikation

In der Literatur existieren eine Vielzahl von Arbeiten, die sich mit der Generierung bzw. Simulation von V2X Nachrichten oder mit dem Test von V2X Kommunikationskomponenten befassen. Nachfolgend wird zunächst auf Arbeiten eingegangen,

[13] Der Begriff mikroskopisch bedeutet, dass jedes Element (Fahrzeug, Fußgänger) in der Simulation abgebildet wird.
[14] `mixim.sourceforge.net`
[15] `wiki.openstreetmap.org/wiki`
[16] `de.mathworks.com/products/simulink.html`

die sich mit der Generierung von V2X Nachrichten befassen und demnach eine Grundlage für das im Rahmen dieser Arbeit zu entwickelnde V2X Modul darstellen. Anschließend werden Arbeiten aufgeführt, die sich mit der Erprobung von V2X Kommunikationskomponenten auseinandersetzen.

In der Arbeit von Riebl et al. [36] wird das Simulationsframework Artery vorgestellt. Artery wurde 2015 als Erweiterung von Veins für die V2X Kommunikation nach ETSI ITS G5 entwickelt und beinhaltet demnach für die Verkehrssimulation die Software SUMO und den eventbasierten Netzwerksimulator OMNeT++. Aufbauend auf den genannten Komponenten wird der Vanetza V2X Stack integriert, der die Kommunikation über ETSI ITS G5 gewährleistet. Vanetza kommuniziert sowohl mit den Softwarekomponenten von Veins, um Informationen der Verkehrssimulation zu erhalten als auch mit Applikationen, die im Anwendungslayer des V2X Stacks zu verorten sind. Das Simulationsframework Artery kann zum aktuellen Zeitpunkt die Nachrichtenformate CAM und DENM nach ETSI ITS G5 versenden. Sowohl die Umwandlung von Nachrichten im ASN.1 Format als auch von OMNeT++ spezifischen Nachrichten ist möglich. Das Versenden von Nachrichten wird anhand eines intern definierten Szenarios umgesetzt. Das interne Szenarioformat ist ähnlich zum OpenSCENARIO Standard aufgebaut. Es existiert jedoch keine Möglichkeit zur Umwandlung von Szenarien im OpenSCENARIO Format in das interne Szenarioformat. Zusammenfassend kann Artery die V2X Kommunikation für den ITS G5 Standard mittels der Verkehrssimulationssoftware SUMO und einem internen Szenarioformat simulieren. Im Rahmen dieser Arbeit ist eine Verwendung von Artery innerhalb des V2X Moduls denkbar. Ein entscheidender Nachteil ist jedoch, dass Artery die Simulation von Szenarien im OpenSCENARIO Format nicht unterstützt. Zudem weist Artery einen Überhang an nicht benötigten Funktionalitäten auf.

Eine weitere Arbeit, die im Jahr 2019 entstanden ist, befasst sich mit der Verkehrssimulation innerhalb eines HiL-Steuergerätetests [34]. Reinold et al. stellen in ihrer Arbeit ein Konzept für die Simulation von V2X Nachrichten in einem HiL-Prüfstand im Rahmen des Projekts „Hy-Nets“ vor. Das Projekt „Hy-Nets“ befasst sich mit der Fragestellung, ob der Energiebedarf von Fahrzeugen mit Hybridantrieb durch intelligente Algorithmen, die V2X Informationen nutzen, gesenkt werden kann. Das

in der Arbeit Reinold et al. vorgestellte System basiert auf zwei Komponenten. Einerseits wird die Umgebung durch Veins und das **E**go **V**ehicle **I**nterface (EVI) [3], einer dedizierten Softwarekomponente von Veins, simuliert. Die zweite Komponente umfasst die Simulation des Ego-Fahrzeugs und des nahen Umfelds sowie die HiL-Anbindung an einen realen Verbrennungs- bzw. Elektromotor, die durch die dSPACE SCALEXIO Software umgesetzt wird. Eine wesentliche Herausforderung in diesem Schritt ist die Umwandlung von Straßen, die in SUMO definiert sind, in Straßen im ASM Format. Die Problematik der unterschiedlichen Straßenformate lösen Reinold et al. durch die Verwendung des standardisierten Dateiformates OpenDRIVE, das sowohl SUMO als auch dSPACE SCALEXIO verarbeiten können. Das Ego-Fahrzeug (dSPACE V2X Lösung) und die Umgebungsfahrzeuge (Veins) kommunizieren anschließend über V2X Nachrichten. Je nach erhaltener Information werden der angebundene Verbrennungs- bzw. Elektromotor angesprochen. Die Arbeit von Reinold et al. ist ähnlich zu der Idee dieser Bachelorarbeit, da ebenfalls V2X Kommunikationskomponenten innerhalb einer HiL Umgebung getestet werden. Der Ansatz, Informationen des Ego-Fahrzeugs und der benachbarten Fahrzeuge durch die globale Verkehrssimulation zu berechnen ist, bedingt durch die Szenariodefinition im OpenSCENARIO Format, die ein Ego-Fahrzeug und Non-Ego-Fahrzeuge umfasst, nicht notwendig. Die Integration von Veins oder der V2X Lösung von dSPACE ist ähnlich zu dem zuvor beschriebenen V2X Simulationsframework Artery möglich, weist aber den Nachteil der fehlenden Integration von Szenarien im OpenSCENARIO Format auf.

Szendrei et al. stellen in ihrer Arbeit ein zu [34] ähnliches HiL Simulationsframework für das Testen von hochautomatisierten Fahrfunktionen vor, das ebenfalls auf SUMO basiert [41]. Das System besteht aus mehreren On-Board Units mit integrierter Benutzerschnittstelle, einer GPS-Simulationsinstanz, der Verkehrssimulationssoftware SUMO und einem sogenannten Orchestrator, der die Verwaltung der Komponenten übernimmt. Die simulierten Fahrzeuginformationen von SUMO werden wie in den vorgestellten Arbeiten [34, 36] mit Hilfe der Software TraCI zur Verfügung gestellt. Die Kommunikation zwischen TraCI und der V2X Komponenten erfolgt über das Datenformat JSON. Als Stack für die Bereitstellung der V2X Kommunikation wird Commsignia OB2 verwendet. Mit dem von Szendrei et al. vorgestellten HiL

V2X Simulationsframework können reale V2X Anwendungen unter der Verwendung mehrerer OBUs entwickelt und erprobt werden. Im Gegensatz zu Szenadrei et al. wird im Rahmen dieser Bachelorarbeit durch die Benutzung des OpenSCENARIO Standards keine zusätzliche Simulationsinstanz für GPS Informationen und keine Verkehrssimulationssoftware benötigt. Die Verwendung von JSON als Dateiformat zur Übertragung der Fahrzeuginformationen zu den V2X Komponenten ist für die Kommunikation zwischen der HiL Laborumgebung des DLR und dem V2X Modul innerhalb dieser Bachelorarbeit in Betracht zu ziehen.

Lee et al. beschreiben in [25] einen bisher völlig neuen Ansatz für die Generierung von V2X Nachrichten durch die Integration eines Moduls für die V2X Kommunikation in die Open Source Simulationsplattform CARLA [9]. CARLA ist ein Fahr- und Verkehrssimulator für die Entwicklung, Erprobung und Validierung von ADAS. Zielsetzung der Arbeit ist die Entwicklung eines V2X Simulationsframeworks zur Generierung von V2X Datensätzen für verschiedene Verkehrsszenarien. Innerhalb der Arbeit werden ein V2X Kommunikationsmodul, das die Kommunikation eines Fahrzeugs über den 802.11p Standard unterstützt, und ein Modul zur Generierung von V2X Datensätzen aus den simulierten Fahrzeuginformationen entwickelt und an CARLA angebunden. Das entwickelte Modul zu Generierung von V2X Daten erhält Informationen des Image und Sensormoduls und generiert aus diesen V2X Nachrichten, die zwischen zwei oder mehreren Fahrzeugen (Vehicle-to-Vehicle) ausgetauscht werden können. Als Anwendungsfall für die entwickelten Module stellen Lee et al. eine im Ego-Fahrzeug integrierte **A**ugmented **R**eality (AR) Anwendung vor, die Sensorinformationen benachbarter Fahrzeuge erkennbar machen soll. Im Rahmen dieser Bachelorarbeit ist die Implementierung und die Integration eines V2X Moduls in eine bestehende Simulationsumgebung, ähnlich zu dem in der Arbeit von Lee et al. präsentierten Ansatz, geplant. Ein Modul, das generierte V2X Datensätze für eine Auswertung speichert ist, ebenso wie eine Visualisierung der V2X Informationen, nicht Teil der Arbeit, kann jedoch als Bestandteil nachfolgender Arbeiten interessant sein.

Zwei Arbeiten, die unterschiedliche Testmethodiken für die Erprobung der V2X Kommunikation thematisieren, sind die Arbeiten von De Ponte Müller et al. [31] und

Wang et al. [45]. De Ponte Müller, vom Institut für Kommunikation und Navigation des DLR, erforscht sich in Zusammenarbeit mit der Universität in Malaga die Konzeptionierung und Durchführung von Tests spezifisch für einzelne Layer des V2X Stacks. De Ponte Müller et al. präsentieren Erprobungsmethodiken für die physische, die Netzwerk- und die Applikationsschicht und setzen diese für konkrete Testfälle um. Die vorgestellte Arbeit von de Ponte Müller et al. ist im Rahmen dieser Bachelorarbeit insofern relevant, als dass sie ein Konzept für das Testen der Funktionsweise des zu entwickelnden V2X Moduls bereitstellt. Wang et al. führen das Thema Erprobung von V2X Kommunikationskomponenten fort und geben in ihrer Arbeit eine Zusammenfassung von bestehenden Testmethoden für die V2X Kommunikation. Als Herausforderung bei der Erprobung von V2X Komponenten auf Anwendungsebene stellen sie heraus, dass verschiedene Applikationen unterschiedliche Anforderungen an beispielsweise die Latenz oder Durchsatz der Übertragung oder die Verlässlichkeit der Nachrichteninformation haben. Da innerhalb dieser Bachelorarbeit lediglich die Funktionalität des Sendens und Empfangens einer CAM bzw. DENM betrachtet wird, ist anzunehmen, dass die genannte Herausforderung hier keine Relevanz hat. Wang et al. erläutern darüber hinaus, dass für einen umfangreichen Test sämtlicher Felder des V2X Standards eine hohe Anzahl an Testfällen benötigt wird. Als Lösung wird das Testen durch virtuelle Funktionstest mit möglichst realistischen Szenarien vorgeschlagen.

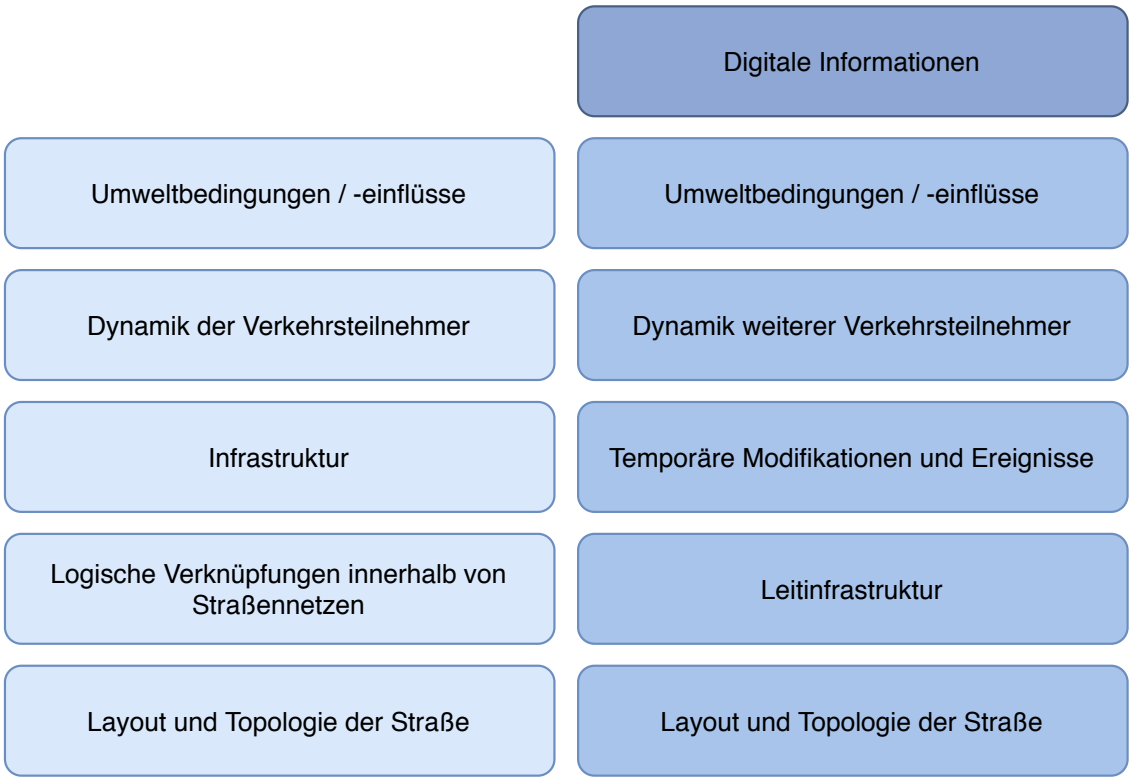
Diese Bachelorarbeit führt den Vorschlag virtuelle Tests mit möglichst realitätsnahen Szenarien umzusetzen weiter, indem das detaillierte und standardisierte Dateiformat OpenSCENARIO zur Generierung von V2X Nachrichten verwendet wird.

3. Grundlagen der szenariobasierten Verkehrssimulation

Dieses Kapitel umfasst die im Rahmen dieser Arbeit relevanten Grundlagen der szenariobasierten Verkehrssimulation. Zunächst erfolgt die Definition des Begriffs *Szenario* innerhalb der Verkehrssimulation und dem Test von ADAS. Daraufhin werden Datenformate zur konkreten Repräsentation eines Verkehrsszenarios eingeführt. Zuletzt werden verwandte Arbeiten im Bereich szenariobasiertes Testen mit dem OpenSCENARIO Standard betrachtet.

3.1. Definition Szenario

Innerhalb der szenariobasierten Verkehrssimulation repräsentiert ein *Szenario* temporär eine Verkehrssituation und besteht aus den in Abbildung 3.1(a) dargestellten Elementen [15]. Die untersten zwei Ebenen beinhalten Informationen über die Topologie und die Logik des zugrundeliegenden Straßennetzes. In der dritten Ebene werden infrastrukturseitige Elemente, wie Lichtsignalanlagen oder Gebäude beschrieben. Die vierte Ebene beinhaltet die Dynamik der unterschiedlichen Verkehrsteilnehmer. In der fünften Ebene ist die Beschreibung von Wetter- bzw. Umweltbedingungen enthalten. Dies umfasst beispielsweise Informationen über die Wetterlage oder die Temperatur. Ein Szenario stellt dabei eine zeitliche Abfolge von Aktionen bzw. Änderungen dar, die innerhalb der definierten Umgebung in Abhängigkeit von Bedingungen ausgeführt werden.



(a) Elemente eines Szenarios nach [15]

(b) Elemente eines Szenarios nach [37] mit zusätzlicher Ebene

Abbildung 3.1.: Definierte Elemente innerhalb eines Verkehrsszenarios

Eine weiterführende Definition des Begriffs Szenario ist in [1] bzw. [37] aufgeführt. Je nach aktueller Phase des Testprozesses wird ein Szenario unterschiedlich definiert und enthält demnach verschiedene Inhalte: Die Konzeptionsphase des Testprozesses enthält in natürlicher Sprache beschriebene Szenarien mit einem hohen Grad an Abstraktion. Eine solche Definition, wie beispielsweise „zwei Fahrzeuge fahren im Regen hintereinander auf einer zweispurigen Straße gemäß der Geschwindigkeitsrichtlinien“ wird als funktionales Szenario bezeichnet. In der zweiten Phase, welche die Testgenerierung umfasst, erfolgt die Umwandlung von funktionalen Szenarien in logische Szenarien. In einem logischen Szenario werden abstrakte Erläuterungen, beispielsweise die Wetterbezeichnung „Regen“ als Parameterräume von Attributen dargestellt. Der Regen kann demnach z. B. durch eine Niederschlagsrate im Bereich von $20 - 100\,\mathrm{mm}$ angegeben werden. Die dritte und letzte Phase beinhaltet die Aus- bzw. Durchführung der Tests. Für das Testen müssen klar definierte und detaillierte Szenarien in einem festgelegten Datenformat vorliegen. Ein Szenario, das genaue Beschreibungen im Gegensatz zu Bereichen enthält und in einem definierten Format vorliegt, wird als konkretes Szenario bezeichnet. Aufgrund des verringerten Abstraktionsgrades nimmt die Anzahl an benötigten Szenarien im Laufe des Testprozesses stark zu.

Die vorangegangene Beschreibung der Elemente eines Szenarios (siehe Abbildung 3.1(a)) findet sich ebenfalls im Pegasus Projekt [1] wieder. Innerhalb des Projekts ist ein Beschreibungsmodell für Szenarien enthalten, das aus 5 Schichten besteht. Das Modell unterscheidet sich zu den in [15] genannten Punkten insofern, als dass die Aspekte Layout, Topologie und logische Verknüpfungen von Straßen zu einem Punkt zusammengefasst werden. Als eigenständiger Aspekt sind hier temporäre Veränderungen der Topologie, beispielsweise durch Baustellen, angeführt. Ergänzend zu dem fünfschichtigen Modell des Pegasus Projekts definiert [37] eine theoretische sechste Ebene, die digitale Informationen enthalten soll. Das Schichtmodell, welches sich aus den Überlegungen von [1] und [37] zusammensetzt, ist in Abbildung 3.1(b) dargestellt. Auf der sechsten Ebene können sämtliche Informationen der V2X Kommunikation sowie Informationen aus digitalen Karten beschrieben werden.

Die vorliegende Arbeit ist innerhalb der Ausführungsphase des Testprozesses einzu-

ordnen. Es existieren klar definierte Szenarien in einem festgelegten Datenformat, die Informationen der 5 definierten Ebenen beinhalten. V2X Informationen, die innerhalb der 6. Ebene beschrieben werden können, sind nicht im Szenario enthalten.

3.2. Datengrundlage

Im folgenden Kapitel werden die Datengrundlage bzw. einzelne Datenformate zur Repräsentation eines in 3.1 definierten Szenarios vorgestellt. Es wird sich dabei auf die in dieser Arbeit relevanten Datenformate, die als (Quasi-)Standard innerhalb der szenariobasierten Verkehrssimulation gelten, beschränkt.

3.2.1. OpenDRIVE

Das Dateiformat OpenDRIVE dient der logischen Beschreibung von Straßennetzen und wurde im Jahr 2006 von Vires Simulationtechnologie entwickelt. OpenDRIVE basiert auf dem **Ex**tensible **M**arkup **L**anguage (XML)-Standard und ist ebenfalls hierarchisch aufgebaut. Es beinhaltet eine analytische Beschreibung von Straßen, wie beispielsweise die Straßenbreite oder die Art der Straße. Darüber hinaus können Kreuzungen und weitere logische Verbindungen zwischen Straßen definiert werden. OpenDRIVE bildet demnach die ersten drei Ebenen der Elemente (siehe Kapitel 3.1) innerhalb eines Szenarios ab.

3.2.2. OpenCRG

Die Initative OpenCRG fokussiert sich auf die detaillierte Beschreibung von Straßenoberflächen. Innerhalb der Initative werden verschiedene Dateiformate und Werkzeuge für das Erstellen, Darstellen und Bewerten von Straßenbelägen zur Verfügung gestellt. OpenCRG kann innerhalb einer OpenDRIVE Spezifikation eines Straßennetzes verwendet werden. Die Beschreibung einer Straße mittels OpenCRG ist innerhalb der untersten Ebene einer Szenariodefinition (Layout und Topologie der Straße) zu verorten.

3.2.3. OpenSCENARIO

Der OpenSCENARIO Standard beinhaltet die Spezifikation und ein Dateiformat für die Beschreibung von dynamischen Inhalten in Verkehrssimulationen. OpenSCENARIO wurde initial innerhalb eines Industriekonsortiums um Vires Simulationstechnologie entwickelt und wurde 2018 zu ASAM e. V. transferiert. Die primäre Zielsetzung des Standards ist die Ergänzung der bestehenden Dateiformate OpenDRIVE und OpenCRG um dynamische Komponenten, sodass komplexe Szenarien mit mehreren beteiligten Fahrzeugen abgebildet werden können. Die Spezifikation des OpenSCENARIO Standards deckt die Ebenen vier (Dynamik der Verkehrsteilnehmer) und fünf (Umweltbedingungen / -einflüsse) einer Szenario Definition ab. Das OpenSCENARIO Dateiformat wird bei der Entwicklung, dem Test und der Validierung von ADAS genutzt [10]. Im Folgenden wird der Begriff OpenSCENARIO für die Beschreibung des im OpenSCENARIO Standard enthaltenen Dateiformat verwendet. Im Rahmen dieser Arbeit wird die Version 1.0.0 betrachtet, die im März 2020 veröffentlicht wurde.

OpenSCENARIO basiert, ähnlich zu OpenDRIVE, auf dem XML-Standard und ist hierarchisch strukturiert. Im OpenSCENARIO User Guide von ASAM [10] werden fünf grundlegende Konzepte von OpenSCENARIO dargelegt, die im Folgenden vorgestellt werden:

1. Das Straßennetz (`RoadNetwork`) beschreibt die statische Infrastruktur und wird innerhalb eines Szenarios mit Entitäten (`Entities`) beispielsweise Fahrzeugen (`Vehicles`) oder Fußgängern (`Pedestrians`) befüllt.
2. Sämtliche Manöver eines Szenarios werden innerhalb eines `StoryBoard`s definiert. Ein `StoryBoard` enthält mindestens ein Element des Typs `Story`. Die Struktur der Elemente innerhalb der `Story` ist vorgegeben und wird im Anschluss an die Vorstellung der Konzepte erläutert.
3. Aktionen (`Actions`) bzw. `Event`s sind mit Bedingungen für das Starten und Stoppen (`Conditions`) verknüpft. Ist die Startbedingung einer Aktion erfüllt, wird die Aktion ausgeführt.

4. Es existieren Kataloge (`Catalogs`), die definierte OpenSCENARIO Elemente beinhalten und der Wiederverwendbarkeit von Szenarien in verschiedenen Anwendungsfällen dienen.
5. Innerhalb eines Katalogs oder Szenarios können Parameter symbolisch definiert werden (`ParameterDeclaration`), um einen inhaltlichen Zusammenhang zu verdeutlichen.

Ein Szenario, das im OpenSCENARIO Dateiformat definiert ist, weist die folgende Struktur auf:

Zu Beginn wird der Header des Dokuments (`FileHeader`) aufgeführt, der das Erstelldatum, den Autor und eine kurze Beschreibung des Szenarios respektive den Namen des Szenarios beinhaltet. Anschließend folgen globale Parameter Deklarationen, die für das gesamte Dokument gelten. Nachfolgend werden Informationen bezüglich verwendeter Kataloge und deren Ablageort (`CatalogLocations`) und der Datenbasis des Straßennetzes, definiert in OpenDRIVE, gegeben.

Im nächsten Abschnitt werden die am Szenario beteiligten Entitäten definiert. Eine Entität wird durch ein `ScenarioObject` definiert und kann ein Fahrzeug, Fußgänger oder `MiscObject` sein. Ein `MiscObject` beschreibt beispielsweise ein Hindernis, ein Gebäude oder eine Verkehrsinsel. Für ein Fahrzeug können neben einer Referenz auf ein bereits existierendes Fahrzeug im Katalog Attribute wie Fahrzeug Kategorie (`vehicleCategory`), `Performance` oder Informationen zum Fahrer (`ObjectController`) angegeben werden.

Innerhalb des Elements `Init` werden die beteiligten Entitäten initialisiert, erhalten demnach eine Startposition und -geschwindigkeit im definierten Szenario. Mögliche Elemente, die im `Init` enthalten sein können, sind in Abbildung A.1 dargestellt. In der `Story` werden sämtliche Aktionen, die nach der Initialisierung beginnen, aufgeführt. Der hierarchische Aufbau der Elemente innerhalb der `Story` ist in Abbildung A.2 veranschaulicht. Eine `Story` kann, wie der Header, Parameter Deklarationen enthalten. Die Parameter, die innerhalb der `Story` definiert werden, gelten jedoch nur für Elemente innerhalb der `Story`. Die `Story` beinhaltet immer einen `Act` mit einer Gruppe von Manövern (`ManeuverGroup`) und einer Startbedingung (`StartTrigger`).

Innerhalb einer Manöver Gruppe werden zunächst die zugehörigen Entitäten definiert (`Actors`). Anschließend folgt die Beschreibung eines oder mehrerer Manöver (`Maneuver`), die jeweils aus einem oder mehreren `Events` bestehen. Ein Event kann eine oder mehrere Aktionen (`Actions`) und eine Startbedingung beinhalten.

Die Dynamik beteiligter Entitäten in einem Szenario wird im Wesentlichen durch Aktionen und mit diesen verknüpften Bedingungen beschrieben. Im Folgenden werden zunächst einige im Rahmen dieser Arbeit relevante Aktionen (siehe Abbildung A.1) und anschließend Bedingungen (siehe Abbildung A.2) erläutert.

Eine abstrakte Aktion kann in drei Aktionen aufgeteilt werden:

Global Action Eine globale Aktion umfasst sämtliche Aktionen, welche die Umwelt betreffen und keiner bestimmten Entität zuzuordnen sind. Zu einer globalen Aktion gehören beispielsweise `EnvironmentActions`, die das Wetter, Straßenverhältnisse und die Zeit beeinflussen, `EntityActions` für das Löschen oder Hinzufügen von Entitäten oder `InfrastructureActions` mit denen Verkehrssignale (Ampeln etc.) gesetzt bzw. verändert werden können.

User Defined Action Eine selbst definierte Aktion beinhaltet Aktionen, die nicht Teil des OpenSCENARIO Standards sind, sondern die ein Nutzer selbst erstellt hat. Innerhalb des Elements `UserDefinedAction` kann z. B. ein Kommando oder ein Skript enthalten sein, das spezifisch für eine Simulationsumgebung ausgeführt wird.

Private Action Ein Element des Typs `PrivateAction` gehört immer zu einer bestimmten Entität. Mit einer privaten Aktion kann die Position, laterale und longitudinale Bewegung sowie die Sichtbarkeit einer Entität beschrieben werden. Es existieren `LongitudinalActions`, welche die Geschwindigkeit oder die Distanz in Relation zu einem Objekt beeinflussen und `LateralActions`, welche die laterale Änderung der Position in absolutem oder relativem Bezug auf ein Objekt betreffen. Überdies existieren `SynchronizeActions`, `VisibilityActions` und `TeleportActions`. Eine `RoutingAction` spezifiziert eine Route, der eine Entität folgen soll. Für die Spezifikation einer Route gibt es die Möglichkeit Wegpunkte und eine Strategie zum Abfahren dieser, Vertices und Interpolation

oder eine Zielposition zu definieren. Wird lediglich eine Zielposition angegeben, soll die Entität den kürzesten Weg von der aktuellen zu angegebenen Position unter Beachtung des Straßennetzes wählen.

Sämtliche Aktionen und ihre übergeordneten Storyboard Elemente (Event, Manöver, Act) weisen einen Status vom Type `StandByState`, `RunningState` oder `CompleteState` während der Laufzeit eines Szenarios auf. Ist die Bedingung für ein Element noch nicht erfüllt, hat das Element den Status *StandBy*. Der Status ändert sich zu *Running*, wenn die Startbedingung des Elements erfüllt ist und die Inhalte des Elements ausgeführt werden. Sind sämtliche Aktionen innerhalb des Elements ausgeführt oder ist die Stoppbedingung erfüllt, wird der Status des Elements auf *Complete* gesetzt.

Die Start- und Stoppbedingungen der Aktionen und Manöver werden durch Trigger bestimmt. Ein Trigger beinhaltet eine Menge an Bedingungen und hat zu jedem Zeitpunkt innerhalb des Szenarios den Wert `True` oder `False`. Im Folgenden werden zunächst ausgewählte Bedingungen, die von einer Entität abhängig sind, vorgestellt. Ein Element des Typs `byEntityCondition` enthält Informationen über die Art der Bedingung und die Entitäten, von denen die Bedingung abhängt. Eine entitätsabhängige Bedingung kann sich auf eine bestimmte Entität oder auf sämtliche im Szenario beteiligten Entitäten beziehen.

ReachPosition Bedingung wird erfüllt, wenn beteiligte Entitäten eine festgelegte Position im Szenario erreicht haben.

Distance Die aktuelle Distanz zwischen zwei Entitäten wird mit einem gegebenen Wert verglichen.

Speed Die Geschwindigkeit der zugehörigen Entität wird mit einer definierten Geschwindigkeit verglichen.

TimeToCollision Die vorhergesagte Zeit einer Kollision der beteiligten Entitäten mit entweder einem weiteren Fahrzeug oder einer bestimmten Position im Szenario wird mit einem Wert verglichen.

EndOfRoad Bedingung ist erfüllt, wenn die zugehörige Entität das Ende der Straße für einen gewissen Zeitraum erreicht hat.

Die zweite Art von Bedingungen (`byValueCondition`) sind abhängig von einem bestimmten Parameter bzw. Wert innerhalb des Szenarios.

TimeOfDay Die simulierte Zeit und das simulierte Datum werden mit einem gegebenen Zeit- bzw. Datumswert verglichen.

SimulationTime Vergleicht die Simulationszeit mit einem festgelegten Wert.

StoryBoardElementState Bedingung ist erfüllt, wenn der Status eines Referenzelementes des `StoryBoards` einen bestimmten Wert aufweist. Der Status eines Elements in OpenSCENARIO kann den Wert `standbyState`, `runningState` und `completeState` haben.

UserDefinedValue Die Bedingung wird für die Definition externer, nicht in OpenSCENARIO enthaltenen Bedingungen verwendet. Ein selbst definierter Wert kann hier mit einem Referenzwert verglichen werden.

TrafficSignal Bedingung ist erfüllt, wenn ein `TrafficSignal`, beispielsweise eine Lichtsignalanlage einen bestimmten Status aufweisen. Die Definition eines `TrafficSignals` erfolgt i.d.R. innerhalb der zugehörigen OpenDRIVE Daten.

Das Vergleichen mehrerer Werte erfolgt durch den `RuleOperator`, der als Enum definiert ist und die Werte `greaterThan`, `lessThan` und `equalTo` annehmen kann.

3.3. Themenbezogene Arbeiten zum OpenSCENARIO Standard

Der OpenSCENARIO Standard wird seit seiner Entwicklung im Jahr 2017 an vielen Stellen in der Literatur genannt. Es existieren sowohl Arbeiten, welche die Generierung von Szenarien im OpenSCENARIO Format darlegen, als auch Arbeiten in denen die Simulation von Verkehrsszenarien im OpenSCENARIO Format behandelt wird.

Eine Arbeit, die einen Grundstein für die Ermittlung von Fahrzeuginformation aus einem in OpenSCENARIO definierten Szenario legt, ist von Heinz et al. [14] aus dem Jahr 2017. Die in Zusammenarbeit von AUDI und der technischen Hochschule Ingolstadt entstandene Arbeit befasst sich mit der Generierung von Trajektorien aus OpenSCENARIO Daten für das deterministische Testen von hochautomatisierten Fahrfunktionen. Zunächst teilen Heinz et al. den Aspekt Fahrzeugnavigation in drei aufeinanderfolgende Aspekte auf: (i) Die Routenplanung zur Ermittlung einer optimalen Route zwischen zwei gegebenen Punkten innerhalb eines Straßennetzes. (ii) Die Manöversortierung, die der Generierung von konkreten Verhaltensweisen durch das chronologische Sortieren von getriggerten Events dient. (iii) Die Generierung von Trajektorien aus den erhaltenen Information durch die Kombination einer Zeit- und Geometrieinformation. Ziel der Arbeit ist nicht die Generierung eines Polynoms, das eine Route für das gesamte Szenario beschreibt, sondern lediglich das Festhalten der Position der beteiligten Fahrzeuge in einer Wegpunktliste. Dies wird in [14] darin begründet, dass bereit fürs das Abbilden eine einzelnen Aktion vom Typ `LaneChange` ein Polynom 5. Grades vonnöten sei. Die Wegpunktliste, welche für ein Szenario generiert werden soll, enthält Zeit- und Positionsinformationen, sowie die Orientierung, Geschwindigkeit und Beschleunigung des Fahrzeugs. Heinz et al. implementieren das vorgeschlagene Konzept zur Generierung von Trajektorien prototypisch für ein Szenario, das auf OpenDRIVE Daten einer AUDI Teststrecke basiert und einen Spurwechsel (`LaneChangeAction`) und Beschleunigungen beinhaltet. Die Bewertung der generierten Trajektorien erfolgt mit dem Verkehrssimulator VTD [44]. Die Idee der Arbeit von Heinz et al. korreliert mit der Zielsetzung dieser Bachelorarbeit, da ebenfalls Fahrzeuginformationen aus OpenSCENARIO ermittelt werden, um eine deterministische Verkehrssimulation anhand fester Positionsdaten durchführen zu können. Dass eine erstellte Trajektorie bzw. Wegpunktliste als Basis für die Generierung von V2X Nachrichten der beschriebenen Fahrzeuge dienen kann, wurde von Heinz et al. jedoch nicht betrachtet.

Pilz stellt in seiner Masterarbeit von 2019 [30] ein Konzept zur Entwicklung eines simulationsbasierten Software-in-the-Loop-Systems zur Validierung hochautomatisierter Fahrfunktionen, basierend auf Szenarien im OpenSCENARIO Format, vor. Ziel ist die Anbindung einer detaillierten 3D-Simulation an die Sensorik eines Fahrzeugs,

wodurch ADAS mit einem in die Simulation integrierten Testfahrer erprobt werden können. Pilz beschriebt die Entwicklung einer Simulationsplattform, die auf CARLA und OpenSCENARIO basiert. Hierfür wird in der Programmiersprache Python ein *Scenario Loader*, ähnlich zu dem OpenSCENARIO Modul im Rahmen dieser Bachelorarbeit, entwickelt und mit dem CARLA Server verknüpft. Der Scenario Loader beinhaltet die Komponenten Szenario Parser, Simulations- und Eventhandling, einen Aktor für das Ego-Fahrzeug und mehrere Aktoren für Non-Ego-Fahrzeuge. Die Unterteilung in *Map*-Aspekte auf der einen und Aktoren auf der anderen Seite basiert auf der Gestaltung von Computerspielen, die eine ähnliche Herangehensweise aufweisen.[30] Das Ego-Fahrzeug kann entweder über die im Szenario enthaltenen Informationen oder über eine hochautomatisierte Fahrfunktion respektive einen angebundenen ADAS Algorithmus gesteuert werden, während die Non-Ego-Fahrzeuge ausschließlich durch das zugrundeliegende Szenario gesteuert werden. Pilz beschränkt sich aufgrund der Komplexität der OpenSCENARIO Standards in seiner Arbeit auf das Parsen und die Simulation von vier aufeinander aufbauenden Szenarien. Als Ergebnis der Arbeit kann festgehalten werden, dass die Ermittlung der benötigten Fahrzeuginformationen mit dem von Pilz entwickelten Scenario Loader ohne Probleme erfolgt. Die korrekte Synchronisation der entwickelten SiL Simulation mit dem CARLA Server ist lediglich zu Anfang der Simulation möglich, da die Implementierung von Pilz durch die Verwendung der Programmiersprache Python keine ausreichende Schnelligkeit aufweist. Die vorgestellte Implementierung kann als Grundlage für die Implementierung des OpenSCENARIO Parsers verwendet werden. Die beschriebene Problematik, dass die Ermittlung der Fahrzeuginformationen bzw. die eigene Simulation dieser nicht ausreichend schnell erfolgt, stellt im Rahmen dieser Bachelorarbeit kein Problem dar, da keine Synchronizität mit einer anderen Simulation gewährleistet werden soll.

4. Konzeption zur Generierung von V2X Nachrichten aus OpenSCENARIO Daten

Das folgende Kapitel umfasst die Konzeption der zu entwickelnden Module für die Generierung von V2X Nachrichten aus OpenSCENARIO Daten. Zunächst wird die HiL Laborumgebung des Instituts für Verkehrssystemtechnik des DLR vorgestellt, in welche die Module integriert werden sollen. Anschließend werden existierende Ansätze zur Kombination des OpenSCENARIO Standards und der V2X Kommunikation erläutert und bezüglich ihrer Anwendbarkeit im Rahmen dieser Arbeit evaluiert. Die Aufgabenstellung dieser Arbeit sieht vor, dass die Laborumgebung um zwei Module erweitert werden soll. Basierend auf der Architektur der Laborumgebung und den Ergebnissen der betrachteten Ansätzen werden Anforderungen an die zu entwickelnden Module aufgestellt. Anschließend wird die konzipierte Architektur anhand einer schematischen Darstellung der Module dargelegt.

4.1. HiL Laborumgebung des Instituts für Verkehrssystemtechnik des DLR

Innerhalb der Zulassung hochautomatisierter Fahrfunktionen (ADAS) nimmt das Verifizieren und Validieren durch umfangreiches Testen eine wesentliche Rolle ein. Da bei der Erprobung aller Komponenten mit zunehmend komplexer werdenden Systemen in realen Tests, also mit einem menschlichen Fahrer im öffentlichen Raum, das

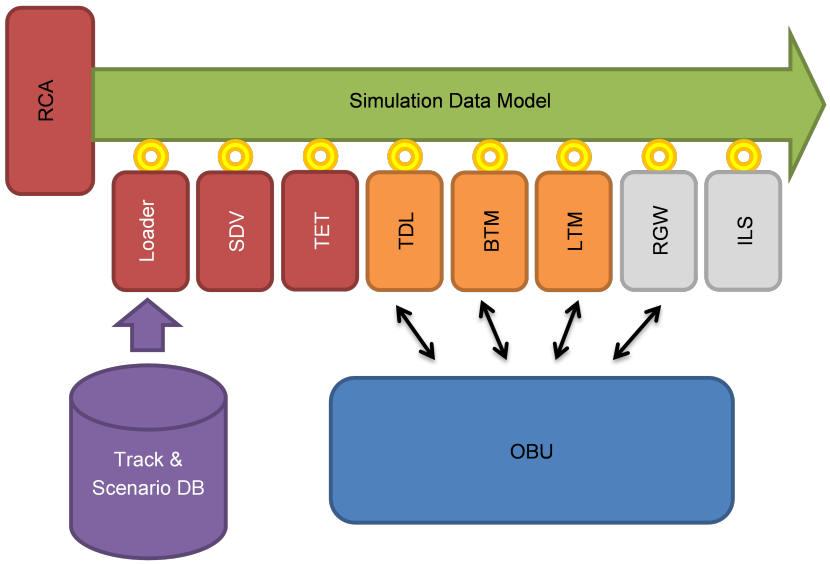
Risiko und die Kosten der Durchführung stark anwachsen, werden ADAS zunehmend in simulierten Umgebungen getestet. HiL beschreibt eine Methodik, in der reale Schnittstellen eines **S**ystem **u**nder **T**est (SUT) an eine Simulationsumgebung angeschlossen werden. Die Simulationsumgebung simuliert Signale an den entsprechenden Hardware-Schnittstellen, sodass das SUT annimmt, es sei in einer realen Situation. Mit HiL-Simulationen ist das reproduzierbare Testen eines SUT in unterschiedlichen Szenarien möglich.[47]

Das **S**imulation **E**nvironment **f**or **E**RTMS **V**erification (SEFEV) ist die Simulationssoftware des RailSiTe Labors des Instituts für Verkehrssystemtechnik des DLR. Das RailSiTe ist eine HiL-Laborumgebung und dient der eisenbahntechnischen Simulation sowie dem Test von Eisenbahnleit- und Sicherungssystemen [33]. Der Test von automatisierten Funktionalitäten eines Zugrechners erfolgt durch die simulierte Informationen, die der OBU des Zugrechners über verschiedene Schnittstellen, beispielsweise Balisen[1], übermittelt werden. Abhängig von den übermittelten Informationen wird ein bestimmtes Verhalten des Zugrechners erwartet. Das Verhalten kann z. B. das Bremsen bzw. Triggern der Bremsfunktion nach Überfahren einer Balise sein. Die simulierten Informationen werden innerhalb der Software aus einem vordefinierten Testszenario generiert. Die Software SEFEV ist in C++ geschrieben und greift auf die Programmbibliothek Qt[2] für grafische Oberflächen zurück.

In Abbildung 4.1 ist der schematische Aufbau der Software dargestellt. Die Software teilt sich in einzelne Komponenten auf, die über **T**ransmission **C**ontrol **P**rotocol (TCP) kommunizieren. Die Kommunikation wird durch die RailSiTe Control Authority (RCA) gesteuert. Die vordefinierten Szenarien befinden sich innerhalb der „Track & Scenario Datenbank“, die sowohl Informationen über den logischen Aufbau des Gleissystems beinhaltet, als auch dynamische Informationen beschreibt. Eine dynamische Information kann beispielsweise eine Kommunikation zwischen der OBU und einer gleisseitigen technischen Einrichtung, wie einer Balise, sein. Der sogenannte „Loader“ lädt Gleis- und Szenarioinformationen aus der Datenbank und übergibt diese an das interne Datenmodell für die Simulation. Innerhalb der Simulation kommunizieren die Module **T**rain **D**ynamics and **L**ogic (TDL), **B**alise **T**ransmission

[1] Balisen sind Informationspunkte im Eisenbahngleis.
[2] `qt.io`

Abbildung 4.1.: SEFEV - Schematische Darstellung



Module (BTM), **L**oop **T**ransmission **M**odule (LTM) und das Radio Gateway mit der OBU des Zugrechners.[32]

Die Simulation eines Szenarios erfolgt über das Einlesen einer sogenannten *Testsequenz*, mit welcher das Modul **T**est **E**vent **P**layer (TEP) abhängig von der aktuellen Position des simulierten Fahrzeugs Events auslösen kann. Eine Testsequenz beschreibt den Ablauf einer Simulation in einzelnen Schritten und wird in SEFEV durch mehrere sqlite[3] Tabellen abgebildet. Der TEP kann Events, bzw. Aktionen anderer Module, die über Schnittstellen mit der OBU kommunizieren, triggern. Beispielsweise ist es möglich ein Szenario zu definieren, das eine Bremsung an einer bestimmten Position vorschreibt. Ist während der Ausführung des Szenarios die angegebene Position erreicht, löst der TEP ein Event aus und aktiviert beispielsweise das Balise Transmission Module, das eine Nachricht an die OBU des Zuges schickt. Innerhalb dieser Bachelorarbeit wird lediglich auf die Tabellen *TCStep* und *SpeedProfile* innerhalb der Testsequenz zurückgegriffen.

SpeedProfile Die Tabelle SpeedProfile beinhaltet die Spalten *TestSequenceID*, wel-

[3] `sqlite.org`

che eine eindeutige Identifikation der zugehörigen Testsequenz in Form einer natürlichen Zahl beinhaltet, *Row* mit der Zeilennummer, sowie *Distance*, *Speed* und *StopTime*, welche die eigentlichen Fahrzeuginformationen enthalten. In Abbildung A.5 ist das Format der Tabelle dargestellt. Die erste Zeile weist für die *Distance* und *Speed* den Wert 0 auf, wodurch das Fahrzeug, solange es an Position 0 ist, die Geschwindigkeit 0 hat. Als *StopTime* ist der Wert 50 angegeben, was bedeutet, dass das simulierte Fahrzeug für 50 Sekunden an der Position 0 stehen bleibt.

TCStep Die Tabelle *TCStep* bezieht sich auf die Informationen der Tabelle Speed-Profile und beinhaltet Events, die abhängig von der aktuellen Position des Fahrzeugs innerhalb der Simulation durch den TEP ausgelöst werden sollen. In Abbildung A.6 ist das Format der Tabelle dargestellt. In der Tabelle *TCStep* sind ebenso wie in der Tabelle SpeedProfile die Spalten *TestSequenceID* und *TCSOrder*, die der Zeilennummer entspricht, enthalten. Die Anzahl der Zeilen in der Tabelle *TCStep* ist i.d.R. größer als in der Tabelle *SpeedProfile*, da für ein Positionselement in *SpeedProfile* mehrere Events in *TCStep* ausgelöst werden. Die Definition der Events erfolgt in der Spalte *ST_DESCRIPTION*. Beispielhafte Events, welche innerhalb der Simulation eines Zuges genutzt werden, sind „On-Board-Unit is powered up“ oder „GENERAL MESSAGE is recorded“. Als zusätzliche Information über die Art des Events existiert die Spalte *ST_IO*, in welcher ein Event mit „I“ (Input) oder „O“ (Output) gekennzeichnet wird. Ein Input Event ist beispielsweise eine Nachricht, die von der Laborumgebung simuliert wird und die das simulierte Fahrzeug erhält, während ein Output Event sämtliche Aktionen umfasst, die von der OBU ausgehen. Die Events werden nacheinander in der durch die Tabelle festgelegten Reihenfolge ausgeführt. Für die Ausführung spricht der TEP je nach Art des Events das jeweilige Modul für die Kommunikation mit der OBU an und löst beispielsweise das Senden einer Balisen Nachricht mit dem „Balise Transmission Module“ aus.

4.2. Existierende Ansätze zur Kombination des OpenSCENARIO Standards und der V2X Kommunikation

In der Literatur existieren viele Arbeiten aus Industrie und Forschung, die sich entweder mit dem Testen von hochautomatisierten Fahrfunktionen auf Basis des OpenSCENARIO Standards (siehe Kapitel 3.3) oder mit der Generierung von V2X Nachrichten aus verschiedenen Datenquellen (siehe Kapitel 2.2) befassen. Die Kombination aus beiden Ansätzen, also die Validierung von vernetzten Fahrfunktionen durch die Generierung von V2X Nachrichten aus Szenarien mit dem OpenSCENARIO Standard, ist ein neuer Aspekt im Bereich szenariobasierter Validierung, der lediglich in wenigen Arbeiten diskutiert wird.

In der Arbeit von Zlocki et al. [50] wird eine von der RWTH Aachen University entwickelte „Ganzheitliche Werkzeugkette für die Entwicklung und Bewertung des automatisierten Fahrens“ vorgestellt, die sowohl Szenariodefinitionen im OpenSCENARIO Format als auch die V2X Kommunikation umfasst. Die beschriebene Werkzeugkette beinhaltet eine Datenbank mit zunächst logischen Verkehrsszenarien, die aus verschiedenen Quellen, wie beispielsweise Fahrversuchen oder Unfalldaten, extrahiert bzw. rekonstruiert werden. Anhand der in [1] entwickelten Ebenendarstellung für Elemente innerhalb eines Verkehrsszenarios (siehe Kapitel 3.1) werden die erstellten logischen Szenarien bezüglich ihrer Vollständigkeit bewertet. In der Datenbank werden zusätzlich zu den genannten Szenarien auch die „für jede Beschreibungsebene mögliche Menge von Ausprägungen der beschreibenden Faktoren“[50] respektive Szenarien mit veränderten Parametern abgelegt. Die logischen Szenarien in der Datenbank werden anschließend in konkrete Szenariodefinitionen im OpenSCENARIO und OpenDRIVE Format umgewandelt. Die Werkzeugkette beinhaltet zum einen verschiedene Simulatoren, die bidirektionale Interaktionen mit dem simulierten Verkehrsszenario ermöglichen, und zum anderen reale Fahrversuche. Die Integration der V2X Kommunikation wird in der Arbeit von Zlocki et al. lediglich im Rahmen der Realfahrten genannt, welche auf einer Teststrecke, ausgestattet mit Gebäudeattrappen zur realistischen Ausbreitung der Funksignale, stattfinden. Innerhalb der

vorgestellten Werkzeugkette sind demnach sowohl der OpenSCENARIO Standard als auch die V2X Kommunikation enthalten. Die Integration der V2X Kommunikation in die Simulationsumgebung erfolgt, laut Zlocki et al. jedoch nicht.

Ein weiterer Ansatz zur Kombination des OpenSCENARIO Standards und der V2X Kommunikation wurde innerhalb des dritten Teils des Pegasus Projekts [1] entwickelt, der das Testen von ADAS mit szenariobasierten Verkehrssimulationen umfasst. Wie in 3.1 erläutert, existieren im Rahmen des Pegasus Projekts funktionale, logische und konkrete Szenarien, die abhängig vom Fortschritt innerhalb des Generierungsprozesses auftreten. Bei der Simulation von konkreten Szenarien im OpenSCENARIO Format wird u. a. die Software CarMaker [4] von IPG Automotive eingesetzt. CarMaker ist eine Simulationsplattform für die Entwicklung und den Test von Algorithmen für PKWs und leichte Nutzfahrzeuge, die sich neben MiL- und SiL-Anwendungsfällen auch für HiL Anwendungsfälle einsetzen lässt. Die Szenarien im OpenSCENARIO Format werden in das interne Szenarioformat von CarMaker umgewandelt, um die anschließende Simulation der V2X Kommunikation zu ermöglichen [1]. In [39] wird das V2X Testsystem der Firma S.E.A Datentechnik[4] mit der Software CarMaker beschrieben. Das vorgestellte System ist sowohl für Open-Loop Erprobungen, mit einem fest definierten, deterministischen Szenario, als auch für Closed-Loop Simulationen, die eine Interaktion zwischen dem Szenario und dem SUT ermöglichen, geeignet. Da im Rahmen dieser Bachelorarbeit lediglich die Simulation von Fahrzeuginformationen innerhalb eines deterministischen Szenarios vorgesehen ist, wird sich auf die Erläuterung der in [39] vorgestellten Open-Loop Simulation beschränkt. Der Ablauf der Open-Loop Lösung gestaltet sich wie folgt: (i) Das Szenario wird aus einer vorhandenen Datenquelle mit der Software CarMaker berechnet. (ii) Die Informationen des Szenarios werden an den *S.E.A V2X Converter* übertragen, der V2X Pakete, **G**lobales **N**avigations**s**atellite**n**system (GNSS) Informationen und **C**ontroller **A**rea **N**etwork (CAN) Pakete generiert. (iii) Die generierten Daten werden über ein Testsystem an das SUT weitergeleitet. (iv) Die Reaktionen des SUT werden im Testsystem aufgezeichnet und anschließend evaluiert.
Die Open-Loop Lösung ähnelt sehr dem Anwendungsfall dieser Bachelorarbeit, da ebenfalls ein SUT durch die Generierung von V2X Nachrichten aus einem vorde-

[4] `sea-gmbh.com/produkte/testsysteme/v2x/`

finierten Szenario erprobt wird. Wie in [1] und [39] gezeigt, kann ein Szenario im OpenSCENARIO Format in ein Szenario im CarMaker Format umgewandelt werden, um daraus V2X Nachrichten zu generieren. Dieser Zwischenschritt soll im Rahmen dieser Bachelorarbeit übersprungen werden, sodass die benötigten Informationen zur Generierung von V2X Nachrichten direkt aus OpenSCENARIO Daten ermittelt werden.

4.3. Anforderungen

Die Anforderungen an die Module zur Generierung von V2X Nachrichten aus OpenSCENARIO leiten sich aus der Architektur der HiL Laborumgebung sowie den Ergebnissen der vorgestellten Arbeiten her und werden separat pro Modul betrachtet. Zunächst werden die Anforderungen an den OpenSCENARIO Parser vorgestellt:

Da ein Szenario im OpenSCENARIO Standard eine detaillierte und realitätsnahe Repräsentation einer Verkehrssituation darstellt, existieren umfangreiche Möglichkeiten zur Spezifikation verschiedener Ereignisse bzw. Aktionen (siehe Kapitel 3.2.3). Wie auch in [30] beschrieben, ist die Implementierung eines Parsers, der den OpenSCENARIO Standard in vollem Umfang verstehen und aus diesem Fahrzeuginformationen ermitteln kann, sehr umfangreich. Daher wird im Rahmen dieser Arbeit nicht der vollständige OpenSCENARIO Standard innerhalb des OpenSCENARIO Parsers implementiert. Für die Funktionalität des OpenSCENARIO Parsers gelten folgende Einschränkungen:

- Beschreibung von Fahrzeugen, Routen, etc. erfolgt ohne die Benutzung von Parameter Deklarationen.
- Es werden lediglich Szenario Objekte inkludiert, die vom Typ „car“ sind.
- Definition von Fahrzeugen innerhalb des Elements `Entities` werden ohne die Nutzung von Katalogen vorgenommen. Einem Fahrzeug, das mittels eines Kataloges definiert ist, wird der Fahrzeugtyp `car` zugewiesen.

- Innerhalb einer `ManeuverGroup` ist mindestens ein Manöver beschrieben. Innerhalb eines Elements vom Typ `Maneuver` ist mindestens ein Event beschrieben.
- Ein `Event` beinhaltet mindestens eine Aktion (`Action`) und eine Startbedingung (`StartTrigger`). Es werden die Aktionen `TeleportAction`, `SpeedAction` und `FollowTrajectoryAction` unterstützt.
- Innerhalb des Elements `Condition` werden die Bedingungen `SpeedCondition`, `ReachPositionCondition` und `SimulationTimeCondition` unterstützt.
- Die Beschreibung der Position eines Fahrzeugs erfolgt ausschließlich über das Element `WorldPosition`
- Das Element `TriggeringEntities`, das innerhalb einer Bedingung gelegen ist, hat für das Attribut `TriggeringEntitiesRule` den Wert „any“.

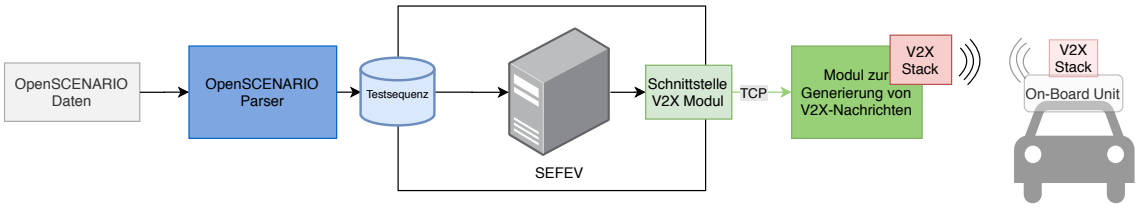
Bezüglich der Integration in die HiL Laborumgebung sind die Gegebenheiten der Laborumgebung respektive der enthaltenen Software SEFEV zu beachten. Wie in 4.1 vorgestellt, bezieht die Software SEFEV die Szenario- bzw. Gleisinformationen aus einer als sqlite-Datenbank abgespeicherten Testsequenz. Die Anforderung besteht demnach darin, die ermittelten Informationen aus der OpenSCENARIO Datei in das Format einer Testsequenz umzuwandeln und diese als sqlite-Datenbank zu speichern. Die Datenbank soll an einem Ort abgelegt werden, auf den die Laborsoftware Zugriff hat. In einer erstellten Testsequenz sollen lediglich solche Informationen enthalten sein, die für die Generierung von V2X Nachrichten innerhalb des V2X Moduls relevant sind. Dies umfasst die Anzahl der im Szenario enthaltenen Fahrzeuge sowie den Fahrzeugtyp, die Position und Geschwindigkeit sämtlicher Fahrzeuge zu jedem Zeitpunkt der Simulation. Zudem sollen Informationen zu Verkehrsereignissen, die durch eine DENM beschrieben werden können, vom OpenSCENARIO Parser geparst und über die Testsequenz an die Software SEFEV übermittelt werden. Die Simulation muss darüber hinaus gewährleisten, dass Aktionen (beispielsweise das Aussenden einer CAM) mindestens ein Mal pro Sekunde ausgeführt werden können.

An das V2X Modul werden ebenfalls Anforderungen gestellt, die nachfolgend erläutert werden: Das V2X Modul soll in die HiL Laborumgebung insofern integriert werden, als dass pro Simulationsschritt innerhalb der Testsequenz das V2X Modul beliebig

oft angesprochen werden kann. Die Kommunikation zwischen der Software SEFEV und dem V2X Modul findet über TCP statt. Mit dem V2X Modul soll die V2X Kommunikation sämtlicher, im Szenario enthaltenen Fahrzeuge simuliert werden. Das Modul muss demnach V2X Nachrichten parallel für mehrere Fahrzeuge generieren und versenden. Das V2X Modul soll die Nachrichtentypen CAM und DENM (siehe Kapitel 2.1.3) unterstützen. Gemäß der Spezifikation einer CAM soll diese mit einem zeitlichen Abstand von maximal 1 s von jedem simulierten Fahrzeug gesendet werden. Eine generierte CAM beinhaltet die ID des zugehörigen Fahrzeugs, den Fahrzeugtyp und die aktuelle Position und Geschwindigkeit. Eine DENM beinhaltet Informationen zu einem im Szenario definierten Verkehrsereignis, das über die Testsequenz übergeben wurde. Die Aussendung einer DENM erfolgt durch das in der Testsequenz spezifizierte Fahrzeug. Die gesendeten Nachrichten des V2X Moduls sollten von einer V2X fähigen OBU empfangen und verarbeitet werden können. Das Empfangen bzw. Verarbeiten von V2X Nachrichten soll im Rahmen der Spezifikation der Nachrichtenformate bzgl. Senderadius und Übertragungsrate und mit Berücksichtigung äußerer Faktoren, wie der Abschirmung durch Gegenstände, stattfinden.

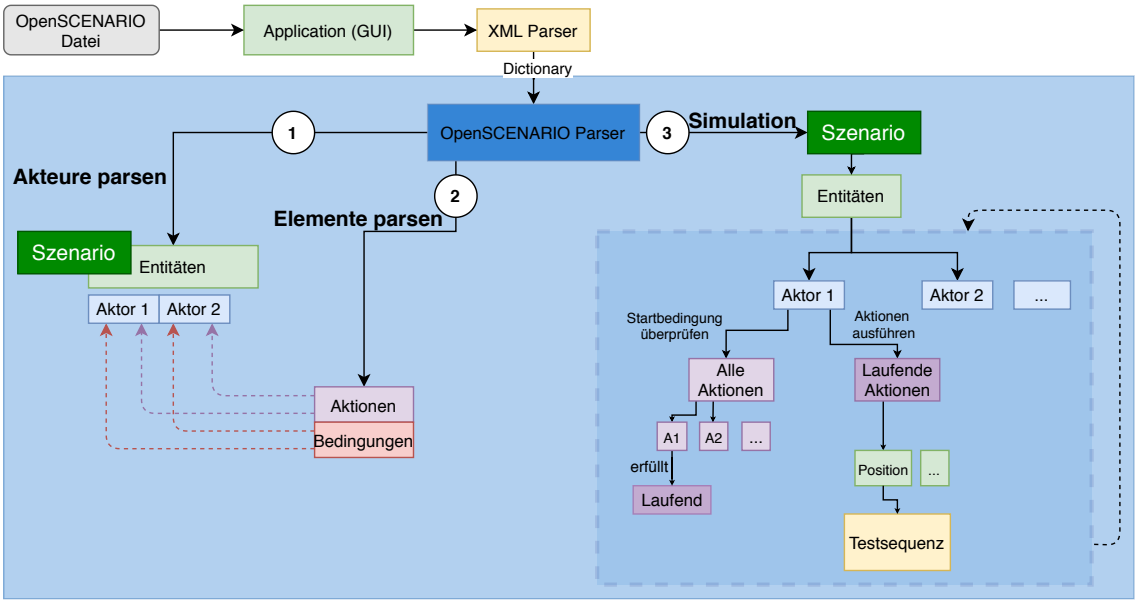
4.4. Architektur

Abbildung 4.2.: Konzeptionierte Architektur - Schematische Darstellung



In Abbildung 4.2 ist die konzeptionierte Architektur der Entwicklung schematisch dargestellt. Die Entwicklung teilt sich in zwei Module, den OpenSCENARIO Parser und ein Modul zu Generierung von V2X Nachrichten auf, die mit der Laborsoftware SEFEV kommunizieren. Der zeitliche Ablauf der Werkzeugkette ist in Abbildung 4.2 von links nach rechts dargestellt.

Abbildung 4.3.: Ablauf innerhalb des OpenSCENARIO Parsers



Zunächst wird Szenario im OpenSCENARIO Format mit dem OpenSCENARIO Parser eingelesen und geparst. Das Parsen beschränkt sich auf die in Kapitel 4.3 genannten Elemente des OpenSCENARIO Standards, jedoch sollen sämtliche der in Kapitel 3.2.3 beschriebenen Elemente des OpenSCENARIO Standards innerhalb der Klassenstruktur des OpenSCENARIO Parsers abgebildet werden. Die Eingliederung sämtlicher OpenSCENARIO Elemente in die interne Klassenstruktur des OpenSCENARIO Parsers soll die Erweiterbarkeit und das schnelle Hinzufügen neuer Funktionalitäten ermöglichen. Die Abhängigkeit der Elemente zueinander soll durch Vererbungstrukturen verdeutlicht werden. Eine detaillierte Ansicht des Ablaufes im OpenSCENARIO Parser ist in Abbildung 4.3 dargestellt. Der dargestellte Prozess, zunächst die Umwandlung der OpenSCENARIO Datei in ein Python `dict()` vorzunehmen, basiert auf der vorgestellten Entwicklung von Pilz [30]. Die Umwandlung in ein `dict()` hat zur Folge, dass der Zugriff auf einzelne Elemente des Szenarios einfacher erfolgen kann. Innerhalb des Parsens soll zunächst ein internes Szenario erstellt werden, das im Laufe des Prozesses mit Informationen befüllt wird. Im ersten Schritt werden die beteiligten Entitäten geparst und dem Szenario als Liste hinzugefügt. Im zweiten Schritt werden die Entitäten betreffende Informationen, wie

OpenSCENARIO `Actions` und `Conditions` geparst und den jeweiligen Entitäten zugeordnet. Zur Gewährleistung des korrekten Ablaufes des Szenarios, sollen geparste Aktionen der Reihenfolge nach in einer Liste hinzugefügt werden, auf welche die zugehörige Entität zugreifen kann. Jeder Aktion werden gemäß dem OpenSCENARIO Standard Start- und Endbedingungen zugeordnet. Darüber hinaus soll der Status einer Aktion (siehe Kapitel 3.2.3) abgebildet werden.

Die Position des Fahrzeugs wird innerhalb des Szenarios in Weltkoordinaten angegeben, die nach Spezifikation abhängig vom inertialen Bezugssystem der Simulationsumgebung sind [10]. Im Rahmen dieser Bachelorarbeit sollen die Weltkoordinaten als Differenz zu einer globalen Position, die als Anfangsposition des Szenarios angesehen werden kann, in Metern angegeben werden. Die globale Bezugsposition soll dabei aus den zugrundeliegenden OpenDRIVE Daten eines Szenarios ermittelt werden.

Die Ermittlung der dynamischen Fahrzeuginformationen, wie die Position eines Fahrzeugs innerhalb eines Szenarios, erfordert die Simulation des Szenarios im OpenSCENARIO Format. Die Simulation stellt den dritten Schritt im Ablauf innerhalb des OpenSCENARIO Parsers dar. Der konzipierte Ablauf der Simulation des geparsten Szenarios ist ebenfalls in Abbildung 4.3 dargestellt und wird im Folgenden erläutert:

Die Simulation des Szenarios erfolgt ab einem vom Benutzer festgelegten Startzeitpunkt (beispielsweise durch das Betätigen eines Buttons) solange bis die beinhalteten Manöver abgeschlossen sind. Die Information, ob ein Manöver bzw. ein anderes enthaltenes Element abgeschlossen ist, wird über den Status des Elements ermittelt. Die Simulation beginnt mit dem Simulationsschritt 0, der (solange keine gegensätzliche Information gegeben ist) einer Simulationszeit von 0 s entspricht. Die Größe eines Simulationsschrittes respektive die Zeit, die während eines Simulationsschrittes verstreicht, soll veränderlich sein, aber einen Wert von 1 s nicht überschreiten (siehe Anforderung in Kapitel 4.3). Pro Simulationsschritt sollen sämtliche im Szenario enthaltenen Entitäten nacheinander betrachtet werden. Für jede Entität werden zunächst die Startbedingungen ihrer zugehörigen Aktionen überprüft. Ist die Startbedingung einer Aktion erfüllt, wird diese, wenn die Überprüfung der Startbedingungen abgeschlossen ist, ausgeführt. Die Ausführung einer Aktion kann die Position, die Geschwindigkeit und die Beschleunigung einer Entität verändern. Bei der Ausführung

Tabelle 4.1.: Konzipierte Informationsquellen der Simulation im Vergleich zu IPG CarMaker [39]

Simulationsaspekte	**S.E.A Open-Loop Lösung**	**SEFEV Simulation**
Position des SUT	GNSS Simulation	berechnete Position des OpenSCENARIO Ego-Fahrzeugs
Geschwindigkeit des SUT	GNSS Simulation	berechnete Geschwindigkeit des OpenSCENARIO Ego-Fahrzeugs
Interne Informationen des SUT	„Internal Car State“	Fahrzeuginformationen in OpenSCENARIO
Position weiterer Fahrzeuge	GNSS Simulation	berechnete Positionen der OpenSCENARIO Non-Ego-Fahrzeuge
V2X Nachrichten weiterer Fahrzeuge	SEA V2X Stack	Open Source V2X Stack

einer Aktion ist zu beachten, dass eine Aktion i.d.R. nicht innerhalb eines Simulationsschrittes gestartet und beendet wird. Aus diesem Grund sollen Aktionen jeweils nur für die Länge, die ein Simulationsschritt andauert, ausgeführt werden. Nachdem sämtliche Aktionen einer Entität, deren Startbedingung erfüllt ist, ausgeführt wurden, wird die nächste Entität betrachtet. Ist die Durchführung der Aktionen der letzten im Szenario enthaltenen Entität beendet, werden die ermittelten Informationen in das Format einer Testsequenz (siehe Kapitel 4.1) umgewandelt und der nächste Simulationsschritt gestartet.

Nach der Beendigung der Simulation des eingelesenen Szenarios wird die vollständige Testsequenz an die Laborsoftware SEFEV übermittelt (siehe Abbildung 4.2). Im Rahmen dieser Arbeit wird die Laborsoftware innerhalb einer virtuellen Maschine mit dem Betriebssystem CentOS ausgeführt. Wie in Kapitel 4.1 dargelegt, kann die Software SEFEV mittels des Moduls TEP die in der Testsequenz spezifizierten Schritte nacheinander simulieren. In Tabelle 4.1 werden die verschiedenen Aspekte

der Simulation und die konzipierte Quelle der Simulationsinformationen im Rahmen dieser Bachelorarbeit mit dem vorgestellten S.E.A. Open-Loop Konzept mit IPG CarMaker (siehe 4.2) verglichen. Die Position und Geschwindigkeit des SUT werden in der Open-Loop Lösung durch die separate Simulation von GNSS Daten, unabhängig vom auszuführenden Szenario, ermittelt. Da in OpenSCENARIO jedoch bereits Informationen zum möglichen SUT in Form des Ego-Aktors bzw. Ego-Fahrzeugs enthalten sind, sollen sowohl die Position als auch die Geschwindigkeit des SUT im Rahmen dieser Bachelorarbeit durch dynamische Informationen innerhalb des geparsten Szenarios ermittelt werden. Wie in Abbildung 4.2 dargestellt, sollen sowohl die dynamischen als auch die statischen Fahrzeuginformationen des Ego-Fahrzeugs innerhalb der Simulation an das SUT übermittelt werden. Durch die erhaltenen Informationen kann SEFEV die Fahrt des Ego-Fahrzeugs simulieren. Die internen, statischen Informationen des SUT werden demnach ebenfalls aus den definierten Informationen zur Entität „Ego“ innerhalb der OpenSCENARIO Datei ermittelt. Die Position weiterer Fahrzeuge wird in der Open-Loop Lösung von S.E.A abermals durch die Simulation der GNSS Daten umgesetzt. Da die dynamischen Informationen, der neben dem Ego-Fahrzeug im Szenario enthaltenen Fahrzeuge ebenfalls in der Szenariodefinition im OpenSCENARIO Format enthalten sind, werden Positions- und zusätzliche dynamische und statische Informationen weiterer Fahrzeuge durch die Betrachtung der Non-Ego-Fahrzeuge extrahiert. Für jedes innerhalb der generierten Testsequenz genannte Non-Ego-Fahrzeug werden unter Berücksichtigung der zugehörigen Fahrzeuginformationen V2X Nachrichten generiert, welche das SUT empfangen kann. Die Simulation der V2X Nachrichten der Non-Ego-Fahrzeuge soll über einen Open Source V2X Stack innerhalb des V2X Moduls erfolgen.

Die Abbildung der im Szenario möglichen Verkehrsereignisse soll, wie in 4.3 erläutert, durch die Generierung einer DENM erfolgen. Die DENM soll während der Simulation von mindestens einem Non-Ego-Fahrzeug ausgesendet werden, sodass das SUT die Nachricht empfangen kann.

Das V2X Modul unterteilt sich in mehrere Komponenten. Zum einen wird eine TCP Kommunikationsschnittstelle zu der Laborsoftware SEFEV benötigt, um die Fahrzeuginformationen pro Simulationsschritt zu empfangen. Zudem soll eine Applikation

entwickelt werden, welche aus den erhaltenen Fahrzeuginformationen V2X Nachrichten, respektive CAMs und DENMs, durch die Integration eines Open Source V2X Stacks generiert. Da die V2X Nachrichten über das IEEE 802.11p Protokoll versendet und empfangen werden sollen, soll die Anwendung innerhalb einer 802.11p-fähigen Hardwareeinheit ausgeführt werden.

5. Entwicklung des OpenSCENARIO Parsers

Die Entwicklung des OpenSCENARIO Parsers unterteilt sich in die Aspekte Implementierung und Integration. Zunächst erfolgt die Erläuterung der Implementierung des Parsers und der Simulation der im OpenSCENARIO Standard enthaltenen Elemente anhand des in Kapitel 4.4 vorgestellten Konzepts. Anschließend wird die Integration des OpenSCENARIO Parsers in die HiL Laborumgebung und die Übertragung der ermittelten Fahrzeuginformation an SEFEV in Form einer Testsequenz dargelegt.

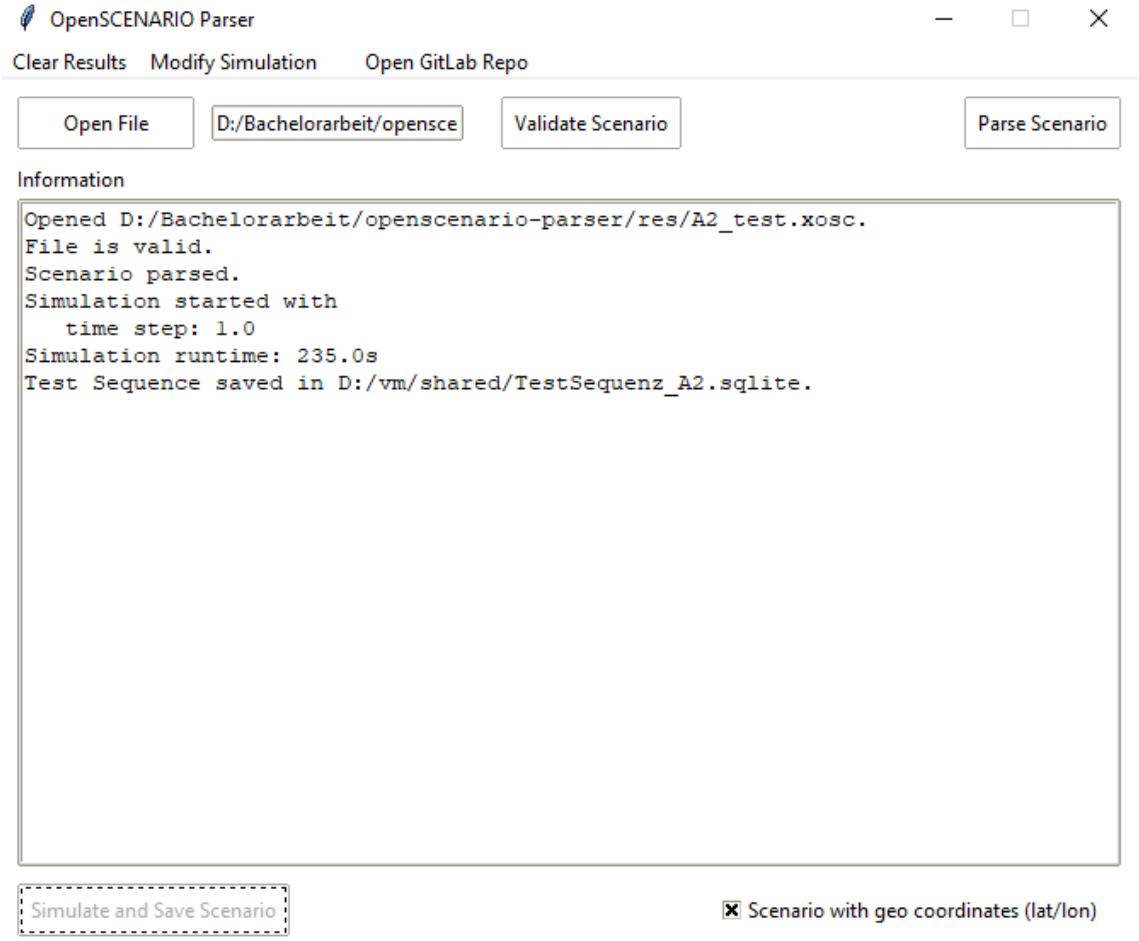
5.1. Implementierung des OpenSCENARIO Parsers

Das in 4.4 vorgestellte Konzept wird mit der Programmiersprache Python implementiert. Das Einlesen und Umwandeln der OpenSCENARIO Datei zu einem Python `dict()` erfolgt mit Hilfe des Pakets *xmlschema* [49], das Funktionen für das Umwandeln eines XML-Dokuments mit einer gegebenen Schema Spezifikation (XSD) bereitstellt. Die grundlegende Struktur des Moduls für das Parsen einer OpenSCENARIO Datei ist in Abbildung B.1 gezeigt. Der OpenSCENARIO Parser wird mit Hilfe der Klasse `Application` gestartet, durch welche sich eine grafische Oberfläche, die in Abbildung 5.1 dargestellt ist, öffnet. Die Benutzeroberfläche basiert auf dem **G**raphical **U**ser **I**nterface (GUI)-Toolkit *tkinter*[1] und beinhaltet u. a. die Widgets

[1] `docs.python.org/3/library/tkinter.html`

`Button`, `Scale`, `Label`, `LabelFrame` und `Entry`. Die Anordnung der Elemente erfolgt mit dem *Grid Geometry Manager*.

Abbildung 5.1.: Grafische Oberfläche des OpenSCENARIO Parsers



Der Benutzer kann nach dem Starten der Anwendung eine Datei mit der Dateiendung .xml (XML) oder .xosc (OpenSCENARIO) durch den Button „Open File“auswählen. Die Datei kann mittels des Buttons „Validate Scenario“ gegen das OpenSCENARIO Schema validiert werden. Mit Hilfe des Buttons „Parse Scenario“ kann die ausgewählte Datei geparst werden. Hierfür wird zunächst eine Instanz der Klasse `XMLParser` erstellt, welche die eingelesenen OpenSCENARIO Datei in ein Python `dict` umwandelt. Das erstellte `dict` wird nachfolgend an ein Objekt der Klasse `OpenSCENARIOParser` übergeben, welches die Ermittlung der Informationen aus

dem definierten Szenario durch die Erstellung einer Instanz der Klasse `Scenario` übernimmt. Das `Scenario` beinhaltet eine Liste an zugehörigen Entitäten sowie eine Liste mit allgemeinen Events im Szenario, die beispielsweise die Inhalte einer `UserDefinedAction` umfassen können. Darüber hinaus enthält das Szenario eine globale Bezugsposition mit der die Umwandlung der in Weltkoordinaten angegebenen Fahrzeugpositionen in geografische Koordinaten ermöglicht wird.

In Abbildung B.2, B.3 und B.4 sind die Klassendiagramme in UML-Notation für die im Rahmen dieser Arbeit verwendeten Elemente des OpenSCENARIO Standards dargestellt. Die Hierarchie der aufgeführten Elemente ist ähnlich zu der in Kapitel 3.2.3 vorgestellten Struktur der Elemente des OpenSCENARIO Standards. Im Folgenden werden daher lediglich die Implementierungen vorgestellt, die entweder nicht im OpenSCENARIO Standard existieren oder sich zum OpenSCENARIO Referenzelement unterscheiden:

Die Klasse `Actor` ist die Umsetzung des Elements `ScenarioObject` des OpenSCENARIO Standards, das eine einzelne Entität innerhalb eines Szenarios darstellt. In OpenSCENARIO kann ein `ScenarioObject` ein `Vehicle` mit Fahrzeug- und Performanceinformationen, sowie bestimmten Abhängigkeiten und einen `ObjectController`, der ein Fahrerobjekt beschreibt, beinhalten. Die Klasse `Actor` enthält Informationen zum Namen (`name`), Fahrzeugtyp (`type`), zur aktuellen Position (`position`), Geschwindigkeit (`velocity`), Beschleunigung (`acceleration`) sowie Listen mit noch nicht ausgeführten Aktionen (`actions`) bzw. aktuell laufenden Aktionen (`running_actions`). Die Aktionen innerhalb der Liste `actions` sind in der Reihenfolge des Parsens, demnach in derselben Reihenfolge wie innerhalb der OpenSCENARIO Datei, angeordnet.

Die Klasse `PrivateAction` weist die Attribute `type`, zur Beschreibung des Aktionstypen, (beispielsweise *FollowTrajectoryAction*), `start_condition` und `end_condition`, die jeweils eine Liste von Objekten der Klasse `Condition` beinhalten, und `state` auf. Die Werte des Attributs `state` orientieren sich an den von ASAM e.V. für den OpenSCENARIO Standard verwendeten Bezeichnungen `StandBy`, `Running` und `Complete` [10]. Wird eine Aktion innerhalb eines Events ausgeführt, weisen, im nach OpenSCENARIO definierten Szenario, lediglich das übergeordnete Event bzw. der übergeordnete Akt eine Startbedingung (`StartTrigger`) auf. Während des Parsens

werden zunächst sämtliche Aktionen innerhalb eines Events geparst. Anschließend werden die Startbedingungen des Events bzw. des Akts den Aktionen innerhalb des Events zugeordnet. Zusätzlich zu den Startbedingungen, die durch das übergeordnete Element entstehen, können weitere Start- und Endbedingungen für einzelne Aktionen definiert werden. Standardmäßig wird beim Parsen der Aktionen zusätzlich die Startbedingung hinzugefügt, dass sämtliche vorherigen Aktionen abgeschlossen sein und demnach den Status `Complete` aufweisen müssen. Dies dient der Vermeidung einer simultanen Ausführung mehrerer Aktionen eines Fahrzeugs.

Die Klasse `UserDefinedAction` wird innerhalb der Szenariodefinition für das Aussenden einer DENM verwendet. Innerhalb eines Szenarios im OpenSCENARIO Format wird eine DENM als `type` einer `CustomCommandAction` innerhalb einer `UserDefinedAction` übergeben (siehe Listing 5.1). Dabei können in fester Reihenfolge die folgenden Parameter angegeben werden: Name der sendenden ITS, *CauseCode* des Events, *SubCauseCode* des Events, geografische Länge und Breite, Anzahl der Wiederholungen.

Nach Beendigung des Parsens, der vom Benutzer angegebenen OpenSCENARIO Datei, wird eine Information in der grafischen Oberfläche der Applikation ausgegeben und der Button „Simulate and Save Scenario“ aktiviert. In der Menüleiste der grafischen Oberfläche wird zudem das Drop-Down Menü „Modify Simulation“aktiviert, wodurch der Benutzer die initiale globale Position respektive die Referenzposition der Simulation (siehe Abbildung A.3) und den zeitlichen Abstand zwischen zwei Simulationsschritten (siehe Abbildung A.4) bestimmen kann. Mit einer Checkbox kann ausgewählt werden, ob die im Szenario verwendeten Weltkoordinaten als geografische Breite und Länge oder als Distanz zu einer Referenzkoordinate angegeben sind. Für sämtliche Berechnungen mit geographischen Koordinaten innerhalb des OpenSCENARIO Parsers wird *pygeodesy*[2] verwendet.

Durch einen Klick auf den Button „Simulate and Save Scenario“ wird der Simulationsprozess innerhalb der Applikation gestartet. In Folge dessen übergibt die Applikation das vom `OpenSCENARIOParser` erstellte `Szenario`, das die geparsten

[2] `/mrjean1.github.io/PyGeodesy/`

Listing 5.1: OpenSCENARIO `UserDefinedAction` mit enthaltener DENM

```
<Actions>
  <UserDefinedAction>
    <CustomCommandAction type="A1,99,0,52.56732,10.45632,5"/>
  </UserDefinedAction>
</Actions>
```

Informationen beinhaltet, an den `ScenarioSimulator`. Der `ScenarioSimulator` simuliert das erhaltene Szenario nach dem in Kapitel 4.4 vorgestellten Ablauf. Die Überprüfung der Bedingungen für die Aktionen wird durch den `ConditionChecker` durchgeführt. Der `ConditionChecker` erhält zunächst für sämtliche Startbedingungen einer Aktion die einzelne Bedingung vom Typ `Condition`, den zugehörigen Aktor und den aktuellen Simulationsschritt. Die Überprüfung, ob eine Bedingung erüllt ist, ist abhängig vom Typ der Bedingung. Im Folgenden wird beispielhaft die Überprüfung von Bedingungen der Typen `SimulationTimeCondition` und `SpeedCondition` vorgestellt:

SimulationTimeCondition wird durch den Vergleich der aktuellen Simulationszeit mit einem festgelegten Wert überprüft. Ist beispielsweise angegeben, dass eine Aktion erst beginnen soll, nachdem 10 s vergangen sind, wird der aktuelle Simulationsschritt zunächst mit der Zeiteinheit pro Simulationsschritt (angegeben in Sekunden) multipliziert, um die aktuelle Simulationzeit zu erhalten. Ist die Simulationszeit größer oder gleich dem definierten Wert, ist die Bedingung erfüllt und der `ConditionChecker` gibt den Wert `True` zurück.

SpeedCondition wird durch den Vergleich der aktuellen Geschwindigkeit des zugehörigen Fahrzeugs mit einem festgelegten Wert überprüft. Der Vergleichsoperator wird durch das in der `Condition` enthaltene Attribut `rule` definiert, das eine Instanz des `RuleOperators` darstellt. Je nach definierter Operation wird überprüft, ob die aktuelle Geschwindigkeit des Fahrzeugs größer, gleich oder kleiner als der definierte Wert ist. Ist die Bedingung erfüllt, gibt der `ConditionChecker` den Wert `True` zurück.

Sind sämtliche Startbedingungen einer Aktion erfüllt, erhält sie den Status `Running`

und wird zu der Liste `running_actions` des Aktors hinzugefügt und aus der Liste `actions` entfernt. Ist die Überprüfung der Startbedingungen für jede Aktion jedes Aktors abgeschlossen, startet die Ausführung der Aktionen mit dem Status `Running`. Der `ScenarioSimulator` startet für jede auszuführende Aktion den `ActionExecutor`, der die Ausführung der Aktion respektive die Berechnung der neuen Position und Geschwindigkeit innerhalb des aktuellen Simulationschrittes übernimmt. Der `ActionExecutor` erhält die auszuführende Aktion vom Typ `Action` und den zugehörigen Aktor. Eine Aktion, deren Simulation gemäß der aufgestellten Anforderungen erfolgen soll, ist die `FollowTrajectoryAction`, die das Folgen einer definierten Trajektorie innerhalb des Szenarios beschreibt. Für die Berechnung der neuen Position eines Fahrzeugs, das eine `FollowTrajectoryAction` ausführt, werden die folgenden Schritte ausgeführt:

1. Die mögliche Distanz, welche das Fahrzeug innerhalb eines Simulationsschrittes unter Beachtung seiner aktuellen Geschwindigkeit und Beschleunigung zurücklegen kann, wird berechnet. Die Formel für die Berechnung lautet: $travel\_dist = speed \times sim\_time + 0.5 \times acceleration \times sim\_time^2$.

2. Die Distanz zwischen der aktuellen Position des Fahrzeugs und dem nächsten Vertex[3] (`vertex_dist`) wird berechnet. Da sowohl die Position des Fahrzeugs als auch der Vertex als OpenSCENARIO `WorldPosition` angegeben sind, wird die Distanz der Positionen im dreidimensionalen Raum, unter der Beachtung der Differenz in x-, y- und z-Richtung, berechnet.

3. Ist die `travel_dist` kleiner als die `vertex_dist`, so tritt der Fall, welche in Abbildung 5.2(a) visualisiert ist, ein und die Berechnung wird in Schritt 5 weitergeführt. Der zweite mögliche Fall tritt ein, wenn die `travel_dist`, die das Fahrzeug innerhalb des aktuellen Simulationsschrittes zurücklegt größer ist als die Distanz zwischen seiner aktuellen Position und dem nächsten Vertex. In diesem Fall erfolgt der vierte Schritt, der solange wiederholt wird, bis die `travel_dist` einen kleineren Wert als die `vertex_dist` aufweist.

[3]Der Begriff Vertex beschreibt einen Eckpunkt einer Trajektorie.

4. In diesem Schritt wird zunächst die `vertex_dist` von der `travel_dist` subtrahiert, da die Entfernung zum nächsten Vertex innerhalb des aktuellen Simulationsschrittes überwunden wird. Anschließend werden ein neuer Start- und Endpunkt der abzufahrenden Trajektorie festgelegt. Der neue Startpunkt ist, wie in Abbildung 5.2(b) dargestellt, die Position des ersten Vertex, welchen das Fahrzeug innerhalb des aktuellen Simulationsschrittes passiert. Der Endpunkt der Trajektorie ist die Position des nächsten abzufahrenden Vertex, welcher innerhalb der `FollowTrajectoryAction` definiert ist.

5. Die neue Position des Fahrzeugs auf der abzufahrenden Trajektorie wird mit der folgenden Rechnung ermittelt. Es wird lediglich die Ermittlung der neuen Position in x- und y-Richtung beschrieben, die sich jedoch ebenfalls auf die Position in z-Richtung anwenden lässt:

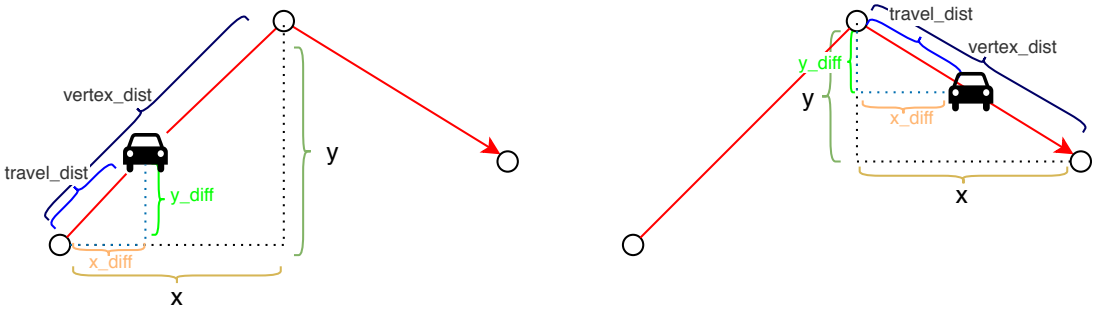
$$x\_diff = \sqrt{\frac{travel\_dist^2}{1 + \frac{y^2}{x^2}}})$$

$$y\_diff = (\frac{y}{x}) \times x\_diff$$

   Die berechneten Differenzen werden anschließend zu der ursprünglichen x- bzw. y-Position des Fahrzeugs addiert. Sind die Positionen der Fahrzeuge innerhalb des Szenarios als Distanz zu einer geographischen Referenzposition angegeben, erfolgt zuletzt die Umwandlung der errechneten Position in geographische Breite und Länge.

Der `ActionExecutor` weist die neue Position dem zugehörigen Aktor zu und gibt diesen an den `ScenarioSimulator` zurück. Ist eine Aktion beendet, erhält sie den Status `Complete`. Weist ein Fahrzeug keine Aktionen auf, die den Status `Running` haben, wird die Funktion `simple_driving()` des `ActionExecutors` ausgeführt, welche das Weiterfahren des Fahrzeugs mit Berücksichtigung der Position, Geschwindigkeit und Richtungsangabe (`WorldPosition.h`[4]) umfasst.

[4]Das Attribut h der `WordPosition` enthält die Richtung des Fahrzeugs (heading), wobei der Wert 0 das Fahren entlang der x-Achse respektive nach Osten beschreibt



(a) Gefahrene Distanz kleiner als Vertex Distanz (b) Gefahrene Distanz größer als Vertex Distanz

Abbildung 5.2.: Visualisierung der Berechnung einer Position auf einer Trajektorie

Nach der Berechnung der Position und Geschwindigkeit aller im Szenario enthaltenen Fahrzeuge werden die ermittelten Informationen mittels des `DatabaseWriters` unter der Verwendung des *sqlite3*[5] Pakets in eine sqlite-Datenbank geschrieben.

5.2. Integration des OpenSCENARIO Parsers in die HiL Laborumgebung

Die Integration des OpenSCENARIO Parsers in die HiL Laborumgebung erfolgt im Wesentlichen durch die Umwandlung der ermittelten Fahrzeuginformationen in das Format einer Testsequenz, respektive in die Tabellen TC_Step und SpeedProfile (siehe Erläuterung in Kapitel 4.1).

Pro Simulationsschritt wird zunächst die Tabelle SpeedProfile mit den ermittelten Informationen des Ego-Fahrzeugs versehen. Anschließend werden innerhalb der Tabelle TC_Step für die aktuelle Position des Ego-Fahrzeugs auszulösende V2X Events hinzugefügt. Abhängig von der Größe eines Simulationsschritts wird das Aussenden von CAM Nachrichten der Non-Ego-Fahrzeuge (nach ETSI Spezifikation jede 0,1 −1 s) ein- oder mehrfach als Event hinzugefügt. Das Format eines V2X Events für die Aussendung einer CAM ist in Abbildung 5.3 dargestellt. Abbildung 5.4 zeigt

[5] `docs.python.org/3/library/sqlite3.html`

ein V2X Event für das Aussenden einer DENM innerhalb der Testsequenz. Die Events werden durch die Information „V2X“ in der Spalte ST_INTERFACE als V2X Events gekennzeichnet. Weiterführende Informationen über den Nachrichteninhalt werden in der Spalte ST_DESCRIPTION festgehalten. Der erste Eintrag respektive die Zahl „1“ oder „2“ gibt den Nachrichtentyp an. Eine „1“ steht dabei für eine CAM und eine „2“ für eine DENM. In Abhängigkeit vom Nachrichtentyp erhalten nachfolgende Variablen eine unterschiedliche Bedeutung. In Bezug auf die Abbildungen 5.3 und 5.4 wird im Folgenden die Reihenfolge der Variablen, zunächst für eine CAM und anschließend für eine DENM erläutert. Die Trennung der Variablen erfolgt durch Kommata.

CAM ID der aussendenden ITS Station, Position in x-, y- und z-Richtung, Geschwindigkeit, Stationstyp der ITS Station

DENM ID der aussendenden ITS Station, Stationstyp, *CauseCode*[6] des Events, *SubCauseCode*[6] des Events, Position in x- und y-Richtung, Geschwindigkeit der aussendenden ITS Station

Abbildung 5.3.: Darstellung einer CAM in einer Testsequenz

...	ST_DESCRIPTION	...	ST_IO	...	ST_INTERFACE	...
	1,12,52.34521,10.43276,1.067,20.2,car		I		V2X	

Abbildung 5.4.: Darstellung einer DENM in einer Testsequenz

...	ST_DESCRIPTION	...	ST_IO	...	ST_INTERFACE	...
	2,12,car,99,0,52.34521,10.43276,20.2		I		V2X	

Innerhalb der Laborsoftware SEFEV wird das Modul TEP modifiziert, sodass lediglich die Tabellen TC_Step und SpeedProfile eingelesen werden. Darüber hinaus wird ein `V2XEvent` erstellt, dass vom TEP ausgelöst wird, wenn als Interface „V2X“ angegeben ist. Das `V2XEvent` wird ähnlich zu bestehenden Events, welche mit existierenden Modulen, beispielsweise BTM, verknüpft sind, eingebunden.

[6] Weiterführende Informationen zu finden in [20].

6. Entwicklung des V2X Moduls

Wie in Kapitel 4.4 dargelegt, werden für die Entwicklung des V2X Moduls mehrere sowohl software- als auch hardwareseitige Komponenten benötigt, die im Rahmen dieser Arbeit implementiert bzw. integriert werden. Im Folgenden wird zunächst der gewählte Open Source V2X Stack, Vanetza, vorgestellt und die Wahl begründet. Im Zuge dessen erfolgt eine Einführung in die Cohda MK5 OBU [28] als gewählte Hardwareeinheit zur Umsetzung der ETSI ITS G5 Kommunikation. Zuletzt wird die Entwicklung einer Anwendung, welche mit der Laborsoftware SEFEV kommuniziert und aus den erhaltenen Informationen V2X Nachrichten mittels des Vanetza Stacks generiert und verschickt, beschrieben und die Integration der Anwendung in die Cohda MK5 OBU erläutert.

6.1. Eingesetzte Soft- und Hardwarekomponenten für das V2X Modul

Es existiert eine Vielzahl an Implementierungen für die Bereitstellung der V2X Kommunikation über den IEEE 802.11p respektive ETSI ITS G5 Standard (siehe Kapitel 2.1.2). Da im Rahmen dieser Arbeit frei verfügbare Implementierungen verwendet werden sollen, werden die V2X Stacks OpenC2X und Vanetza im Folgenden in Bezug auf die in Kapitel 4.3 gestellten Anforderungen verglichen.

Sowohl Vanetza als auch OpenC2X unterstützen die Nachrichtenformate CAM und DENM, die im Rahmen dieser Arbeit verwendet werden. Zudem ist es möglich beide V2X Stacks in eine 802.11p fähige OBU zu integrieren. Im Gegensatz zu OpenC2X, das als eigenständige Anwendung verstanden werden kann, muss für die Verwendung

des Vanetza V2X Stacks immer ein Trigger von einer weiteren Anwendung erfolgen. Bei der Entwicklung des V2X Moduls weist Vanetza in diesem Punkt demnach einen Vorteil auf, da der V2X Stack direkt in eine eigene Anwendung integriert werden kann. Vanetza unterstützt darüber hinaus weitere Funktionalitäten innerhalb der ETSI Spezifikation, wie das GeoNetworking und Security, die in OpenC2X nicht enthalten sind. Der letzte Aspekt, der in Vanetza ebenfalls besser umgesetzt ist, ist die Dokumentation. Während Vanetza eine Dokumentation [43] und ein aktives Github Repository[1] umfasst, finden sich für OpenC2X nur wenige offizielle Dokumentationen der Komponenten. Zusammenfassend überwiegen die Vorteile für die Verwendung von Vanetza, weshalb die Entwicklung des V2X Moduls im Rahmen dieser Bachelorarbeit mit dem V2X Protokoll Stack von Vanetza erfolgt.

Innerhalb des Vanetza V2X Stacks wird *socktap* verwendet. Socktap führt Vanetza auf Basis von *Linux raw packet sockets*[2] und ermöglicht die Erprobung von einigen in Vanetza enthaltenen Funktionalitäten, ohne dass spezielle V2X Hardware benötigt wird.

Als Hardwarekomponente zur Umsetzung der Kommunikation über den IEEE Standard 802.11p wird eine Cohda MK5 OBU verwendet. Dies ist zum einen darin begründet, dass OBUs dieses Typs dem Institut für Verkehrssystemtechnik des DLR zur Verfügung stehen und bereits mehrfach in Projekten eingesetzt wurden (siehe Kapitel 2.1.4). Zum anderen bietet Cohda das Cohda SDK, eine virtuelle Maschine für das Entwickeln und Cross-kompilieren von Anwendungen für eine Cohda MK5 OBU, an, wodurch der Entwicklungsprozess vereinfacht werden kann. Darüber hinaus ist die Integration von Vanetza als V2X Stack in einer Cohda MK5 OBU möglich.

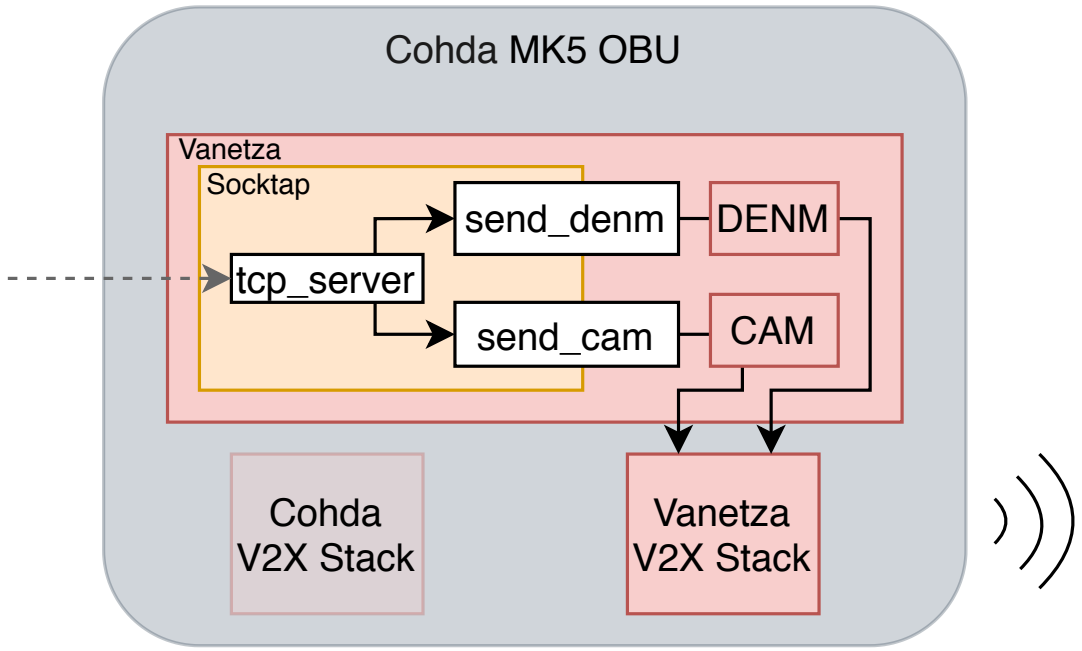
An die Cohda MK5 OBU, abgebildet in A.7, werden zunächst zwei Rundstrahlantennen und ein GPS-Modul angeschlossen. Anschließend wird die initiale Konfiguration der OBU vorgenommen, die unter anderem das Zuweisen einer statischen IP-Adresse für den Zugriff über *ssh*[3] und die Aktivierung von Netzwerkschnittstellen umfasst.

[1] `github.com/riebl/vanetza`
[2] Linux raw packets sockets ermöglichen das Senden oder Empfangen von Daten ohne vordefinierte Header.
[3] Secure Shell protocol (`ssh.com/ssh/`)

Daraufhin können Anwendungen, die mittels des Cohda SDKs gebaut wurden, in die OBU integriert werden.

Die Architektur des V2X Moduls innerhalb der Cohda MK5 OBU ist in Abbildung 6.1 dargestellt. Die abgebildeten Komponenten werden in den nachfolgenden Kapiteln an passender Stelle erläutert.

Abbildung 6.1.: Aufbau des V2X Moduls



6.2. Integration des V2X Moduls in die HiL Laborumgebung

Die Kommunikation zwischen der Laborsoftware SEFEV innerhalb der HiL Laborumgebung und dem V2X Stack von Vanetza erfolgt, gemäß den in Kapitel 4.3 aufgestellten Anforderungen, über das Protokoll TCP. Im Folgenden wird die Implementierung der Kommunikationsfunktionalität zunächst von Seiten der Laborumgebung und anschließend von Seiten des V2X Stacks vorgestellt.

Innerhalb der Software SEFEV wird zusätzlich zu den existierenden Modulen (z. B. BTM) ein `V2XModul` implementiert. Innerhalb des `V2XModuls` werden ein `V2XInterface` und ein `V2XSocket` implementiert. Die genannte Aufteilung orientiert sich an dem bereits existierenden **R**adio **T**ransmission **M**odule (RTM), das, ausgelöst durch ein `RadioEvent`, Radionachrichten an die OBU verschicken kann. Das `V2XSocket` stellt eine Implementierung des Qt `QAbstractSockets` dar, das Funktionen für die Kommunikation über die Protokolle TCP und **U**ser **D**atagram **P**rotocol (UDP) zur Verfügung stellt. Das `V2XInterface` beinhaltet ein Objekt vom Typ `V2XSocket` und wird mit dem in Kapitel 5.2 beschriebenen `V2XEvent` über das Qt Signal-Slot-Konzept verknüpft. Wird ein `V2XEvent` durch den TEP ausgelöst, erfolgt die Übertragung der in der Tabelle TC_Step enthaltenen Fahrzeuginformationen an das `V2XInterface`. Das `V2XInterface` formatiert die erhaltenen Informationen, fügt einen Zeitstempel hinzu und versendet die gesamte Nachricht über das `V2XSocket` an eine gegebene Adresse. Die Definition der Adresse und des Ports des Empfängers erfolgt innerhalb der Konfigurationsdatei von SEFEV.

Auf der Seite des V2X Stacks respektive der Cohda MK5 OBU wird eine TCP Kommunikationsschnittstelle in Form eines TCP Servers in C++ implementiert (siehe Abbildung 6.1). Der TCP Server wird in die Anwendung zur Generierung von Nachrichten, die in Kapitel 6.3 erläutert wird, integriert. Innerhalb des TCP Servers werden zunächst der Port, die Hostadresse und das zu verwendende Protokoll festgelegt. Als Kommunikationsprotokoll wird IPv4 verwendet. Im Anschluss daran wird ein Socket mit den festgelegten Informationen erstellt. Daraufhin wartet das Socket auf einen Verbindungsaufbau durch einen Client. Ist ein Client verbunden, werden Nachrichten mit einer Länge von 64 Bytes empfangen und weiterverarbeitet. Die Weiterverarbeitung erfolgt durch die nachfolgend beschriebene Anwendung zur Generierung von V2X Nachrichten.

6.3. Anwendung zur Generierung von V2X Nachrichten

Die Funktionalität zur Generierung von CAMs und DENMs wird in die Anwendung *socktap* von Vanetza integriert (siehe Abbildung 6.1), sodass sie mit der CMake Option `-DBUILD_SOCKTAP` gebaut werden kann.

Nach dem Start der Anwendung werden zunächst der TCP Server (siehe Kapitel 6.2) und die Laufzeitumgebung innerhalb von Vanetza gestartet. Wird eine Nachricht über TCP empfangen, werden je nach ermitteltem Nachrichtentyp (durch die erste Stelle der empfangenen Nachricht) Funktionen zur Generierung einer CAM oder DENM aufgerufen. Die Generierung einer CAM erfolgt durch die Zuweisung von Werten zu den nach ASN.1 definierten Variablen (siehe Anhang C), die in Vanetza innerhalb des Namespaces `vanetza::asn1::Cam` zu finden sind. Innerhalb des *BasicContainers* werden die Position und die Geschwindigkeit der sendenden ITS angegeben. Hierfür werden die Positionswerte, die als `double` übergeben wurden, in `GeoAngles` umgewandelt sowie der Geschwindigkeitswert mit der *Boost*[4]-Einheit `meter_per_second` multipliziert. Eine DENM wird ebenfalls durch die Zuweisung von Werten generiert. Ein Unterschied zwischen der Generierung einer CAM und DENM ist jedoch, dass bei einer DENM die einzelnen Nachrichtensektionen (siehe Kapitel 2.1.3) zuvor alloziert werden müssen. Dies ist darin begründet, dass für Definition einer DENM die Managementsektion ausreichend ist, wobei die Situations- und Ortssektion optional sind. Im Anschluss daran wird sowohl bei einer CAM als auch bei einer DENM die generierte Nachricht über den Vanetza Protokoll Stack verschickt, der neben dem vorhandenen Cohda V2X Stack auf der Cohda MK5 OBU installiert wurde. Da die Generierung von V2X Nachrichten in der Applikationsebene erfolgt, wird zunächst ein `DownPacket` innerhalb der Applikationsebene erstellt und die Nachricht über den `move`-Konstruktor in einen `ByteBuffer` innerhalb des `DownPackets` verschoben. Nachfolgend wird ein BTP `DataRequest` (siehe ASN.1 Darstellung in Abbildung C.1) erstellt. Die Länge der Nachricht wird in Form des Enums `vanetza::aid::DEN` bzw. `vanetza::aid::CAM` angegeben. Als Transport Typ wird `SHB` (Single Hop

[4]C++-Bibliothek (`boost.org`)

Broadcast) und als Kommunikationsprofil `ITS_G5` festgelegt. Der `DataRequest` und das Paket werden über den `Router` an den `DCCPassthrough`[5] weitergeleitet, in dem ein hardwarespezifischer Header (beispielsweise für die Cohda MK5) hinzugefügt und das Paket an die definierte Adresse (hier: Broadcast) gesendet wird.

Die Integration von Vanetza Socktap in die Cohda MK5 OBU erfolgt nach dem in der Dokumentation von Vanetza beschriebenen Verfahren[6], das im Folgenden kurz zusammengefasst wird:
Zunächst werden Vanetza und vorkompilierte Abhängigkeiten innerhalb des Cohda SDKs heruntergeladen. Daraufhin erfolgt die Anpassung von Paketvariablen, sodass in der Cohda MK5 OBU vorinstallierte Abhängigkeiten verwendet werden können. Anschließend werden Vanetza und Vanetza Socktap gebaut und ausführbare Dateien erzeugt. Der letzte Schritt der Integration ist das Kopieren der ausführbaren Dateien und der vorkompilierten Bibliotheken auf die Cohda MK5 OBU.

[5] DCC (Decentralized Congestion Control) ist ein Mechanismus zur Verhinderung der Überlastung von Übertragungskanälen.[21]

[6] `vanetza.org/recipes/cohda-sdk-build`

7. Validierung

Die Validierung dieser Bachelorarbeit fokussiert sich auf die folgenden Aspekte:

(i) Die Betrachtung der Implementierung des OpenSCENARIO Parsers und der Integration des OpenSCENARIO Standards, beschrieben in Kapitel 5.1. Das Hauptaugenmerk liegt hierbei einerseits auf der Abbildung der in Kapitel 3.2.3 vorgestellten Elemente des OpenSCENARIO Standards und andererseits auf der Berechnung der dynamischen Fahrzeuginformationen und die Erstellung der Testsequenz, deren Aufbau in Kapitel 4.1 erläutert wurde.

(ii) Die Analyse bzw. Validierung der mit dem V2X Modul generierten V2X Nachrichten, dessen Entwicklung in Kapitel 6 dargelegt wurde. Die Grundlagen für die Analyse stellen die Informationen über den Aufbau des V2X Protokoll Stacks und der V2X Nachrichten, erläutert in Kapitel 2.1.2 und 2.1.3, dar. Die Validierung der gesendeten Nachrichten erfolgt zunächst mit dem Paket Sniffer[1] *tcpdump*[2] und anschließend mit einem Analyse Tablet für V2X Nachrichten (WaveBee touch [8]).

(iii) Die Bewertung des Konzepts (siehe Kapitel 4.4) und der Umsetzung der gesamten, im Rahmen dieser Arbeit entwickelten Werkzeugkette, erläutert in Kapitel 5 und 6. Die Bewertung erfolgt anhand der in Kapitel 4.3 aufgestellten Anforderungen.

[1] Paket-Sniffing bezeichnet das Protokollieren von Paketen an einer Netzwerkschnittstelle, unabhängig von der Adressierung des Pakets.[48]

[2] `tcpdump.org`

7.1. OpenSCENARIO Parser

Der OpenSCENARIO Parser ist ein Modul zur Generierung einer Testsequenz aus einem eingelesenen Szenario im OpenSCENARIO Format. Die Testsequenz beinhaltet Events für gesendete CAMs und DENMs der im Szenario enthaltenen Fahrzeuge während des gesamten Szenarios. Die Implementierung des OpenSCENARIO Parsers erfolgte mit der Programmiersprache Python in der Version 3.7. Da an die Laufzeit der Simulation für die Berechnung der dynamischen Fahrzeuginformationen innerhalb eines Szenarios keine Anforderung gestellt wurde und die Simulation nicht, wie bei dem Szenario Parser von Pilz [30], mit einer weiteren Simulationsumgebung synchronisiert werden muss, ist die Programmiersprache Python ausreichend schnell genug.

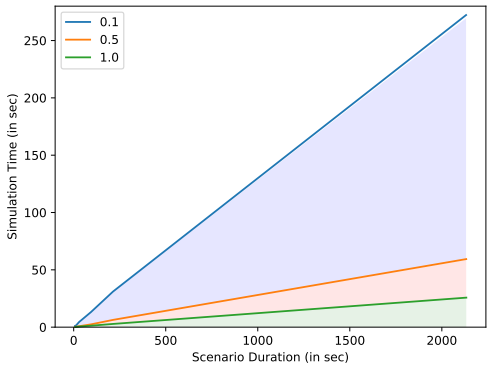
In Abbildung 7.1(a) ist die Dauer für die Erstellung der Testsequenz innerhalb des OpenSCENARIO Parsers in Abhängigkeit von der Dauer des enthaltenen Szenarios (bedingt durch die Geschwindigkeit und Zielpositionen der Fahrzeuge) dargestellt. Die benötigte Zeit ist für die Zeitabstände 0,1 s, 0,5 s und 1 s zwischen den Simulationsschritten dargestellt. Die Angabe von Positionen innerhalb des Szenarios erfolgt als Distanz in Metern zu einer Referenzposition. Die Dauer der Datenbankzugriffe ist als farbige Fläche unterhalb des zugehörigen Graphen dargestellt. Erkennbar ist, dass der Datenbankzugriff mit ca. 99% fast den kompletten Teil der Simulationszeit in Anspruch nimmt. Die Graphen weisen einen annähernd linearen Verlauf auf, wobei der Anstieg der benötigten Zeit bei einem Zeitschritt von 0,1 s in Abhängigkeit von der Dauer des Szenarios am höchsten ist. Der schnelle Anstieg in Abbildung 7.1(a) ist darin begründet, dass die Anzahl der Datenbankzugriffe pro Sekunde des Szenarios um $Anzahl\_Non\_Ego\_Fahrzeuge \times 10$ (ohne DENM-Events) ansteigt, da pro Simulationsschritt für jedes Non-Ego Fahrzeug ein CAM Event erstellt wird. Zur Verbesserung der Laufzeit wurde daher nachträglich die Datenbankzugriffe betrachtet und optimiert.

In den Abbildungen 7.1(b) bis 7.1(f) ist die Laufzeit der Simulation mit optimierten Datenbankzugriffen dargestellt. Die Optimierung wurde bei der Tabelle TC_Step angewandt und zeichnet sich dadurch aus, dass neue Einträge zunächst in einer Liste

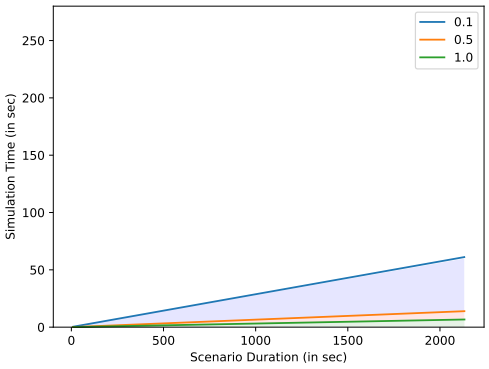
gespeichert werden. Nach einer bestimmten Anzahl von Simulationsschritten werden die gesammelten Einträge mittels der *sqlite3* Funktion `executemany()` anstelle der Funktion `execute()` hinzugefügt, um mehrere Datenbankzugriffe gleichzeitig auszuführen. Die Anzahl der Simulationsschritte, die zwischen der Ausführung des SQL Statements liegen, sind in der Bildunterschrift der Abbildungen zu sehen. Die Abbildung 7.2 fasst die Abbildungen 7.1(a) bis 7.1(f) zusammen und stellt die Simulationszeit in Abhängigkeit von den übersprungenen Simulationsschritten dar. Zu sehen ist, dass die Verringerung der Simulationszeit zwischen dem Datenbankzugriff in jedem Simulationsschritt (Ausführung der Funktion `execute()`) und 10 übersprungenen Simulationsschritten (Ausführung von `executemany()`) am stärksten ist. Insgesamt konnte die Simulationszeit durch den optimierten Datenbankzugriff ca. um den Faktor 11 reduziert werden.

Als Anforderung an den OpenSCENARIO Parser wurde in Kapitel 4.3 festgehalten, dass die Elemente `TeleportAction`, `SpeedAction`, `FollowTrajectoryAction`, `ReachPositionCondition`, `SpeedCondition` und `SimulationTimeCondition` unterstützt werden sollen. In Anhang D.2 dieser Arbeit sind zwei Szenarien zu finden, welche die genannten Elemente beinhalten und zur Validierung des OpenSCENARIO Parsers verwendet werden. Die Implementierung der Aktionen wurde im Rahmen dieser Arbeit insofern erfolgreich umgesetzt, als dass `TeleportActions` und `SpeedActions` innerhalb des Abschnitts `Init` des Szenarios das Fahrzeug mit einer definierten Position bzw. Geschwindigkeit initialisieren. Innerhalb der `Story` sind beide genannten Aktionen ebenfalls so implementiert, dass ein Fahrzeug während der laufenden Simulation an eine bestimmte Position gesetzt wird bzw. eine Geschwindigkeit erhält, auch wenn dies in der Realität nicht möglich ist. Eine Vorgabe, dass `TeleportActions` oder `SpeedActions` außerhalb der `Init` als Zielwert eines Fahrzeugs anzusehen sind, wird von ASAM e.V. nicht genannt [10].

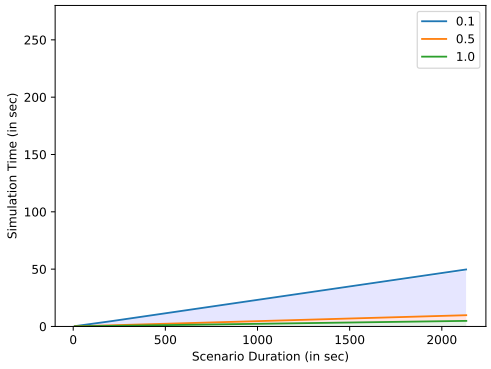
Die Implementierung der OpenSCENARIO `WorldPosition` ist auf zwei Weisen erfolgt: Zum einen können innerhalb des in OpenSCENARIO definierten Szenarios Positionen als geografische Breite und Länge angegeben werden. Zum anderen können Fahrzeugpositionen als Distanz zu einer geografischen Referenzposition im Szenario definiert werden. Die Referenzposition, anhand der die Aktualisierung



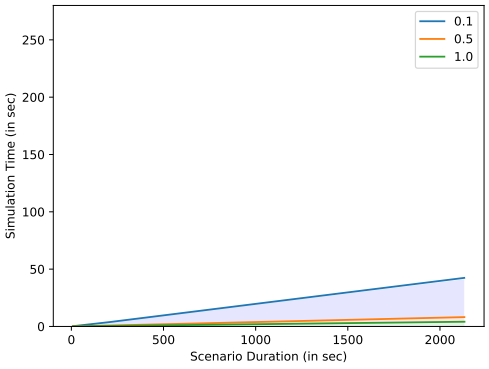
(a) Jeder Simulationsschritt



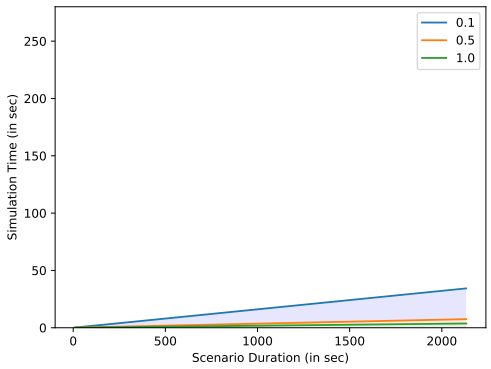
(b) 10 Simulationsschritte



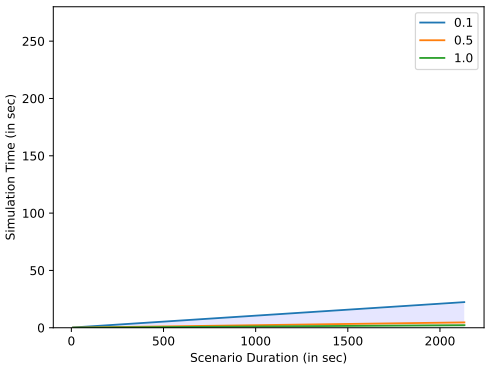
(c) 20 Simulationsschritte



(d) 50 Simulationsschritte



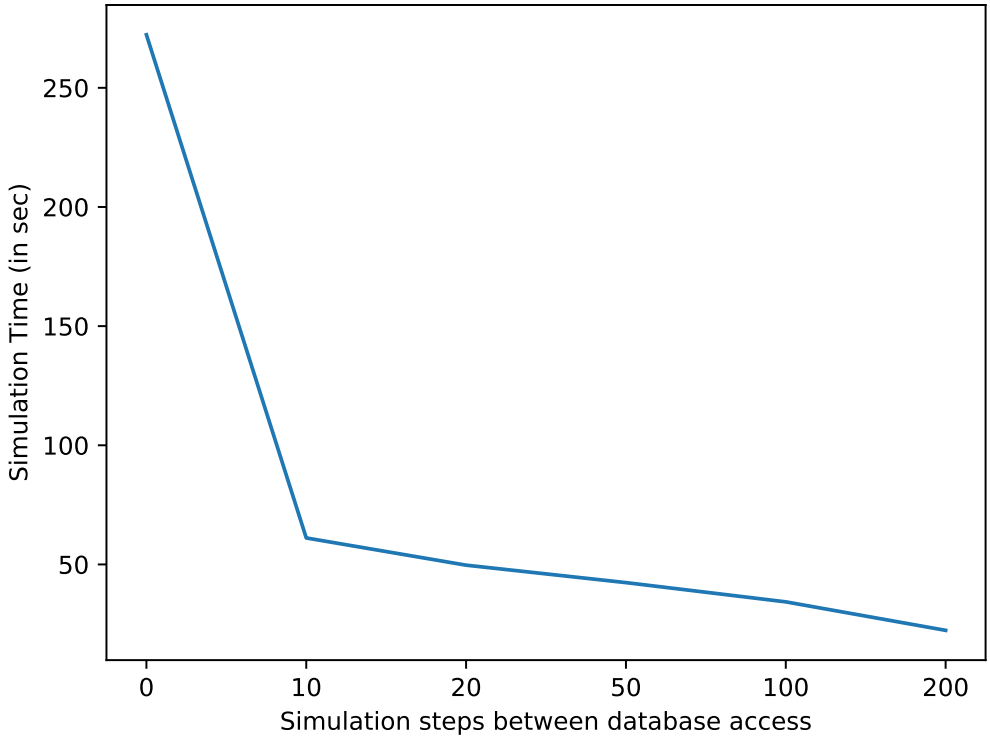
(e) 100 Simulationsschritte



(f) 200 Simulationsschritte

Abbildung 7.1.: Benötigte Zeit zur Erstellung der Testsequenz in Abhängigkeit von der Szenariodauer

Abbildung 7.2.: Simulationszeit in Abhängigkeit von übersprungenen Simulationsschritten (Zeitschritt: 0.1)



der Fahrzeugposition während der Simulation erfolgt, ist mit Hilfe der grafischen Oberfläche des Parsers veränderbar.

Die Erstellung von CAM Events in der Tabelle TC_Step innerhalb der Testsequenz erfolgt ein Mal pro Simulationsschritt. Für jedes enthaltene Non-Ego Fahrzeug wird dabei ein CAM Event hinzugefügt. Der zeitliche Abstand zweier Simulationsschritte kann vor dem Beginn der Simulation in der grafischen Oberfläche verändert werden. Da der Maximalwert eine Sekunde beträgt, ist sichergestellt, dass sämtliche simulierten Fahrzeuge mindestens ein Mal pro Sekunde eine CAM senden. DENM Events werden ebenfalls anhand der im Szenario festgelegten Anzahl an Wiederholungen zur Testsequenz hinzugefügt.

Zusammenfassend kann der entwickelte Parser für Szenarien im OpenSCENARIO Format die gestellten Anforderungen erfüllen. Der Parser kann zukünftig für die Umwandlung von OpenSCENARIO Daten in für SEFEV lesbare Testsequenzen verwendet werden.

7.2. Generierung der V2X Nachrichten

Die Validierung des V2X Moduls ist nach [31] in erster Linie als Test der Applikationsschicht zu betrachten, da die Generierung von V2X Nachrichten, wie einer CAM bzw. DENM, innerhalb der Applikationsschicht des V2X Protokoll Stacks erfolgt. Der Fokus liegt hierbei zum einen auf der Überprüfung, ob die V2X Nachrichten von der sendenden Station (Cohda MK5 OBU mit V2X Modul) korrekt enkodiert und daraufhin von einem Empfänger decodiert werden können (Interoperabilitätstest). Zum anderen werden die Geonetworking- und die Nachrichtenebene der empfangenen V2X Nachrichten analysiert. Die Validierung teilt sich in die zwei folgenden Aspekte auf:

Der erste Schritt ist Überprüfung der Generierung von V2X Nachrichten innerhalb des Vanetza Protokoll Stacks und die Übertragung der generierten Nachrichten an das ITS G5 Modem der Cohda MK5 OBU, welches die Kommunikation über den 802.11p Standard ermöglicht. Die Analyse wird mit dem Paket Sniffer *tcpdump* vorgenommen,

der die Ethernet Netzwerkschnittstelle zwischen dem Vanetza Protokoll Stack und dem ITS G5 Modem der Cohda MK5 OBU überwacht. Der zweite Schritt ist die Validierung der von dem V2X Modul respektive dem ITS G5 Modem der Cohda MK5 OBU ausgesendeten V2X Nachrichten. Die Validierung erfolgt mit einer WaveBee touch (siehe Abbildung A.8), einem Analyse Tablet für V2X Kommunikation, das in der Nähe der Cohda MK5 OBU platziert wird und die generierten V2X Nachrichten empfangen soll.

Die Durchführung der ersten Stufe der Validierung (Analyse der V2X Nachrichten, die vom Vanetza Stack zum ITS G5 Modem der Cohda übertragen werden) erfordert, dass der gewünschte Inhalt der generierten V2X Nachrichten im Vorhinein klar festgelegt wird. Aus diesem Grund und um eine Unabhängigkeit von möglichen Fehlfunktionen der Laborsoftware SEFEV zu gewährleisten, wird die erste Validierung nicht mit einer durch SEFEV simulierten, vollständigen Testsequenz, sondern mit einem simplen TCP Client, der periodisch eine fest definierte Nachricht an die TCP Schnittstelle des V2X Moduls sendet, durchgeführt. Die TCP Nachrichten weisen das Format der Nachrichten auf, die von SEFEV, während der Simulation der Testsequenz, an das V2X Modul gesendet werden. Die Reihenfolge der enthaltenen Informationen, die entweder die Generierung einer CAM oder DENM im V2X Modul auslösen sollen, ist in den Abbildungen 5.3 bzw. 5.4 zu sehen. Für die Validierung wird an die abgebildeten Informationen zusätzlich ein Zeitstempel, welcher im Normalfall durch SEFEV generiert wird, angehängt. Das V2X Modul erhält demnach in diesem Schritt zunächst periodisch eine festgelegte Nachricht vom TCP Client zur Generierung einer CAM und einer DENM. Der Paket Sniffer *tcpdump* wird an der Ethernet Netzwerkschnittstelle zwischen dem Vanetza Protokoll Stack und dem ITS G5 Modem der Cohda MK5 OBU eingesetzt. Die übertragenen Pakete werden von *tcpdump* mitgeschnitten und als **p**acket **cap**ture (pcap)[3] gespeichert.

Die Analyse der übertragenen Pakete wird mit einer speziell für V2X Nachrichten ausgelegten Version von *Wireshark*[4] vorgenommen. Die Abbildungen D.1 und D.2 zeigen

[3] pcap ist eine Programmierschnittstelle für das Mitschneiden von Paketen in einem Netzwerk. Die pcap Funktionalität wird in unixartigen Betriebssystemen über die libpcap Bibliothek und unter Windows mit WinPcap umgesetzt. [27]

[4] Version 2.2.2-Cohda-106051, `wireshark.org`

eine vom Vanetza Stack an das ITS G5 Modem der Cohda MK5 OBU übertragene DENM bzw. CAM. Zu sehen ist, dass beide V2X Nachrichtenformate vollständig von Wireshark decodiert werden konnten, wodurch die korrekte Übertragung der generierten V2X Nachrichten vom Vanetza Stack zur Cohda belegt ist.

Die zweite Stufe der Validierung, die Überprüfung der korrekten Aussendung der Nachrichten über die Schnittstelle IEEE 802.11p, erfolgt mit Hilfe einer WaveBee touch [8], die das Empfangen, Aufzeichnen und Analysieren von V2X Kommunikation live ermöglicht. Da die korrekte Generierung der V2X Nachrichten im Vanetza Protokoll Stack innerhalb des V2X Moduls bereits bewiesen ist, wird für die Validierung der korrekten Aussendung der V2X Nachrichten über das ITS G5 Modem der Cohda MK5 OBU eine vollständige Testsequenz, generiert durch den OpenSCENARIO Parser, mit SEFEV simuliert und in der Testsequenz enthaltene CAM bzw. DENM Events über TCP an das V2X Modul gesendet. Das zugrundeliegende Szenario der Testsequenz ist das erste Szenario im Anhang D.2 dieser Arbeit. Die Live-Auswertung der empfangenen V2X Nachrichten mit der WaveBee touch ist in Abbildung A.8 zu sehen. Auf der Karte sind die drei CAMs dargestellt, die durch das V2X Modul generiert und versendet wurden. Zu sehen ist, dass die Nachrichten als CAM identifiziert und dekodiert wurden. Die Information, dass es sich bei den Sendern um mehrere „Passenger Cars“ handelt, ist ebenfalls ersichtlich. Empfangene Nachrichten können mit der WaveBee touch zur weiteren Analyse aufgezeichnet und als pcap übertragen werden. Die Abbildungen D.3 und D.4 zeigen die Auswertung der mit der WaveBee touch empfangenen Nachrichten in Wireshark. Zu sehen ist, dass sowohl in der Geonetworking- als auch in der Nachrichtenebene die Positionsinformationen dekodiert werden können. Bei der empfangenen DENM sind zusätzlich u. a. innerhalb der Managementsektion der Stationstyp und in der Situationssektion die Informationsqualität und der *Cause Code* zu sehen, die korrekt übertragen wurden. In der CAM sind u. a. der Zeitstempel, die Geschwindigkeit und die Fahrrichtung des Fahrzeugs auf Werte aus dem in OpenSCENARIO definierten Szenario gesetzt.

Die gezeigten Analysen der übertragenen Pakete (Anhang D.3) belegen, dass das entwickelte V2X Modul valide V2X Nachrichten der Formate CAM und DENM generieren und an ITS in seiner Umgebung versenden kann.

7.3. Bewertung der gesamten Werkzeugkette

Die Werkzeugkette zur Generierung von V2X Nachrichten aus OpenSCENARIO Daten beinhaltet einen OpenSCENARIO Parser und ein V2X Modul, die in die Laborsoftware SEFEV integriert sind. Die Erprobung der gesamten Werkzeugkette erfolgt, wie in Kapitel 4.4 beschrieben, mit einer virtuellen Maschine, in der die Laborsoftware SEFEV ausgeführt wird. Die durch den OpenSCENARIO Parser generierten Testsequenzen werden in einem *shared folder* der virtuellen Maschine abgelegt, sodass SEFEV diese einlesen kann. Die Verbindung der virtuellen Maschine mit der Cohda MK5 OBU erfolgt durch die in Anhang D.1.1 und D.1.3 beschriebenen Schritte. Die Simulation wird, wie in Anhang D.1.4 (Anleitung zur Konfiguration der einzelnen Module und der Laborumgebung SEFEV) erläutert, gestartet.

Zunächst wird die Benutzerfreundlichkeit der entwickelten Werkzeugkette betrachtet. Die Generierung der Testsequenz ist, bedingt durch die umfangreiche Funktionalität der grafischen Oberfläche des OpenSCENARIO Parsers, leicht durchzuführen. Die Funktionalitäten umfassen beispielsweise die Veränderung der Referenzposition und die Anpassung des zeitlichen Abstands der Simulationsschritte. Die Dauer der Berechnung der Fahrzeugpositionen umfasst bei kurzen Szenarien nicht mehr als 2 s (siehe Kapitel 7.1), wodurch der Benutzer nur eine sehr kurze Wartezeit hat.

Als zweiter Aspekt wird die Funktionsweise der gesamten Werkzeugkette validiert. Wie in Kapitel 7.1 und 7.2 erläutert, erfolgt sowohl die Generierung der Testsequenz als auch die Aussendung der V2X Nachrichten durch das V2X Modul und die Cohda MK5 OBU fehlerfrei. Die Integration der entwickelten Module in die HiL Laborumgebung SEFEV und die Ausführung der Werkzeugkette erfolgt ebenfalls ohne Probleme. Die gesendeten V2X Nachrichten (siehe Wireshark Analysen in Anhang D.3), welche die im eingelesenen Szenario definierten Fahrzeuge abbilden und sämtliche übertragenen Informationen beinhalten, belegen, dass die entwickelte Werkzeugkette korrekt funktioniert. Die Zielsetzung dieser Bachelorarbeit, die Generierung von V2X Nachrichten aus OpenSCENARIO Daten ist demnach erreicht.

In Bezug auf die themenbezogenen Arbeiten sowohl zur V2X Kommunikation als auch zum OpenSCENARIO Standard lässt sich die entwickelte Werkzeugkette einordnen.

Mit der entwickelten Werkzeugkette lassen sich, ähnlich zu den vorgestellten Arbeiten [25, 34, 36] V2X Nachrichten der Formate CAM und DENM generieren und versenden. Ein neuer Ansatz, der innerhalb der Werkzeugkette umgesetzt wurde, ist die Verwendung des OpenSCENARIO Standards zur Beschreibung des Verkehrsszenarios, das der simulierten V2X Kommunikation zugrunde liegt. Die Integration des OpenSCENARIO Standards in V2X Simulationsframeworks in keiner der vorgestellten Arbeiten erfolgt.

Auf den Aspekt der Wirtschaftlichkeit der entwickelten Werkzeugkette wurde im Rahmen dieser Arbeit kein Fokus gelegt. Der größte Kostenpunkt innerhalb der Werkzeugkette stellt das V2X Modul respektive die Hardware für die Kommunikation über den IEEE Standard 802.11p dar. Die Cohda MK5 OBU wurde als Hardwareeinheit gewählt, da sie dem Institut für Verkehrssystemtechnik zum Zeitpunkt der Arbeit zur Verfügung stand. Die Verwendung von anderen, möglicherweise günstigeren, Kommunikationseinheiten wird daher lediglich im Ausblick dieser Arbeit diskutiert.

8. Schlussbetrachtung

Die abschließende Betrachtung der vorliegenden Bachelorarbeit „Konzeption und Entwicklung eines Moduls für die Generierung von V2X Nachrichten aus OpenSCENARIO Daten als Teil einer HiL Laborumgebung“ teilt sich in die Aspekte Fazit und Ausblick auf. Im Fazit werden die Entwicklungen im Rahmen dieser Bachelorarbeit zusammengefasst. Der Ausblick umfasst mögliche Weiterentwicklungen und Verbesserungen der entwickelten Module, die nach Abschluss dieser Arbeit umgesetzt werden können.

8.1. Fazit

Innerhalb dieser Bachelorarbeit wurden zunächst verschiedene Ansätze für die Generierung von V2X Nachrichten und Grundlagen der Verkehrssimulation mit dem OpenSCENARIO Standard betrachtet. Aufgrund der Themenstellung, Szenarien im OpenSCENARIO Format für die Generierung von V2X Nachrichten zu verwenden, wurden anschließend diesbezüglich existierende Arbeiten vorgestellt. Bedingt durch einen Mangel an Entwicklungen, die eine direkte Simulation von V2X Nachrichten zum Testen von ADAS aus OpenSCENARIO Daten beinhalten, wurde nachfolgend ein eigenes Konzept für die Integration eines solchen Moduls in die HiL Laborumgebung des Instituts für Verkehrssystemtechnik des DLR entwickelt. Die beschriebene HiL Laborumgebung, die zuvor lediglich der Validierung von Zugrechnern gegen die ERTMS Spezifikation durch die Simulation einer Zugstrecke diente, soll durch das Modul langfristig um die Möglichkeit der Validierung von ADAS erweitert werden. Das entwickelte Modul besteht aus den zwei Teilaspekten OpenSCENARIO Parser

und V2X Generierung, die in die HiL Laborumgebung eingebunden wurden. Die Kommunikation zwischen dem OpenSCENARIO Parser und der Software innerhalb der Laborumgebung wird über eine sogenannte Testsequenz umgesetzt, welche für die Steuerung der Eventsimulation für Zugrechner zur ERTMS Verifikation verwendet wird.

Der OpenSCENARIO Parser umfasst Funktionalitäten zum Einlesen und Parsen eines Szenarios im OpenSCENARIO Format, sowie die Simulation des Szenarios zur Ermittlung der dynamischen Informationen der enthaltenen Fahrzeuge. Die dynamischen Informationen, wie die Position und Geschwindigkeit werden anschließend an die Laborsoftware weitergeleitet. Die Laborsoftware simuliert das Szenario durch die Generierung von V2X Nachrichten für sämtliche enthaltenen Non-Ego-Fahrzeuge über das entwickelte V2X Modul. Das V2X Modul basiert auf dem Open Source Stack Vanetza, der in eine Cohda MK 5 OBU integriert wurde und unterstützt die Nachrichtenformate CAM und DENM.

Zusammenfassend wurde ein Modul zur Simulation der V2X Kommunikation innerhalb eines Szenarios im OpenSCENARIO Format entwickelt, das in eine HiL Laborumgebung integriert wurde. Mit dem entwickelten Modul können V2X Nachrichten an ein zu testendes System gesendet und die Reaktionen ausgewertet werden. Die generierten Nachrichten konnten erfolgreich mit einer WaveBee touch empfangen und dekodiert werden.

8.2. Ausblick

Aufbauend auf dem im Rahmen dieser Bachelorarbeit entwickelten Konzept und den umgesetzten Funktionalitäten, sind mehrere Weiterführungen denkbar. In erster Linie ist eine Erweiterung des OpenSCENARIO Parsers um zusätzliche Elemente des OpenSCENARIO Standards geplant. Dies führt dazu, dass auch aus Szenarien, die nicht den in Kapitel 4.3 genannten Kriterien entsprechen, V2X Nachrichten innerhalb der Laborumgebung generiert und an ein SUT gesendet werden können.

Bezüglich der Einbindung der OBU des SUT in die Laborumgebung ist es sinnvoll, die OBU per TCP an ein OBU Modul anzubinden. Durch das Modul TDL innerhalb von SEFEV, der die Simulation des Zuges (bzw. hier des Autos) übernimmt, kann die Selbstverortung des SUT innerhalb des Szenarios ermöglicht werden. Auf dieser Basis kann die Erprobung von automatisierten Fahrfunktionen, wie einem Bremsassistenten, in Abhängigkeit von empfangenen V2X Nachrichten durchgeführt werden. Der Anschluss eines Human Machine Interfaces für die Visualisierung von erhaltenen DENMs in Form einer Gefahrenmeldung für den Fahrzeugführer ist ebenfalls denkbar.

Eine dritte Option, die vorstellbar wäre, ist die Integration weiterer Nachrichtenformate, wie einer SPATEM oder IVIM (siehe Kapitel 2.1.3). Da der OpenSCENARIO Standard die Definition von Infrastrukturkomponenten ermöglicht, ist die Generierung von V2X Nachrichten, welche von beispielsweise Ampeln oder Baustellen ausgehen, möglich. Die Laborumgebung könnt somit neben den enthaltenen Fahrzeugen, die das SUT umgeben, die definierte Infrastruktur durch die Aussendung von V2X Nachrichten simulieren.

Der Aspekt Wirtschaftlichkeit der Werkzeugkette, der bereits in Kapitel 7 angeführt wurde, kann im Anschluss dieser Arbeit weiterführend betrachtet werden. Eine Idee ist hier der Austausch der verwendeten Hardware für die Bereitstellung der V2X Kommunikation über den ETSI Standard ITS G5 (Cohda MK5 OBU) durch eine kostengünstigere Alternative. Eine Option ist hier eine innerhalb des Instituts für Verkehrssystemtechnik entwickelte prototypische RSU [13]. Der Prototyp basiert auf einem Routerboard der Firma *PC Engines*[1] und stellt eine V2X Kommunikation über den OpenC2X Stack bereit. Die Kosten für die eigens entwickelte RSU betragen ca. 300€. Die günstigste kommerzielle Lösung hingegen ist für ca. 1750€ erwerbbar.[13]

[1] `pcengines.ch`

Literatur

[1] G. Bagschik. *Pegasus Project: Requirements and Conditions –Stand 4: Scenario Description.* URL: https://www.pegasusprojekt.de/files/tmpl/PDF-Symposium/04_Scenario-Description.pdf (besucht am 29. 06. 2020).

[2] Roberto Baldessari u. a. *Car-2-Car Communication Consortium - Manifesto.* Techn. Ber. 2007. URL: https://elib.dlr.de/48380/.

[3] D. S. Buse u. a. „Bridging worlds: Integrating hardware-in-the-loop testing with large-scale VANET simulation“. In: *2018 14th Annual Conference on Wireless On-demand Network Systems and Services (WONS).* 2018, S. 33–36.

[4] *CarMaker.* URL: https://ipg-automotive.com/de/produkte-services/simulation-software/carmaker/ (besucht am 23. 06. 2020).

[5] *Cohda Wireless.* URL: https://cohdawireless.com/solutions/v2x-stack/ (besucht am 23. 06. 2020).

[6] *Commsignia V2X Software Stack.* URL: https://www.commsignia.com/software/ (besucht am 23. 06. 2020).

[7] *Dedicated Short Range Communication for Transport Information and Control Systems. ARIB STD-T55 Version 1.0.* Standard. Association of Radio Industries und Businesses, 11/1997. URL: http://www.arib.or.jp/english/html/overview/doc/5-STD-T55v1_0-E.pdf.

[8] *DescriptionwaveBEE®touch.* Datasheet. Nordsys, 09/2016. URL: https://nordsys.de/images/sampledata/asimages/C2X/c2xdatasheets/NO_waveBEEtouch_db_09_2016_rev1.6_de.pdf.

[9] Alexey Dosovitskiy u. a. „CARLA: An Open Urban Driving Simulator“. In: *Proceedings of the 1st Annual Conference on Robot Learning.* 2017, S. 1–16.

[10] ASAM e.V. „ASAM OpenSCENARIO: User Guide“. In: *ASAM OpenSCENARIO. Dynamic content in driving simulation.* 2020. URL: https://www.asam.net/index.php?eID=dumpFile&t=f&f=3496&token=df4fdaf41a8463e585495001cc3db3298b57d426.

[11] *Environment Simulator Minimalistic (esmini).* URL: https://github.com/esmini/esmini (besucht am 23. 06. 2020).

[12] „Eyes on the Road“. In: *dSPACE Magazine* (01/2014), S. 44–52.

[13] Katharina Hartmann. *Aufbau und Konzeption eines Demonstrators für die Visualisierung von Vehicle-to-everything(V2X)-Nachrichten*. DLR e.V., 2019.

[14] Aaron Heinz, Wolfram Remlinger und Johann Schweiger. „Track- / Scenario-based Trajectory Generation for Testing Automated Driving Functions“. In: *8. Tagung Fahrerassistenz*. Lehrstuhl für Fahrzeugtechnik mit TÜV SÜD Akademie. München, 2017.

[15] Martin Herrmann. „Methodology for the generation and execution of scenarios for the virtual driving test with automated driving functions“. In: *Proceedings*. Springer Fachmedien Wiesbaden, 11/2019, S. 145–154. DOI: 10.1007/978-3-658-27990-5_13.

[16] „IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments“. In: *IEEE Std 802.11p-2010 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, IEEE Std 802.11n-2009, and IEEE Std 802.11w-2009)* (2010), S. 1–51.

[17] „IEEE Standard for Wireless Access in Vehicular Environments (WAVE)– Identifiers“. In: *IEEE Std 1609.12-2019 (Revision of IEEE Std 1609.12-2016)* (2019), S. 1–17.

[18] *Intelligent Transport Systems (ITS);Testing; Conformance test specifications for Facilities layer protocols and communication requirements for infrastructure services; Part 1: Test requirements and Protocol Implementation Conformance Statement (PICS) pro forma*. Standard. Sophia Antipolis Cedex - FRANCE: ETSI, 03/2017.

[19] *Intelligent Transport Systems (ITS);Vehicular Communications;Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service*. Standard. Sophia Antipolis Cedex - FRANCE: ETSI, 03/2011.

[20] *Intelligent Transport Systems (ITS);Vehicular Communications;Basic Set of Applications; Part 2: Specification of Decentralized Environmental Notification Basic Service*. Standard. Sophia Antipolis Cedex - FRANCE: ETSI, 09/2010.

[21] *Intelligent Transport Systems (ITS);Vehicular Communications;GeoNetworking; Part 3: Network architecture*. Standard. Sophia Antipolis Cedex - FRANCE: ETSI, 03/2010.

[22] *Intelligent Transport Systems (ITS);Vehicular Communications;GeoNetworking; Part 5: Transport Protocols; Sub-part 1: Basic Transport Protocol*. Standard. Sophia Antipolis Cedex - FRANCE: ETSI, 09/2014.

[23] Frank Köster, Jens Mazzega und Sascha Knake-Langhorst. „Automatisierte und vernetzte Systeme Effizient erprobt und evaluiert“. In: *ATZextra* 23.S5 (2018), S. 26–29. DOI: 10.1007/s35778-018-0040-9.

[24] Sven Laux u. a. „OpenC2X - An Open Source Experimental and Prototyping Platform Supporting ETSI ITS-G5“. In: *8th IEEE Vehicular Networking Conference (VNC 2016), Demo Session*. Hrsg. von Onur Altintas u. a. Columbus, OH: IEEE, 12/2016, S. 152–153. DOI: 10.1109/VNC.2016.7835955.

[25] T. Lee u. a. „Building a V2X Simulation Framework for Future Autonomous Driving“. In: *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. 2019, S. 1–6.

[26] Pablo Alvarez Lopez u. a. „Microscopic Traffic Simulation using SUMO“. In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 11/2018, S. 2575–2582. URL: https://elib.dlr.de/127994/.

[27] *Man page of PCAP*. URL: https://www.tcpdump.org/manpages/pcap.3pcap.html (besucht am 02. 09. 2020).

[28] *MK5 OBU*. Cohda Wireless. 08/2018. URL: https://www.cohdawireless.com/wp-content/uploads/2018/08/CW_Product-Brief-sheet-MK5-OBU.pdf.

[29] *Nordsys waveBEE*. URL: https://www.nordsys.de/v2x-produkte.html (besucht am 23. 06. 2020).

[30] Christoph Pilz. „Development of a Scenario Simulation Platform to Support Autonomous Driving Verification“. Magisterarb. Graz: Graz University of Technology, 2019.

[31] Fabian de Ponte Müller u. a. „Interoperability Testing Suite for C2X Communication Components“. In: *Nets4Cars/Nets4Trains 2011*. Hrsg. von Thomas Strang u. a. Bd. LNCS 6. Lecture Notes in Computer Science. Springer, März/2011, S. 203–215. URL: https://elib.dlr.de/69373/.

[32] *RailSiTe SEFEV Lab Manual*. English. DLR. 29. 08. 2013. 49 S.

[33] *RailSiTe: Railway Simulation an Testing*. URL: https://www.dlr.de/ts/desktopdefault.aspx/tabid-11368/19985_read-3254/ (besucht am 23. 06. 2020).

[34] Peter Reinold u. a. „Verkehrssimulation im Hardware-in-the-Loop-Steuergerätetest“. In: *Simulation und Test 2018*. Hrsg. von Johannes Liebl. Wiesbaden: Springer Fachmedien Wiesbaden, 2019, S. 253–269.

[35] Christian Ress und Martin Wiecker. „Potenzial der V2X-Kommunikation für Verkehrssicherheit und Effizienz“. In: *ATZ - Automobiltechnische Zeitschrift* 118.1 (2016), S. 16–21. DOI: 10.1007/s35148-015-0154-y. URL: https://doi.org/10.1007/s35148-015-0154-y.

[36] R. Riebl u. a. „Artery: Extending Veins for VANET applications“. In: *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*. 2015, S. 450–456.

[37] Jan Sauerbier u. a. „Definition von Szenarien zur Absicherung automatisierter Fahrfunktionen“. In: *ATZ - Automobiltechnische Zeitschrift* 121.1 (2019), S. 42–45. DOI: 10.1007/s35148-018-0205-2. URL: https://doi.org/10.1007/s35148-018-0205-2.

[38] Robert K. Schmidt, Tim Leinmüller und Dr. Bert Böddeker. „V2X Kommunikation“. In: *Aacherner Kolloquium Fahrzeug- und Motorentechnik* 17 (2008). URL: https://www.tu-ilmenau.de/fileadmin/media/telematik/schmidt/080718_V2X-Kommunikation.pdf.

[39] Dr. Gerd Schmitz. *Vehicle-to-X Communication and ADAS Test Solutions with CarMaker and the National Instruments Platform*. S.E.A. Datentechnik GmbH. 11. 09. 2018. URL: https://ipg-automotive.com/fileadmin/user_upload/content/Download/PDF/Events/Apply_Innovate_2018/Presentations/Apply_and_Innovate_2018_SEADatentechnik_Schmitz.pdf (besucht am 23. 06. 2020).

[40] C. Sommer, R. German und F. Dressler. „Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis“. In: *IEEE Transactions on Mobile Computing* 10.1 (2011), S. 3–15.

[41] Zsolt Szendrei, Norbert Varga und László Bokor. „A SUMO-Based Hardware-in-the-Loop V2X Simulation Framework for Testing and Rapid Prototyping of Cooperative Vehicular Applications“. In: *Vehicle and Automotive Engineering 2*. Hrsg. von Károly Jármai und Betti Bolló. Cham: Springer International Publishing, 2018, S. 426–440.

[42] *The Case for Cellular V2X for Safety and Cooperative Driving*. URL: https://5gaa.org/wp-content/uploads/2017/10/5GAA-whitepaper-23-Nov-2016.pdf (besucht am 23. 07. 2020).

[43] *Vanetza*. URL: https://www.vanetza.org/ (besucht am 23. 06. 2020).

[44] *Virtual test Drive*. URL: https://www.mscsoftware.com/de/virtual-test-drive (besucht am 07. 07. 2020).

[45] Jian Wang u. a. „A Survey of Vehicle to Everything (V2X) Testing“. In: *Sensors* 19.2 (01/2019), S. 334. DOI: 10.3390/s19020334.

[46] Axel Wegener u. a. „TraCI: An Interface for Coupling Road Traffic and Network Simulators“. In: *Proceedings of the 11th Communications and Networking Simulation Symposium*. CNS '08. Ottawa, Canada: Association for Computing Machinery, 2008, S. 155–163. DOI: 10.1145/1400713.1400740. URL: https://doi.org/10.1145/1400713.1400740.

[47] *What Is Hardware-in-the-Loop?* URL: https://www.ni.com/de-de/innovations/white-papers/17/what-is-hardware-in-the-loop-.html (besucht am 23. 06. 2020).

[48] *What is packet sniffing?* URL: https://www.paessler.com/it-explained/packet-sniffing (besucht am 02. 09. 2020).

[49] *xmlschema*. URL: https://xmlschema.readthedocs.io/en/latest/ (besucht am 02. 07. 2020).

[50] Adrian Zlocki u. a. „Ganzheitliche Werkzeugkette für die Entwicklung und Bewertung des automatisierten Fahrens“. In: *ATZextra* 23.S5 (2018), S. 16–21. DOI: 10.1007/s35778-018-0046-3. URL: https://link.springer.com/article/10.1007%2Fs35778-018-0046-3#citeas.

Anhang A.

Bilder

Abbildung A.1.: Struktur des Init-Elements in OpenSCENARIO

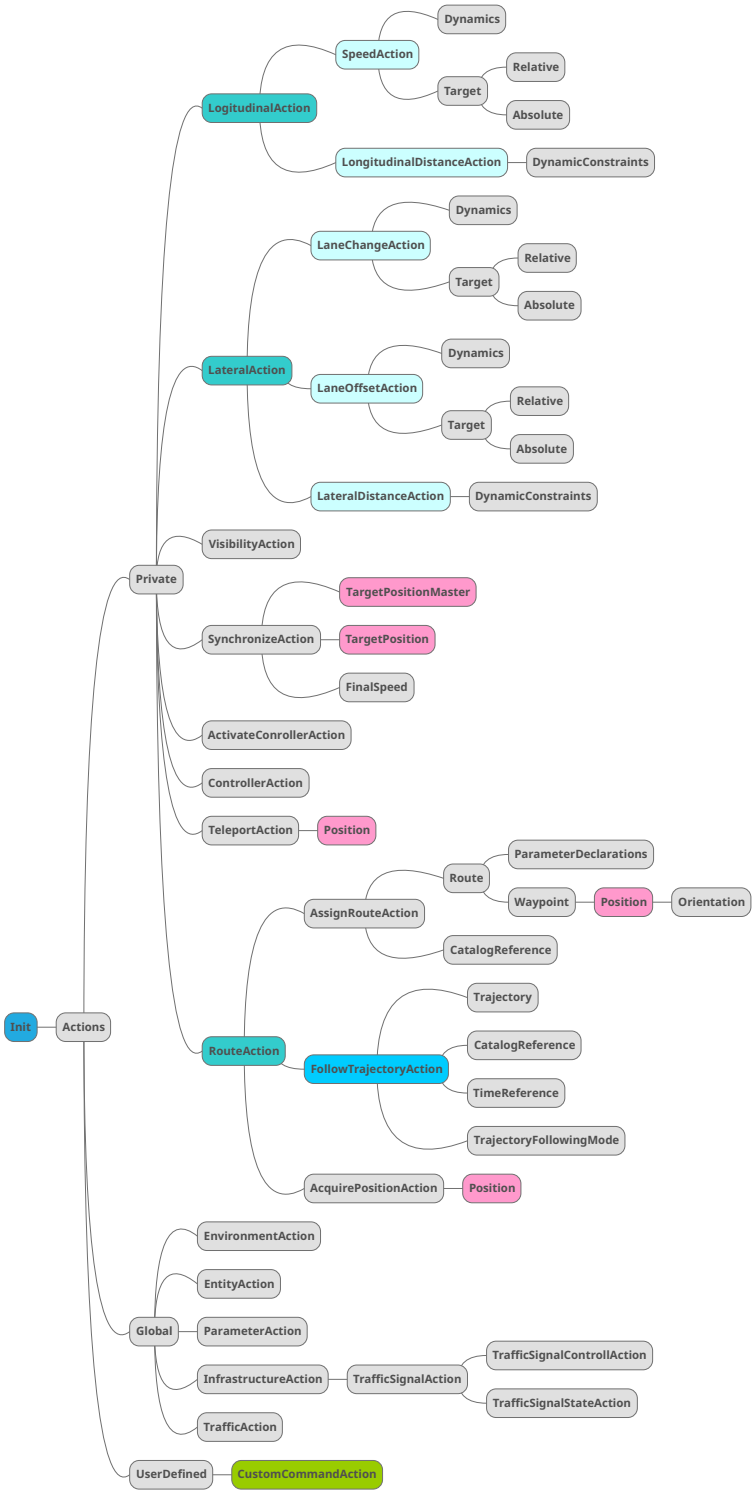


Abbildung A.2.: Struktur des Story-Elements in OpenSCENARIO

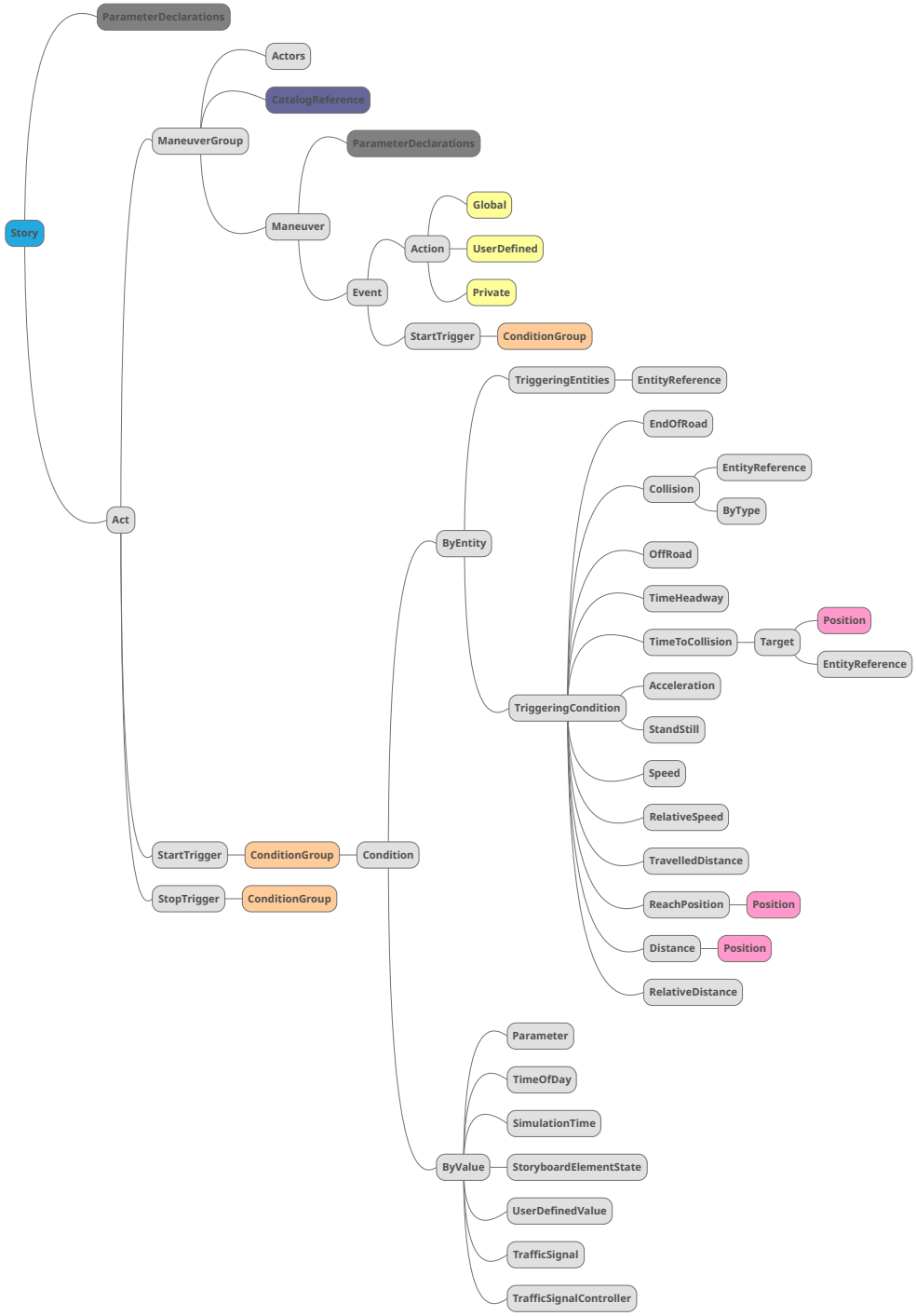


Abbildung A.3.: Popup Fenster zur Festlegung der initialen globalen Position

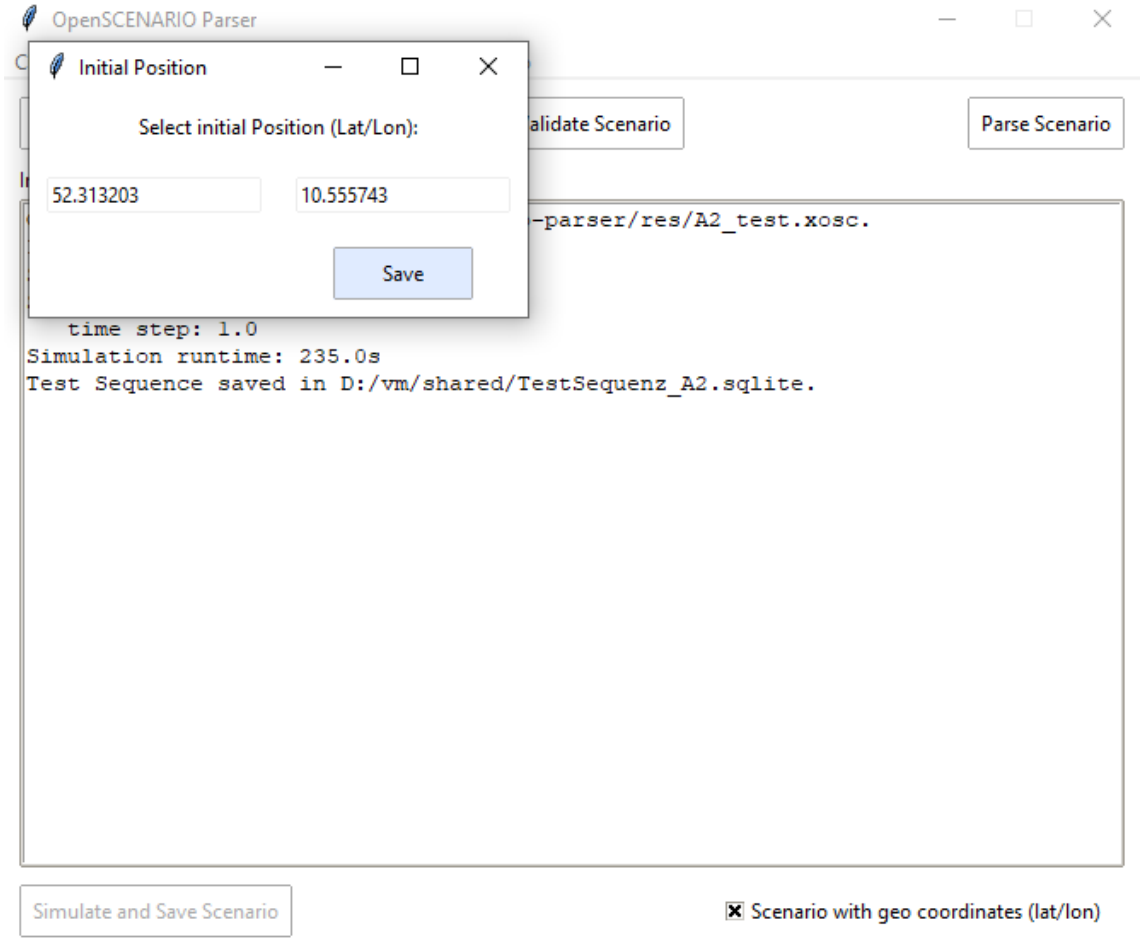


Abbildung A.4.: Popup Fenster zur Festlegung des zeitlichen Abstands zweier Simulationsschritte

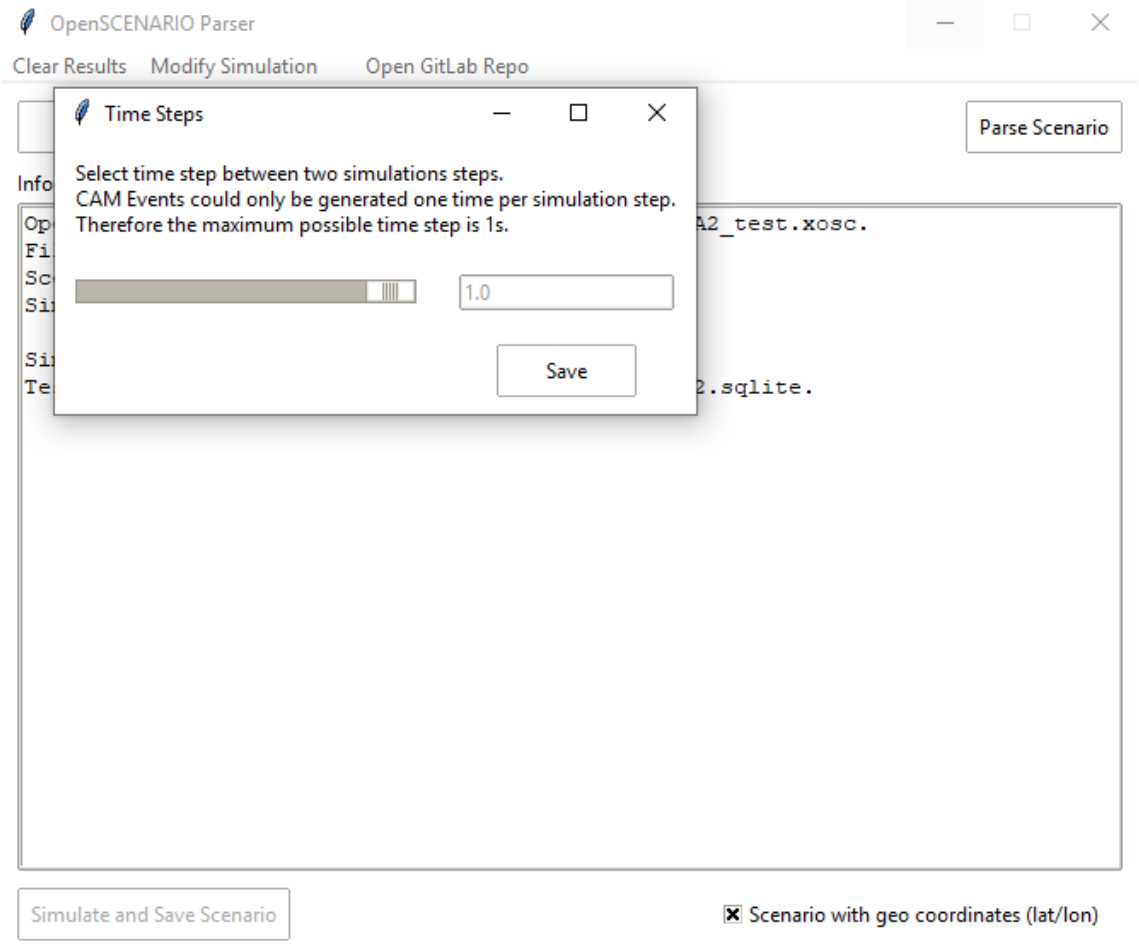


Abbildung A.5.: Format der Tabelle *SpeedProfile*

TestSequenceID	Zahl	ID of the Test Sequence
Row	Zahl	
Distance	Zahl	Absolute Distance
Speed	Zahl	Speed
StopTime	Zahl	Length of the Stop (if speed == 0)

Abbildung A.6.: Format der Tabelle *TC_Step*

TestModeOut	Text	
TestSequenceID	Zahl	ID of the Test Sequence
TCSOrder	Zahl	User-defined Order of this Test Case Step
Distance	Zahl	Absolute Distance of this Test Case Step
FT_NUMBER	Zahl	Feature number
TC_NUMBER	Zahl	Testcase number within feature
ST_ROW	Zahl	Order number of the step
ST_STEP	Text	Step of sequence within testcase
TFRAME_NUM	Zahl	Frame number
SUB_SEQ_NUM	Zahl	Sub Sequence number
ST_DESCRIPTION	Memo	Description of events
UserComment	Memo	
ST_IO	Text	Possible values are: None, I(nput), O(utput) or Both
ST_LANGUAGE	Text	
ST_INTERFACE	Text	Interface name
ST_COMMENTS	Memo	Comments
ST_RESULT	Text	Result of Sequence
ST_PREV_LEVEL	Text	Previous level
ST_PREV_MODE	Text	Previous mode
ST_NEXT_LEVEL	Text	Next level
ST_NEXT_MODE	Text	Next mode
Status	Zahl	0: normal, 1:moved, 2:deleted, 3:replacing
Reason	Memo	Reason why the step was deleted
TC_VERSION	Text	
ReplacingStep	Zahl	
TimeShift	Zahl	
Priority	Zahl	

Abbildung A.7.: Cohda MK5 OBU [28]

Abbildung A.8.: Empfangene V2X Nachrichten in der WaveBee touch

Anhang B.

Klassendiagramme

Abbildung B.1.: Grundlegende Klassenstruktur innerhalb des OpenSCENARIO Parsers

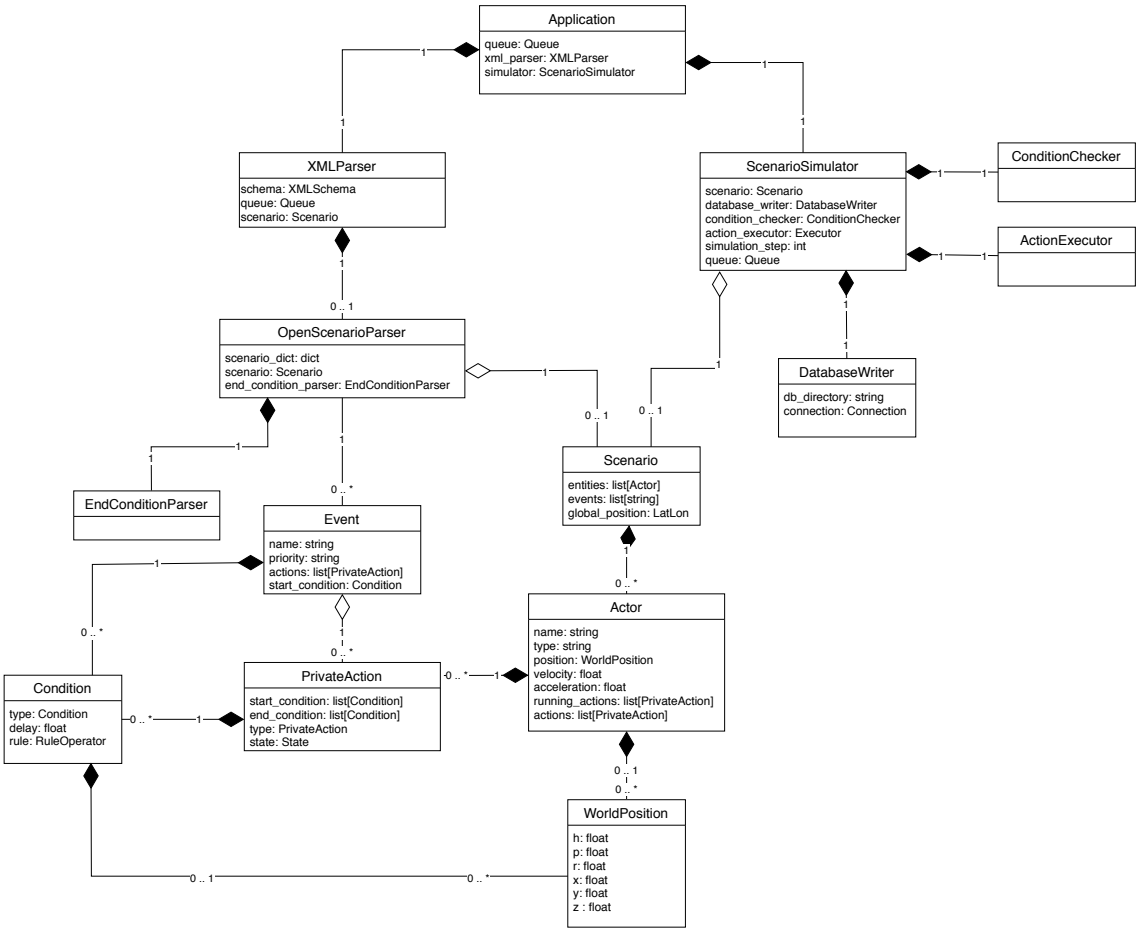


Abbildung B.2.: Klassendiagramm der `PrivateAction`

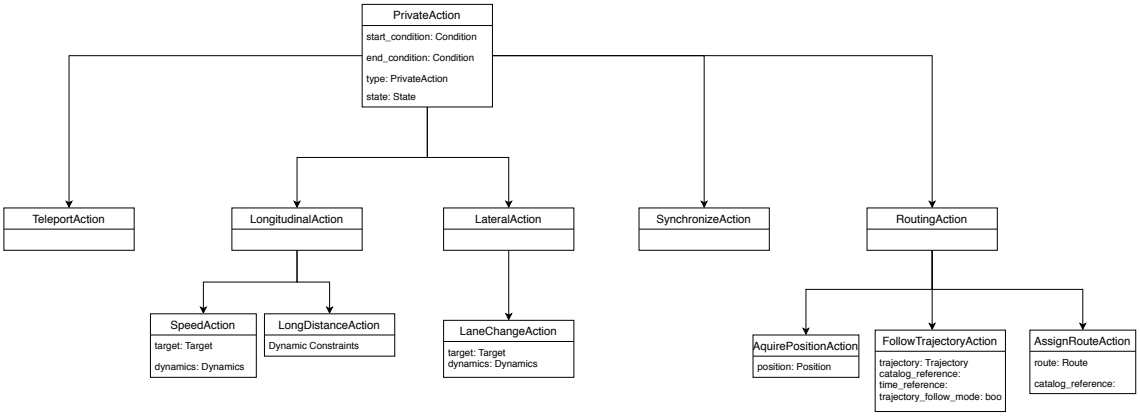


Abbildung B.3.: Klassendiagramm der `Condition`

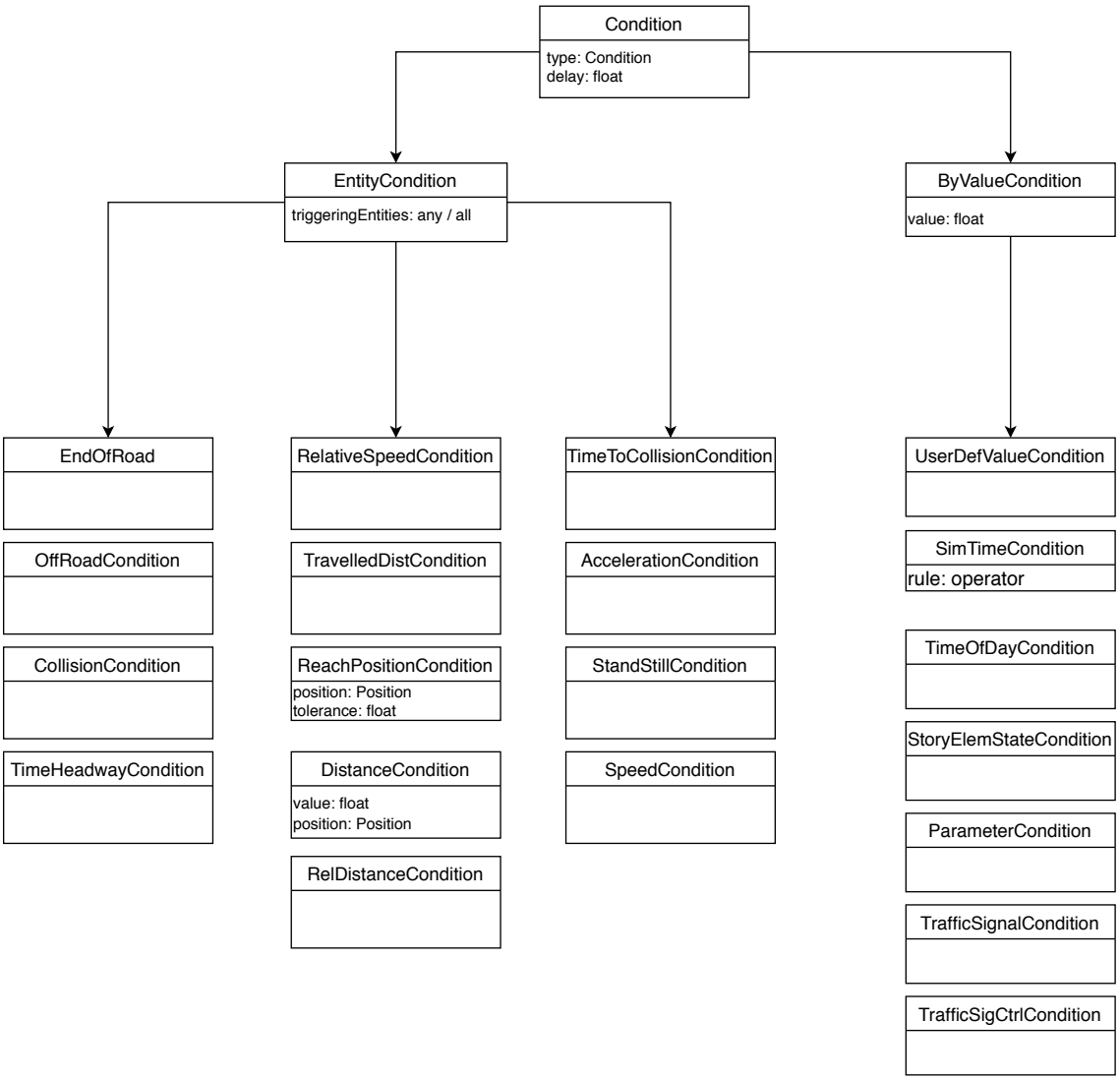
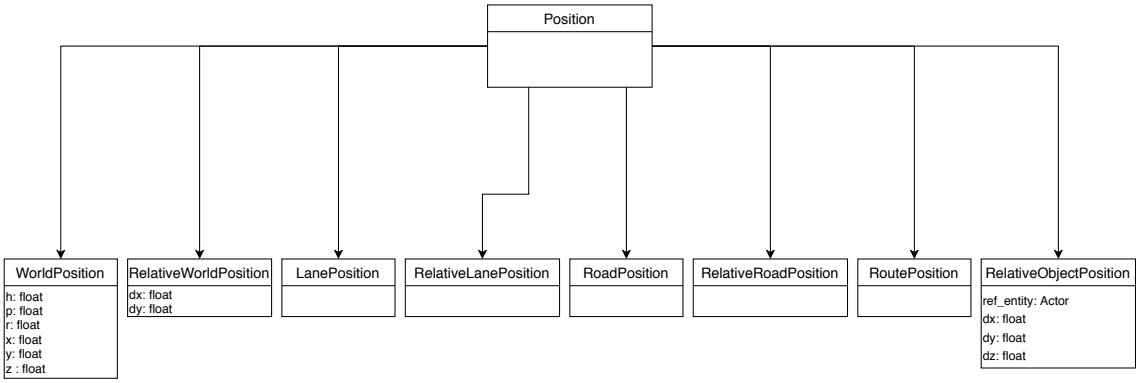


Abbildung B.4.: Klassendiagramm der `Position`



Anhang C.

ETSI Referenzen

Abbildung C.1.: Parameter eines BTP-`DataRequests` [22]

A.2 BTP-Data.request

The **BTP-Data.request** primitive is used by the ITS Facilities protocol entity to request sending a BTP-PDU. Upon reception of the **BTP-Data.request** primitive, the BTP protocol delivers the BTP-SDU to the GeoNetworking protocol entity via the GN-SAP.

The parameters of the **BTP-Data.request** are as follows:

BTP-Data.request (
BTP type,
Source port, (optional)
Destination port,
Destination port info, (optional)
GN Packet transport type,
GN Destination address,
GN Communication profile,
GN Security profile, (optional)
GN Maximum packet lifetime, (optional)
GN Repetition interval, (optional)
GN Maximum repetition time, (optional)
GN Maximum hop limit, (optional)
GN Traffic class,
Length,
Data
)

NOTE: *GN* in the primitive parameters indicates that this parameter is passed to the GeoNetworking protocol entity via the GN-SAP without being used by BTP.

Annex A (normative): ASN.1 specification of CAM

This annex provides the ASN.1 specification of CAM.

NOTE: Some of the optional data elements and data frames conditions for the availability are specified in annex B.

```
CAM-PDU-Descriptions {
itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) en (302637) cam (2) version (1)
}

DEFINITIONS AUTOMATIC TAGS ::=

BEGIN

IMPORTS
ItsPduHeader, CauseCode, ReferencePosition, AccelerationControl, Curvature,
CurvatureCalculationMode, Heading, LanePosition, EmergencyPriority, EmbarkationStatus, Speed,
DriveDirection, LongitudinalAcceleration, LateralAcceleration, VerticalAcceleration, StationType,
ExteriorLights, DangerousGoodsBasic, SpecialTransportType, LightBarSirenInUse, VehicleRole,
VehicleLength, VehicleWidth, PathHistory, RoadworksSubCauseCode, ClosedLanes, TrafficRule,
SpeedLimit, SteeringWheelAngle, PerformanceClass, YawRate, ProtectedCommunicationZone, PtActivation,
Latitude, Longitude, ProtectedCommunicationZonesRSU, CenDsrcTollingZone FROM ITS-Container {
itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (102894) cdd (2) version (1)
};


--  The root data frame for cooperative awareness messages

CAM ::= SEQUENCE {
    header ItsPduHeader,
    cam CoopAwareness
}

CoopAwareness ::= SEQUENCE {
    generationDeltaTime GenerationDeltaTime,
    camParameters CamParameters
}

CamParameters ::= SEQUENCE {
    basicContainer BasicContainer,
    highFrequencyContainer HighFrequencyContainer,
    lowFrequencyContainer LowFrequencyContainer OPTIONAL,
    specialVehicleContainer SpecialVehicleContainer OPTIONAL,
    ...
}

HighFrequencyContainer ::= CHOICE {
    basicVehicleContainerHighFrequency BasicVehicleContainerHighFrequency,
    rsuContainerHighFrequency RSUContainerHighFrequency,
    ...
}

 LowFrequencyContainer ::= CHOICE {
  basicVehicleContainerLowFrequency BasicVehicleContainerLowFrequency,
    ...
}

SpecialVehicleContainer ::= CHOICE {
    publicTransportContainer PublicTransportContainer,
    specialTransportContainer SpecialTransportContainer,
    dangerousGoodsContainer DangerousGoodsContainer,
    roadWorksContainerBasic RoadWorksContainerBasic,
    rescueContainer RescueContainer,
    emergencyContainer EmergencyContainer,
    safetyCarContainer SafetyCarContainer,
    ...
}

BasicContainer ::= SEQUENCE {
    stationType StationType,
    referencePosition ReferencePosition,
```

```
    ...
}

BasicVehicleContainerHighFrequency ::= SEQUENCE {
    heading Heading,
    speed Speed,
    driveDirection DriveDirection,
    vehicleLength VehicleLength,
    vehicleWidth VehicleWidth,
    longitudinalAcceleration LongitudinalAcceleration,
    curvature Curvature,
    curvatureCalculationMode CurvatureCalculationMode,
    yawRate YawRate,
    accelerationControl AccelerationControl OPTIONAL,
    lanePosition LanePosition OPTIONAL,
    steeringWheelAngle SteeringWheelAngle OPTIONAL,
    lateralAcceleration LateralAcceleration OPTIONAL,
    verticalAcceleration VerticalAcceleration OPTIONAL,
    performanceClass PerformanceClass OPTIONAL,
    cenDsrcTollingZone CenDsrcTollingZone OPTIONAL
}

BasicVehicleContainerLowFrequency  ::= SEQUENCE {
    vehicleRole VehicleRole,
    exteriorLights ExteriorLights,
    pathHistory PathHistory
}

PublicTransportContainer ::= SEQUENCE {
    embarkationStatus EmbarkationStatus,
    ptActivation PtActivation OPTIONAL
}

SpecialTransportContainer ::= SEQUENCE {
    specialTransportType SpecialTransportType,
    lightBarSirenInUse LightBarSirenInUse
}

 DangerousGoodsContainer ::= SEQUENCE {
  dangerousGoodsBasic DangerousGoodsBasic
 }

 RoadWorksContainerBasic ::= SEQUENCE {
  roadworksSubCauseCode RoadworksSubCauseCode OPTIONAL,
  lightBarSirenInUse LightBarSirenInUse,
  closedLanes ClosedLanes OPTIONAL
 }

RescueContainer ::= SEQUENCE {
    lightBarSirenInUse LightBarSirenInUse
}

EmergencyContainer ::= SEQUENCE {
    lightBarSirenInUse LightBarSirenInUse,
    incidentIndication CauseCode OPTIONAL,
    emergencyPriority EmergencyPriority OPTIONAL
}

SafetyCarContainer ::= SEQUENCE {
    lightBarSirenInUse LightBarSirenInUse,
    incidentIndication CauseCode OPTIONAL,
    trafficRule TrafficRule OPTIONAL,
    speedLimit SpeedLimit OPTIONAL
}

RSUContainerHighFrequency ::= SEQUENCE {
    protectedCommunicationZonesRSU ProtectedCommunicationZonesRSU OPTIONAL,
    ...
}

GenerationDeltaTime ::= INTEGER { oneMilliSec(1) } (0..65535)

END
```

Annex A (normative): ASN.1 specification of DENM

```
DENM-PDU-Descriptions {itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) en
(302637) denm (1) version (2)
}

DEFINITIONS AUTOMATIC TAGS ::=

BEGIN

IMPORTS
ItsPduHeader, CauseCode, Speed, InformationQuality, ReferencePosition, ClosedLanes,
DangerousGoodsExtended, Heading, LanePosition, LightBarSirenInUse, RoadType, HeightLonCarr,
PosLonCarr, PosCentMass, PositioningSolutionType, RequestResponseIndication, StationType,
SpeedLimit, StationarySince, TimestampIts, WheelBaseVehicle, TurningRadius, PosFrontAx,
PositionOfOccupants, Temperature, VehicleMass, VehicleIdentification, EnergyStorageType, ActionID,
ItineraryPath, NumberOfOccupants, PositionOfPillars, RelevanceTrafficDirection, RestrictedTypes,
Traces, TransmissionInterval, ValidityDuration, RelevanceDistance, EventHistory, TrafficRule,
DeltaReferencePosition FROM ITS-Container {
itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (102894) cdd (2) version (2)
};


DENM ::= SEQUENCE {
    header ItsPduHeader,
    denm DecentralizedEnvironmentalNotificationMessage
}

DecentralizedEnvironmentalNotificationMessage ::= SEQUENCE {
    management ManagementContainer,
    situation SituationContainer OPTIONAL,
    location LocationContainer OPTIONAL,
    alacarte AlacarteContainer OPTIONAL
}

ManagementContainer ::= SEQUENCE {
    actionID ActionID,
    detectionTime TimestampIts,
    referenceTime TimestampIts,
    termination Termination OPTIONAL,
    eventPosition ReferencePosition,
    relevanceDistance RelevanceDistance OPTIONAL,
    relevanceTrafficDirection RelevanceTrafficDirection OPTIONAL,
    validityDuration ValidityDuration DEFAULT defaultValidity,
    transmissionInterval TransmissionInterval OPTIONAL,
    stationType StationType,
    ...
}

SituationContainer ::= SEQUENCE {
    informationQuality InformationQuality,
    eventType CauseCode,
    linkedCause CauseCode OPTIONAL,
    eventHistory EventHistory OPTIONAL,
    ...
}

LocationContainer ::= SEQUENCE {
    eventSpeed Speed OPTIONAL,
    eventPositionHeading Heading OPTIONAL,
    traces Traces,
    roadType RoadType OPTIONAL,
    ...
}

ImpactReductionContainer ::= SEQUENCE {
    heightLonCarrLeft HeightLonCarr,
    heightLonCarrRight HeightLonCarr,
    posLonCarrLeft PosLonCarr,
    posLonCarrRight PosLonCarr,
    positionOfPillars PositionOfPillars,
    posCentMass PosCentMass,
    wheelBaseVehicle WheelBaseVehicle,
    turningRadius TurningRadius,
```

```
    posFrontAx PosFrontAx,
    positionOfOccupants PositionOfOccupants,
    vehicleMass VehicleMass,
    requestResponseIndication RequestResponseIndication
}

RoadWorksContainerExtended ::= SEQUENCE {
    lightBarSirenInUse LightBarSirenInUse OPTIONAL,
    closedLanes ClosedLanes OPTIONAL,
    restriction RestrictedTypes OPTIONAL,
    speedLimit SpeedLimit OPTIONAL,
    incidentIndication CauseCode OPTIONAL,
    recommendedPath ItineraryPath OPTIONAL,
    startingPointSpeedLimit DeltaReferencePosition OPTIONAL,
    trafficFlowRule TrafficRule OPTIONAL,
    referenceDenms ReferenceDenms OPTIONAL
 }

StationaryVehicleContainer ::= SEQUENCE {
    stationarySince StationarySince OPTIONAL,
    stationaryCause CauseCode OPTIONAL,
    carryingDangerousGoods DangerousGoodsExtended OPTIONAL,
    numberOfOccupants NumberOfOccupants OPTIONAL,
    vehicleIdentification VehicleIdentification OPTIONAL,
    energyStorageType EnergyStorageType OPTIONAL
}

AlacarteContainer ::= SEQUENCE {
    lanePosition LanePosition OPTIONAL,
    impactReduction ImpactReductionContainer OPTIONAL,
    externalTemperature Temperature OPTIONAL,
    roadWorks RoadWorksContainerExtended OPTIONAL,
    positioningSolution PositioningSolutionType OPTIONAL,
    stationaryVehicle StationaryVehicleContainer OPTIONAL,
    ...
}

defaultValidity INTEGER ::= 600

Termination ::= ENUMERATED {isCancellation(0), isNegation (1)}

ReferenceDenms ::= SEQUENCE (SIZE(1..8, ...)) OF ActionID

END
```

Anhang D.

Testdurchführung

D.1. Anleitung

Das im Rahmen dieser Arbeit entwickelte System besteht aus den Modulen OpenSCENARIO Parser und V2X Modul, welche in die Software SEFEV innerhalb der Laborumgebung RailSiTe integriert sind. Im Folgenden sind die zu tätigenden Schritte für das Aufsetzen der Module und die Herstellung der Verbindung beschrieben.

D.1.1. SEFEV

Für die Einrichtung von SEFEV wird eine virtuelle Maschine mit dem Betriebssystem CentOS 7/8 und installierten Gasterweiterungen benötigt. Darüber hinaus ist eine Einrichtung eines *shared folder* zwischen Gast- und Hostsystem sinnvoll. Die virtuelle Maschine muss zur Gewährleistung der Kommunikation mit dem restlichen Netzwerk wie folgt konfiguriert werden:

Netzwerkmodus: Bridged Adapter (Netzwerkbrücke)

Netzwerkadapter: Ethernet

Promiscuous Modus: erlauben

IP-Adresse: 192.168.1.15

Subnetzmaske: 255.255.255.0

Innerhalb der virtuellen Maschine müssen zunächst Qt und weitere Softwarekomponenten[1] installiert werden. Anschließend wird die im Rahmen dieser Bachelorarbeit

[1] Weiterführende Informationen im RailSiTe Handbuch [33].

modifizierte Version von SEFEV aus dem SVN-Repository (`\branches\v2x`) heruntergeladen und gebaut.

D.1.2. OpenSCENARIO Parser

Der OpenSCENARIO Parser kann aus dem Gitlab-Repository der Gruppe Verifikationstechnologien[2] heruntergeladen werden. Für die Ausführung wird Python 3 benötigt. Die Installation der benötigten Pakete erfolgt mit Hilfe der Datei `requirements.txt` automatisch. Das Starten des OpenSCENARIO Parsers erfolgt durch das Starten von `application.py`. Anschließend kann ein Beispielszenario aus `\res\` ausgewählt und geparst werden. Die Testsequenz kann am einem Ort der Wahl (vorzugsweise im *shared folder* mit der SEFEV virtuellen Maschine) durch die Erstellung einer neuen `.sqlite` oder `.db` Datei gespeichert werden.

D.1.3. V2X Modul in Cohda MK5 OBU

Für das V2X Modul werden das Cohda SDK und eine Cohda MK5 OBU mit der Version 16 und der IP-Adresse 192.168.1.17 benötigt. Das V2X Modul kann aus dem Gitlab-Repository der Gruppe Verifikationstechnologien in das Cohda SDK heruntergeladen werden. Anschließend erfolgt das Cross-Kompilieren der Anwendung, das nach dem in [43] beschriebenen Prinzip abläuft. Die benötigten Bibliotheken und die Anwendung werden anschließend auf die Cohda mittels *ssh* kopiert. Daraufhin müssen die Netzwerkschnittstellen `cw-mon-tx` und `cw-mon-rx` aktiviert werden (`ifconfig <interface> up`). Das Senden über die Funkschnittstelle 802.11p erfordert zusätzlich die folgende Konfiguration der Radioschnittstelle: `chconfig -s -w CCH -c 180 -r a`.

D.1.4. Start der Simulation

1. (Optional) Starten des Programms *tcpdump* in der Cohda MK5 OBU in einem separaten Screen mit dem Befehl `tcpdump -i cw-mon-tx -s 2000 -w <file>`.

[2] `gitlab.dlr.de`

2. Starten der Anwendung zur Generierung von V2X Nachrichten in der Cohda MK5 OBU mit dem Befehl `sudo \home\user\vanetza\bin\socktap-cam -i cw-llc0`.
3. Starten der Laborsoftware SEFEV.
4. Öffnen der Konfigurationsdatei (`config\v2x.xml`) und des Szenarios im *shared folder*. Möglicherweise ist hier noch eine Anpassung der Rechte für den aktuellen Benutzer notwendig.
5. Laden und starten der Simulation. Innerhalb der Cohda werden die Größe der gesendeten Nachricht und die Empfängeradresse bei Auslösung eines Events angezeigt.

D.2. OpenSCENARIO Testszenarien

```
<?xml version="1.0" encoding="UTF-8"?>
<OpenSCENARIO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="OpenSCENARIO.xsd">
  <FileHeader revMajor="1" revMinor="0" date="2020-02-21T10:00:00"
description="Cut-In example" author="ASAM e.V."/>
  <ParameterDeclarations/>
  <CatalogLocations/>
  <RoadNetwork>
    <LogicFile filepath="Databases/AB_RQ31_Straight.xodr"/>
    <SceneGraphFile filepath="Databases/AB_RQ31_Straight.opt.osgb"/>
  </RoadNetwork>
  <Entities>
    <ScenarioObject name="Ego">
      <Vehicle name="HAF" vehicleCategory="car">
        <ParameterDeclarations/>
        <Performance maxSpeed="69.444" maxAcceleration="200"
maxDeceleration="10.0"/>
        <BoundingBox>
          <Center x="1.5" y="0.0" z="0.9"/>
          <Dimensions width="2.1" length="4.5" height="1.8"/>
        </BoundingBox>
        <Axles>
          <FrontAxle maxSteering="0.5" wheelDiameter="0.6" trackWidth="1.8"
positionX="3.1" positionZ="0.3"/>
          <RearAxle maxSteering="0.0" wheelDiameter="0.6" trackWidth="1.8"
positionX="0.0" positionZ="0.3"/>
        </Axles>
        <Properties/>
      </Vehicle>
      <ObjectController>
        <Controller name="HAF_Driver">
          <Properties>
            <Property name="weight" value="60.0"/>
            <Property name="height" value="1.8"/>
            <Property name="eyeDistance" value="0.065"/>
            <Property name="age" value="28"/>
            <Property name="sex" value="female"/>
          </Properties>
        </Controller>
      </ObjectController>
    </ScenarioObject>
    <ScenarioObject name="A1">
      <Vehicle name="Default_Car" vehicleCategory="car">
        <ParameterDeclarations/>
        <Performance maxSpeed="69.444" maxAcceleration="200"
maxDeceleration="10.0"/>
        <BoundingBox>
          <Center x="1.4" y="0.0" z="0.8"/>
          <Dimensions width="2.0" length="4.2" height="1.6"/>
        </BoundingBox>
        <Axles>
          <FrontAxle maxSteering="0.5" wheelDiameter="0.5" trackWidth="1.75"
positionX="2.8" positionZ="0.25"/>
          <RearAxle maxSteering="0.0" wheelDiameter="0.5" trackWidth="1.75"
positionX="0.0" positionZ="0.25"/>
        </Axles>
        <Properties/>
      </Vehicle>
      <ObjectController>
        <Controller name="Default_Driver">
          <Properties>
            <Property name="weight" value="80.0"/>
            <Property name="height" value="1.88"/>
            <Property name="eyeDistance" value="0.07"/>
```

```
            <Property name="age" value="32"/>
            <Property name="sex" value="male"/>
          </Properties>
        </Controller>
      </ObjectController>
    </ScenarioObject>
    <ScenarioObject name="A2">
      <Vehicle name="Default_Car" vehicleCategory="car">
        <ParameterDeclarations/>
        <Performance maxSpeed="69.444" maxAcceleration="200"
maxDeceleration="10.0"/>
        <BoundingBox>
          <Center x="1.4" y="0.0" z="0.8"/>
          <Dimensions width="2.0" length="4.2" height="1.6"/>
        </BoundingBox>
        <Axles>
          <FrontAxle maxSteering="0.5" wheelDiameter="0.5" trackWidth="1.75"
positionX="2.8" positionZ="0.25"/>
          <RearAxle maxSteering="0.0" wheelDiameter="0.5" trackWidth="1.75"
positionX="0.0" positionZ="0.25"/>
        </Axles>
        <Properties/>
      </Vehicle>
      <ObjectController>
        <Controller name="Default_Driver">
          <Properties>
            <Property name="weight" value="80.0"/>
            <Property name="height" value="1.88"/>
            <Property name="eyeDistance" value="0.07"/>
            <Property name="age" value="32"/>
            <Property name="sex" value="male"/>
          </Properties>
        </Controller>
      </ObjectController>
    </ScenarioObject>
  </Entities>
  <Storyboard>
    <Init>
      <Actions>
        <GlobalAction>
          <EnvironmentAction>
            <Environment name="Environment1">
              <TimeOfDay animation="false" dateTime="2020-02-21T12:00:00"/>
              <Weather cloudState="free">
                <Sun intensity="1.0" azimuth="0.0" elevation="1.571"/>
                <Fog visualRange="100000.0"/>
                <Precipitation precipitationType="dry" intensity="0.0"/>
              </Weather>
              <RoadCondition frictionScaleFactor="1.0"/>
            </Environment>
          </EnvironmentAction>
        </GlobalAction>
        <Private entityRef="Ego">
          <PrivateAction>
            <LongitudinalAction>
              <SpeedAction>
                <SpeedActionDynamics dynamicsShape="step" value="0"
dynamicsDimension="time"/>
                <SpeedActionTarget>
                  <AbsoluteTargetSpeed value="0.078"/>
                </SpeedActionTarget>
              </SpeedAction>
            </LongitudinalAction>
          </PrivateAction>
```

```
          <PrivateAction>
            <TeleportAction>
              <Position>
                <WorldPosition x="3.0" y="0.0" z="0.0" h="1.96" p="4.0"
r="5.0" />
              </Position>
            </TeleportAction>
          </PrivateAction>
        </Private>
        <Private entityRef="A1">
          <PrivateAction>
            <LongitudinalAction>
              <SpeedAction>
                <SpeedActionDynamics dynamicsShape="step" value="0"
dynamicsDimension="time"/>
                <SpeedActionTarget>
                  <AbsoluteTargetSpeed value="0.078"/>
                </SpeedActionTarget>
              </SpeedAction>
            </LongitudinalAction>
          </PrivateAction>
          <PrivateAction>
            <TeleportAction>
              <Position>
                <WorldPosition x="-13.0" y="0.0" z="11.0" h="1.96" p="4.0"
r="5.0" />
              </Position>
            </TeleportAction>
          </PrivateAction>
        </Private>
        <Private entityRef="A2">
          <PrivateAction>
            <LongitudinalAction>
              <SpeedAction>
                <SpeedActionDynamics dynamicsShape="step" value="0"
dynamicsDimension="time"/>
                <SpeedActionTarget>
                  <AbsoluteTargetSpeed value="0.078"/>
                </SpeedActionTarget>
              </SpeedAction>
            </LongitudinalAction>
          </PrivateAction>
          <PrivateAction>
            <TeleportAction>
              <Position>
                <WorldPosition x="-13.0" y="-13.0" z="11.0" h="1.96" p="4.0"
r="5.0" />
              </Position>
            </TeleportAction>
          </PrivateAction>
        </Private>
      </Actions>
    </Init>
    <Story name="MyStory">
      <ParameterDeclarations>
        <ParameterDeclaration parameterType="string" name="$owner" value="A2"/>
      </ParameterDeclarations>
      <Act name="Act1">
        <ManeuverGroup maximumExecutionCount="1" name="Sequence1">
          <Actors selectTriggeringEntities="false">
            <EntityRef entityRef="A2"/>
          </Actors>
          <Maneuver name="Maneuver1">
            <Event name="Event1" priority="overwrite">
```

```
              <Action name="Action1">
                <PrivateAction>
                  <RoutingAction>
                    <FollowTrajectoryAction>
                      <Trajectory name="Trajectory1" closed="false">
                        <Shape>
                          <Polyline>
                            <Vertex time="0.0">
                              <Position>
                                <WorldPosition x="-13.0" y="0.0" z="0.0" h="3.0"
p="4.0" r="5.0" />
                              </Position>
                            </Vertex>
                            <Vertex time="0.5">
                              <Position>
                                <WorldPosition x="-13.0" y="50.0" z="0.0"
h="3.0" p="4.0" r="5.0" />
                              </Position>
                            </Vertex>
                            <Vertex time="1.0">
                              <Position>
                                <WorldPosition x="-14.0" y="80.0" z="0.0"
h="3.0" p="4.0" r="5.0" />
                              </Position>
                            </Vertex>
                            <Vertex time="1.5">
                              <Position>
                                <WorldPosition x="-13.0" y="100.0" z="0.0"
h="3.0" p="4.0" r="5.0" />
                              </Position>
                                        </Vertex>
                                        <Vertex time="1.5">
                              <Position>
                                <WorldPosition x="-13.0" y="120.0" z="0.0"
h="3.0" p="4.0" r="5.0" />
                              </Position>
                                        </Vertex>
                                        <Vertex time="1.5">
                              <Position>
                                <WorldPosition x="-20.0" y="150.0" z="0.0"
h="3.0" p="4.0" r="5.0" />
                              </Position>
                                        </Vertex>
                          </Polyline>
                        </Shape>
                      </Trajectory>
                      <TimeReference>
                                <Timing domainAbsoluteRelative="relative"
scale="1.0" offset="0.0"/>
                      </TimeReference>
                      <TrajectoryFollowingMode followingMode="follow"/>
                    </FollowTrajectoryAction>
                  </RoutingAction>
                </PrivateAction>
              </Action>
              <StartTrigger>
                <ConditionGroup>
                            <Condition name="" delay="0"
conditionEdge="rising">
                                  <ByValueCondition>
                                        <SimulationTimeCondition value="0"
rule="greaterThan"/>
                                  </ByValueCondition>
                            </Condition>
```

```
                    </ConditionGroup>
            </StartTrigger>
          </Event>
                <Event name="Event1" priority="overwrite">
                 <Action name="dangerous_situation">
                     <UserDefinedAction>
                           <CustomCommandAction type="A2,99,0,3.0,92,1"/>
                     </UserDefinedAction>
                 </Action>
                 <StartTrigger>
              <ConditionGroup>
                <Condition name="" delay="0" conditionEdge="rising">
                 <ByValueCondition>
                   <SimulationTimeCondition value="52" rule="greaterThan"/>
                 </ByValueCondition>
              </Condition>
              </ConditionGroup>
            </StartTrigger>
                 </Event>
        </Maneuver>
      </ManeuverGroup>
          <ManeuverGroup maximumExecutionCount="1" name="Sequence1">
        <Actors selectTriggeringEntities="false">
          <EntityRef entityRef="Ego"/>
        </Actors>
        <Maneuver name="Maneuver1">
          <Event name="Event1" priority="overwrite">
            <Action name="Action1">
              <PrivateAction>
                <RoutingAction>
                  <FollowTrajectoryAction>
                    <Trajectory name="Trajectory1" closed="false">
                      <Shape>
                        <Polyline>
                          <Vertex time="0.0">
                            <Position>
                              <WorldPosition x="3.0" y="10.0" z="0.0" h="3.0"
p="4.0" r="5.0" />
                            </Position>
                          </Vertex>
                          <Vertex time="0.5">
                            <Position>
                              <WorldPosition x="3.0" y="50.0" z="0.0" h="3.0"
p="4.0" r="5.0" />
                            </Position>
                          </Vertex>
                          <Vertex time="1.0">
                            <Position>
                              <WorldPosition x="3.0" y="80.0" z="0.0" h="3.0"
p="4.0" r="5.0" />
                            </Position>
                          </Vertex>
                          <Vertex time="1.5">
                            <Position>
                              <WorldPosition x="3.0" y="100.0" z="0.0" h="3.0"
p="4.0" r="5.0" />
                            </Position>
                                      </Vertex>
                                      <Vertex time="1.5">
                            <Position>
                              <WorldPosition x="3.0" y="120.0" z="0.0" h="3.0"
p="4.0" r="5.0" />
                            </Position>
                                      </Vertex>
```

```
                                        <Vertex time="1.5">
                                <Position>
                                  <WorldPosition x="3.0" y="150.0" z="0.0" h="3.0"
p="4.0" r="5.0" />
                                </Position>
                                        </Vertex>
                            </Polyline>
                          </Shape>
                        </Trajectory>
                        <TimeReference>
                                <Timing domainAbsoluteRelative="relative"
scale="1.0" offset="0.0"/>
                        </TimeReference>
                        <TrajectoryFollowingMode followingMode="follow"/>
                      </FollowTrajectoryAction>
                    </RoutingAction>
                  </PrivateAction>
                </Action>
                <StartTrigger>
                  <ConditionGroup>
                                <Condition name="" delay="0"
conditionEdge="rising">
                                        <ByValueCondition>
                                                <SimulationTimeCondition value="20"
rule="greaterThan"/>
                                        </ByValueCondition>
                                </Condition>
                          </ConditionGroup>
                </StartTrigger>
              </Event>
            </Maneuver>
          </ManeuverGroup>
              <ManeuverGroup maximumExecutionCount="1" name="Sequence1">
            <Actors selectTriggeringEntities="false">
              <EntityRef entityRef="A1"/>
            </Actors>
            <Maneuver name="Maneuver1">
              <Event name="Event1" priority="overwrite">
                <Action name="Action1">
                  <PrivateAction>
                    <RoutingAction>
                      <FollowTrajectoryAction>
                        <Trajectory name="Trajectory1" closed="false">
                          <Shape>
                            <Polyline>
                              <Vertex time="0.0">
                                <Position>
                                  <WorldPosition x="-13.0" y="0.0" z="0.0" h="3.0"
p="4.0" r="5.0" />
                                </Position>
                              </Vertex>
                              <Vertex time="0.5">
                                <Position>
                                  <WorldPosition x="-13.0" y="50.0" z="0.0"
h="3.0" p="4.0" r="5.0" />
                                </Position>
                              </Vertex>
                              <Vertex time="1.0">
                                <Position>
                                  <WorldPosition x="-14.0" y="80.0" z="0.0"
h="3.0" p="4.0" r="5.0" />
                                </Position>
                              </Vertex>
                              <Vertex time="1.5">
```

```
                                <Position>
                                  <WorldPosition x="3.0" y="100.0" z="0.0" h="3.0"
p="4.0" r="5.0" />
                                </Position>
                                          </Vertex>
                                          <Vertex time="1.5">
                                <Position>
                                  <WorldPosition x="3.0" y="120.0" z="0.0" h="3.0"
p="4.0" r="5.0" />
                                </Position>
                                          </Vertex>
                                          <Vertex time="1.5">
                                <Position>
                                  <WorldPosition x="3.0" y="150.0" z="0.0" h="3.0"
p="4.0" r="5.0" />
                                </Position>
                                          </Vertex>
                                          <Vertex time="1.5">
                                <Position>
                                  <WorldPosition x="3.0" y="160.0" z="0.0" h="1.8"
p="4.0" r="5.0" />
                                </Position>
                                          </Vertex>
                           </Polyline>
                         </Shape>
                       </Trajectory>
                       <TimeReference>
                                <Timing domainAbsoluteRelative="relative"
scale="1.0" offset="0.0"/>
                       </TimeReference>
                       <TrajectoryFollowingMode followingMode="follow"/>
                     </FollowTrajectoryAction>
                   </RoutingAction>
                 </PrivateAction>
               </Action>
               <StartTrigger>
                 <ConditionGroup>
                                <Condition name="" delay="0"
conditionEdge="rising">
                                      <ByValueCondition>
                                            <SimulationTimeCondition value="0"
rule="greaterThan"/>
                                      </ByValueCondition>
                                </Condition>
                         </ConditionGroup>
               </StartTrigger>
             </Event>
           </Maneuver>
         </ManeuverGroup>
         <StartTrigger>
           <ConditionGroup>
             <Condition name="" delay="0" conditionEdge="rising">
               <ByValueCondition>
                 <SimulationTimeCondition value="0" rule="greaterThan"/>
               </ByValueCondition>
             </Condition>
           </ConditionGroup>
         </StartTrigger>
       </Act>
     </Story>
     <StopTrigger/>
   </Storyboard>
</OpenSCENARIO>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<OpenSCENARIO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="OpenSCENARIO.xsd">
  <FileHeader revMajor="1" revMinor="0" date="2020-02-21T10:00:00"
description="Cut-In example" author="ASAM e.V."/>
  <ParameterDeclarations/>
  <CatalogLocations/>
  <RoadNetwork>
    <LogicFile filepath="Databases/AB_RQ31_Straight.xodr"/>
    <SceneGraphFile filepath="Databases/AB_RQ31_Straight.opt.osgb"/>
  </RoadNetwork>
  <Entities>
    <ScenarioObject name="Ego">
      <Vehicle name="HAF" vehicleCategory="car">
        <ParameterDeclarations/>
        <Performance maxSpeed="69.444" maxAcceleration="200"
maxDeceleration="10.0"/>
        <BoundingBox>
          <Center x="1.5" y="0.0" z="0.9"/>
          <Dimensions width="2.1" length="4.5" height="1.8"/>
        </BoundingBox>
        <Axles>
          <FrontAxle maxSteering="0.5" wheelDiameter="0.6" trackWidth="1.8"
positionX="3.1" positionZ="0.3"/>
          <RearAxle maxSteering="0.0" wheelDiameter="0.6" trackWidth="1.8"
positionX="0.0" positionZ="0.3"/>
        </Axles>
        <Properties/>
      </Vehicle>
      <ObjectController>
        <Controller name="HAF_Driver">
          <Properties>
            <Property name="weight" value="60.0"/>
            <Property name="height" value="1.8"/>
            <Property name="eyeDistance" value="0.065"/>
            <Property name="age" value="28"/>
            <Property name="sex" value="female"/>
          </Properties>
        </Controller>
      </ObjectController>
    </ScenarioObject>
    <ScenarioObject name="A2">
      <Vehicle name="Default_Car" vehicleCategory="car">
        <ParameterDeclarations/>
        <Performance maxSpeed="69.444" maxAcceleration="200"
maxDeceleration="10.0"/>
        <BoundingBox>
          <Center x="1.4" y="0.0" z="0.8"/>
          <Dimensions width="2.0" length="4.2" height="1.6"/>
        </BoundingBox>
        <Axles>
          <FrontAxle maxSteering="0.5" wheelDiameter="0.5" trackWidth="1.75"
positionX="2.8" positionZ="0.25"/>
          <RearAxle maxSteering="0.0" wheelDiameter="0.5" trackWidth="1.75"
positionX="0.0" positionZ="0.25"/>
        </Axles>
        <Properties/>
      </Vehicle>
      <ObjectController>
        <Controller name="Default_Driver">
          <Properties>
            <Property name="weight" value="80.0"/>
            <Property name="height" value="1.88"/>
            <Property name="eyeDistance" value="0.07"/>
```

```
            <Property name="age" value="32"/>
            <Property name="sex" value="male"/>
          </Properties>
        </Controller>
      </ObjectController>
    </ScenarioObject>
      <ScenarioObject name="A1">
      <Vehicle name="Default_Car" vehicleCategory="car">
        <ParameterDeclarations/>
        <Performance maxSpeed="69.444" maxAcceleration="200"
maxDeceleration="10.0"/>
        <BoundingBox>
          <Center x="1.4" y="0.0" z="0.8"/>
          <Dimensions width="2.0" length="4.2" height="1.6"/>
        </BoundingBox>
        <Axles>
          <FrontAxle maxSteering="0.5" wheelDiameter="0.5" trackWidth="1.75"
positionX="2.8" positionZ="0.25"/>
          <RearAxle maxSteering="0.0" wheelDiameter="0.5" trackWidth="1.75"
positionX="0.0" positionZ="0.25"/>
        </Axles>
        <Properties/>
      </Vehicle>
      <ObjectController>
        <Controller name="Default_Driver">
          <Properties>
            <Property name="weight" value="80.0"/>
            <Property name="height" value="1.88"/>
            <Property name="eyeDistance" value="0.07"/>
            <Property name="age" value="32"/>
            <Property name="sex" value="male"/>
          </Properties>
        </Controller>
      </ObjectController>
    </ScenarioObject>

      <ScenarioObject name="A3">
      <Vehicle name="Default_Car" vehicleCategory="car">
        <ParameterDeclarations/>
        <Performance maxSpeed="69.444" maxAcceleration="200"
maxDeceleration="10.0"/>
        <BoundingBox>
          <Center x="1.4" y="0.0" z="0.8"/>
          <Dimensions width="2.0" length="4.2" height="1.6"/>
        </BoundingBox>
        <Axles>
          <FrontAxle maxSteering="0.5" wheelDiameter="0.5" trackWidth="1.75"
positionX="2.8" positionZ="0.25"/>
          <RearAxle maxSteering="0.0" wheelDiameter="0.5" trackWidth="1.75"
positionX="0.0" positionZ="0.25"/>
        </Axles>
        <Properties/>
      </Vehicle>
      <ObjectController>
        <Controller name="Default_Driver">
          <Properties>
            <Property name="weight" value="80.0"/>
            <Property name="height" value="1.88"/>
            <Property name="eyeDistance" value="0.07"/>
            <Property name="age" value="32"/>
            <Property name="sex" value="male"/>
          </Properties>
        </Controller>
      </ObjectController>
```

```
    </ScenarioObject>
  </Entities>
  <Storyboard>
    <Init>
      <Actions>
        <GlobalAction>
          <EnvironmentAction>
            <Environment name="Environment1">
              <TimeOfDay animation="false" dateTime="2020-02-21T12:00:00"/>
              <Weather cloudState="free">
                <Sun intensity="1.0" azimuth="0.0" elevation="1.571"/>
                <Fog visualRange="100000.0"/>
                <Precipitation precipitationType="dry" intensity="0.0"/>
              </Weather>
              <RoadCondition frictionScaleFactor="1.0"/>
            </Environment>
          </EnvironmentAction>
        </GlobalAction>
        <Private entityRef="Ego">
          <PrivateAction>
            <LongitudinalAction>
              <SpeedAction>
                <SpeedActionDynamics dynamicsShape="step" value="0"
dynamicsDimension="time"/>
                <SpeedActionTarget>
                  <AbsoluteTargetSpeed value="30.78"/>
                </SpeedActionTarget>
              </SpeedAction>
            </LongitudinalAction>
          </PrivateAction>
              <PrivateAction>
            <TeleportAction>
              <Position>
                <WorldPosition x="52.313420" y="1.531632" z="11.0" h="1.96"
p="4.0" r="5.0" />
              </Position>
            </TeleportAction>
          </PrivateAction>
        </Private>
        <Private entityRef="A2">
          <PrivateAction>
            <LongitudinalAction>
              <SpeedAction>
                <SpeedActionDynamics dynamicsShape="step" value="0"
dynamicsDimension="time"/>
                <SpeedActionTarget>
                  <AbsoluteTargetSpeed value="30.56"/>
                </SpeedActionTarget>
              </SpeedAction>
            </LongitudinalAction>
          </PrivateAction>
               <PrivateAction>
            <TeleportAction>
              <Position>
                <WorldPosition x="52.314313" y="10.556422" z="11.0" h="1.96"
p="4.0" r="5.0" />
              </Position>
            </TeleportAction>
          </PrivateAction>
        </Private>
            <Private entityRef="A1">
          <PrivateAction>
            <LongitudinalAction>
              <SpeedAction>
```

```
                <SpeedActionDynamics dynamicsShape="step" value="0"
dynamicsDimension="time"/>
                <SpeedActionTarget>
                  <AbsoluteTargetSpeed value="30.56"/>
                </SpeedActionTarget>
              </SpeedAction>
            </LongitudinalAction>
          </PrivateAction>
               <PrivateAction>
            <TeleportAction>
              <Position>
                <WorldPosition x="52.313284" y="10.550474" z="11.0" h="1.96"
p="4.0" r="5.0" />
              </Position>
            </TeleportAction>
          </PrivateAction>
        </Private>
            <Private entityRef="A3">
          <PrivateAction>
            <LongitudinalAction>
              <SpeedAction>
                <SpeedActionDynamics dynamicsShape="step" value="0"
dynamicsDimension="time"/>
                <SpeedActionTarget>
                  <AbsoluteTargetSpeed value="30.56"/>
                </SpeedActionTarget>
              </SpeedAction>
            </LongitudinalAction>
          </PrivateAction>
               <PrivateAction>
            <TeleportAction>
              <Position>
                <WorldPosition x="52.313117" y="10.564093" z="11.0" h="1.96"
p="4.0" r="5.0" />
              </Position>
            </TeleportAction>
          </PrivateAction>
        </Private>
      </Actions>
    </Init>
    <Story name="MyStory">
      <ParameterDeclarations>
        <ParameterDeclaration parameterType="string" name="$owner" value="A2"/>
      </ParameterDeclarations>
      <Act name="Act1">
             <ManeuverGroup maximumExecutionCount="1" name="Sequence1">
          <Actors selectTriggeringEntities="false">
            <EntityRef entityRef="A1"/>
          </Actors>
          <Maneuver name="Maneuver1">
            <Event name="Event1" priority="overwrite">
              <Action name="Action1">
                <PrivateAction>
                  <RoutingAction>
                    <FollowTrajectoryAction>
                      <Trajectory name="Trajectory1" closed="false">
                        <Shape>
                          <Polyline>
                            <Vertex time="0.0">
                              <Position>
                                <WorldPosition x="52.313463" y="10.530966"
z="0.0" h="3.0" p="4.0" r="5.0" />
                              </Position>
                            </Vertex>
```

```
                                        <Vertex time="0.0">
                                          <Position>
                                              <WorldPosition h="3.0" p="4.0"
r="5.0" x="52.313463" y="10.530966" z="0.0" />
                                          </Position>
                                        </Vertex>
                          </Polyline>
                        </Shape>
                      </Trajectory>
                      <TimeReference>
                                  <Timing domainAbsoluteRelative="relative"
scale="1.0" offset="0.0"/>
                      </TimeReference>
                      <TrajectoryFollowingMode followingMode="follow"/>
                    </FollowTrajectoryAction>
                  </RoutingAction>
                </PrivateAction>
              </Action>
              <StartTrigger>
                <ConditionGroup>
                  <Condition name="StartCondition1" delay="0"
conditionEdge="rising">
                    <ByEntityCondition>
                      <TriggeringEntities triggeringEntitiesRule="any">
                        <EntityRef entityRef="Ego"/>
                      </TriggeringEntities>
                      <EntityCondition>
                        <ReachPositionCondition tolerance="0.0">
                          <Position>
                            <WorldPosition x="52.313420" y="1.531632" z="11.0"
h="1.96" p="4.0" r="5.0" />
                          </Position>
                        </ReachPositionCondition>
                      </EntityCondition>
                    </ByEntityCondition>
                  </Condition>
                </ConditionGroup>
              </StartTrigger>
            </Event>
          </Maneuver>
        </ManeuverGroup>
            <ManeuverGroup maximumExecutionCount="1" name="Sequence1">
          <Actors selectTriggeringEntities="false">
            <EntityRef entityRef="A3"/>
          </Actors>
          <Maneuver name="Maneuver1">
            <Event name="Event1" priority="overwrite">
              <Action name="Action1">
                <PrivateAction>
                  <RoutingAction>
                    <FollowTrajectoryAction>
                      <Trajectory name="Trajectory1" closed="false">
                        <Shape>
                          <Polyline>
                            <Vertex time="0.0">
                              <Position>
                                <WorldPosition x="52.313463" y="10.530966"
z="0.0" h="3.0" p="4.0" r="5.0" />
                              </Position>
                            </Vertex>
                            <Vertex time="0.5">
                              <Position>
                                <WorldPosition x="52.313466" y="10.530966"
z="0.0" h="3.0" p="4.0" r="5.0" />
```

```
                              </Position>
                                        </Vertex>
                          </Polyline>
                        </Shape>
                      </Trajectory>
                      <TimeReference>
                                  <Timing domainAbsoluteRelative="relative"
scale="1.0" offset="0.0"/>
                      </TimeReference>
                      <TrajectoryFollowingMode followingMode="follow"/>
                    </FollowTrajectoryAction>
                  </RoutingAction>
                </PrivateAction>
              </Action>
              <StartTrigger>
                <ConditionGroup>
                            <Condition name="" delay="0"
conditionEdge="rising">
                                  <ByValueCondition>
                                        <SimulationTimeCondition value="12"
rule="greaterThan"/>
                                  </ByValueCondition>
                            </Condition>
                        </ConditionGroup>
              </StartTrigger>
            </Event>
          </Maneuver>
        </ManeuverGroup>
            <ManeuverGroup maximumExecutionCount="1" name="Sequence1">
          <Actors selectTriggeringEntities="false">
            <EntityRef entityRef="Ego"/>
          </Actors>
          <Maneuver name="Maneuver1">
            <Event name="Event1" priority="overwrite">
              <Action name="Action1">
                <PrivateAction>
                  <RoutingAction>
                    <FollowTrajectoryAction>
                      <Trajectory name="Trajectory1" closed="false">
                        <Shape>
                          <Polyline>
                            <Vertex time="0.0">
                              <Position>
                                <WorldPosition x="52.314568" y="10.518382"
z="0.0" h="3.0" p="4.0" r="5.0" />
                              </Position>
                                        </Vertex>
                                  <Vertex time="1.0">
                              <Position>
                                <WorldPosition x="52.315605" y="10.518382"
z="0.0" h="3.0" p="4.0" r="5.0" />
                              </Position>
                                        </Vertex>
                          </Polyline>
                        </Shape>
                      </Trajectory>
                      <TimeReference>
                                  <Timing domainAbsoluteRelative="relative"
scale="1.0" offset="0.0"/>
                      </TimeReference>
                      <TrajectoryFollowingMode followingMode="follow"/>
                    </FollowTrajectoryAction>
                  </RoutingAction>
                </PrivateAction>
```

```
              </Action>
              <StartTrigger>
                <ConditionGroup>
                              <Condition name="" delay="0"
conditionEdge="rising">
                                    <ByValueCondition>
                                          <SimulationTimeCondition value="0"
rule="greaterThan"/>
                                    </ByValueCondition>
                              </Condition>
                        </ConditionGroup>
              </StartTrigger>
            </Event>
          </Maneuver>
        </ManeuverGroup>
        <StartTrigger>
          <ConditionGroup>
            <Condition name="" delay="0" conditionEdge="rising">
              <ByValueCondition>
                <SimulationTimeCondition value="0" rule="greaterThan"/>
              </ByValueCondition>
            </Condition>
          </ConditionGroup>
        </StartTrigger>
      </Act>
    </Story>
    <StopTrigger/>
  </Storyboard>
</OpenSCENARIO>
```

D.3. Validierungsergebnisse

Abbildung D.1.: Übertragene DENM vom Vanetza Stack zum ITS G5 Modem der Cohda MK5 OBU - analysiert mit Wireshark

```
▾ Frame 2: 165 bytes on wire (1320 bits), 165 bytes captured (1320 bits)
    Encapsulation type: IEEE 802.11 plus radiotap radio header (23)
    Arrival Time: Aug 14, 2020 11:05:39.606458000 Mitteleurop�ische Sommerzeit
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1597395939.606458000 seconds
    [Time delta from previous captured frame: 2.721114000 seconds]
    [Time delta from previous displayed frame: 2.721114000 seconds]
    [Time since reference or first frame: 2.721114000 seconds]
    Frame Number: 2
    Frame Length: 165 bytes (1320 bits)
    Capture Length: 165 bytes (1320 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: radiotap:wlan_radio:wlan:llc:gn:btp:etsiits]
▸ Radiotap Header v0, Length 44
  802.11 radio information
▾ IEEE 802.11 QoS Data, Flags: ........
    Type/Subtype: QoS Data (0x0028)
  ▸ Frame Control Field: 0x8800
    .000 0000 1110 1100 = Duration: 236 microseconds
    Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)
    Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
    Transmitter address: e2:7a:14:00:6b:1e (e2:7a:14:00:6b:1e)
    Source address: e2:7a:14:00:6b:1e (e2:7a:14:00:6b:1e)
    BSS Id: Broadcast (ff:ff:ff:ff:ff:ff)
    .... .... .... 0000 = Fragment number: 0
    0000 0000 0000 .... = Sequence number: 0
  ▸ Qos Control: 0x0007
▾ Logical-Link Control (LPD)
  ▸ DSAP: SNAP (0xaa)
  ▸ SSAP: SNAP (0xaa)
  ▸ Control field: U, func=UI (0x03)
    Organization Code: Encapsulated Ethernet (0x000000)
    Type: ETSI GN (0x8947)
▾ GeoNetworking: Common (TSB Single Hop)
  ▾ Basic Header
      0001 .... = Version: 1
      .... 0001 = Next Header: Common (1)
      Reserved: 0
    ▸ Lifetime 60000 ms
      Remaining Hop Limit: 1
  ▸ Common Header
  ▾ Topology-Scoped Broadcast
    ▸ Source Position Vector
    ▸ Reserved (G5 DCC)
▾ Basic Transport Protocol (Type B)
    Destination Port: 2001
    Destination Port Info: 0
▾ ETSI ITS (DENM)
  ▾ DENM
    ▾ header
        protocolVersion: 2
        messageID: denm (1)
        stationID: 17 (0x00000011)
    ▾ denm
      ▾ management
        ▾ actionID
            originatingStationID: 111 (0x0000006f)
            sequenceNumber: 222 (0x00de)
          detectionTime: 2003-12-31 23:59:55.007 (0.007 s) (7/0x000007)
          referenceTime: 2003-12-31 23:59:55.000 (0.000 s) (0/0x000000)
        ▾ eventPosition
            latitude: Unknown (333400000) (33.3400000 deg)
            longitude: Unknown (354600000) (35.4600000 deg)
          ▾ positionConfidenceEllipse
              semiMajorConfidence: unavailable (4095)
              semiMinorConfidence: unavailable (4095)
              semiMajorOrientation: unavailable (3601)
          ▾ altitude
              altitudeValue: Unknown (14432) (144.32 m)
              altitudeConfidence: alt-002-00 (7)
          stationType: passengerCar (5)
      ▾ situation
          informationQuality: highest (7)
        ▾ eventType
            causeCode: dangerousSituation (99)
            subCauseCode: 0
```

Abbildung D.2.: Übertragene CAM vom Vanetza Stack zum ITS G5 Modem der Cohda MK5 OBU - analysiert mit Wireshark

```
Frame 5: 163 bytes on wire (1304 bits), 163 bytes captured (1304 bits)
    Encapsulation type: IEEE 802.11 plus radiotap radio header (23)
    Arrival Time: Aug 14, 2020 11:05:39.908487000 Mitteleurop�ische Sommerzeit
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1597395939.908487000 seconds
    [Time delta from previous captured frame: 0.101396000 seconds]
    [Time delta from previous displayed frame: 0.101396000 seconds]
    [Time since reference or first frame: 3.023143000 seconds]
    Frame Number: 5
    Frame Length: 163 bytes (1304 bits)
    Capture Length: 163 bytes (1304 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: radiotap:wlan_radio:wlan:llc:gn:btp:etsiits]
Radiotap Header v0, Length 44
  802.11 radio information
IEEE 802.11 QoS Data, Flags: ........
    Type/Subtype: QoS Data (0x0028)
  Frame Control Field: 0x8800
    .000 0000 1110 1100 = Duration: 236 microseconds
    Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)
    Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
    Transmitter address: e2:7a:14:00:6b:1e (e2:7a:14:00:6b:1e)
    Source address: e2:7a:14:00:6b:1e (e2:7a:14:00:6b:1e)
    BSS Id: Broadcast (ff:ff:ff:ff:ff:ff)
    .... .... .... 0000 = Fragment number: 0
    0000 0000 0000 .... = Sequence number: 0
  Qos Control: 0x0007
Logical-Link Control (LPD)
GeoNetworking: Common (TSB Single Hop)
  Basic Header
      0001 .... = Version: 1
      .... 0001 = Next Header: Common (1)
      Reserved: 0
    Lifetime 60000 ms
      Remaining Hop Limit: 1
  Common Header
  Topology-Scoped Broadcast
Basic Transport Protocol (Type B)
    Destination Port: 2001
    Destination Port Info: 0
ETSI ITS (CAM)
  CAM
    header
        protocolVersion: 2
        messageID: cam (2)
        stationID: 17 (0x00000011)
    cam
        generationDeltaTime: Unknown (78) (0x004e, 0.078 sec)
      camParameters
        basicContainer
            stationType: passengerCar (5)
          referencePosition
            latitude: Unknown (333000000) (33.3000000 deg)
            longitude: Unknown (343000000) (34.3000000 deg)
          positionConfidenceEllipse
              semiMajorConfidence: unavailable (4095)
              semiMinorConfidence: unavailable (4095)
              semiMajorOrientation: unavailable (3601)
          altitude
              altitudeValue: Unknown (14432) (144.32 m)
              altitudeConfidence: alt-002-00 (7)
        highFrequencyContainer: basicVehicleContainerHighFrequency (0)
          basicVehicleContainerHighFrequency
            heading
              headingValue: wgs84North (0) (0.0 deg)
              headingConfidence: equalOrWithinOneDegree (10)
            speed
              speedValue: Unknown (20) (0.20 m/s, 0.72 km/h)
              speedConfidence: equalOrWithinOneCentimeterPerSec (1) (0.01 m/s)
            driveDirection: forward (0)
          vehicleLength
            vehicleWidth: unavailable (62)
          longitudinalAcceleration
          curvature
            curvatureCalculationMode: yawRateUsed (0)
          yawRate
```

Abbildung D.3.: Mit der WaveBee touch empfangene DENM - analysiert mit Wireshark

```
> Frame 42: 101 bytes on wire (808 bits), 101 bytes captured (808 bits)
> Ethernet II, Src: a2:3f:49:bd:89:89 (a2:3f:49:bd:89:89), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
v GeoNetworking: Common (TSB Single Hop)
  v Basic Header
      0001 .... = Version: 1
      .... 0001 = Next Header: Common (1)
      Reserved: 0
    > Lifetime 60000 ms
      Remaining Hop Limit: 1
  > Common Header
  v Topology-Scoped Broadcast
    v Source Position Vector
      v GN Address: 0x8000a23f49bd8989
          1... .... .... .... = Assignment: Manual (1)
          .000 00.. .... .... = Station Type: Unknown (0)
          .... ..00 0000 0000 = Reserved: 0
          Link-Layer Address: a2:3f:49:bd:89:89 (a2:3f:49:bd:89:89)
        Timestamp: 1980577.939s (1980577939)
        Latitude: 52.3155683 (52°18'56.05"N) (523155683)
        Longitude: 10.5608092 (10°33'38.91"E) (105608092)
        1... .... .... .... = PAI: 1
        .000 0000 0000 0101 = Speed: 0x0005 (0.05 m/s) (0.18 km/h) (5)
        Heading: 217.5° (2175)
    v Reserved (G5 DCC)
        CBR_L_0: 0
        CBR_L_1: 0
        Output power: 0 dB
        MCO: 0
> Basic Transport Protocol (Type B)
v ETSI ITS (DENM)
  v DENM
    v header
        protocolVersion: 2
        messageID: denm (1)
        stationID: 115 (0x00000073)
    v denm
      v management
        v actionID
            originatingStationID: 115 (0x00000073)
            sequenceNumber: 222 (0x00de)
          detectionTime: 2004-01-01 00:01:00.535 (65.535 s) (65535/0x00ffff)
          referenceTime: 2003-12-31 23:59:55.000 (0.000 s) (0/0x000000)
        v eventPosition
            latitude: Unknown (104757610) (10.4757610 deg)
            longitude: Unknown (523287640) (52.3287640 deg)
          > positionConfidenceEllipse
          > altitude
          stationType: passengerCar (5)
      v situation
          informationQuality: highest (7)
        v eventType
            causeCode: dangerousSituation (99)
            subCauseCode: 0
```

Abbildung D.4.: Mit der WaveBee touch empfangene CAM - analysiert mit Wireshark

```
> Frame 21: 99 bytes on wire (792 bits), 99 bytes captured (792 bits)
> Ethernet II, Src: a2:3f:49:bd:89:89 (a2:3f:49:bd:89:89), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
v GeoNetworking: Common (TSB Single Hop)
  v Basic Header
      0001 .... = Version: 1
      .... 0001 = Next Header: Common (1)
      Reserved: 0
    > Lifetime 60000 ms
      Remaining Hop Limit: 1
  > Common Header
  v Topology-Scoped Broadcast
    v Source Position Vector
      > GN Address: 0x8000a23f49bd8989
        Timestamp: 1980574.936s (1980574936)
        Latitude: 52.3155681 (52°18'56.05"N) (523155681)
        Longitude: 10.5608095 (10°33'38.91"E) (105608095)
        1... .... .... .... = PAI: 1
        .000 0000 0000 0001 = Speed: 0x0001 (0.01 m/s) (0.04 km/h) (1)
        Heading: 217.5° (2175)
    > Reserved (G5 DCC)
> Basic Transport Protocol (Type B)
v ETSI ITS (CAM)
  v CAM
    v header
        protocolVersion: 2
        messageID: cam (2)
        stationID: 114 (0x00000072)
    v cam
        generationDeltaTime: Unknown (65535) (0xffff, 65.535 sec)
      v camParameters
        v basicContainer
            stationType: passengerCar (5)
          v referencePosition
              latitude: Unknown (523133010) (52.3133010 deg)
              longitude: Unknown (105486600) (10.5486600 deg)
            v positionConfidenceEllipse
                semiMajorConfidence: unavailable (4095)
                semiMinorConfidence: unavailable (4095)
                semiMajorOrientation: unavailable (3601)
            v altitude
                altitudeValue: Unknown (13424) (134.24 m)
                altitudeConfidence: alt-002-00 (7)
        v highFrequencyContainer: basicVehicleContainerHighFrequency (0)
          v basicVehicleContainerHighFrequency
            > heading
            v speed
                speedValue: Unknown (20) (0.20 m/s, 0.72 km/h)
                speedConfidence: equalOrWithinOneCentimeterPerSec (1) (0.01 m/s)
              driveDirection: forward (0)
            > vehicleLength
              vehicleWidth: unavailable (62)
            > longitudinalAcceleration
            > curvature
              curvatureCalculationMode: yawRateUsed (0)
            > yawRate
```