Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Besnik Mulici

Integration von CANoe in die OMNeT++ System Level Simulation für automobile Netzwerke

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Besnik Mulici

Integration von CANoe in die OMNeT++ System Level Simulation für automobile Netzwerke

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

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

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Wolfgang Fohl Zweitprüfer

Eingereicht am: 4. Mai 2016

Besnik Mulici

Thema der Arbeit

Integration von CANoe in die OMNeT++ System Level Simulation für automobile Netzwerke

Stichworte

CANoe Integration, BLF-Datei, PCAP-Datei, Simulationsdaten, CAN-Frame, Ethernet-Frame

Kurzzusammenfassung

Die heutigen Bussysteme erreichen langsam ihre Grenzen und deswegen sucht die Automobilindustrie nach einem neuen Bussystem, das die steigenden Anforderungen erfüllt. Eine mögliche Technologie, die eine hohe Bandbreite und eine deterministische Echtzeitkommunikation ermöglicht, ist das Time-Triggered-Ethernet. Die vorliegende Arbeit befasst sich mit der Integration von CANoe in die OMNeT++ System Level Simulation für automobile Netzwerke. CANoe ist in der Automobilindustrie ein weit verbreitetes Simulationswerkzeug der Firma Vector Informatik GmbH. Die Integration ermöglicht es den Autobauern, in ihrem gewohnten Simulationswerkzeug mit den OMNet++ System Level Simulationsdaten zu arbeiten.

Besnik Mulici

Title of the paper

Integration of CANoe in the OMNeT++ System Level Simulation for automotive networks

Keywords

CANoe integration, BLF-File, PCAP-File, simulation data, CAN-Frame, Ethernet-Frame

Abstract

Today's bus systems slowly reach their limits. For this reason the automotive industry is looking for a new bus system that meets the increasing demands. One possible technology that allows a high bandwidth and a deterministic real-time communication is the time-triggered Ethernet. This thesis deals with the integration of CANoe in the OMNeT ++ System Level Simulation for automotive networks. CANoe is a widely used simulation tool in the automotive industry from Vector Informatik GmbH. This integration allows carmakers to work in their usual simulation tool CANoe with the OMNeT++ system level simulation data.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

1 Einleitung

Moderne Fahrzeuge werden mittlerweile mit immer mehr Sensoren und Steuergeräten ausgestattet (siehe Abbildung 1.1), die die Sicherheit, die Zuverlässigkeit, die Leistung und den Komfort im Automobil steigern. Dadurch werden die Kommunikationsnetze immer komplexer und die zu übertragenden Datenmengen, die teilweise in Echtzeit übertragen werden müssen, steigen stetig an [vgl. Schauffele und Zurawka, 2006].

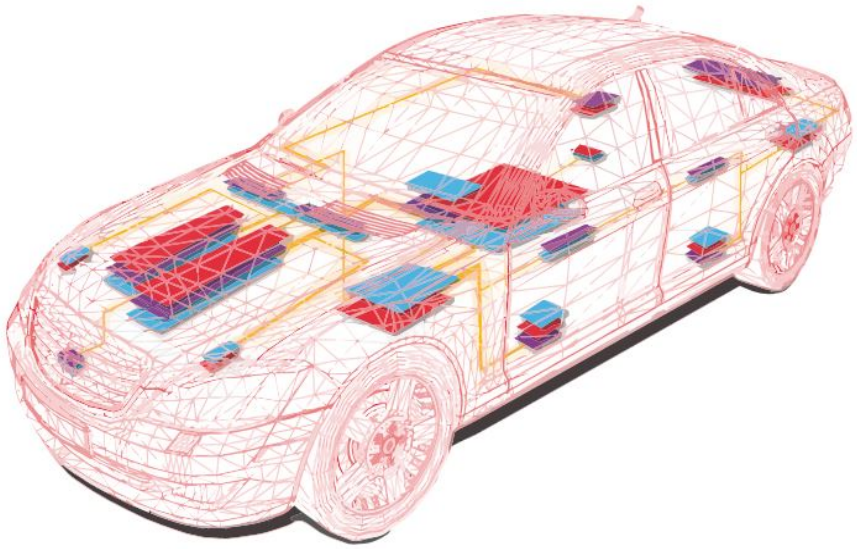

Abbildung 1.1: Elektronische Komponenten und deren Vernetzung in einem modernen Oberklassefahrzeug [CoRE Research Group, a]

Das hat zur Folge, dass die heutigen Bussysteme mit ihren geringen Bandbreiten (CAN 500 kBit/s, Highspeed-CAN 1MBit/s, FlexRay 10 MBit/s) in der Zukunft an ihre Grenzen stoßen werden. Eine mögliche Technologie, die eine hohe Bandbreite und eine deterministische Echtzeitkommunikation ermöglicht, ist das Time-Triggered-Ethernet. Die

vorliegende Arbeit ist in der Communication over Real-time Ethernet (CoRE) Arbeitsgruppe an der HAW Hamburg entstanden. Die CoRE-Arbeitsgruppe befasst sich mit der Realisierung von Echtzeit-Ethernet-Kommunikation im Auto und arbeitet mit der Simulationsumgebung OMNeT++ an verschiedenen Simulationsmodellen mit Echtzeit-Ethernet-Protokollen und Feldbussen.

In einer Phase, in der noch keine Hardware vorhanden ist, wird das Kommunikationsnetzwerk eines Autos mit der OMNeT++ System Level Simulation simuliert. Das Kommunikationsnetzwerk ist mit einem Ethernet ausgestattet, an das über Gateways verschiedene Feldbusse (z.B. FlexRay und CAN) angeschlossen werden können. So kann das Kommunikationsnetzwerk früh analysiert werden, um z.B. die Bandbreite oder die Auslastung auf einem Link zu untersuchen.

Da davon ausgegangen werden kann, dass die System Level Simulation in naher Zukunft für die Automobilindustrie von Bedeutung sein wird, geht es in dieser Arbeit darum, CANoe in die OMNeT++ System Level Simulation zu integrieren. CANoe ist eine Entwicklungssoftware der Firma Vector Informatik GmbH [Vector Informatik GmbH, 2015] und in der Automobilindustrie ein weit verbreitetes Werkzeug für Simulation, Analyse, Test und Diagnose von einzelnen Steuergeräten und Steuergerätenetzwerken. Die Vector Informatik GmbH wurde am 1. April 1988 gegründet und wurde zu einem internationalen, stetig wachsenden Unternehmen. Vector unterstützt Hersteller und Zulieferer der Automobilindustrie mit Werkzeugen, Softwarekomponenten und Dienstleistungen zur Entwicklung von eingebetteten Systemen.

Durch die Integration von CANoe wird dem Autobauer der Einstieg in die OMNeT++ System Level Simulation erleichtert, da dieser in seinem gewohnten Simulationswerkzeug CANoe mit den System Level Simulationsdaten arbeiten kann.

1.1 Ziel der Arbeit

Ziel der vorliegenden Arbeit ist es, Nutzern die Möglichkeit zu geben, die OMNeT++ System Level Simulationsdaten in die CANoe Simulationsumgebung zu integrieren. Diese Integration setzt sich aus mehreren Teilen zusammen. Zuerst muss ein Verfahren entwickelt und implementiert werden, das die Simulationsdaten aus der OMNeT++ Simulation in ein Logging-Format speichert. Dabei müssen Anforderungen untersucht werden, wie z.B. welche Daten dafür benötigt werden. Anschließend muss das Logging-Format in die CANoe-Simulationsumgebung eingelesen werden können, damit der CANoe-Nutzer die System Level Simulationsdaten in seiner gewohnten Simulationsumgebung weiterverarbeiten kann, wie z.B. die Auswertung der Linkauslastung im Netzwerk.

1.2 Aufbau der Arbeit

In Kapitel 2 werden Grundlagen beschrieben, die zum besseren Verständnis dieser Arbeit beitragen. Es werden die Grundmodelle der Simulation, das OSI-Schichtenmodell, die Simulationsumgebung OMNeT++ und eine Einführung in CANoe vorgestellt. Kapitel 3 beschäftigt sich mit den funktionalen und nicht-funktionalen Anforderungen des Aufzeichnungsprogramms und den funktionalen Anforderungen an die Datenerfassung. Auf Basis dieser Anforderungen wird in Kapitel 4 das Konzept entwickelt. Kapitel 5 beschriebt die Implementierung des Konzepts. Daraufhin wird in Kapitel 6 das in dieser Arbeit entwickelte Aufzeichnungsprogramm evaluiert. Abschließend wird die gesamte Arbeit zusammengefasst.

2 Grundlagen

In diesem Kapitel werden die notwendigen Grundlagen besprochen, die zum Verständnis dieser Arbeit beitragen. Zu Beginn werden in Abschnitt 2.1 die Grundlagen einer Simulation beschrieben. In Abschnitt 2.2 wird das verwendete Simulationswerkzeug OMNeT++ erläutert. Anschließend wird das OSI-Schichtenmodell kurz erläutert. Zum Abschluss folgt eine Einführung in CANoe.

2.1 Grundbegriff Simulation

Um Erkenntnisse über das reale System zu gewinnen, wird mithilfe von Computern und Simulationen das Verhalten von dynamischen, nicht trivialen Systemen und Prozessen analysiert.

Simulation ist das *„Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierbaren Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind; insbesondere werden die Prozesse über die Zeit entwickelt.“* [vgl. Verein Deutscher Ingenieure, 2014, S. 3]

Ein Modell ist die *„vereinfachte Nachbildung eines geplanten oder real existierenden Originalsystems und Prozesses in einem anderen begrifflichen oder gegenständlichen System. Es unterscheidet sich hinsichtlich der untersuchungsrelevanten Eigenschaften nur innerhalb eines vom Untersuchungsziel abhängigen Toleranzrahmens vom Bild.“* [vgl. Verein Deutscher Ingenieure, 2014, S. 3]

Simulationen werden nicht nur genutzt, um komplexe Systeme zu vereinfachen, sondern auch um ein Gefühl für dieses System zu erhalten. Bei der „Worst-Case-Time-Analyse“ auf Netzwerken wird das Zeitverhalten theoretisch analysiert. Dafür kann eine Simulation

eingesetzt werden, um den Paketverlauf nachverfolgen zu können. Abbildung 2.1 zeigt, dass das reine Experimentieren am realen System ebenfalls zum Simulationsmodell führen kann. Je nach untersuchtem System kann es verschiedene Gründe geben, die eine Untersuchung am realen System für nicht sinnvoll erscheinen lassen. Nach *Bossel* betragen die Kosten für die Modellerstellung und Simulation (siehe Abbildung 2.1) nur einen Bruchteil dessen, was die ähnliche Untersuchung am realen System kosten würde [vgl. Bossel, 2004, S. 15]. Außerdem könnte die Untersuchung am realen System zu aufwendig bzw. gefährlich sein oder das System beschädigen. Natürlich sind den Simulationen auch Grenzen gesetzt, da z.B. die verwendeten Modelle in der Simulation oft eine grobe und vereinfachte Form der Realität darstellen. Naturgemäß beeinträchtigt dies die Genauigkeit der Simulationsergebnisse.

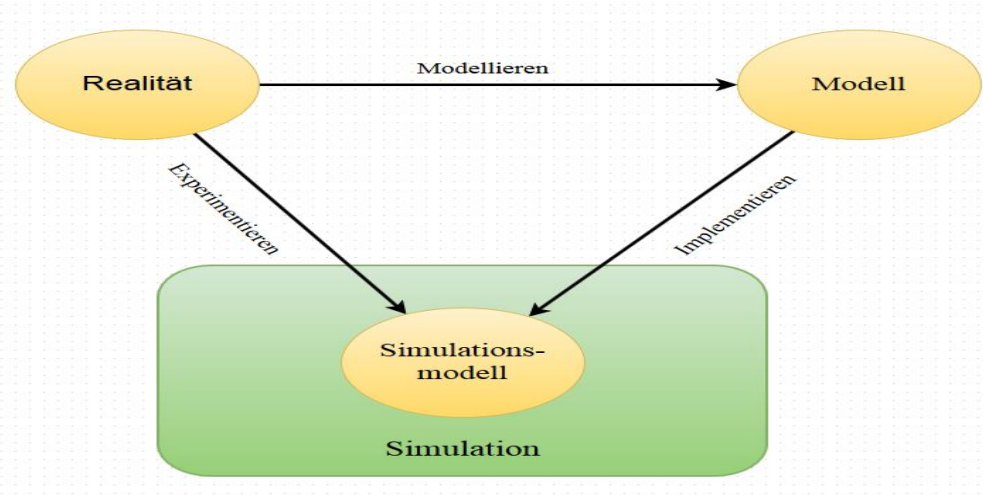

Abbildung 2.1: Gesamtansicht Simulation und Modellbildung [TU München, 2014]

2.1.1 Grundmodelle der Simulation

Grundsätzlich lassen sich dynamische Vorgänge auf zwei Grundmodelle zurückführen: auf kontinuierliche oder diskrete Systeme. Wie in Abbildung 2.2 zu sehen ist, ändert sich der Zustand bei der kontinuierlichen Simulation fortlaufend, wobei die Zeit bei solchen Systemen kontinuierlich oder diskret modelliert werden kann. Dementsprechend handelt es sich dann um eine numerische oder zeitdiskrete Simulation. Bei der ereignisdiskreten Simulation ändern sich sowohl der Zustand als auch die Zeit stets diskret. Diese Grundformen werden im Folgenden näher betrachtet.

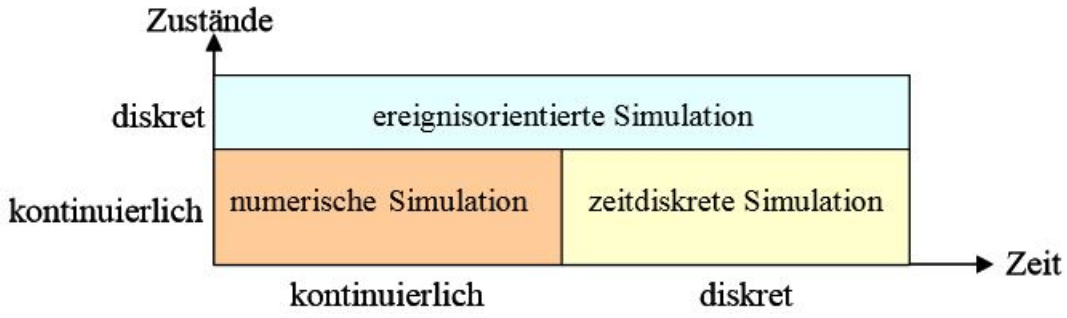

Abbildung 2.2: Klassifizierung von Simulationen [TU München, 2014]

2.1.2 Kontinuierlich numerische Simulation

Bei der kontinuierlich numerischen Simulation kann sich der Zustand in einem endlichen Zeitintervall unendlich oft ändern, wobei sich die Zeit in nur sehr kleinen Schritten ändert (siehe Abbildung 2.3). Diese Art der Simulation wird z.B. bei der Untersuchung von Schwingungen [Grudzinski u. a., 1992] verwendet.

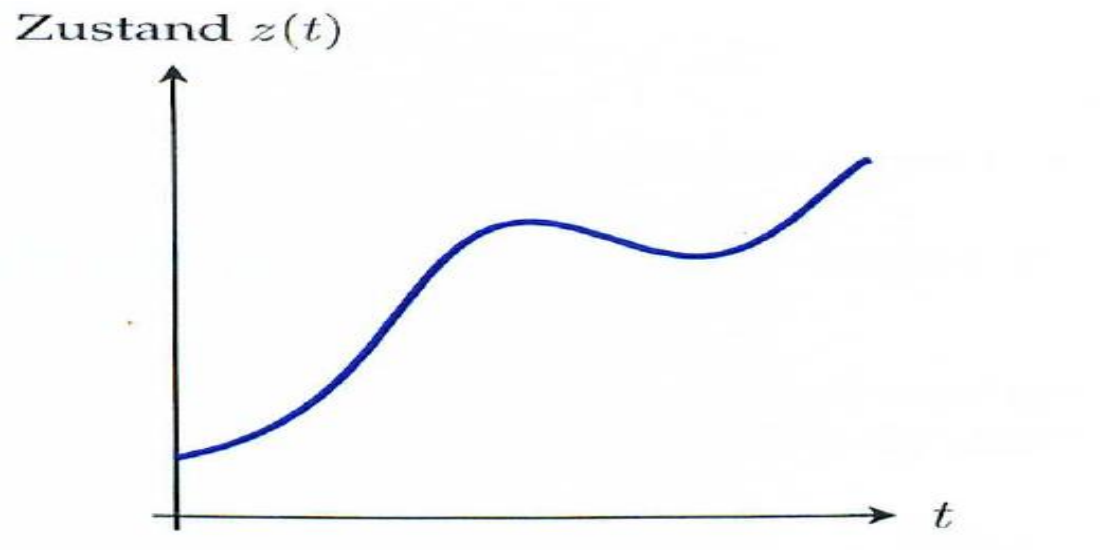

Abbildung 2.3: Numerische Simulation: Zustandsverlauf über die Zeit [Kiencke und Puente León, 2013]

2.1.3 Kontinuierlich zeitdiskrete Simulation

Wie in Abbildung 2.4 dargestellt ist, ändert sich bei der zeitdiskreten Simulation die Zeit in Intervallen Δt mit konstanter Länge. Der Zustand kann in den festgelegten Zeitabständen jeden beliebigen Wert annehmen. Im Gegensatz zu der numerischen Simulation ändert die zeitdiskrete Simulation ihre Zustandsgrößen sprunghaft. Wie im Vergleich der Abbildungen 2.3 und 2.4 zu sehen ist, nähert sich eine zeitdiskrete Simulation einer numerischen Simulation lediglich an.

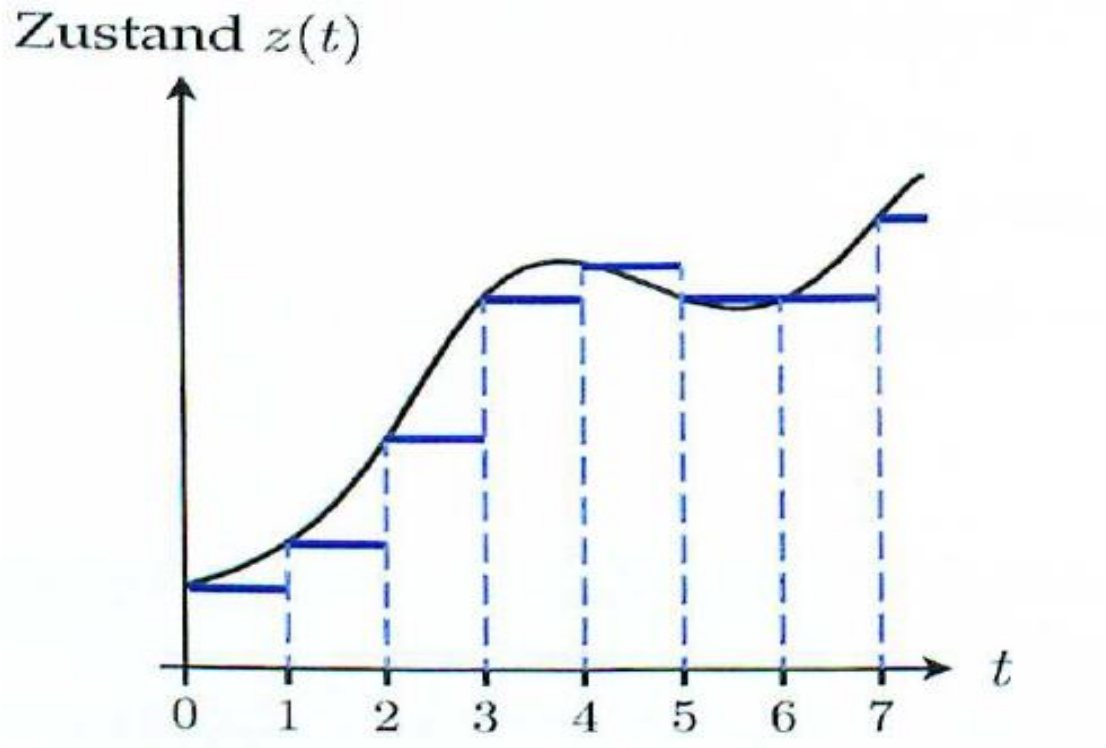

Abbildung 2.4: Zeitdiskrete Simulation: Zustandsverlauf über die Zeit [Kiencke und Puente León, 2013]

2.1.4 Ereignisdiskrete Simulation

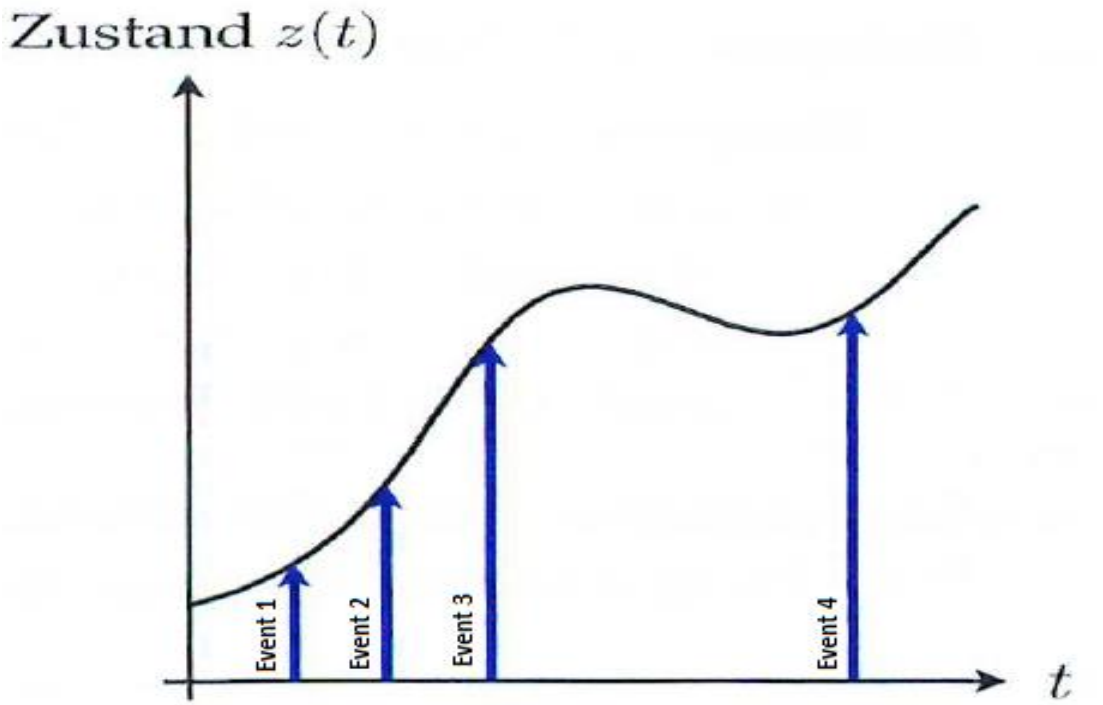

Abbildung 2.5: Ereignisdiskrete Simulation: Zustandsverlauf über die Zeit [Kiencke und Puente León, 2013]

Die Simulation ereignisdiskreter Systeme wird z.B. bei Simulationsmodellen von Fertigungseinrichtungen, Netzwerken oder Warteschlangen verwendet. Wie in Abbildung 2.5 zu sehen ist, ändert sich der Systemzustand ausschließlich in diskreten Zeitpunkten. Diese Zeitpunkte werden Ereignisse (Events) genannt, treten völlig unvorhergesehen auf und lösen die Zustandsänderung aus. [vgl. Kiencke und Puente León, 2013, S. 6]. Die wichtigsten Komponenten für die Simulation eines ereignisdiskreten Systems sind der aktuelle Zustand, die Simulationszeit und eine Liste mit den künftigen Events [vgl. Steinbach, 2011, S. 28]. Abbildung 2.6 zeigt den Ablauf einer ereignisdiskreten Simulation. Diese beginnt damit, dass das Modell initialisiert und dabei die Simulationszeit auf 0

gestellt wird. Anschließend werden alle Events aus der Liste innerhalb der Simulationsschleife abgearbeitet. Dabei wird der Systemzustand verändert und gegebenenfalls werden neue Ereignisse generiert. Die Simulationszeit wird immer mit dem Zeitpunkt des nächsten Events aktualisiert. Die Simulationsschleife wird bei Simulationsende abgebrochen und anschließend kann mit der Auswertung der Daten begonnen werden [vgl. Steinbach, 2011, S. 28 - 29].

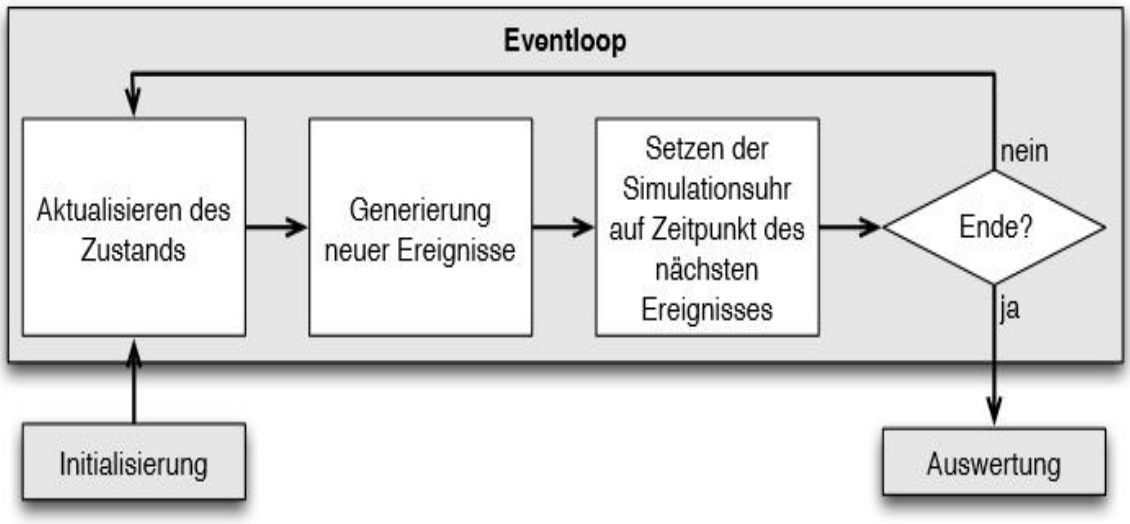

Abbildung 2.6: Ablauf einer ereignisdiskreten Simulation [Steinbach, 2011]

2.2 Simulationsumgebung OMNeT++

Objective Modular Network Testbed in C++ (OMNeT++) ist eine ereignisdiskrete Simulationsumgebung. Diese wird besonders bei der Modellierung von Kommunikationsnetzwerken, von verteilten und parallelen Systemen verwendet [vgl. OMNeT++ Community, b]. OMNeT++ ist eine Open Source-Umgebung, die für akademische sowie private Zwecke unter der GNU General Public License (GPL) kostenlos verwendbar ist [Varga, 2001]. Mit OMNEST wird auch eine kommerzielle Version angeboten.

Die OMNeT++ Entwicklungsumgebung basiert auf Eclipse [Eclipse Foundation] und kann unter Windows, Mac OS X und Linux betrieben werden. OMNeT++ Modelle bestehen aus Modulen, die über Nachrichtenaustausch miteinander kommunizieren. Die Module werden mit der OMNeT++ spezifischen Sprache Network Description (NED) erstellt. Es existieren zwei Arten von Modulen:

- *Simple-Module* beinhalten die Logik bzw. das Verhalten eines Modells. Diese werden mit der Programmiersprache C++ implementiert.
- *Compound-Module* können beliebig viele Simple- und Compound-Module enthalten. Abbildung 2.7 zeigt ein Compound-Modul, das aus weiteren Simple- bzw. Compound-Modulen besteht.

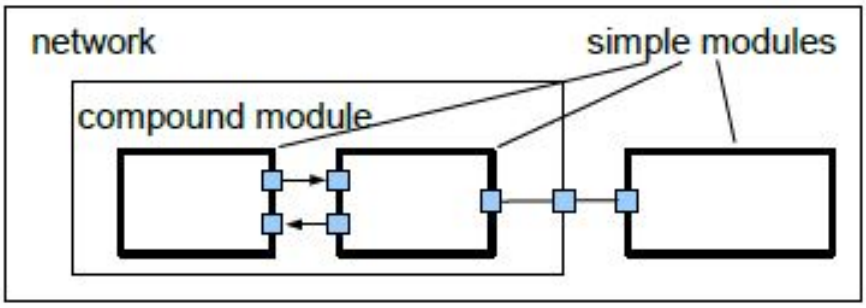

Abbildung 2.7: OMNeT++ Modell-Aufbau [Varga und Hornig, 2008]

Jedes Modul verfügt über eine Anzahl an Gates (siehe blaue Kästchen in Abbildung 2.7). Sie werden mithilfe von Connections verbunden, über welche der Nachrichtenaustausch erfolgt [vgl. Varga und Hornig, 2008]. Module und Connections können Konfigurationsparameter enthalten. Diese können in einer Initialisierungsdatei konfiguriert werden. Eine Connection mit spezifischen Eigenschaften wird *Channel* genannt.

Trifft eine Nachricht in einem Modul ein, wird vom Simulations-Kernel die *handleMessage*-Funktion aufgerufen. In dieser Funktion kann die weitere Vorgehensweise definiert werden. Über eine *send*-Funktion kann ein Modul eine Nachricht an das Gate eines anderen Moduls versenden. Mit der *scheduleAt*-Funktion kann ein Modul auch eine Nachricht an sich selbst versenden.

So besteht die Möglichkeit, jedes Netzwerkprotokoll umzusetzen und zu untersuchen. Der modulare Aufbau von OMNeT++ ermöglicht es, viele Netzwerkprotokolle und Teilnehmergeräte in unterschiedlichen Frameworks zu implementieren. In dieser Arbeit werden vier dieser Frameworks verwendet.

INET Framework

Das INET-Framework [OMNeT++ Community, a] ist eine Open Source Zusammenstellung verschiedener Netzwerk-Protokolle. Zu diesen implementierten Protokollen für OMNeT++ gehören z.B. UDP, TCP, SCTP, IP, IPv6 und Ethernet.

CoRE Frameworks

Die Communication over Real-time Ethernet (CoRE) Research Group [CoRE Research Group, a] arbeitet an Echtzeit-Netzwerksimulationen und hat mehrere Frameworks in OMNeT++ entwickelt. In der vorliegenden Arbeit werden drei dieser Frameworks für die Nutzung von CAN-Bussen und CAN-Komponenten im Echtzeit-Ethernet-Netzwerk verwendet.

CoRE4INET Framework

CoRE4INET ist eine Erweiterung des INET-Frameworks für die ereignisdiskrete Echtzeit-Ethernet-Simulation in OMNeT++ [vgl. CoRE4INET CoRE Research Group, b]. Diese wurde in der HAW Hamburg (Hochschule für Angewandte Wissenschaften) von der CoRE und INET (Internet Techologies) Research Group entwickelt und bietet z.B. folgende Erweiterungen:

- Time-Triggered Ethernet (TTEthernet)[1] (AS6802): bestehend aus den Time-Triggered (TT), Rate-constrained (RC) und Best-effort (BE) Traffic-Klassen.
- IEEE 802.1 Audio Video Bridging (AVB)[2]: bestehend aus den AVB Stream Klassen.

FiCo4OMNeT Framework

Die CoRE Research Group implementiert mit dem FiCo4OMNeT (Fieldbus Communication for OMNeT) Framework [CoRE Research Group, c] die Automobil-Feldbustechnologie CAN und FlexRay. Die Modelle wurden nach den neuesten Spezifikationen implementiert und mit etablierten kommerziellen Tools (wie CANoe) oder analytischen Modellen ausgewertet [vgl. Steinbach u. a., 2015].

Signals and Gateways Framework

Das Signals and Gateways Framework [CoRE Research Group, d] dient der Erstellung von Netzwerken bestehend aus Switches, Gateways und ECU's (Steuerungseinheiten). CAN-Nachrichten aus verschiedenen Fahrzeugsteuerungen treffen in den Gateways ein und werden in Ethernet-Frames umgewandelt. Im Ziel-Gateway werden die Ethernet-Frames wieder in CAN-Frames umgewandelt und an den Ziel-Feldbus versendet. Gateway-Konfigurationen werden über ein XML-File konfiguriert [vgl. Steinbach u. a., 2015].

[1] Ein Ethernet-basiertes Netzwerk mit Echtzeitverhalten.

[2] AVB ist ein Audionetzwerk-Protokoll mit dem Ziel der Festlegung eines international anerkannten Standards für die Echtzeit-Übertragung von Audio- und Videosignalen über ein Ethernet-basiertes Netzwerk [vgl. Heyden, 2012, S. 59].

2.3 Das OSI-Schichtenmodell

Mit dem ISO 7498 wurde das OSI-Modell (Open Systems Interconnection) für die Datenkommunikation in offenen Systemen entwickelt [vgl. Ernst u. a., 2015]. Das Hauptmerkmal ist eine hierarchische Strukturierung in sieben Schichten (Layer) (siehe Abbildung 2.8). Jede Schicht stellt unabhängig von allen anderen Schichten Dienste für Kommunikations- und Steueraufgaben zu Verfügung, womit die Funktion der darüber liegenden Schicht unterstützt wird. Art und Ablauf der Kommunikation werden durch die Protokolle festgelegt [vgl. Ernst u. a., 2015]. Das Ethernet- und CAN-Protokoll, das für diese Arbeit relevant ist, wird in der Bitübertragungs- und der Sicherungsschicht definiert.

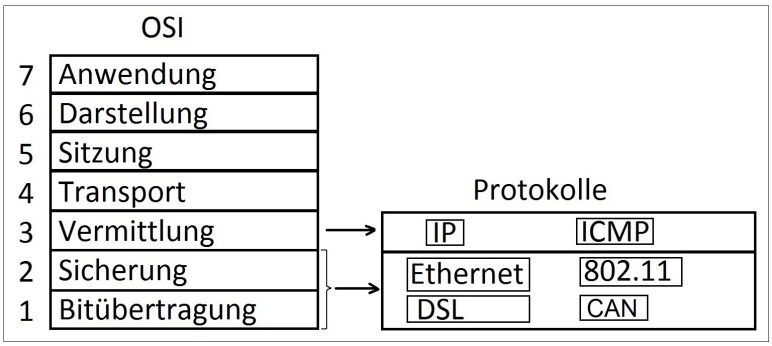

Abbildung 2.8: OSI-Schichtenmodell

1. Schicht (Bitübertragungsschicht, Phyisical Layer)

Die Bitübertragungsschicht definiert die elektrische, mechanische und funktionale Schnittstelle zum Übertragungsmedium, wobei das Übertragungsmedium nicht Bestandteil der ersten Schicht ist [vgl. Tanenbaum und Wetherall, 2012].

2. Schicht (Sicherungsschicht, Data Link Layer)

Die Sicherungsschicht sorgt für eine zuverlässige Verbindung zwischen Endgerät und Übertragungsmedium. Diese enthält Funktionen zur Fehlererkennung, Fehlerbehebung und Datenkontrolle. Außerdem findet in dieser Schicht die Adressierung von Datenpaketen statt.

3. Schicht (Vermittlungsschicht, Network Layer)

Die Vermittlungsschicht steuert die zeitlich und logisch getrennte Kommunikation zwischen den Endgeräten. Erstmals erfolgt in dieser Schicht die logische Adressierung der Endgeräte [vgl. Tanenbaum und Wetherall, 2012].

4. Schicht (Transportschicht, Transport Layer)

Die wesentlichen Aufgaben dieser Schicht sind die Bereitstellung verbindungsloser und verbindungsorientierter Transportmechanismen und Mechanismen zur Stau-[1] und Flusskontrolle[2].

5. Schicht (Sitzungsschicht, Session Layer)

Diese Schicht organisiert die Verbindungen zwischen den Endsystemen.

6. Schicht (Darstellungsschicht, Presentation Layer)

In dieser Schicht werden die Daten zu oder von der Anwendungsschicht in ein geeignetes Format umgewandelt.

7. Schicht (Anwendungsschicht, Application Layer)

Diese Schicht enthält die Anwendungsprotokolle und darauf aufbauende Dienste unter anderem zur Datenübertragung, Synchronisierung und Fernsteuerung von Rechnern und der Namensauflösung [vgl. Baun, 2013].

[1] Reaktion auf Überlast im Netz
[2] Laststeuerung durch den Empfänger

2.4 CANoe

CANoe ist eine Entwicklungssoftware der Firma Vector Informatik GmbH [Vector Informatik GmbH, 2015] für Simulation, Analyse, Test und Diagnose von Steuergerätenetzwerken. Dabei wird der Benutzer über den gesamten Entwicklungsprozess von der Planung bis hin zur Inbetriebnahme von Steuergeräten oder verteilten Systemen unterstützt [vgl. User Manual Vector Informatik GmbH, b]. CANoe ist nicht nur auf die Entwicklung CAN-basierter Netzwerke ausgelegt, sondern unterstützt auch die Bussysteme LIN, MOST, Ethernet und FlexRay. Durch die Vielzahl an Bussystemen eignet sich CANoe besonders gut für die Entwicklung von Steuergeräten für konventionelle sowie Hybrid- und Elektrofahrzeuge. CANoe ist ein aus vielen Einzeltools aufgebautes Softwarepaket (siehe Abbildung 2.9) und besteht aus den folgenden Komponenten:

- *CANoe Hauptprogramm*
- *CAPL Browser*: Programmierumgebung
- *Panel Editor*: Erstellung von Nutzeroberflächen
- *Datenbank CANdb++*: Für die Verwaltung und Definition der Nachrichten

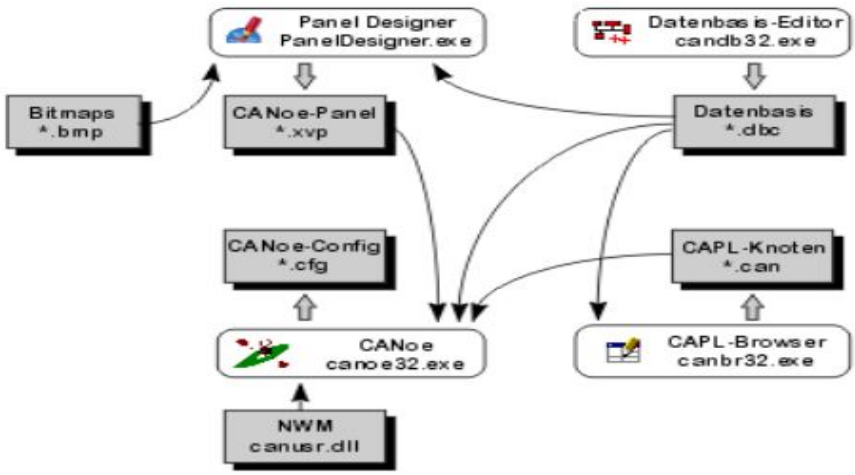

Abbildung 2.9: CANoe Systemübersicht [User Manual Vector Informatik GmbH, b]

In CANoe werden Nachrichten als Botschaften und übertragene Daten als Signale bezeichnet. CANoe verfügt mit CAPL über eine eigene event-orientierte Programmiersprache. So können mit CAPL Timer gesetzt, Signale verändert und Botschaften gesendet bzw. empfangen werden.

2.4.1 Einführung in CANoe

Bevor eine Simulation in CANoe dargestellt werden kann, muss das Netzwerk im Simulations-Setup konfiguriert werden (siehe Abbildung 2.10). Dieses besteht aus dem Simulationsaufbau- und dem Messaufbau-Fenster.

Simulationsaufbau-Fenster

In diesem Fenster wird das Gesamtsystem mit den Netzen und Geräten graphisch dargestellt. CANoe kann sowohl reale als auch simulierte Netzknoten enthalten. Es ist sowohl eine komplette Simulation als auch eine Restbussimulation möglich. Dies wird im Simulationsaufbau-Fenster durch die Darstellung der Busanbindung simulierter Netzknoten mittels roter Linien und realer Netzknoten mittels schwarzer Linien (siehe Abbildung 2.10) verdeutlicht. Die folgenden simulierten Netzknoten können an den simulierten Bus angeschlossen werden:

- *Knoten*: Repräsentieren unterschiedliche ECUs, werden in CAPL oder anderen Programmen (.NET, MATLAB/Simulink) implementiert.
- *Replay-Block*: Ermöglicht die Wiedergabe bereits aufgezeichneter Messabläufe mithilfe einer Logging-Datei.
- *Generator-Block*: Erzeugt und sendet Botschaften (z.B. auf einen Tastendruck).
- *Interaktiver Generator-Block*: Erzeugt und sendet Botschaften. Ermöglicht das Übertragen von Botschaften zwischen zwei Bussen (Gateway-Funktionalität).

Anbindungen an den realen Bus werden mithilfe von PC-Einsteckkarten realisiert.

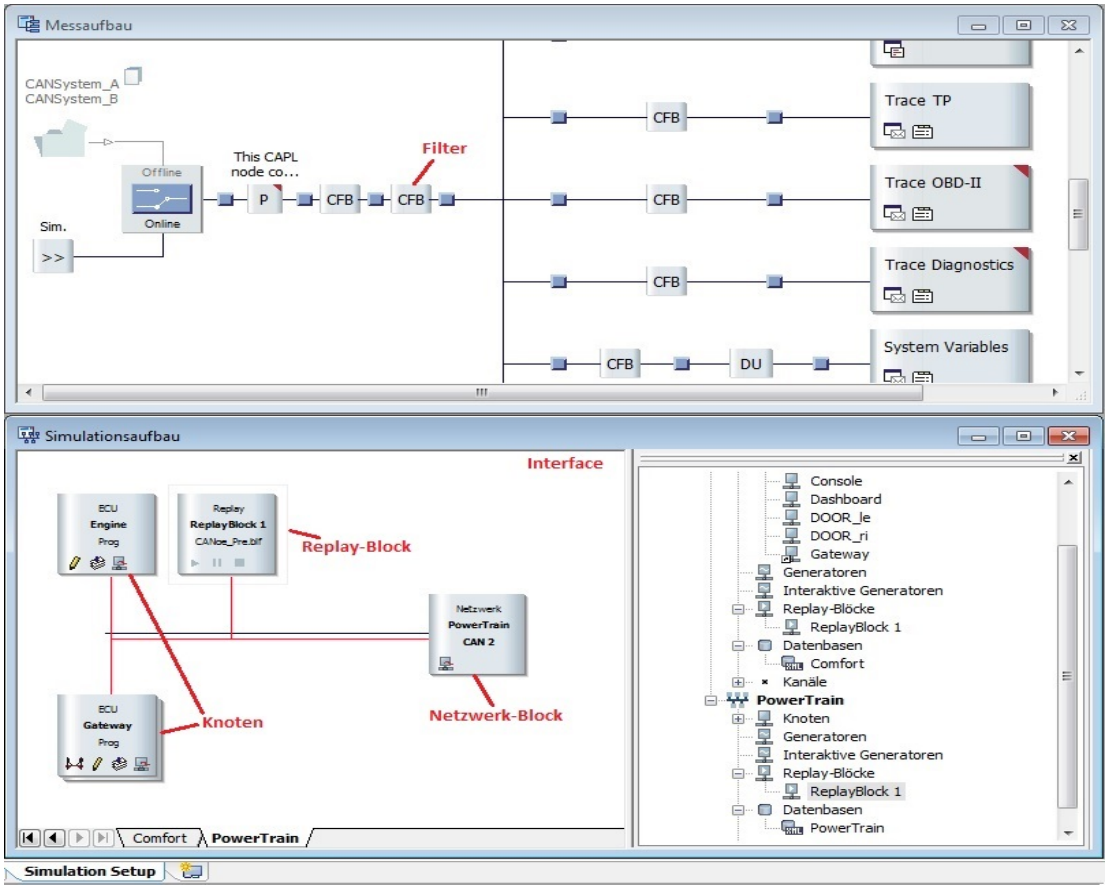

Abbildung 2.10: Simulations-Setup [Vector Informatik GmbH, 2015]

Messaufbau-Fenster

Im Messaufbau-Fenster (siehe Abbildung 2.10) wird der Datenfluss des Netzwerkes in CANoe graphisch dargestellt. Über diesen können Auswertungswerkzeuge symbolisch am Bus angeschlossen werden [Spinczyk u. a., 2007]. Dazu gehören z.B.:

- *Grafik-Block*: Graphische Darstellung der Signale von Botschaften.
- *Trace-Block*: Ermöglicht die Anzeige von Busaktivitäten.
- *Logging-Block*: Ermöglicht die Aufzeichnung von Busaktivitäten.

CANoe-Simulation

Wenn alle Konfigurationen im Simulations-Setup abgeschlossen sind, kann die Simulation gestartet werden. In Abbildung 2.11 wird anhand eines Beispiels [Vector Informatik GmbH, 2015, CANoe Sample Configurations] gezeigt, wie so eine Simulation aussehen kann. In dem Beispiel wurden das Panel Editor Dashboard und das Control-Fenster erstellt. In dem Trace-Fenster wird die Busaktivität analysiert und protokolliert. Dabei können mithilfe von Filtern Botschaften aus dem Trace-Fenster isoliert werden. Im Grafik-Fenster der Abbildung 2.11 können Signale, Umgebungsvariablen und Systemvariablen während der Simulation graphisch dargestellt und nach Simulationsende analysiert werden.

Abbildung 2.11: CANoe Simulation [Vector Informatik GmbH, 2015]

3 Anforderungen

Im Mittelpunkt dieser Arbeit stehen die Entwicklung und Implementierung eines Aufzeichnungsprogramms, um die System-Level-Simulationsdaten von OMNeT++ in CANoe zu übertragen. In diesem Kapitel werden die Anforderungen an das Aufzeichnungsprogramm erläutert, welche in funktionale und nicht-funktionale Anforderungen aufgeteilt werden können.

3.1 Nicht-funktionale Anforderungen

Die Integration von CANoe und der OMNeT++ System-Level-Simulation muss über eine Schnittstelle gelöst werden. CANoe bietet mit den Replay-Blocks die Möglichkeit, den bereits abgeschlossenen Nachrichtenaustausch über ein Logging-Format einzulesen. Es muss dafür gesorgt werden, dass die OMNeT++ Simulation ein Logging-Format generiert, das die CANoe-Simulationsumgebung lesen kann. OMNeT++ bietet einige Methoden an, um Simulationsdaten in einer Logdatei abzuspeichern.

3.2 Funktionale Anforderungen

In diesem Abschnitt werden die funktionalen Anforderungen analysiert.

Ethernet-Netzwerke

Das Aufzeichnungsprogramm muss gewährleisten, dass der Austausch von Botschaften im Ethernet-Netzwerk aufgezeichnet wird.

CAN-Netzwerke

CAN-Netzwerke müssen ebenfalls unterstützt werden. Das Aufzeichnungsprogramm muss eine Kombination aus Ethernet- und CAN-Netzwerken beherrschen.

Gateway Nachrichten-Transformation

Gateways befassen sich mit den sieben Schichten des OSI-Schichtenmodells (siehe Abschnitt 2.3) und koppeln die unterschiedlichen Protokolle und Übertragungsverfahren miteinander [vgl. Borowka, 1992, S. 54]. Ethernet-Netzwerke werden mit CAN-Netzwerken über Gateways miteinander verbunden, damit sich die ECUs trotz der unterschiedlichen Protokolle miteinander unterhalten können. Dementsprechend muss das Aufzeichnungsprogramm die Gateway-Nachrichten (Gateway Aggregation Message und Unit Message) aufzeichnen (dazu mehr im Abschnitt 4.4.6).

Simulationsgeschwindigkeit

Die Simulationsgeschwindigkeit darf vom Aufzeichnungsprogramm nicht stark beeinflusst werden.

Format der Logging-Datei

Die vom Aufzeichnungsprogramm erzeugten Logging-Dateien müssen von der CANoe-Simulationsumgebung gelesen und abgespielt werden können.

Vollständigkeit der Nachrichten

Es muss sichergestellt werden, dass keine Nachrichten verloren gehen, ansonsten entstehen Lücken, die wiederum eine Verfälschung der Aufzeichnung verursachen.

Duplikate

Es dürfen keine Nachrichten doppelt (Duplikate) aufgezeichnet werden.

Zeitstempel

Bei der Erfassung und Aufzeichnung der Daten muss gewährleistet werden, dass ein Zeitstempel gespeichert wird. Dabei sind die Genauigkeit (in Nanosekunden) und Auflösung der Zeitstempel zu beachten.

Serializer

Nachrichten, die während der Simulation versendet und empfangen werden, sind Objekte mit Eigenschaften. Diese Objekte müssen über Serializer in einen Datenstrom (Bytestrom) umgewandelt werden, um diese in einer Logging-Datei abzuspeichern. Daher müssen Serializer für Ethernet-, CAN- und Gateway-Nachrichten implementiert werden.

4 Konzept

Im Folgenden wird das Konzept für den Aufbau des Aufzeichnungsprogramms erläutert.

4.1 Datenerfassung

Problemstellung

In den heutigen Fahrzeugen werden Steuergeräte über verschiedene Systembusse (CAN, Ethernet, MOST, LIN, FlexRay) miteinander verbunden. Steuergeräte erzeugen Daten und teilen diese mit anderen Steuergeräten. Die versendeten Nutzdaten werden durch die Protokollschicht in sogenannten Frames (z.B Ethernet- und CAN-Frames) verpackt, die sowohl die Nutzdaten als auch die Steuer- und Kontrollinformationen beinhalten [Hüning, 2016].

Diese Frames sollen vom Aufzeichnungsprogramm in einer Logging-Datei gespeichert werden, damit diese in CANoe eingelesen werden können. Anschließend kann der CANoe-Anwender die Simulationsdaten in seiner gewohnten Umgebung auswerten. Das Erfassen und Speichern dieser Simulationsdaten ist ein Hauptbestandteil dieser Arbeit. Dafür stehen drei Methoden zur Auswahl.

Um eine Auswahl zwischen den Methoden treffen zu können, wird im Folgenden Abschnitt jede Methode kurz beschrieben und ihre Vor- und Nachteile aufgelistet. Schließlich wird eine Entscheidung getroffen, welche Methode am besten für diese Arbeit geeignet ist.

4.1.1 PcapRecorder

Das INET-Framework bietet mit dem *PcapRecorder* eine Methode zur Aufzeichnung von Simulationsdaten. Dieser ist als NED-Modul implementiert und muss als Submodul in das Modul, in dem die Aufnahme durchgeführt werden soll, eingefügt werden.

Der PcapRecorder unterstützt nur die Aufzeichnung von IPv4-, IPv6-, SCTP- und ICMP-Nachrichten, die innerhalb eines Netzwerkes versendet oder empfangen wurden. Nach Simulationsende wird eine Packet Capture (PCAP) Datei erstellt. Die PCAP-Datei ist ein weitverbreitetes Standard-Logging-Format für Kommunikationsdaten.

Nach dem OSI-Referenzmodel arbeitet der PcapRecorder auf der dritten und vierten Schicht. Das Ziel dieser Arbeit ist die Aufzeichnung von Ethernet- und CAN-Nachrichten, die der PcapRecoder nicht unterstützt.

Vorteile

- Bereits in OMNeT++ integriert.
- CANoe kann die PCAP-Datei lesen und abspielen.
- Simulationsgewindigkeit wird kaum verringert.
- Moderate Dateigröße.

Nachteile

- Die Aufzeichnung von Ethernet- und CAN-Nachrichten wird nicht unterstützt.

4.1.2 Eventlog

OMNeT++ bietet mit dem Eventlog (ab Version 4.0) eine weitere Möglichkeit, Simulationsdaten aufzuzeichnen. Alle aufgetretenen Events werden geloggt, wie zum Beispiel:

- Senden und Empfangen von Nachrichten.
- Erstellen und Löschen von Modulen.
- Erstellen und Löschen von Connections usw.

Dabei werden alle Events mit einer ID und einem Zeitstempel versehen [vgl. Manual OMNeT++ Community, c, Kapitel 13.]. Nach Simulationsende wird eine .elog-Datei in dem dafür vorgesehenen *results*-Ordner erstellt. Jede Zeile beginnt mit einem Identifier, welcher bekannt gibt, um welche Daten es sich handelt. Im OMNeT++ Manual gibt es eine genaue Auflistung aller Identifier [vgl. Manual OMNeT++ Community, c, Kapitel 27.]. Optional können auch alle Nachrichten, die während der Simulation versendet oder empfangen wurden, aufgezeichnet werden. Die Aufzeichnung der Nachrichten erhöht die Dateigröße dramatisch und verringert die Simulationsgeschwindigkeit [vgl. Manual OMNeT++ Community, c, Kapitel 13.2.4]. Das Eventlog-Verfahren wird über einer Initialisierungs-Datei aktiviert. Beim Testen der Methoden ist aufgefallen, dass die Simulationsgeschwindigkeit im Vergleich zum Einsatz des PcapRecorders deutlich verringert war. In Abschnitt 6.2 wird dies näher betrachtet und tabellarisch festgehalten.

Vorteile

- Bereits in OMNeT++ integriert.
- Alle Events und Nachrichten können aufgezeichnet werden.

Nachteile

- CANoe kann nativ keine Eventlog-Datei lesen.
- Die .elog-Datei müsste in ein anderes Logging-Format umgewandelt werden.
- Die daraus resultierende Dateigröße steigt bei der Aufzeichnung der Nachrichten stark an.

- Die Eventlog-Datei enthält viele Informationen, die für diese Arbeit nicht relevant sind. Das Parsen der relevanten Daten ist wegen der Dateigröße zeitaufwendig.
- Die Simulationsgeschwindigkeit ist im Vergleich zum Einsatz des PcapRecorders deutlich verringert.

4.1.3 Eigener Logging-Recorder

Die letzte Methode, die in diesem Abschnitt vorgestellt wird, ist nicht in OMNeT++ integriert. Der eigene Logging-Recorder baut auf dem PcapRecorder-Prinzip (vom INET-Framework) auf. Der Logging-Recorder wird als NED-Modul implementiert, dabei wird die dazugehörige Logik in C++ geschrieben (siehe Abschnitt 5.1 und Abschnitt 5.2). Wie der PcapRecorder wird dieser als Submodul in das Modul, in dem die Aufnahme durchgeführt werden soll, eingebunden. Es ermöglicht die Aufzeichnung aller gesendeten und empfangenen Ethernet- und CAN-Nachrichten eines Moduls im Netzwerk. Diese Nachrichten werden in einer Logging-Datei gespeichert. Das Logging-Format der resultierenden Datei wird so ausgewählt, dass diese in CANoe nativ eingelesen und abgespielt werden kann.

Vorteile

- CANoe kann die resultierende Logging-Datei nativ lesen und abspielen.
- Die Simulationsgeschwindigkeit wird durch den PcapRecorder kaum verringert.
- Die Logging-Datei enthält nur die gewünschten Simulationsdaten (Ethernet- und CAN-Frames).

Nachteile

- Programm muss implementiert werden.

4.1.4 Zusammenfassung und Wahl der Methode

Die Eventlog-Datei speichert alle Events und Nachrichten, die während der Simulation aufgetreten sind. Die daraus resultierende Dateigröße steigt stark an und darüber hinaus verringert sich die Simulationsgeschwindigkeit drastisch. Die Eventlog-Datei müsste in ein Format umgewandelt werden, das in CANoe eingelesen werden kann. Das Parsen der relevanten Daten ist aufgrund der Dateigröße zeitaufwendig. Wegen dieser Nachteile ist dieses Verfahren für diese Arbeit ungeeignet.

Der PcapRecorder liefert hingegen mit der PCAP-Datei ein Format, das CANoe nativ lesen und abspielen kann. Die Simulationsgeschwindigkeit wird durch den PcapRecorder nur leicht beeinflusst. Da die Ethernet- und CAN-Nachrichten nicht aufgezeichnet werden, ist der PcapRecorder für diese Arbeit ungeeignet.

Beim eigenen Logging-Recorder entsteht ein Aufwand für dessen Implementierung, dieser erfüllt jedoch alle Anforderungen aus dem letzten Kapitel. Der eigene Logging-Recorder wird vom PcapRecorder (INET-Framework) erben[1]. Es wird ein Logging-Format erzeugt, das die CANoe-Simulationsumgebung nativ lesen und abspielen kann. Alle gewünschten Simulationsdaten sind enthalten. Aus den genannten Gründen fiel die Wahl auf den eigenen Logging-Recorder.

[1] Vererbung ist ein Mechanismus, der es ermöglicht, Attribute und Methoden von Klassen an andere Klassen weiterzugeben.

4.2 Architekturkonzept

Das Ziel dieser Arbeit ist es, eine Schnittstelle zwischen CANoe und OMNeT++ aufzubauen. Um das zu erreichen, müssen mehrere Schritte vorgenommen werden (siehe Abbildung 4.2):

1. Ein Netzwerk auswählen, in dem das Aufzeichnungsprogramm angebunden werden soll. Ein Netzwerk kann Ethernet-Nodes, Switches, Gateways, CAN-Busse und CAN-Nodes beinhalten. Abbildung 4.1 zeigt ein geeignetes Netzwerk, das mit dem Logging-Recorder aufgenommen werden kann.

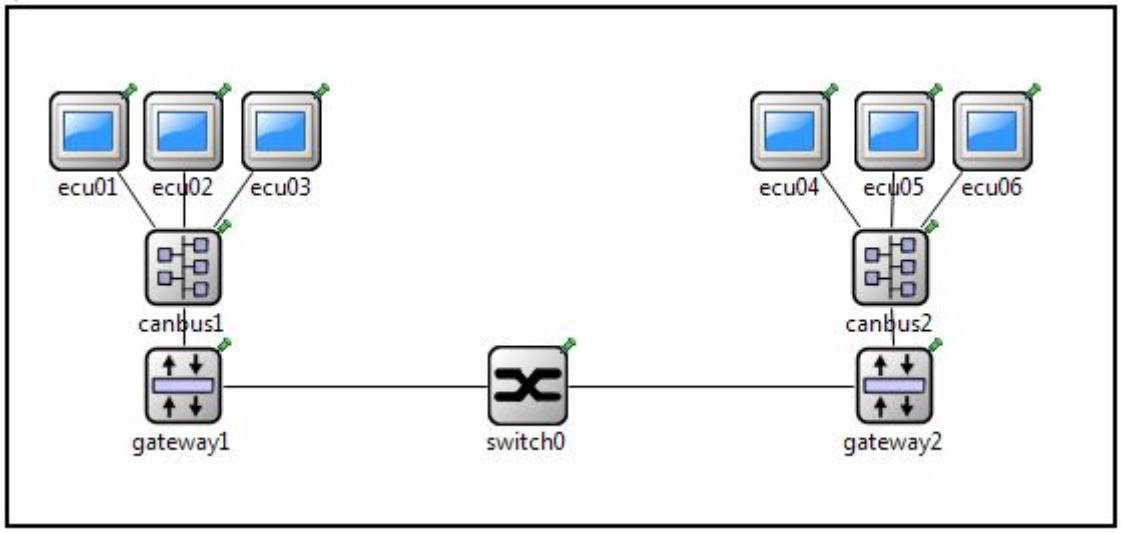

Abbildung 4.1: OMNeT++ Netzwerk aus Ethernet- und CAN-Komponenten

2. Der Logging-Recorder wird als Submodul eingebunden. Dafür gibt es zwei Varianten:

   - Der Logging-Recorder kann pro Modul (Node, Switch, Gateway etc.) im Netzwerk angebunden werden, dann wird pro Modul eine Logging-Datei erstellt.

- In Abbildung 4.1 ist ein Switch (switch0) zu sehen, an den zwei Gateways (gateway1 und gateway2) angeschlossen sind. Damit sind zwei physikalische Ports des Switches besetzt. Dementsprechend kann der Logging-Recorder an dem phyikalischen Port angebunden werden und es wird für beide besetzten Port jeweils eine Logging-Datei erstellt. Das ist sinnvoll, wenn in CANoe nachverfolgt werden soll, wann eine Nachricht an einem bestimmten Port ankommt oder diesen verlässt.

Für jede Logging-Datei, die in CANoe über einen Replay-Block angebunden wurde, wird ein Nachrichtenkanal in CANoe besetzt. Dabei ist zu beachten, dass CANoe maximal 32 Ethernet-Kanäle und 32 CAN-Kanäle verwalten kann [vgl. User Manual Vector Informatik GmbH, b].

3. Wurde der Logging-Recorder angebunden, muss dieser in der omnetpp.ini Datei konfiguriert werden.

4. Anschließend kann die Simulation gestartet werden. Dabei werden alle Ethernet- und CAN-Frames aufgezeichnet. Als Output werden eine oder mehrere Logging-Dateien erzeugt.

5. Schließlich können alle erzeugten Logging-Dateien als Input in CANoe eingelesen werden (siehe Abbildung 4.2). Somit kann der CANoe-Anwender die OMNeT++ System Level Simulationsdaten in seiner gewohnten Simulationsumgebung mit der Programmiersprache CAPL weiter verarbeiten und diese in CANoe visualisieren. Abbildung 4.2 veranschaulicht diesen Gesamtprozess.

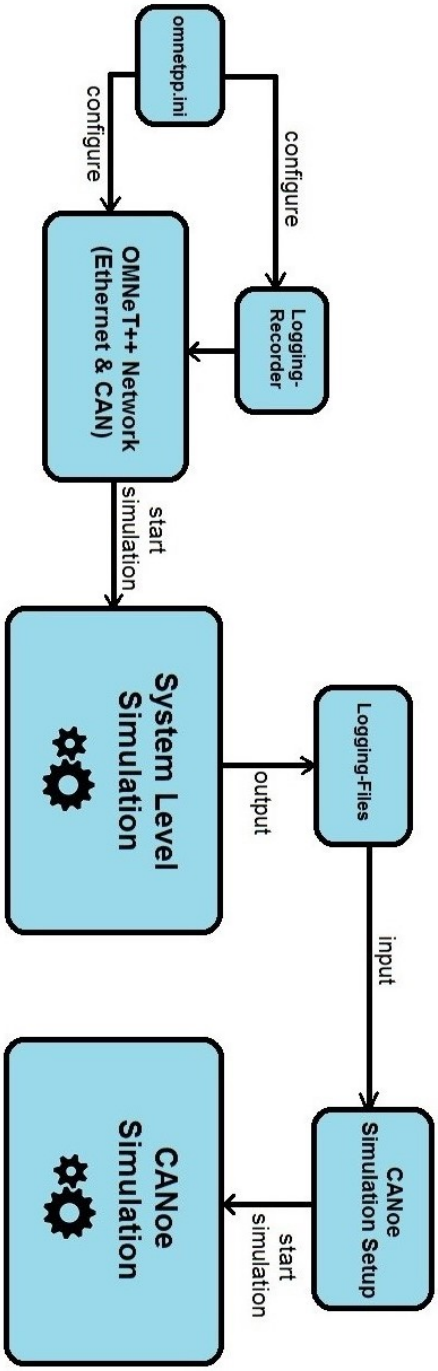

Abbildung 4.2: Gesamtprozessverlauf des Logging-Recorders

4.3 Logging-Formate

In diesem Abschnitt werden die verwendeten Logging-Formate näher erläutert.

4.3.1 Packet Capture Format

Das Packet Capture (PCAP) Logging-Format ermöglicht die Abspeicherung von Netzwerkkommunikationsdaten in einer binären Datei. In dieser Arbeit wird es für die Aufzeichnung von Ethernet- und CAN-Frames genutzt. Unix-Betriebssysteme können die *libpcap*-Bibliothek nutzen. Diese ist Bestandteil des Programmes *Tcpdump* [The Tcpdump team] und bietet eine Menge von Funktionen, die den bequemen Zugriff auf empfangene Datenpakete ermöglichen. Beim Mitlauschen von Paketströmen wird dabei zwischen Online- und Offline-Capturing unterschieden. Online-Capturing ist in dieser Arbeit der Prozess, mit dem Nachrichten zur Simulationszeit erfasst und in einer PCAP-Datei aufgezeichnet werden. Offline-Capturing wird angewendet, wenn zuvor aufgezeichnete Nachrichten aus einer PCAP-Datei ausgelesen und analysiert werden.

Diese Funktionalitäten werden auch für Windows-PCs mit der *WinPcap*-Bibliothek [Riverbed Technology] angeboten. Die *WinPcap*-Bibliothek wird im Programm *Wireshark* genutzt und für die Analyse von Netzwerkkommunikationsverbindungen (Sniffer) verwendet.

Abbildung 4.3: Schematischer Aufbau der PCAP-Datei

Für das PCAP-Format sind bereits Formate für viele Netzwerk-Protokolle definiert. Zum Beispiel werden das Ethernet- und das CAN-Protokoll unterstützt, die für diese Arbeit eine besondere Rolle spielen. Die Struktur bzw. das Format der PCAP-Datei ist für alle Nutzer frei einsehbar und wird deswegen in vielen Programmen zur Netzwerkanalyse genutzt, so z.B. auch in CANoe.

```
struct pcap_hdr {
    uint32 magic_number;   /* magic number */
    uint16 version_major;  /* major version number */
    uint16 version_minor;  /* minor version number */
    int32  thiszone;       /* GMT to local correction */
    uint32 sigfigs;        /* accuracy of timestamps */
    uint32 snaplen;        /* max length of captured packets, in octets */
    uint32 network;        /* data link type */
};
```

Listing 4.1: PCAP Global Header

Abbildung 4.3 zeigt den strukturellen Aufbau einer PCAP-Datei. Der *Global Header* (siehe Listing 4.1) wird als Struktur implementiert und muss bei der Erzeugung einer PCAP-Datei einmal definiert werden. Der *Global Header* aus Listing 4.1 ist wie folgt aufgebaut:

- ***magic_number* (4 Byte):** Bestimmt die Byte-Ordnung der PCAP-Datei. Diese können nach Big-Endian (0xa1b2c3d4) oder Little-Endian (0xd4c3b2a1) geordnet sein.
- ***version_major & minor* (4 Byte):** Die Versionsnummer des Dateiformats (aktuelle Version: 2.4) . Beide Zahlen werden mit je zwei Bytes repräsentiert.
- ***thiszone* (4 Byte):** Zeitzonen-Offset.
- ***sigfigs* (4 Byte):** Diese Zahl soll theoretisch die Genauigkeit der Zeitstempel in der Datei angeben.
- ***snaplen* (4 Byte):** Gibt die maximale Paketgröße an.
- ***network* (4 Byte):** Diese wird genutzt, um Netzwerk-Protokolle zu definieren. Für diese Arbeit werden das Ethernet-Protokoll (0x1) und das CAN-Protokoll (0xE3) benötigt.

```
struct pcaprec_hdr{
   int32 ts_sec;      /* timestamp seconds */
   uint32 ts_usec;    /* timestamp microseconds */
   uint32 incl_len;   /* number of octets of packet saved in file */
   uint32 orig_len;   /* actual length of packet */
};
```

Listing 4.2: PCAP Packet Header

Wie in Abbildung 4.3 zu sehen ist, muss der *Packet Header* (siehe Listing 4.2) für jedes erfasste Packet definiert werden. Dieser wird wie der *Global Header* als Struktur implementiert (siehe Listing 4.2). Die *Packet-Header-Struktur* ist wie folgt aufgebaut:

- ***ts_sec* (4 Byte) und *ts_usec* (4 Byte):** Die ersten vier Byte (ts_sec) geben die vergangenen Sekunden seit dem 1. Januar 1970 00:00:00 GMT an und die nächsten vier Byte (ts_usec) die Zeitdifferenz in Microsekunden.
- ***incl_len* (4 Byte):** Die tatsächlich aufgezeichnete Länge des Datenpaketes.
- ***orig_len* (4 Byte):** Die tatsächliche Länge des Datenpaketes.

PCAP-Format in der CANoe-Simulationsumgebung

Wie schon beschrieben, können in der PCAP-Datei sowohl Ethernet- als auch CAN-Nachrichten aufgezeichnet werden. Im Laufe dieser Arbeit zeigte sich jedoch, dass CANoe nur Ethernet-Nachrichten aus PCAP-Dateien lesen kann [vgl. Hilfe Vector Informatik GmbH, 2015]. Aus diesem Grund musste ein anderes Format für den Logging-Recorder eingesetzt werden.

Da PCAP ein weit verbreitetes Standard-Format zur Aufzeichnung von Netzwerk-Daten ist, wurde dieses Format dennoch beibehalten. Das zweite Logging-Format wird im Folgenden näher erläutert.

4.3.2 Binary Logging Format (BLF)

Das Binary Logging Format (BLF) wurde von der Vector Informatik GmbH entwickelt und wird in CANoe und CANalyzer verwendet. Es ist wie PCAP ein Binärformat und wird zur Aufzeichnung von Nachrichten in Ethernet- und CAN-Netzwerken verwendet. Auch die Aufzeichnung von anderen Bussystemen wie LIN, MOST und FlexRay ist mit diesem Format möglich.

Was dieses Format besonders auszeichnet, ist die Speicherung der Daten in einer sehr effizienten Art und Weise bezüglich der Dateigröße, Lese- und Schreibperformance. Vector bietet für Entwickler, die das BLF Format nutzen wollen, folgende Dateien an:

- *binlog_objects.h*: Header mit allen BLF-Strukturen und Definitionen (defines).
- *binlog.h*: Header mit allen Funktionsdeklarationen (Prototypen).
- *binlog.dll*: BLF-Bibliothek mit den implementieren Methoden zu den Funktionsdeklarationen aus der binlog.h-Datei.

Abbildung 4.4 zeigt den Aufbau einer BLF-Datei für Ethernet oder CAN. Für den Aufbau spielen vier verschiedene Strukturen (Header) eine wichtige Rolle und werden deshalb näher erläutert.

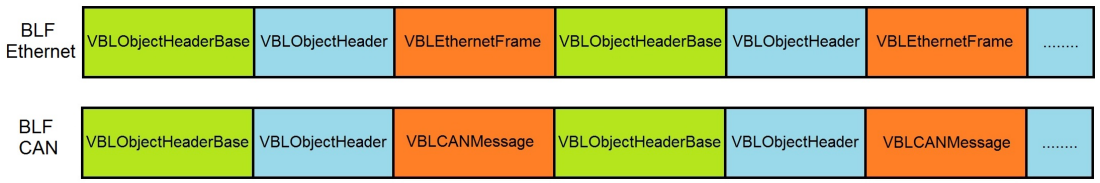

Abbildung 4.4: Schematischer Aufbau der BLF-Datei für Ethernet oder CAN

VBLObjectHeaderBase Struktur

```
/* --------------------- VBLObjectHeaderBase ----------------------------*/
typedef struct VBLObjectHeaderBase{
   uint32     mSignature;      // Object signature, must be BL_OBJ_SIGNATURE.
   uint16     mHeaderSize;     // Size of header in bytes
   uint16     mHeaderVersion;  // Version number of object header
   uint32     mObjectSize;     // Object size in bytes
   uint32     mObjectType;     // Object type (BL_OBJ_TYPE_*)
} VBLObjectHeaderBase;
```

Listing 4.3: Objekt Base Header (VBLObjectHeaderBase) Struktur

Listing 4.3 zeigt den Aufbau der *VBLObjectHeaderBase* Struktur. In dieser werden Informationen über die anderen verwendeten Strukturen gespeichert. Sie ist wie folgt aufgebaut:

- ***mSignature* (4 Byte):** Wird immer auf BL_OBJ_SIGNATURE (0x4A424F4C) gesetzt.
- ***mHeaderSize* (2 Byte):** Definiert die Größe des Objekt Header (VBLObjectHeader) in Bytes.
- ***mHeaderVersion* (2 Byte):** Definiert die Versionsnummer des Objekt Headers (VBLObjectHeader).
- ***mObjectSize* (4 Byte):** Gibt die Größe des Objekts an. Ergibt sich aus der Größe von VBLObjectHeaderBase, VBLObjectHeader sowie der darauf folgenden Struktur (z.B. VBLEthernetFrame).
- ***mObjectType* (4 Byte):** Definiert den Typ der Nachricht (z.B. Ethernet oder CAN).

VBLObjectHeader Struktur

Listing 4.4 zeigt den Objekt Header (*VBLObjectHeader*). In dieser Struktur werden zusätzliche Informationen über die versendete Nachricht gespeichert.

```
/* ------------------------ VBLObjectHeader ----------------------------*/
typedef struct VBLObjectHeader{
   VBLObjectHeaderBase mBase;      // Base object header
   uint32   mObjectFlags;          // Object flags
   uint16   mClientIndex;          // For internal use.
   uint16   mObjectVersion;        // Object specific version
   uint64   mObjectTimeStamp;      // Object timestamp
} VBLObjectHeader;
```

Listing 4.4: Objekt Header (VBLObjectHeader) Struktur

- ***mBase*:** Definiert einen Pointer auf den Objekt Base Header (VBLObjectHeaderBase).
- ***mObjectFlags* (4 Byte):** Definiert die Genauigkeit des Zeitstempels (z.B. Mikro- oder Nanosekunden).
- ***mClientIndex* (2 Byte):** Wird standardmäßig mit „0“ definiert.
- ***mObjectVersion* (2 Byte):** Wird standardmäßig mit „0“ definiert.
- ***mObjectTimeStamp* (8 Byte):** Zeitstempel des Objekts in Abhängigkeit von *mObjectFlags*.

VBLEthernetFrame Struktur

Listing 4.5 zeigt die BLF-Ethernet-Struktur (*VBLEthernetFrame*). Diese Struktur wird mit den Informationen definiert, die das Ethernet-Frame (siehe Abschnitt 4.4.2) enthält. Das hat im Vergleich zum PCAP-Format den Vorteil, dass nicht das gesamte Ethernet-Frame serialisiert werden muss.

```
/* ------------------------------ VBLEthernetFrame  ------------------------------*/
typedef struct VBLEthernetFrame{
  VBLObjectHeader mHeader;              // Object header
  uint8     mSourceAddress[6];          // Source MAC Address
  uint8     mDestinationAddress[6];     // Destination MAC Address
  uint16    mChannel;                   // The channel of the frame
  uint16    mDir;                       // Direction flag: 0=Rx, 1=Tx, 2=TxRq
  uint16    mType;                      // EtherType
  uint16    mPayLoadLength;             // Number of valid mPayLoad bytes
  uint8*    mPayLoad;                   // Payload (Max 1582)
} VBLEthernetFrame;
```

Listing 4.5: VBLEthernetFrame Struktur

VBLCANMessage2 Struktur

Listing 4.6 zeigt die BLF-CAN-Struktur (*VBLCANMessage*). Diese Struktur definiert Informationen über das CAN-Frame (siehe Abschnitt 4.4.5).

```
/* ---------------------------- VBLCANMessage2  ----------------------------*/
typedef struct VBLCANMessage2{
  VBLObjectHeader mHeader;      // Object header
  uint16    mChannel;           // Channel the frame was sent or received.
  uint8     mFlags;             // CAN Message Flags
  uint8     mDLC;               // Data length 0 - 8 Byte
  uint32    mID;                // Identifier
  uint8     mData[8];           // CAN data
} VBLCANMessage2;
```

Listing 4.6: VBLCANMessage Struktur

4.4 Serializer und Netzwerkprotokolle

4.4.1 Serializer

Damit die Ethernet- und CAN-Frames in einer Logging-Datei gespeichert werden können, müssen diese in einen Bytestrom umgewandelt werden. Dafür werden Serializer eingesetzt. Sie müssen den Aufbau der Frames genau einhalten, damit alle Informationen wieder richtig aus den PCAP- und BLF-Dateien gelesen werden können. In dieser Arbeit werden Serializer für *CAN-* , *Ethernet-* (insklusive TTEthernet und AVB) und Gateway-Nachrichten (*Gateway Aggregation* Message und *Unit Message*) eingesetzt. Deswegen werden diese Protokolle im Folgenden vorgestellt.

4.4.2 Ethernet-Frame

Ethernet wurden 1976 von den Amerikanern Bob Metclaf und David Blogg in Hawaii entwickelt und basiert auf dem ALOHA-Protokoll, um die Netze aus den verschiedenen Inseln miteinander zu verbinden [vgl. Tanenbaum und Wetherall, 2012, S. 99]. Beim Ethernet werden das Zugriffsverfahren, das Adressierungsschema und die Paketstruktur zur Datenversendung im Netzwerk innerhalb der ersten beiden Schichten des OSI-Modells festgelegt. Abbildung 4.5 zeigt den typischen Aufbau von Ethernet-Frames.

Ein Ethernet-Frame kann insgesamt zwischen 64 und 1518 Byte lang sein. Wie in Abbildung 4.5 zu sehen ist, wird der Ethernet-Header durch eine sechs Byte Ziel-Adresse, eine sechs Byte Quell-Adresse und ein zwei Byte großes Type-Feld gebildet.

Anschließend beschreibt die bis zu 1500 Byte große Payload die eigentlichen Daten, die transportiert werden sollen. Die Payload darf nie kleiner als 46 Byte sein und muss gegebenenfalls aufgefüllt werden.

Das Frame wird durch das vier Byte lange CRC-Feld abgeschlossen. Dieses wird für die Fehlererkennung genutzt, die während der Übertragung entstehen können.

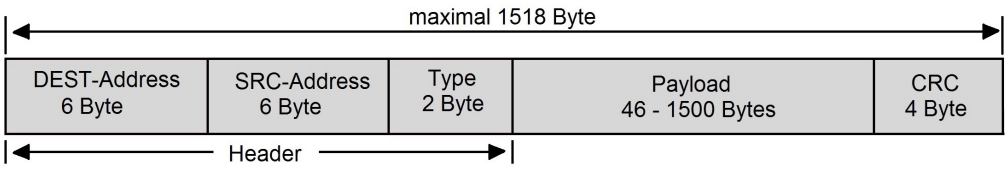

Abbildung 4.5: Aufbau des Ethernet-Frames

4.4.3 TTEthernet-Frame

Time-Triggered Ethernet (TTEthernet) wurde von der Firma TTTech [vgl. TTTech Computertechnik AG] entwickelt, um Echtzeit-Erweiterung für die Ethernet-basierte Kommunikation zu ermöglichen. Mit TTEthernet können herkömmliche PCs, Multimediasysteme, Echtzeitsysteme (mit höhsten Anforderungen) sowie sicherheitskritische Systeme ein und dasselbe Netzwerk nutzen [TTTech Computertechnik AG, 2009, vgl.]. Dies wurde im SAE AS6802 Standard standardisiert [vgl. Society of Automotive Engineers - AS-2D Time Triggered Systems and Architecture Committee, 2011].

TTEthernet definiert drei Nachrichtenklassen:

- *Time-Triggered-Traffic (TT)*: Wird für den zeitkritischen Datenverkehr verwendet. Diese Nachrichtenklasse hat die höchste Priorität im TTEthernet und kann nicht von anderen Nachrichtenklassen verdrängt werden.
- *Rate-Constrained-Traffic (RC)*: Die Nachrichtenklasse stellt sicher, dass Anwendungen im Voraus eine bestimmte Bandbreite zur Verfügung steht, um Verzögerungszeiten und zeitliche Abweichungen in Grenzen halten.
- *Best-Effort-Traffic (BE)*: Diese Nachrichtenklasse entspricht dem Standard-Ethernet-Verkehr. Sie hat die niedrigste Priorität und nutzt deswegen nur die restliche Bandbreite im System.

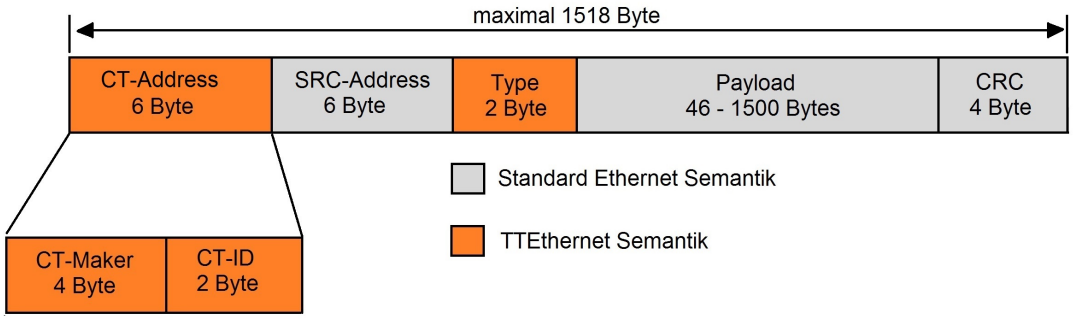

Abbildung 4.6: Aufbau des TTEthernet-Frames [vgl. Bartols, 2010]

Wie in Abbildung 4.6 zu sehen ist, basiert das TTEthernet-Frame auf dem Standard-Ethernet-Frame. Die Priorität der Nachrichtenklassen kann mit dem Standard-Ethernet-Frame nicht abgedeckt werden, da es dafür kein gesondertes Feld gibt. Deswegen wird im TTEthernet-Frame das Zieladressfeld (CT-Address) anders gehandhabt.

Wie in Abbildung 4.6 zu sehen ist, wird das CT-Address (Critical Traffic Address) Feld durch ein vier Byte großes TT-Marker-Feld und ein zwei Byte großes TT-ID-Feld definiert. Zeitkritische Nachrichten werden mit dem CT-Address-Feld gekennzeichnet. Mit dem CT-ID-Feld wird identifiziert, ob es sich um ein TT- oder ein RC-Frame handelt. Dementsprechend können bis zu 4096 verschiedene zeitkritische Nachrichten in einem TT-System definiert werden.

Außerdem unterscheiden sich Standard-Ethernet-Frames von TTethernet-Frames im Typ-Feld des Frames. Nach IEEE wurde für den zeitkritischen Datenverkehr das Bitmuster 0x88D7 reserviert [IEEE, 2010].

4.4.4 Audio/ Video Transport Protocol (AVTP) Ethernet-Frame (IEEE 802.1 Q)

Audio Video Bridging (AVB) wurde von der AVB Task Group der IEEE 802.1 Working Group entwickelt. Diese setzt sich aus einer Reihe von Standards zusammen, die in dem IEEE 802.1BA Standard zusammengefasst wurden [vgl. Institute of Electrical and Electronics Engineers, 2011b].

Das Verpacken von digitalen Mediendaten in Ethernet-Frames wird als Audio/Video-Transport Protocol (AVTP) im IEEE Standard 1722 spezifiziert (IEEE 1722 2011). Das AVTP-Frame basiert auf dem IEEE 802.1 Q Ethernet-Frame. Der 802.1Q-Standard erlaubt, ein physisches Netzwerk in mehrere Virtual Local Area Networks (VLANs), welche anhand von IDs identifiziert werden, zu unterteilen. Durch diesen Mechanismus kann der Netzwerkverkehr voneinander separiert werden, ohne physische Verbindungen zu verändern.

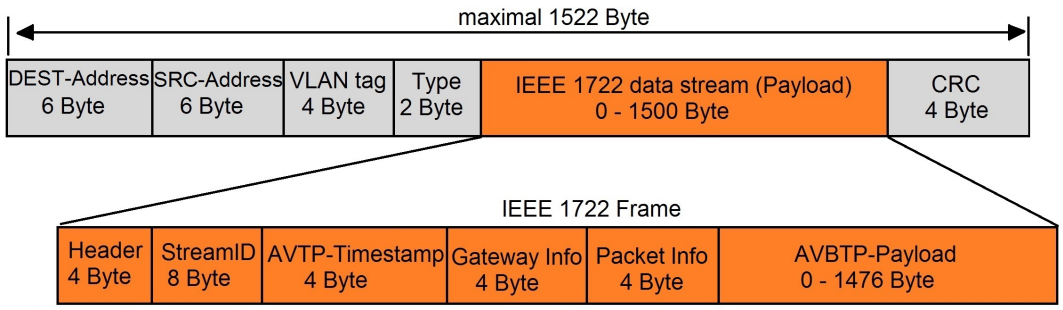

Abbildung 4.7: Aufbau des AVTP Ethernet-Frames (IEEE 802.1Q) [vgl. Lim u. a., 2012]

Abbildung 4.7 zeigt ein AVTP-Frame. Dieser hat im Gegensatz zum Standard-Ethernet-Frame (siehe Abbildung 4.5) ein vier Byte großes *VLAN Tag*-Feld. Der AVB-Standard legt fest, dass zeitkritische Streams anhand einer speziellen VLAN ID gekennzeichnet werden [vgl. Institute of Electrical and Electronics Engineers, 2011b, S.18].

Die AVB Streaming-Daten werden über ein IEEE 1722 Frame [Institute of Electrical and Electronics Engineers, 2011a] spezifiziert (siehe Abbildung 4.7). Die Daten sind in dem *AVBTP-Payload* Feld enthalten und können maximal 1476 Byte groß sein [vgl. Lim u. a., 2012].

Die *StreamID* wird verwendet, um einen Stream zu identifizieren bzw. um zwischen verschiedene Streams unterscheiden zu können [vgl. Lim u. a., 2012].

AVTP-Timestamp-Feld gibt den Zeitpunkt des Versands des Pakets und die Intervallzeit der Nachrichtenklasse an [vgl. Lim u. a., 2012]. Für AVB gibt es drei verschiedene Nachrichten Klassen:

- AVB-Nachrichten Klasse A: Intervall 125 µs.
- AVB-Nachrichten Kassen B: Intervall 250 µs.
- Best Effort-Nachrichten: Kein Intervall relevant.

Das *Gateway-Info*-Feld ist für AVTP Gateways reserviert, die nicht in IEEE 1722 definiert sind. Das *Paket-Info*-Feld enthält Informationen über die Daten (Payload), wie z.B. die Datenlänge [vgl. Lim u. a., 2012].

4.4.5 CAN-Frame

Das Controller Area Network (CAN) wurde 1983 von der Robert Bosch GmbH [Robert Bosch GmbH, 1991] als Kommunikationsprotokoll entwickelt und 1987 nach der Zusammenarbeit mit Intel erstmals vorgestellt. CAN-Netzwerke werden in Personen- bzw. Nutzkraftwagen, Schiffen, Bahnen, Flugzeugen und zunehmend in der Automatisierungstechnik und im Maschinenbau eingesetzt. Nach dem OSI-Referenzmodel (siehe Abschnitt 2.3) definiert das CAN-Protokoll die ersten zwei Schichten. Die Sicherungsschicht (Data Link Layer) definiert die CAN-Spezifikationen. Die Bitübertragungsschicht (Physical Layer) beschreibt den Aufbau der CAN-Nachricht.

Abbildung 4.8 zeigt ein CAN-Frame. Das erste Bit des CAN-Frames ist das *Startbit* und dient zur Synchronisation der Busteilnehmer.

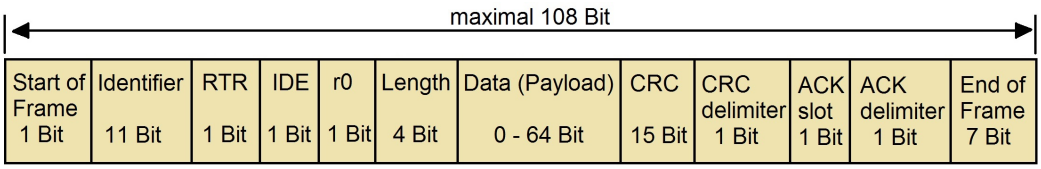

Abbildung 4.8: CAN-Frame [vgl. Lawrenz und Obermöller, 2011, S. 19]

Das Arbitrations-Feld (*Identifier*) hat bei der Version CAN 2.0A eine Länge von elf Bit. Dieses Feld definiert die Priorität und die logische Adresse der Nachricht und je kleiner der Wert, desto höher ist die Priorität [vgl. Lawrenz und Obermöller, 2011, S. 19]. Dementsprechend sind 2048 verschiedene logische Adressen möglich. Mit der Version CAN 2.0B wird der Adressraum auf 29 Bit erweitert.

Das Remote Transmission Request (RTR) Feld gibt an, ob es von einem anderen Knoten Daten anfordert oder diese selber enthält.

Die nächsten sechs Bits (siehe Abbildung 4.8) werden als Kontroll-Feld bezeichnet und enthalten ein Bit Identifier Extension (IDE) Feld, ein Bit Reserved (r0) Feld und einen vier Bit Feld mit den Längeninformationen der Payload (Daten). Sollte das IDE-Bit gesetzt sein, dann folgt nach dem IDE-Bit noch ein weiterer Adressbereich mit 18 Bits für die CAN 2.0B Erweiterung. Die Daten (Payload), die in einem CAN-Frame versendet werden können, sind maximal acht Byte groß.

Das 16 Bit große *CRC*-Feld ist ein Verfahren, um Fehler bei der Übertragung oder Speicherung erkennen zu können. Dabei können bis zu sechs Einzelbitfehler erkannt werden.

Das zwei Bit große *Ack*-Feld wird dazu genutzt, um zu überprüfen, ob eine Nachricht empfangen wurde. Das sieben Bit große End-of-Frame Feld kennzeichnet das Ende des CAN-Frames.

4.4.6 Gateway Aggregation und Unit Message Frame

Wie schon in Abschnitt 3.2 erwähnt wurde, werden in dieser Arbeit über Gateways Ethernet- und CAN-Komponenten im Netzwerk verbunden. Mit dem Signals and Gateways Framework [CoRE Research Group, d] wurden die Gateway Nachrichten *Gateway Aggregation Message* und *Unit Message* in OMNeT++ eingeführt.

Mithilfe dieser kann das Gateway eine CAN-Nachricht in einer *Unit Message* und mehrere *Unit Messages* in einer *Gateway Aggregation Message* verpacken, um diese letztendlich über Ethernet Nachrichten zu versenden.

Abbildung 4.9 zeigt ein Ethernet-Frame, das in der Payload ein *Gateway Aggregation* Frame enthält.

Das ein Byte große *Units*-Feld beschreibt, wie viele *Unit Messages* das *Gateway Aggregation* Frame enthält.

Das ein Byte große *Sequenz Number*-Feld wird verwendet, wenn die Nachricht fragmentiert wurde. Die nächsten zwei Felder geben dann an, in wie viele Fragmente die Nachricht unterteilt wurde und welche Fragment-Nummer diese Nachricht enthält.

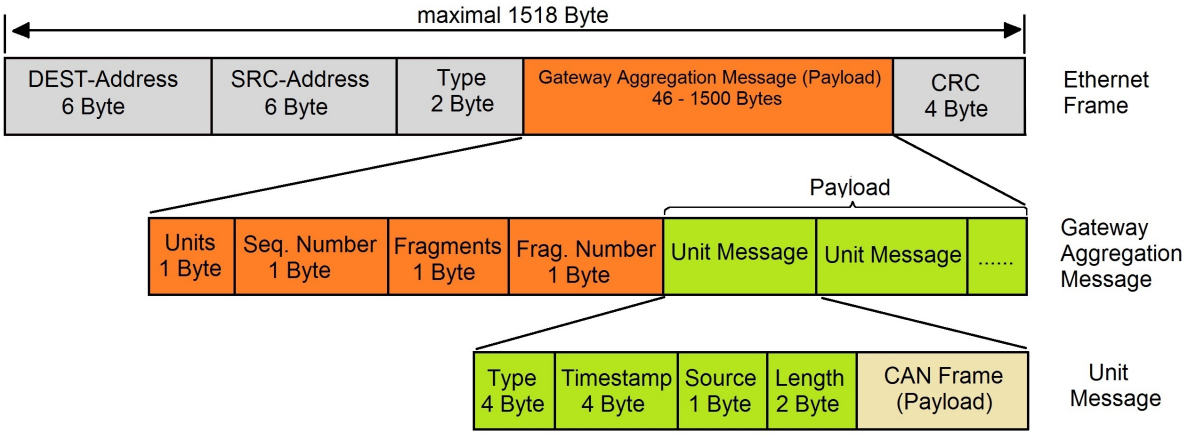

Abbildung 4.9: Gateway Aggregation- und Unit Message-Frame

In der Payload des *Gateway Aggregation* Frames befinden sich eine oder mehrere *Unit Messages* (siehe Abbildung 4.9).

In der Payload jeder *Unit Message* befindet sich jeweils ein CAN-Frame. Der Header eines *Unit Message*-Frames besteht aus einem vier Byte großen *Type*-Feld, einem vier Byte große *Timestamp*-Feld, einem ein Byte großen *Source*-Feld und einem zwei Byte großen *Length*-Feld (siehe Abbildung 4.9).

Das *Type*-Feld gibt den Type der Nachricht an, die in der Payload einer *Unit Message* vorhanden ist. Das nächste Feld (*Timestamp*) gibt den Zeitpunkt an, wann die CAN-Nachricht verpackt wurde. Das *Source*-Feld enthält die ID von dem Bus, von dem die Nachricht her kam. So können CAN-Nachrichten derselben ID, die von verschiedenen Bussen versendet wurden, unterschieden werden. Die letzten zwei Byte geben die Länge der Payload an.

5 Implementierung

Dieses Kapitel beschreibt die Implementierung des BLFRecorders und des CoREPcapRecorders.

5.1 BLFRecorder

Der BLFRecorder wird zur Erstellung einer BLF-Datei genutzt, um Ethernet- und CAN-Nachrichten aufzunehmen, die während der Simulation ausgetauscht wurden. Die Integration von CANoe in die OMNeT++ System Level Simulation kann mithilfe des BLFRecorders realisiert werden. In Abschnitt 4.2 wurde der konzeptionelle Gesamtprozessablauf dieser Arbeit beschrieben, um eine Schnittstelle zwischen CANoe und OMNeT++ aufzubauen. In diesem Abschnitt wird die Implementierung des BLFRecorders näher erläutert.

5.1.1 OMNeT++ Signale

Damit der BLFRecorder auf die bestimmten Nachrichten reagieren kann, wird dieser von Signalen getriggert. Diese Möglichkeit bietet OMNeT++ ab Version 4.1 an. Dazu müssen Signale in einer NED-Klasse definiert werden.

```
1 @signal[txPk](type=EtherFrame);    //Signal for transmitted Ethernet frames
2 @signal[rxPkOk](type=EtherFrame);  //Signal for received Ethernet frames
3 @signal[txDF](type=CanDataFrame);  //Signal for transmitted CAN data frames
4 @signal[rxDF](type=CanDataFrame);  //Signal for received CAN data frames
```

Listing 5.1: Deklaration von Signalen für Ethernet- und CAN-Frames

Listing 5.1 zeigt, wie Signale in einer NED-Klasse deklariert werden. Das *txPk* Signal wird in dieser Arbeit genutzt, um auf versendete Ethernet-Nachrichten reagieren zu

können. Das *rxPkOk* wird genutzt, um auf empfangene Ethernet-Frames reagieren zu können. Für jeden Nachrichtentyp gibt es eigene Signale (siehe Listing 5.1). Mithilfe der OMNeT++ Methode *registerSignal* werden die gewünschten Signale registriert.

Die OMNeT++ cListener-Klasse verfügt über eine *receiveSignal* Methode, die aufgerufen wird, wenn ein registriertes Signal ausgelöst wurde. Die *receiveSignal* Methode wurde beim PcapRecorder (INET-Framework) überschrieben (override). Der BLFRecorder erbt[1] diese Methode vom PcapRecorder. In dieser Arbeit werden Signale ausgelöst, wenn eine Nachricht versendet oder empfangen wurde. Wie in Abbildung 5.1 zu sehen ist, besteht

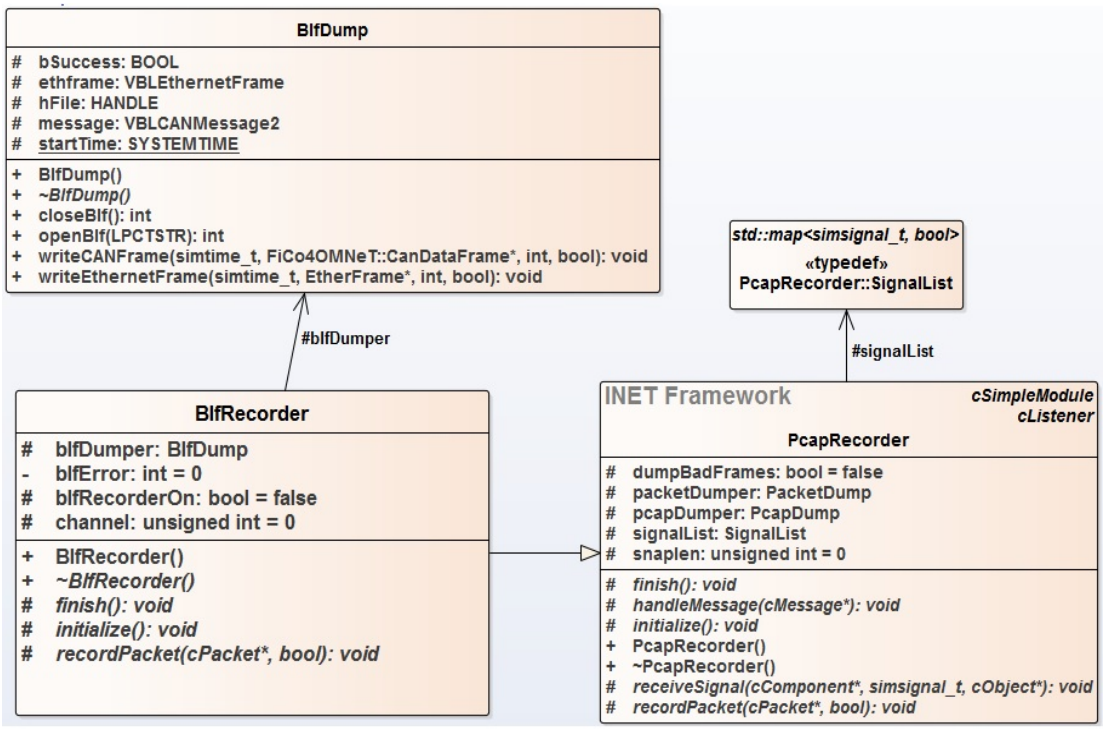

Abbildung 5.1: Klassenmodell des BLFRecorders

der BLFRecorder (NED Modul) aus der BlfDump.cc- und der BLFRecorder.cc-Datei. Auf diese wird im Folgenden näher eingegangen.

[1] Vererbung ist ein Mechanismus, der es ermöglicht, Attribute und Methoden von Klassen an andere Klassen weiterzugeben.

5.1.2 BLFRecorder.ned

Die *BLFRecorder.ned*-Datei repräsentiert ein Modul, das in einem anderen Modul als Submodul angebunden werden muss, um dessen Nachrichtenaustausch aufzunehmen und in einer BLF-Datei abzuspeichern.

In der *BLFRecorder.ned*-Datei werden Parameter deklariert und mit Standardwerten definiert. Mithilfe dieser Parameter kann der BLFRecorder konfiguriert werden:

- *blfFile*: Dieser Parameter repräsentiert den Namen der erzeugten BLF-Datei und muss in einer ini-Datei definiert werden (siehe Listing 5.2).
- *moduleNamePatterns*: Ein Modul (z.B. ein Switch) besteht aus vielen Submodulen und in diesem Parameter werden Submodule eingetragen, in denen Signale definiert wurden, wie z.B. das *EtherMACFullDuplex* Submodul (INET-Framework) für Ethernet-Komponenten. Sollte sich eines dieser Submodule in einem weiteren Submodul befinden, dann muss dieses mit eingetragen werden.
- *sendingSignalNames*: Eine Liste mit den Signalen, die auf gesendete Ethernet- und CAN-Frames verweisen.
- *receivingSignalNames*: Eine Liste mit den Signalen, die auf empfangene Ethernet- und CAN-Frames verweisen.
- *channelID*: Diesen Parameter repräsentiert einen Integer-Wert, der pro Modul einzigartig sein muss. Der Nutzer definiert diesen Parameter eigenständig in einer .ini-Datei (siehe Listing 5.2). Damit wird jeder BLF-Datei in CANoe ein Kanal zugeordnet.
- *recording*: Dieser Parameter bestimmt, ob der BLFRecorder eingeschaltet oder ausgeschaltet ist. Standardmäßig ist dieser ausgeschaltet und muss über eine .ini-Datei eingeschaltet werden (siehe Listing 5.2).

```
1 **.gateway1.BLFRecorder.recording = true
2 **.gateway1.BLFRecorder.channelID = 1
3 **.gateway1.BLFRecorder.blfFile = "results/gateway1.blf"
```

Listing 5.2: Konfiguration des BLFRecorders in der .ini-Datei

5.1.3 BLFRecorder.cc

Der BLFRecorder.cc-Datei (siehe Abbildung 5.1) ist für die Initialisierung der Module und die Filterung der Nachrichtentypen zuständig.

Methode *initialize*

Abbildung 5.2 stellt die Methode *initialize* in einem Ablaufdiagramm dar. Die Methode *initialize* prüft zuerst, ob der BLFRecorder für dieses Modul überhaupt eingeschaltet wurde.

Die BLFRecorder.cc-Klasse verfügt über eine Signalliste (SignalList), die zunächst mit den eingetragenen Signalen aus den Parametern (sendingSignalNames und receivingSignalNames) der NED-Datei aufgefüllt werden muss (siehe Abbildung 5.1).

Als nächstes wird ein Array erzeugt, das mit den eingetragenen Modulen aus dem Parameter *moduleNamePatterns* (siehe Abschnitt 5.1.2) aufgefüllt werden muss. Dann wird dieses Array durchlaufen und dabei muss geprüft werden, ob in diesem Modul die Signale aus der Signalliste definiert wurden. Wenn das der Fall ist, wird diesem Modul ein Listener (callback obbject) zugewiesen.

Der zugewiesene Listener ruft die Methode *receiveSignal* auf, wenn ein Signal ausgelöst wurde. Diese Methode bekommt die *BLFRecorder.cc*-Klasse vom *PcapRecorder* (INET-Framework) vererbt (siehe Abbildung 5.1).

Wurde das Array komplett durchlaufen, wird die Methode *openBLF* aufgerufen. Anschließend wird die Methode *initialize* verlassen und die BLF-Datei ist bereit Nachrichten abzuspeichern.

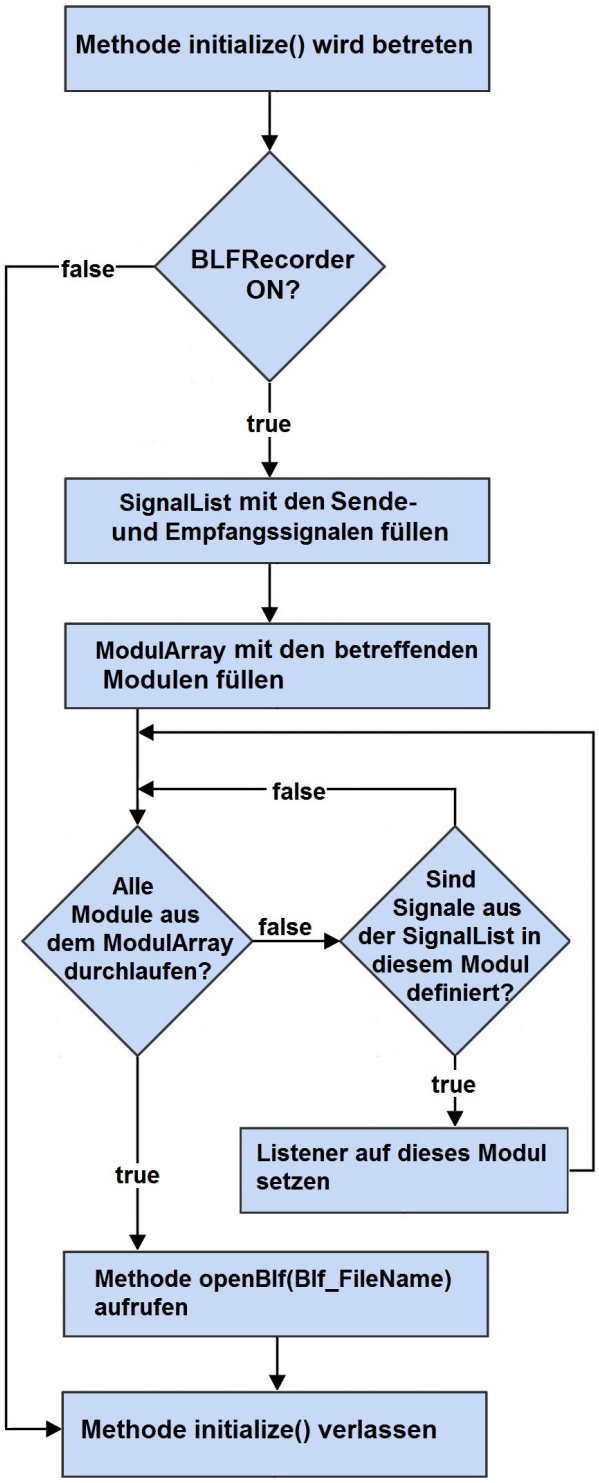

Abbildung 5.2: Ablaufdiagramm der Methode *initialize* (BLFRecorder.cc)

Methode *recordPacket*

Wenn ein Signal ausgelöst wurde, wie z.B. beim Versenden eines Ethernet-Frames, wird über die Methode *receiveSignal* die Methode *recordPacket(cPacket *msg, bool l2r)* aufgerufen. Diese stellt den Nachrichtentyp fest und je nachdem, ob es ein CAN- oder ein Ethernet-Frame ist, wird diese individuell weiterverarbeitet. Stellt die Methode fest, dass es sich um ein CAN-Frame handelt, wird die Methode *writeCANFrame* aufgerufen. Handelt es sich dabei aber um ein Ethernet-Frame, wird die Methode *writeEthernetFrame* aufgerufen. Diese Methoden sind in der *BLFDump*-Klasse (siehe Abbildung 5.1) implementiert. Der Parameter l2r gibt an, ob das ankommende Paket (*msg) vom Modul empfangen oder versendet wurde.

5.1.4 BLFDump.cc

Die BLFDump.cc-Datei ist für das Schreiben der Daten in die BLF-Datei zuständig. Die von Vector bereit gestellte *binlog.dll* Bibliothek sowie die binlog_objects und binlog Header, die in Abschnitt 4.3.2 vorgestellt wurden, werden für diese Klasse benötigt. Die *binlog.dll* Bibliotheksmethoden (siehe Listing 5.3) und drei Methoden der *BLFDump.cc*-Klasse (siehe Abbildung 5.1) werden im Folgenden näher betrachtet.

Binlog Bibliothek

```
1 HANDLE BLCreateFile(LPCTSTR lpFileName, DWORD dwDesiredAccess);
2 BOOL BLSetApplication(HANDLE hFile, BYTE appID, BYTE appMajor, BYTE appMinor, BYTE appBuild);
3 BOOL BLSetMeasurementStartTime(HANDLE hFile, const LPSYSTEMTIME lpStartTime);
4 BOOL BLSetWriteOptions(HANDLE hFile, DWORD dwCompression, DWORD dwReserved)
5 BOOL BLWriteObject(HANDLE hFile, VBLObjectHeaderBase* pBase);
6 BOOL BLCloseHandle(HANDLE hFile);
```

Listing 5.3: Methoden der binlog.dll Bibliothek [Vector Informatik GmbH, a]

Listing 5.3 zeigt die Methoden der binlog.dll Bibliothek, die für den BLFRecorder benötigt werden.

Die Methode *BLCreateFile* öffnet eine BLF-Datei mit den benötigten Zugriffsrechten (Parameter *dwDesiredAccess*) und spezifizierten Dateinamen (Parameter *lpFileName*). Wenn die Methode fehlerfrei ausgeführt wurde, ist der Rückgabewert ein *HANDLE* Objekt (*hfile*), das die Datei spezifiziert [vgl. Binlog Manual Vector Informatik GmbH, a].

Die Methode *BLSetApplication* wird genutzt, um die Applikation zu spezifizieren, welche die BLF-Datei erzeugt.

Mit der Methode *BLSetMeasurementStartTime* wird die Messstartzeit eingestellt.

Mit der Methode *BLSetWriteOptions* wird die Komprimierung eingestellt [vgl. Manual Vector Informatik GmbH, a]. Die Komprimierung beschreibt die Reduktion von Daten, um Speicherplatz oder Übertragungszeit zu sparen.

Die Methode *BLWriteObject* wird genutzt, um das *Binary Logging Object* in die Datei zu schreiben. Ein *Binary Logging Object* enthält Strukturen, die in Abschnitt 4.3.2 vorgestellt wurden (wie z.B. *VBLObjectHeaderBase*). Diese Methode schreibt letztendlich die VBLEthernetFrame- und VBLCANMessage2-Struktur in die BLF-Datei.

Die Methode *BLCloseHandle* wird genutzt, um die BLF-Datei zu schließen, die mit der Methode *BLCreateFile* geöffnet wurde. Diese Methode wird beim Beenden der Simulation aufgerufen.

Methode *openBlf*

Die Methode *openBlf* der Klasse BLFDump.cc (siehe Abbildung 5.1) wird dazu genutzt, um die BLF Datei zu öffnen und Konfigurationen vorzunehmen, die notwendig sind, bevor die eigentlichen Daten gespeichert werden können. Dafür werden die Methoden der *Binlog* Bibliothek genutzt. Als Parameter wird der Name der BLF-Datei übergeben. Wenn die Datei geöffnet wurde und alle Einstellungen korrekt vorgenommen worden sind, ist der Rückgabewert 0 und die Datei ist bereit Daten abzuspeichern.

Methode *writeEthernetFrame*

Ist die BLF-Datei geöffnet und ein Signal ausgelöst worden, mit dem signalisiert werden soll, das ein Ethernet-Frame empfangen bzw. versendet wurde, wird die Methode *writeEthernetFrame* (siehe BLFDump Abbildung 5.1) aufgerufen. In dieser Methode werden die Strukturen *VBLObjectHeaderBase*, *VBLObjectHeader* und *VBLEthernetFrame* definiert (Wertzuweisung), die in Abschnitt 4.3.2 erläutert wurden.

Die Methode *writeEthernetFrame* bekommt als Parameter übergeben:

- ***stime***: Der aktuelle Simulationszeitpunkt (Systemzeit).
- ***etherPacket***: Das aufzunehmende Ethernet-Frame.
- ***channelID***: Der Channel (ID), an den das Paket versendet wurde.
- ***l2r***: Gibt an, ob das aufzunehmende Paket (etherPacket) vom Modul empfangen oder versendet wurde.

Zu Beginn dieser Methode werden diese Strukturen mithilfe der Information des Ethernet-Frames (etherPacket) definiert. Anschließend muss die Payload (Daten) des Ethernet-Frames geholt werden. Dazu wird ein Buffer mit der Länge der Payload (Daten) erzeugt. Die Payload wird mithilfe der Methode *lookupAndSerialize* (INET Framework) in einen Buffer geschrieben. Diese Methode prüft zuerst, ob ein Serializer vorhanden ist. Am Ende werden die im Buffer gesicherten Informationen in die *VBLEthernetFrame* Struktur übertragen.

Methode *writeCANFrame*

Ist die BLF-Datei geöffnet und ein Signal ausgelöst worden, mit dem signalisiert wird, dass ein CAN-Frame empfangen bzw. versendet wurde, wird die Methode *writeCANFrame* (siehe BLFDump Abbildung 5.1) aufgerufen. Diese Methode schreibt CAN-Frames in die BLF-Datei. Der Aufbau und die Parameter dieser Methode gleichen der Methode *writeEthernetFrame*, nur dass diese Methode mit einem CAN-Frame arbeitet und die *VBLCANMessage2* Struktur nutzt.

5.2 CoREPcapRecorder

Der *CoREPcapRecorder* wird zur Erstellung einer PCAP-Datei genutzt, um den Nachrichtenaustausch von Ethernet-Netzwerken aufzunehmen. Es ist zwar möglich, auch CAN Netzwerke aufzunehmen, aber CANoe kann keine CAN-Daten aus der PCAP-Datei lesen.

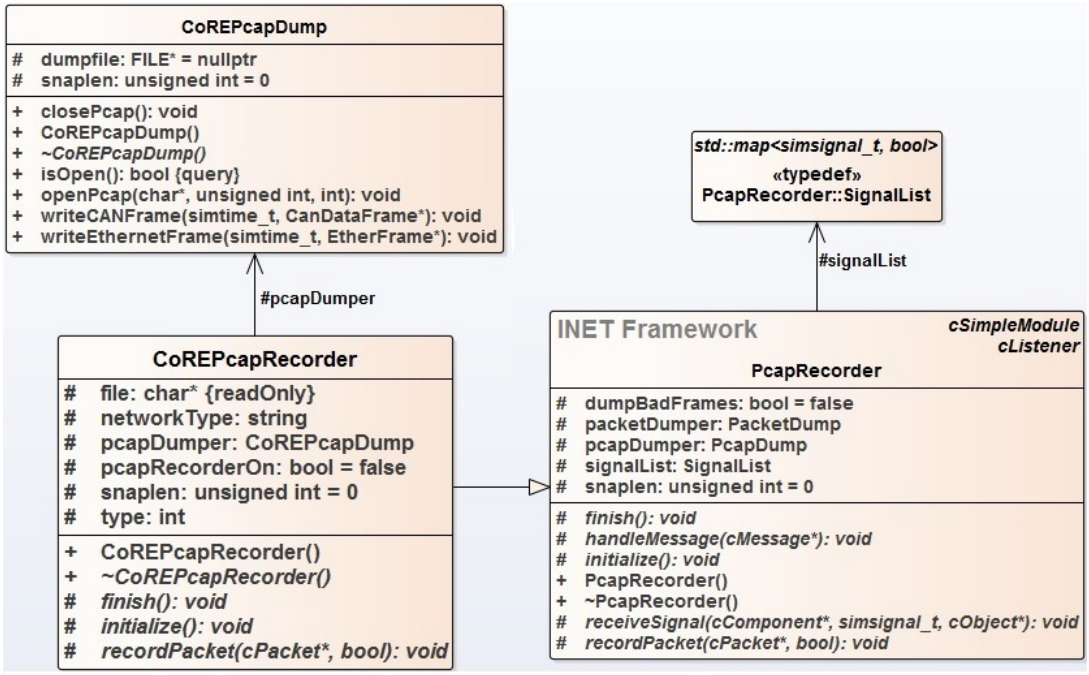

Abbildung 5.3: Klassenmodell des *CoREPcapRecorders*

Der *CoREPcapRecorder* besteht aus drei Dateien: *CoREPcapRecorder.ned*, *CoREPcapRecorder.cc* und *CoREPcapDump.cc*. Auch der *CoREPcapRecorder* erbt vom PcapRecorder (INET-Framework). Wie in Abbildung 5.3 zu sehen ist, gibt es im Vergleich zum *BLFRecorder* kaum Änderungen. Aus diesem Grund wird dieser nicht näher erläutert.

6 Evaluierung

In diesem Kapitel werden die Kriterien festgelegt, nach denen evaluiert wird. Darauffolgend werden die zuvor ermittelten Kriterien untersucht.

6.1 Evaluierungskriterien

Damit der BLF/CoREPcapRecorder erfolgreich genutzt werden kann, müssen die folgenden Fragen mit Ja beantwortet werden.

- Werden BLF- bzw. PCAP-Dateien erzeugt?
- Können die BLF- bzw. PCAP-Dateien in CANoe gelesen werden?
- Werden alle Nachrichtenpakete (Frames) vom *BLFRecorder* bzw. *CoREPcapRecorder* erfasst?
- Werden die Nachrichtenpakete (Ethernet- und CAN-Frames) richtig aufgezeichnet?
- Es findet keine Aufzeichnung von doppelten Paketen (Duplikate) statt?
- Werden alle Nachrichtenpakete beim Abspeichern mit einem Zeitstempeln versehen?

Zum Testen des BLFRecorders wurde das *Ethernet- und CAN-Netzwerk* aus Abbildung 4.1 verwendet. Das *Ethernet- und CAN-Netzwerk* besteht aus zwei Gateways, einem Switch, sechs ECUs und zwei CAN-Bussen. Der BLFRecorder wurde so konfiguriert, dass für jede Komponente im Netzwerk jeweils eine BLF-Datei erzeugt wird. Der CoREPcapRecorder wurde mit dem Ethernet-Netzwerk *Small_Network* (siehe Abbildung 6.1) getestet. Außerdem wurden CANoe-Simulationsnetzwerke verwendet, um den Inhalt der BLF- und PCAP-Dateien zu Testen.

Die verwendeten OMNeT++ und CANoe Netzwerke befinden sich im separaten Projektordner.

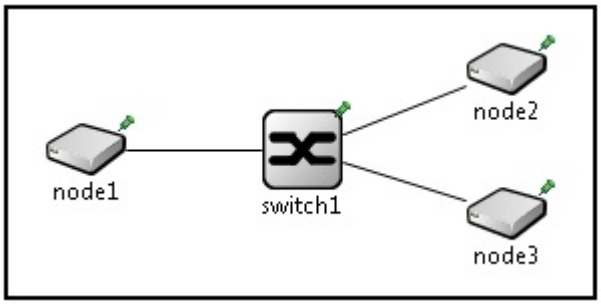

Abbildung 6.1: Small Network [CoRE4INET Framework vgl. CoRE Research Group, b]

Tabelle 6.1 zeigt alle getesteten Anforderungen bezüglich der Evaluierungskriterien. BLFRecorder bzw. CoREPcapRecorder konnten diese Kriterien erfolgreich erfüllen. Die Evaluierung für den BLFRecorder und CoREPcapRecorder ist somit erfolgreich.

Evaluierungskriterien	Ethernet- und CAN-Netzwerk (BLFRecorder)	Small_Network (CoREPcapRecorder)
Werden BLF- bzw. PCAP-Dateien erzeugt?	Ja	Ja
Können die BLF- bzw. PCAP-Dateien in CANoe eingelesen werden?	Ja	Ja
Werden alle Nachrichtenpakete (Frames) vom *BLFRecorder* bzw. *CoREPcapRecorder* erfasst?	Ja	Ja
Werden die Nachrichtenpakete (Ethernet und CAN Frames) richtig aufgezeichnet?	Ja	Ja
Wird die Aufzeichnung von doppelten Paketen (Duplikate) verhindert?	Ja	Ja
Werden alle Nachrichtenpakete beim Abspeichern mit einem Zeitstempeln versehen?	Ja	Ja

Tabelle 6.1: Testkriterien für den BLFRecorder auf dem *Ethernet- und CAN-Netzwerk*

6.2 Simulationsgeschwindigkeit und Dateigröße

In diesem Abschnitt werden die Simulationszeit der Recorder und die daraus resultierenden Dateigrößen gemessen. Der Test verläuft in der OMNeT++ Simulation mit dem *Ethernet- und CAN-Netzwerk* aus Abbildung 4.1. Für die Bewertung der entwickelten Aufzeichnungsmethoden wird die bereits in OMNeT++ integrierte Aufzeichnungsmethode Eventlog tabellarisch mit aufgelistet. Als System wird ein PC mit folgender Ausstattung genutzt:

- Betriebssystem: Windows 7
- OMNeT++ Version 5.01b
- Prozessor: Intel Core i7 M640 2,80GHz
- Arbeitspeicher: 4 GB RAM
- Festplatte: SSD 250GB

Tabelle 6.2 zeigt die Simulationsgeschwindigkeit (gemessene Zeit) der Aufzeichnungsmethoden. Ohne den Einsatz einer Aufzeichnungsmethode werden 240 Sekunden Simulationszeit in 22 Sekunden erreicht. Es ist gut zu erkennen, dass der BLFRecorder bezüglich der Simulationsgeschwindigkeit am effektivsten ist. Mit dem BLFRecorder verlängert sich die gemessene Zeit nur um fünf Sekunden. Auch der CoREPcapRecorder hat bezüglich der Simulationsgeschwindigkeit gut abgeschlossen. Die beiden in dieser Arbeit entwickelten Aufzeichnungsmethoden haben die Anforderungen bezüglich der Simulationsgeschwindigkeit erfüllt.

Aufzeichnungsmethode	Dauer bei 240 sec Simulationszeit
Ohne Aufzeichnungsmethode	22 sec
BLFRecorder	27 sec
CoREPcapRecorder	35 sec
Eventlog (OMNeT++)	01:58 min

Tabelle 6.2: Evaluation der Simulationsgeschwindigkeiten

Tabelle 6.3 zeigt die Dateigrößen der Aufzeichnungsmethoden nach 240 Sekunden Simulationslaufzeit. Auch hier schneidet der BLFRecorder bezüglich der Dateigröße am besten ab. Die Gesamtgröße aller Dateien beträgt beim BLFRecorder nur 1,33 MB und ist 14 mal kleiner als beim CoREPcapRecorder. Der Grund für die geringe Dateigröße ist die Komprimierung der gespeicherten Daten beim BLFRecorder (siehe Abschnitt 5.1.4). Auch bezüglich der Dateigröße haben beide entwickelten Aufzeichnungsmethoden die Anforderung erfüllt.

Aufzeichnungsmethode	Dateigröße bei 240 sec Simulationszeit
BLFRecorder	1,33 MB
CoREPcapRecorder	19.0 MB
Eventlog (OMNeT++)	811 MB

Tabelle 6.3: Evaluation der Dateigrößen nach 240 Sekunden Simulationslaufzeit

7 Zusammenfassung

Das Ziel dieser Arbeit war es, ein Aufzeichnungsprogramm zu entwickeln, um die Integration von CANoe in die OMNeT++ System Level Simulation zu ermöglichen. Der BLFRecorder bzw. CoREPcapRecorder wird als OMNeT++ Submodul in einem Echtzeit-Kommunikationsnetzwerk eingebunden und speichert mithilfe der Serializer alle Ethernet- und CAN-Nachrichten in eine BLF- bzw. PCAP-Datei. Beide Formate sind der CANoe-Simulationsumgebung bekannt und können eingelesen werden.

Um eine einheitliche Wissensbasis zu schaffen, wurden zu Beginn der Arbeit einige Grundlagen vermittelt. Zuerst wurden die Grundmodelle der Simulation und das OSI-Schichtenmodell erläutert. Anschließend wurden die OMNeT++ und CANoe Simulationsumgebung vorgestellt.

Im Hauptteil der Arbeit wurden die Anforderungen an das Programm analysiert. Anhand der gestellten Anforderungen konnte das Konzept für den BLF- und CoREPcapRecorder erstellt werden, welche daraufhin erfolgreich implementiert wurden.

Aufgrund der Evaluieurung kann gesagt werden, dass das Ziel der Arbeit erreicht wurde. Die Recorder können an jedes beliebige Ethernet- und CAN-Modul bzw. Netzwerk angebunden werden und nehmen daraufhin den Nachrichtenaustausch der Netzwerkkomponenten auf. Die Daten werden in einer BLF- bzw. PCAP-Datei gespeichert. Diese Dateien können in CANoe eingelesen werden, um anschließend den Nachrichtenaustausch der Komponenten wiedergeben zu können. Durch die Nutzung der entwickelten Recorder kann der CANoe-Nutzer in seiner gewohnten Simulationsumgebung die System-Level-Simulationsdaten analysieren und anschließend auswerten. Dabei sollte erwähnt werden, dass dieser Nutzer kein großes Verständnis für die OMNeT++ Simulationsumgebung haben muss. Lediglich eine Kenntnis der Netzwerktopologie ist für eine Zuordnung der in den Logging-Dateien enthaltenen Nachrichten vonnöten. Somit ist es gelungen, mithilfe

der entwickelten Recorder einen leichteren Einstieg in die System Level Simulation von OMNeT++ zu erreichen.

Ausblick

Diese Arbeit ermöglicht weitere künftige Arbeiten mit CANoe und OMNeT++ in Kooperation mit Vector Informatik GmbH [Vector Informatik GmbH, 2015]. Der BLFRecorder wird schon in der Masterthesis von Jonathan Wendeborn [Wendeborn, 2016] verwendet, um den Nachrichtenverkehr in jedem Port im Netzwerk aufzuzeichnen. Auf Basis der in den BLFs gespeicherten Events werden Buslast, Latenz und Jitter in CANoe berechnet. Der Automobilbauer befindet sich dabei in seiner gewohnten Umgebung und kann diese Daten anschließend weiterverarbeiten. Um eine Bewertung (gut/schlecht) des Netzwerks vornehmen zu können, werden für diese Metriken Grenzwerte eingestellt. Das Ergebnis dieser Überprüfungen wird letztendlich übersichtlich dargestellt. Eine Erweiterung des BLFRecorders kann z.B. darin bestehen, diesen um weitere Bussysteme wie FlexRay, LIN und MOST zu erweitern. Diese Möglichkeit unterstützt das BLF-Format. Für die neu dazu kommenden Nachrichtentypen (z.B. LIN-Frames) müssen lediglich neue Serializer geschrieben werden.

Literaturverzeichnis

[Bartols 2010] Bartols, Florian: *Leistungsmessung von Time-Triggered Ethernet Komponenten unter harten Echtzeitbedingungen mithilfe modifizierter Linux-Treiber.* Hamburg, Hochschule für Angewandte Wissenschaften Hamburg, bachelorsthesis, Juli 2010

[Baun 2013] Baun, Christian: *Computernetze kompakt.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2013. – ISBN 978-3-642-41653-8

[Borowka 1992] Borowka, Petra: *Brücken und Router. Wege zum strukturierten Netzwerk.* Bergheim : DATACOM-Verlag, 1992. – ISBN 9783892380573

[Bossel 2004] Bossel, H.: *Systeme, Dynamik, Simulation: Modellbildung, Analyse und Simulation komplexer Systeme.* Norderstedt : Books on Demand, 2004

[CoRE Research Group a] CoRE Research Group. – http://core.informatik.haw-hamburg.de/en/ - Zugriffsdatum: 2015-10-22

[CoRE Research Group b] CoRE Research Group: *CoRE4INET Framework.* – URL http://core4inet.realmv6.org/trac/. – Zugriffsdatum: 2015-10-22

[CoRE Research Group c] CoRE Research Group: *FiCo4OMNeT Framework.* – URL https://core4inet.core-rg.de/trac/wiki/FiCo4OMNeT_Background. – Zugriffsdatum: 2015-10-22

[CoRE Research Group d] CoRE Research Group: *Signals and Gateways Framework.* – URL https://core4inet.core-rg.de/trac/wiki/SignalsAndGateways_Background. – Zugriffsdatum: 2015-10-22

[Eclipse Foundation] Eclipse Foundation: *Eclipse IDE.* – URL http://www.eclipse.org/. – Zugriffsdatum: 2015-10-18

[Ernst u. a. 2015] Ernst, H. ; Schmidt, J. ; Beneken, G.: *Grundkurs Informatik: Grundlagen und Konzepte für die erfolgreiche IT-Praxis - Eine umfassende, praxisorientierte Einführung*. Springer Fachmedien Wiesbaden, 2015. – ISBN 9783658016289

[Grudzinski u. a. 1992] Grudzinski, J. ; Kissing, W. ; Zaplata, L.: *Untersuchung selbsterregter Reibungsschwingungen mit Hilfe eines numerischen Simulationsverfahrens*. 1992. – URL http://www.uni-magdeburg.de/ifme/zeitschrift_tm/1992_Heft1/Grudzinski_Kissing_Zaplata.pdf. – Zugriffsdatum: 2015-10-15

[Heyden 2012] Heyden, M.: *Untersuchung von Qualitätsparametern ethernetbasierter Audionetzwerke*. Diplom.de, 2012. – ISBN 9783842827967

[Hüning 2016] Hüning, Felix: *Sensoren und Sensorschnittstellen*. De Gruyter, 2016 (De Gruyter Studium). – ISBN 9783110438550

[IEEE 2010] IEEE: *IEEE EtherType Registration Authority*. 2010. – URL http://standards.ieee.org/regauth/ethertype/eth.txt. – Zugriffsdatum: 2016-02-01

[Institute of Electrical and Electronics Engineers 2011a] Institute of Electrical and Electronics Engineers: IEEE 1722 - IEEE Standard for Layer 2 Transport Protocol for Time Sensitive Applications in a Bridged Local Area Network / IEEE. 2011. – Standard

[Institute of Electrical and Electronics Engineers 2011b] Institute of Electrical and Electronics Engineers: IEEE 802.1BA - IEEE Standard for Local and metropolitan area networks - Audio Video Bridging (AVB) Systems / IEEE. September 2011 (IEEE 802.1BA-2011). – Standard. – ISBN 987-0-7381-7639-8

[Kiencke und Puente León 2013] Kiencke, U. ; Puente León, F.: *Ereignisdiskrete Systeme: Modellierung und Steuerung verteilter Systeme*. München : Oldenbourg Verlag, 2013. – ISBN 978-3-486-73574-1

[Lawrenz und Obermöller 2011] Lawrenz, W. ; Obermöller, N.: *Controller-Area-Network: CAN ; Grundlagen, Design, Anwendungen, Testtechnik*. VDE-Verlag, 2011. – ISBN 9783800733323

[Lim u. a. 2012] Lim, Hyung-Taek ; Herrscher, Daniel ; Waltl, Martin J. ; Chaari, Firas: Performance analysis of the IEEE 802.1 ethernet audio/video bridging standard. In: *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*. New York : ACM-DL, März 2012, S. 27–36. – ISBN 978-1-4503-1510-4

[OMNeT++ Community a] OMNeT++ Community: *INET Framework Vers. 3.0 für OMNeT++ 5.0*. – URL https://inet.omnetpp.org/. – Zugriffsdatum: 2015-10-09

[OMNeT++ Community b] OMNeT++ Community: *OMNeT++ 5.0b1*. – URL http://www.omnetpp.org/. – Zugriffsdatum: 2015-10-05

[OMNeT++ Community c] OMNeT++ Community: *OMNeT++ Manual*. OMNeT++ Community. – URL https://omnetpp.org/doc/omnetpp/manual/usman.html. – Zugriffsdatum: 2015-10-24

[Riverbed Technology] Riverbed Technology: *WinPcap - Windows Packet Capture library*. – URL http://www.winpcap.org/. – Zugriffsdatum: 2016-01-28

[Robert Bosch GmbH 1991] Robert Bosch GmbH: *Controller Area Network Specification*. 1991. – URL http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf. – Zugriffsdatum: 2016-04-03

[Schauffele und Zurawka 2006] Schauffele, J. ; Zurawka, T.: *Automotive Software Engineering: Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen*. Vieweg Verlag, 2006 (ATZ/MTZ-Fachbuch). – ISBN 3834800511

[Society of Automotive Engineers - AS-2D Time Triggered Systems and Architecture Committee 2011] Society of Automotive Engineers - AS-2D Time Triggered Systems and Architecture Committee: *Time-Triggered Ethernet AS6802*. SAE Aerospace. 2011. – URL http://standards.sae.org/as6802/. – Zugriffsdatum: 2016-02-01

[Spinczyk u. a. 2007] Spinczyk, Olaf ; Engel, Michael ; Schirmeier, Horst ; Streicher, Jochen: *AutoLab - Eine Experimentierplattform für automotive Softwareentwicklung*. 2007. – URL http://ess.cs.tu-dortmund.de/Teaching/PGs/autolab/AutoLab-Endbericht.pdf. – Zugriffsdatum: 2015-11-22

[Steinbach 2011] Steinbach, Till: *Echtzeit-Ethernet für Anwendungen im Automobil: Metriken und deren simulationsbasierte Evaluierung am Beispiel von TTEthernet*, Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, 2011

[Steinbach u. a. 2015] Steinbach, Till ; Meyer, Philipp ; Buschmann, Stefan ; Korf, Franz ; Schmidt, Thomas C.: Demo: Prototyping Next-Generation In-Car Backbones Using System-Level Network Simulation. In: *2015 IEEE Conference on Local Computer Networks (LCN)*, Oktober 2015. – Accepted for publication

[Tanenbaum und Wetherall 2012] Tanenbaum, A.S. ; Wetherall, D.J.: *Computernetzwerke*. München : Pearson, 2012. – ISBN 9783868941371

[The Tcpdump team] The Tcpdump team: *PCAP - Packet Capture library*. – URL http://www.tcpdump.org/pcap3_man.html. – Zugriffsdatum: 2016-01-28

[TTTech Computertechnik AG] TTTech Computertechnik AG: . – URL http://www.tttech.com. – Zugriffsdatum: 2016-02-01

[TTTech Computertechnik AG 2009] TTTech Computertechnik AG: *Skalierbare Echtzeit-Ethernet-Plattform*. TTTech Computertechnik AG. 2009. – URL http://www.funkschau.de/fileadmin/media/whitepaper/files/WP_TTEthernet_Artikel-g.pdf. – Zugriffsdatum: 2016-02-03

[TU München 2014] TU München: *Technische Universität München: Praktikum Simulationstechnik*. 2014. – URL https://www.ais.mw.tum.de/fileadmin/w00bdq/www/Lehre/SimT/Praktikum_Simulationstechnik_-_Einfuehrung.pdf. – Zugriffsdatum: 2015-10-17

[Varga 2001] Varga, Andras: The OMNeT++ discrete event simulation system. In: *European Simulation Multiconference (ESM'2001)*, EMC, Juni 2001. – URL http://www.omnetpp.org/download/docs/papers/esm2001-meth48.pdf

[Varga und Hornig 2008] Varga, András ; Hornig, Rudolf: An overview of the OMNeT++ simulation environment. In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, networks and systems & workshops*. New York : ACM-DL, März 2008, S. 60:1–60:10. – URL http:

//portal.acm.org/citation.cfm?id=1416222.1416290. – ISBN 978-963-9799-20-2

[Vector Informatik GmbH a] Vector Informatik GmbH: *BINLOG DLL Manual Version 1.5*. Vector Informatik GmbH

[Vector Informatik GmbH b] Vector Informatik GmbH: *CANoe User Manual*. Vector Informatik GmbH. – URL http://vector.com/portal/medien/cmc/manuals/CANoe75_Manual_EN.pdf. – Zugriffsdatum: 2015-10-24

[Vector Informatik GmbH 2015] Vector Informatik GmbH: *CANoe Version 8.5.62*. Vector Informatik GmbH. 2015. – URL http://www.vector.com. – Zugriffsdatum: 2015-10-24

[Verein Deutscher Ingenieure 2014] Verein Deutscher Ingenieure: *VDI3633-1: Simulation von Logistik-, Materialfluss- und Produktionssystemen - Grundlagen*. Berlin : Beuth Verlag, 2014

[Wendeborn 2016] Wendeborn, Jonathan: *Grafikbasierte Analyse und Bewertung heterogener automobiler Netzwerke*, Hochschule für Angewandte Wissenschaften Hamburg, Masterarbeit, 2016

Abbkürzungsverzeichnis

OMNeT++ Objective Modular Network Testbed in C++ 11

GPL GNU General Public License 11

NED Network Description 11

CoRE Communication over Real-time Ethernet 2

AVB Audio Video Bridging 13

TT Time-Triggered 13

TTEthernet Time-Triggered Ethernet 13

Open Systems Interconnection OSI-Modell 14

RC Rate-constrained 13

BE Best-effort 13

PCAP Packet Capture vi

AVTP Audio/ Video Transport Protocol v

VLAN Virtual Local Area Network 41

RTR Remote Transmission Request 43

IDE Identifier Extension 43

r0 Reserved 43

BLF Binary Logging Format iv

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

Hamburg, 4. Mai 2016 Besnik Mulici