Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Sebastian Schrade

Kopplung einer AUTOSAR eventbasierten Simulation mit der OMNET++ Simulation von Automotive-Netzwerken

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Sebastian Schrade

Kopplung einer AUTOSAR eventbasierten Simulation mit der OMNET++ Simulation von Automotive-Netzwerken

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang 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. Andreas Meisel

Eingereicht am: 26. März 2015

Sebastian Schrade

Thema der Arbeit

Kopplung einer AUTOSAR eventbasierten Simulation mit der OMNET++ Simulation von Automotive-Netzwerken

Stichworte

Event-Simulation, Synchronisation, AUTOSAR, VEOS, SystemDesk, Omnet++, VECU, ECU

Kurzzusammenfassung

Ziel der Arbeit ist die Untersuchung von Kopplungen im Simulationsbereich am Beispiel der Simulationsplattformen VEOS und Omnet++. Die AUTOSAR Steuergerätesimulation VEOS ist anzupassen und zu ergründen wie eine Kopplung zu erreichen ist, so dass die Netzwerksimulation Omnet++ die Kommunikation der Steuergeräte vornehmen kann.

Sebastian Schrade

Title of the paper

Coupling of an AUTOSAR event-based simulation with the OMNeT ++ simulation of automotive networks

Keywords

Event simulation, synchronization, AUTOSAR, VEOS, SystemDesk, Omnet++, VECU, ECU

Abstract

The aim of this work is the investigation of couplings in simulation using the example of simulation platforms VEOS and OMNeT ++. The AUTOSAR ECU simulation VEOS is to adapt and to explore how a coupling can be reached, so that the network simulation OMNeT ++ can perform the communication between the ECU.

Inhaltsverzeichnis

1. Einleitung

Kernthematik dieser Arbeit ist das Ergründen der Kopplungsmöglichkeiten von Simulationsumgebungen in Theorie und realer Implementation. In diesem Rahmen soll beleuchtet werden, wie ausgewählte Simulationen im speziellen Umfeld miteinander kooperieren können, beziehungsweise zur Kooperation gebracht werden. Hierzu ist behandelt, wie die Kopplung zustande kommt und was hierfür nötig ist. Die Kopplung ist der Hauptbereich der Arbeit und ist in die grundlegenden Bestandteile zerlegt behandelt, um die Arbeitsweise und Arbeitsschritte, die notwendig sind offenzulegen und zu analysieren. Es sind die gewählten Simulationen näher erläutert, sowie der AUTOSAR Simulationsinhalt in Form eines Steuergeräteverbunds. Das letzte Anliegen der Arbeit, ist die Behandlung des Aktes der Simulationskopplung mit Wort und Bild in dem Maße, dass eine Reproduktion des beschriebenen Erreichten möglich ist. Dennoch ist dies keine Dokumentation für den Softwareentwickler mit allen Implementierungsdetails, stattdessen sind Implementierungsprinzipien dargelegt, die von den realen Gegebenheiten abhängen.

Die Frage, die diese Arbeit initiierte ist die, nach "Dem Nutzen von Simulationen und deren Rolle in der Entwicklung der modernen technisierten Gesellschaft". Eine einzelne Simulation bietet die Möglichkeit, Erreichtes zu testen und zu validieren ohne die kostenaufwendige Nutzung realer Prototypen. Mit dem Komplexerwerden von Systemen, vor allem in solche, die mehrere Subsysteme beinhalten, wird systemübergreifendes Simulieren immer wichtiger. Im Kontext des Automobils ist dies gut zu verdeutlichen. Hier sind viele unterschiedliche Systeme verbaut, Simulation ist hier der Schlüssel um das Zusammenspiel der Komponenten zu verifizieren. Ohne diese Möglichkeiten hätte die Industrie mit höheren Kosten und Entwicklungszeiten zu kämpfen. Im Rahmen der CoRE Gruppe wird an neuen Netzwerken geforscht, die z.B. Realtime Ethernet in den Automotive Bereich, als Ergänzung oder Ersatz für die alt

eingesessenen Übertragungsmedien wie den CAN-Bus, bringen sollen. Die Forschung stützt sich auf die Simulation der neuen Netzwerke. An jener Stelle setzt diese Arbeit ein. Es wird ein bestehendes AUTOSAR Simulationssystem mit einer in der CoRE Gruppe entwickelten Netzwerksimulation verbunden. Der Nutzen ist, der Blick auf ein übergreifendes Simulationssystem, das originale AUTOSAR Komponenten beinhaltet und zur Kommunikation die CoRE Netzwerksimulation für z.B. Automotive Realtime Ethernet oder einen virtuellen CAN-Bus nutzt, was die Forschung der Realität durch Simulation näher bringt ohne hohe Kosten zu produzieren. Der Rahmen in der sich

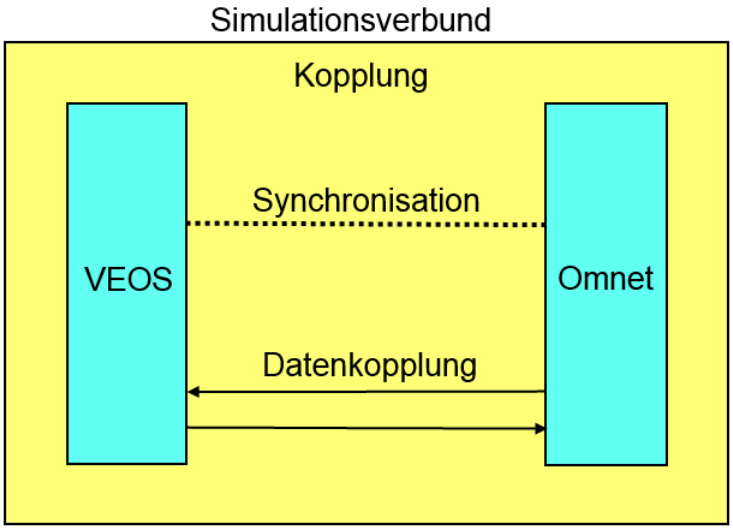

Abbildung 1.1.: Übersicht der zu erreichenden Simulation

diese Arbeit bewegt, sieht wie folgt aus: Es gibt zwei Simulationsumgebungen VEOS aus dem Hause DSpace und die Omnet++ Simulation für komplexe Netzwerke. Beide sind für einen eigenständigen unabhängigen Betrieb konzipiert. VEOS simuliert ein AUTOSAR System bestehend aus drei Steuergeräten die über einen virtuellen CAN-Bus verbunden sind. Omnet++ stellt eine Netzwerksimulation bereit, in diesem Fall

wird ein gesonderter CAN-Bus simuliert. Das zu lösende Problem liegt darin, die Kommunikation der Steuergeräte in VEOS so umzulenken, dass sie Zeit-richtig über die Omnet++ Netzwerksimulation ablaufen kann. Das Ergebnis ist eine durch Kopplung erzeugte Gesamtsimulation. Abbildung 1.1 vermittelt einen ersten grafischen Eindruck vom Ziel der Arbeit.

Die Struktur ist wie folgt: Im Verlauf der Arbeit sind zunächst die Grundlagen aufgearbeitet auf denen die weiteren Schritte gründen. Dann ist untersucht welche Maßnahmen möglich sind um das Ziel der Simulationskopplung zu erreichen und wie die Kopplung beschaffen sein kann. Auf Basis dieser Analyse ist ein Lösungsweg weiterverfolgt und ausführlich behandelt, während andere aufgezeigt und bewertet sind. Es ist ein Konzept erarbeitet wie die anschließende Implementation zu arbeiten hat. Diese Phase orientiert sich an den Gegebenheiten von VEOS und OMNET++ und vereinbart diese mit den Ergebnissen der Analyse. Der implementierte Prototyp ist im Rahmen der Evaluation, den Anforderungen aus der Analyse gegenübergestellt und die Richtigkeit der Funktion ist untersucht. Der erste Schritt dieser Arbeit ist die Auseinandersetzung mit den Grundlagen und den ersten frühen Theorien, die aus diesen erwachsen sind.

2. Grundlagen

Dieser Abschnitt behandelt Wissen, das zum Verstehen der zu entwickelnden Konzepte und Lösungen notwendig ist. Die Arbeit beschäftigt sich mit Simulationen, diese sind im Folgenden ebenso erläutert, wie das eigentliche Objekt der Simulation, das AUTOSAR Steuergerät mit seinen Charakteristiken.

2.1. AUTOSAR

AUTOSAR steht für AUTomotive Open System ARchitecture, es ist ein offener Industriestandard für Softwarearchitekturen im Automobilbereich auf den sich ein Zusammenschluss von Automobilherstellern, Subunternehmen und Zulieferer verständigt hat. Die Ziele von AUTOSAR sind die Schaffung standardisierter Funktionalität für Steuergeräte und Schnittstellen, so dass eine bessere Zusammenarbeit der unterschiedlichen Partner möglich ist und Grenzen im Automobilbereich zwischen den Entwicklungen durch den Standard abgebaut werden. Formell gegründet wurde die AUTOSAR Entwicklungspartnerschaft im Sommer 2003 von namhaften Firmen der Automobilbranche, wie BMW, Daimler, Volkswagen und Bosch um nur einige zu nennen. Seitdem nahm die Zahl der AUTOSAR Partner stetig zu. In dieser Arbeit basieren die Steuergeräte auf dem AUTOSAR Standard und somit ist ein grundlegendes Verständnis und Wissen, der im Standard beschriebenen Konzepte notwendig. Weitere Informationen mit umfassenden Detailgrad zu AUTOSAR können auf deren Internetauftritt unter Specifications (Release 4.2) eingesehen werden. In den nachfolgenden Abschnitten sind die für diese Arbeit wichtigen AUTOSAR Elemente behandelt. [8] [7]

2.1.1. ECU

ECU steht für electronic control unit im Deutschen sagt man Steuergerät. AUTOSAR definiert ECU in seinem Sinne wie folgt: “In the AUTOSAR sense an ECU means a microcontroller plus peripherals and the according software/configuration. Therefore, each microcontroller requires its own ECU Configuration.“ [1, 3.73] Im Bereich von Automobilsystemen kommen eine Vielzahl von ECUs zum Einsatz, die miteinander und mit ihrer Umgebung, also Sensoren und Aktoren, verbunden sind. Für diese Arbeit liegt der Schwerpunkt auf der Kommunikation der ECUs miteinander. Es ist wichtig die Interna der ECU und somit der artverwandten VECU (virtuelle ECU für Simulationen - siehe Abschnitt 2.3) im Bereich Kommunikation zu verstehen und zu analysieren, wie sie für das Ziel der Arbeit anzupassen sind.

2.1.2. Schichtenarchitektur von AUTOSAR

Aus einer hohen Abstraktionssicht betrachtet, ist der AUTOSAR Standard klar in Schichten, in der AUTOSAR-Terminologie Layer, unterteilt. Diese strukturieren die ECU in verschiedene funktionelle Teilbereiche. Die wichtigen Schichten für diese Arbeit sind das AUTOSAR Runtime Environment kurz RTE und der Service Layer, wobei auch Blicke in den ECU Abstraction Layer und den Microcontroller Abstraction Layer erfolgen. In der Grafik 2.1 können die Schichten in ihrer vertikalen Lage eingesehen werden. Die horizontale Lage untergliedert weiter, z.B. ist zu erkennen, dass sich das Element der Kommunikation über drei Layer erstreckt. Für die Grundlagenaufbereitung erfolgt die Betrachtung von oben nach unten, beginnend mit der Runtime Envionment.

2.1.3. RTE

AUTOSAR umschreibt das RTE einleitend so:
“The Run-Time Environment (RTE) is at the heart of the AUTOSAR ECU architecture. The RTE is the realization (for a particular ECU) of the interfaces of the AUTOSAR Virtual Function Bus (VFB). The RTE provides the infrastructure services that enable communication to occur between AUTOSAR software-components as well as acting as

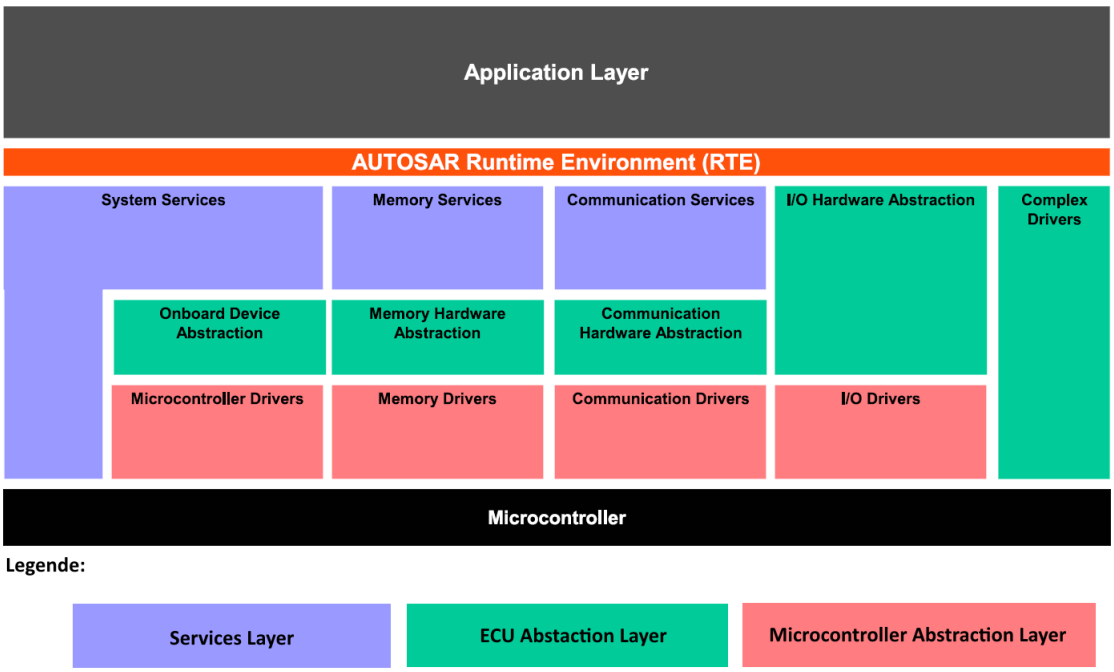

Abbildung 2.1.: ECU Architecture Overview[3, Figure 3.1]

the means by which AUTOSAR software-components access basic software modules including the OS and communication service.“[6, 2.1]

Das RTE fungiert als Middleware zwischen den AUTOSAR Elementen, ist also der Hauptträger der Kommunikation innerhalb der ECU. Es ist verbunden mit dem Application Layer, dem Service Layer und dem ECU Abstraction Layer. Es besteht keine direkte Verbindung zum Microcontroller Abstraction Layer oder zum Microcontroller. In Abb. 2.2 erfolgt die Betrachtung über die Layergrenzen hinweg nach Funktionalität. Die Grundstruktur bleibt, zwecks Vergleichbarkeit zum Abschnitt Schichtenarchitektur von AUTOSAR (Abb. 2.1), erhalten. Die Abbildung 2.2 zeigt die Übergabestellen zwischen den Funktionsblöcken. Folgendes ist für die Arbeit von besonderer Relevanz. Primär verläuft die Aufrufstruktur über das RTE. Die Verbindungsstellen zu den Elementen sind standardisierte Interfaces. Die Hauptanbindung des Operating System mit seinem angegliederten Scheduler erfolgt ebenfalls über das RTE.

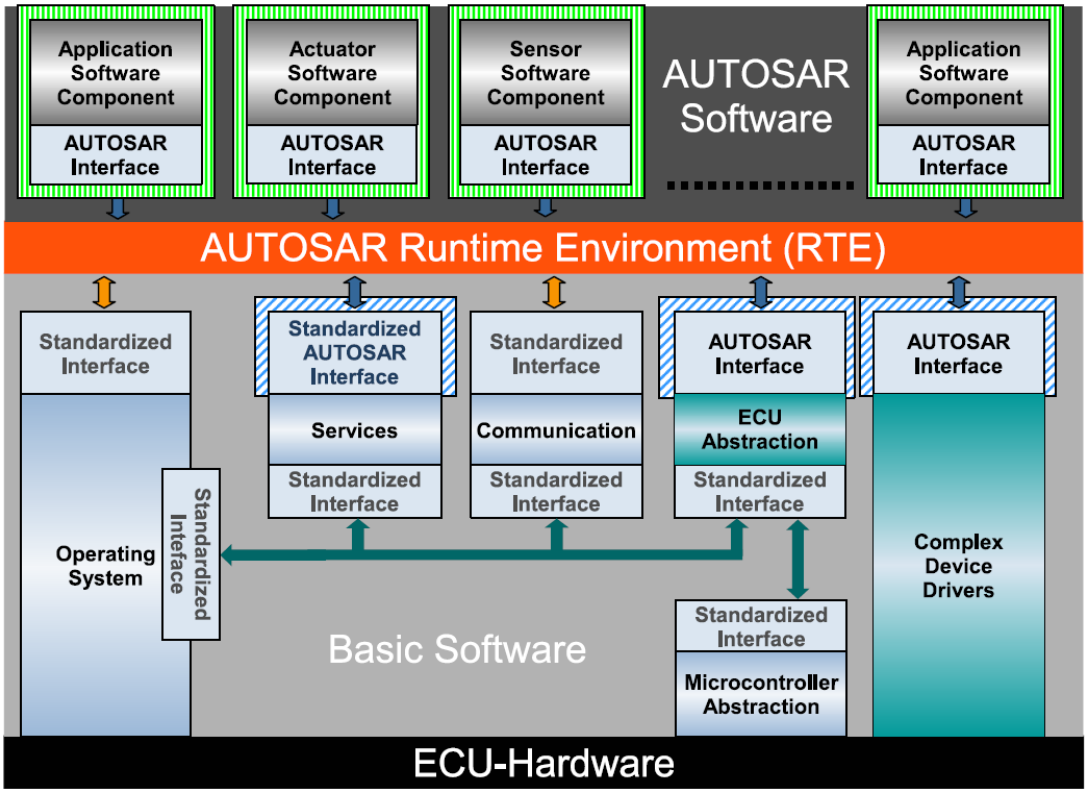

Abbildung 2.2.: ECU Architecture[6, Figure 4.2]

2.1.4. Service Layer und Operating System

Der AUTOSAR Service Layer befindet sich direkt unter dem RTE, siehe Abb. 2.1 und über dem ECU Abstraction Layer. Es beinhaltet neben den Memory Services, das Operating System und Communication Services. Diese letzten zwei Punkte mit ihren standardisierten Interfaces, inklusive der verbundenen RTE, sind die Kernbereiche für die zu entwickelnde Kopplung. Der Communication Service wird in einem eigenen Abschnitt 2.1.5 betrachtet.

Operating System

Das Operating System kurz OS, ist dem Service Layer zugeordnet und ist ein wichtiger Teil der AUTOSAR ECU (Abb. 2.2). Für das Operating System, kommt OSEK OS als

bewährte Technologie zum Einsatz. An OSEK wurde bis 2007 von der Siemens AG entwickelt. OSEK ist aktuell Eigentum von der Continental Automotive GmbH. [9] AUTOSAR beschreibt es so: “OSEK OS is an event-triggered operating system. This provides high flexibility in the design and maintenance of AUTOSAR based systems. Event triggering gives freedom for the selection of the events to drive scheduling at runtime, for example angular rotation, local time source, global time source, error occurrence etc.“ [4, 7.1.1]
Der Scheduler ist eng an das OS und das RTE gebunden. Aufgaben in Form von Runables (standardisierter Codeblock) können im Generierungsprozess der ECU als Task in den Ablauf aufgenommen und im vorher zu definierenden Intervall zur Ausführung gebracht werden. AUTOSAR spezifiziert das so: “The configuration of an AUTOSAR system maps the »runnables« of a »software component« to (one or more) tasks that are scheduled by the operating system.“[4, 4.3]
Eine weitere Eigenart des OS “The API of the operating system is defined as C function calls or macros. If other languages are used they must adapt to the C interface.’"[4, 4.6.2]
Die Betriebssystem Programmierschnittstellen sind C basierte Funktionsaufrufe. Das Kernwissen ist, man kann Runables mit definiertem Zeitverhalten in das System einbinden.

2.1.5. AUTOSAR Communication

Das AUTOSAR Communication verläuft über drei Layer hinweg. Der Service Layer stellt mit seinen Interfaces die direkte Anbindung zu RTE her. Der ECU Abstraction Layer stellt die standardisierte Interface Brücke zum Microcontroller Abstraction Layer dar, wo die Treiber für die Hardware erreichbar sind. Abbildung 2.3 zeigt dies detailliert auf (in groß Abb. A.1 S.73). Neben den vielen Verbindungen, die in der Abbildung zu sehen sind, kann man den Hauptnachrichtenpfad ausmachen. Die wichtigen Teile der PDU ROUTER, das AUTOSAR COM und das Signal selbst sind im Anschluss behandelt. Der Nachrichtenpfad beginnt oben beim RTE mit der Übertragung von Signalen an AUTOSAR COM. Dann erfolgt die Übergabe an den PDU Router und die Weiterleitung an das entsprechende Medium über die Hardware Abstractions Interfaces

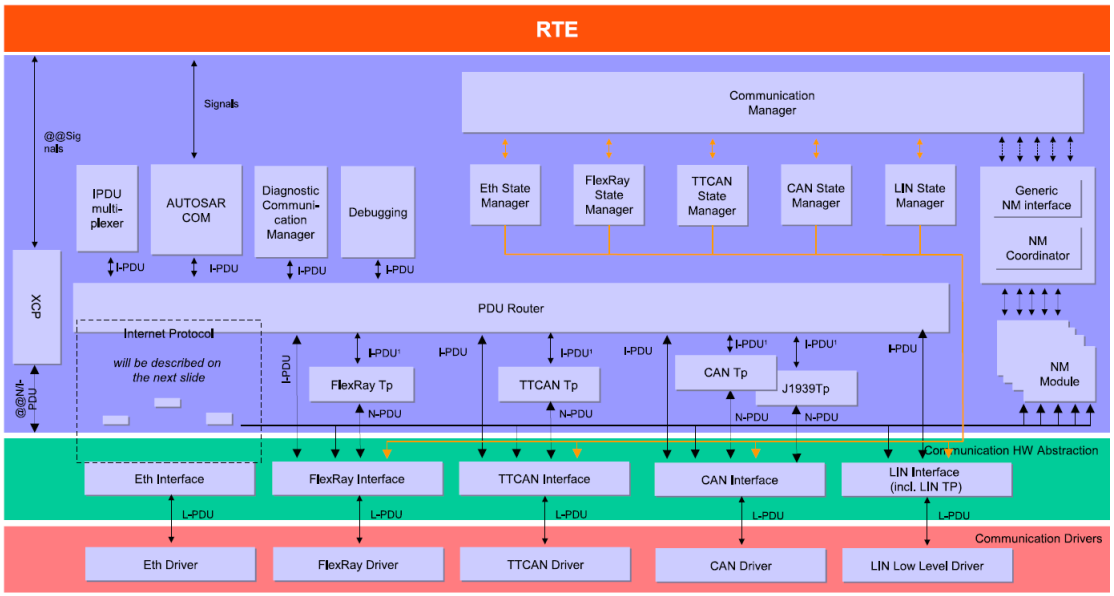

Abbildung 2.3.: ECU Communication (vergrößert auf S.73)[3, Figure 3.10]

an die Gerätetreiber. Das Wissen um den allgemeinen Verlauf von Kommunikation ist wichtig, um sich an später Stelle Möglichkeiten des Eingriffs in diese Abläufe herleiten zu können.[3]

2.1.6. Signals

Ein Signal in AUTOSAR ist grundsätzlich formlos in dem Sinne, dass es unabhängig vom Übertragungsmedium ist. Wenn man also auf Signalebene denkt und analysiert ist einzig die Frage, “wohin wird es übertragen“, interessant und nicht das wie. Signale sind vom Übertragungsmedium abstrahierte, losgelöste Nachrichten, die erst einen konkreten Charakter in der Übertragung durch den Com Services erhalten, der sie z.B. als Ethernet Nachricht überträgt. Es ist also wichtig zu wissen, wann noch Signale vorliegen und wann es konkrete Übertragungen werden. Diese Konkretisierung beginnt im AUTOSAR COM.

2.1.7. AUTOSAR COM

AUTOSAR COM ist primär ein Signal Gateway. In Abbildung 2.3 ist zu sehen, dass AUTOSAR COM Signale vom RTE entgegennimmt. AUTOSAR COM ist eine Vorverarbeitungsstelle für den angegliederten PDU Router (siehe Abschnitt 2.1.8). Signale werden aufgenommen und für das Routing als I-PDU (Interaction Layer Protocol Data Unit [1, 3.133]) verpackt und weiter versandt. Das I-PDU enthält Informationen (I-PDU IDs) die dem PDU Router die Zuordnung, mittels seiner Routing-Tabelle an ein Übertragungsmedium, erlauben. AUTOSAR sagt zudem:

"I-PDUs are identified by static I-PDU IDs. The PDU Router module determines the destination of an I-PDU by using the I-PDU ID in a static configuration table. I-PDUs are used for the data exchange of the modules directly above the PDU Router module, e.g. the COM module and DCM module. The routing operation of the PDU Router module does not modify the I-PDU, it simply forwards the I-PDU to the destination module." [5, 1.3]

Im Detail kann die Arbeit von AUTOSAR COM in der "Specification of Communication; AUTOSAR Release 4.2.1" Figure 5 bis 7 eingesehen werden, die Gateway Funktionalität wird in dem Dokument in Abschnitt 7.2.5 beschrieben. Es ist zu vermerken, dass am Ausgang nicht mehr reine Signale vorliegen, sondern die Vorstufe zu der realen Nachricht über ein konkretes Medium. Die finale Zuordnung an das Übertragungsmedium nimmt der PDU Router vor.[5, 2]

2.1.8. PDU Router

PDU steht für Protocol Data Unit. Der PDU Router verteilt I-PDU Nachrichten, die aus Signalen an AUTOSAR COM (2.1.7) gebildet wurden, an die Übertragungsmedien z.B CAN.

In Abbildung 2.3 ist verdeutlicht, dass der PDU Router ein zentrales Element in AUTOSAR Communication ist. Hier werden I-PDU, basierend auf einer Routing-Tabelle, physischen Übertragungsmedien zugewiesen, die Zuweisung erfolgt anhand ihrer IDs. AUTOSAR beschreibt die Routing Tabelle so:

"The PDU Router routing tables: static routing tables describing the routing attributes for each I-PDU to be routed. The routing tables can be (if supported) updated

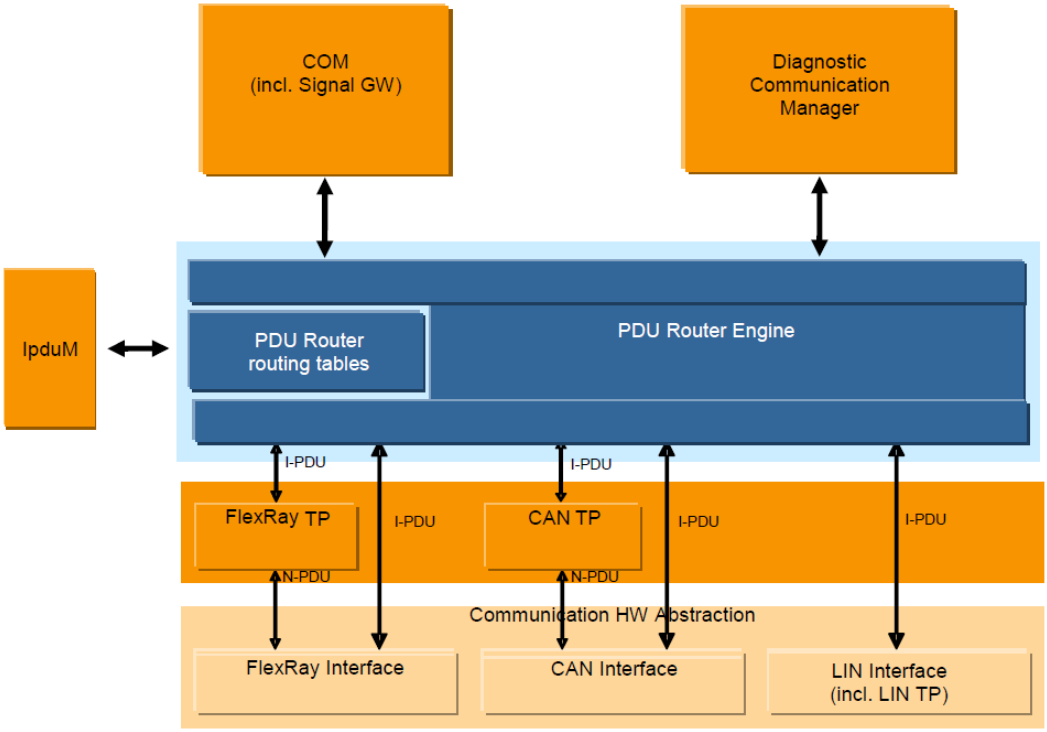

Abbildung 2.4.: PDU Router[5, Figure 2]

post-build loadable in the programming state of the ECU or selected when initializing the PDU router by post-build selectable.“ [5, 1.2]

Diese Beschreibung enthält eine sekundäre Information, es wird gesagt, dass die Routing-Tabelle post-build aktualisierbar ist. Der Nutzen in dieser Information liegt darin, dass man eine Möglichkeit hat die Routingtabelle ohne größere Eingriffe auch nach der Generierung der ECU zu ändern und so direkten Einfluss auf die Datenkommunkation nehmen kann. In Abbildung 2.4 ist der PDU Router zu sehen und seine lokalen Anbindungen. Die gerouteten I-PDU werden nach unten zum Communication HW Abstraction Layer weiter geleitet, damit ist das Übertragungsmedium direkt festgelegt. Das Wissen, dass alle Nachrichten von und nach außen hier passieren müssen, ist von besonderem Belang. Eine erste Eingriffsmöglichkeit in den Kommunikationsablauf ist das Anpassen der Routing-Tabelle.

2.2. Eventbasierte Simulation

Bei einer eventbasieten Simulation liegen die Ereignisse (engl. Events) in einer zeitlichen und kausalen Ordnung vor. Sobald die Simulationszeit den Zeitpunkt eines Ereignisses erreicht hat wird dies ausgeführt. Da zwischen den Ereignissen, die im Ablaufplan eingereiht sind nichts passiert, bewegt sich der zeitliche Verlauf direkt entlang der von den Ereignissen beschriebenen Zeitpunkten. Neue Ereignisse können zur Laufzeit in diesen Ablaufplan aufgenommen werden. Jedes ausgeführte Ereignis überführt das System in den nächsten zeitgebundenen Simulationszustand. Zusammenfassend, die Simulation schreitet in einer Zeitstempel-Ordnung (engl. Timestamp Order) voran, deren Ablauf durch die jeweiligen Vorbedingungen eindeutig festgelegt ist.

2.3. VECU

Die VECU ist eine Variation der ECU (Abs.2.1.1). Das “V“ steht für virtuell, also eine virtuelle ECU. Diese wird in der Simulation verwendet und hat die Eigenschaften einer physischen ECU, wobei die physischen Anteile durch virtuelle Simulationsteile ersetzt werden. Als Beispiel, braucht die ECU einen angeschlossenen CAN-Bus auf dem sie kommuniziert. Dieser reale Teil ist in der reinen Softwaresimulation nicht vorhanden und wird in der VECU von einem virtuellen CAN Adapter übernommen. In dieser Arbeit wird sich mit Simulationen beschäftigt, daher finden VECUs Anwendung.
Das Simulieren von ECUs stellt eine Möglichkeit zur Kostenersparnis dar, da keine Hardware angefertigt werden muss, um zu testen. Des Weiteren werden keine Prüfstände benötigt, da Erprobung in erster Phase auf dem Rechner ausgeführt werden kann. Es werden schnell kostengünstige Ergebnisse erzielt, Schlagwort "rapid prototyping".

2.4. Arbeitsmittel

Dieser Abschnitt beschreibt die verwendeten Arbeitsmittel und Plattformen. Die genutzten Programmiersprachen sind C und C++, diese sind als bekannt vorausgesetzt und nicht näher erläutert.

2.4.1. SystemDesk

SystemDesk aus dem Hause dSPACE ist ein Werkzeug zur Generierung von ECUs, VECUs und ganzer Systeme aus diesen. Es erlaubt die Steuergeräte aktiv zu modellieren. So ist es möglich die Interna bestehender ECU frei zu gestalten und z.B. Schnittstellen neu zu definieren oder neue Ausführungsblöcke in den Ausführungsplan der ECU einbringen. In Abschnitt 5 *Umsetzung und Implementierung* ist SystemDesk verwendet, die nötigen Details sind im Rahmen des Anwendungsfalls erläutert.
SystemDesk wurde gewählt, da es die nötige "Toolchain", die Werkzeugkette zum Erstellen eines AUTOSAR konformen Steuergeräts vereint und weitere Build Tools unnötig macht. Des Weiteren finden die generierten VECUs Anwendung im Simulationswerkzeug VEOS.

2.4.2. VEOS

VEOS ist die Simulationsplattform für VECU, diese sind sowohl einzeln als auch im Systemverbund simulierbar.
Wie auch SystemDesk, kommt VEOS aus dem Hause dSPACE und ist damit kompatibel zu deren anderen Produkten und Formaten. Im Regelanwendungsfall wird das Simulationsobjekt von einer externen Software generiert, dieses liegt dann in einem passenden Format vor und kann von VEOS zur Simulation genutzt werden. Die aktive Simulation innerhalb von VEOS genannt Simulationskernel, ist ein gesonderter Betriebssystem Prozess, der die Teilobjekte instantiiert, zum Beispiel ein System von drei VECU, die mit einem virtuellen Bus verbunden sind. Die VECU verhalten sich in ihrer Simulationsumgebung wie ihre realen ECU Gegenstücke. Für situationsspezifische Testszenarien passt man die virtuelle Umgebung an und gleicht die Eingangssituation der Sensoren mit der erwarteten Ausgabe an den virtuellen Schnittstellen der Aktoren ab. VEOS selbst kann während der Simulation Informationen auf der Konsole ausgeben. Auch wenn die VEOS Simulation sich auf mehrere VECU Teilelemente verteilt liegt eine eventbasierte Simulation, mit einer globalen Simulationszeit für alle VECUs, vor. VEOS mit seiner AUTOSAR Simulation ist das erste Objekt der Simulationskopplung in dieser Arbeit und es soll mit der Omnet++ Simulation verbunden werden.

2.4.3. Omnet++

Omnet++ ist eine Simulation für komplexe Kommunikationsnetzwerke und arbeitet eventbasiert. Die CoRE Gruppe der HAW arbeitet intensiv mit dieser Simulationsplattform, um unter anderem die Einsetzbarkeit von Realtime-Ethernet im Automotive Bereich zu untersuchen. In diesem Rahmen findet auch die Entwicklung dieser Arbeit statt. Es soll die Omnet++ Simulation mit der VEOS Simulation von AUTOSAR Steuergeräten gekoppelt werden. Die Omnet++ Simulation besteht aus Modulen auch Knoten genannt, die mit Kommunikationspfaden für den Nachrichtenaustausch verbunden sind. Im Omnet++ Kontext wird eine Nachricht als Message bezeichnet und ist dort als Klassenrepräsentation zu finden. Omnet++ mit seinen objektorientierten Konzepten nutzen die Sprache C++. Als Gegenstelle zu VEOS ist Omnet++ ein zentrales Element dieser Arbeit, im weiteren Verlauf ist auf Omnet++ im praktischen Anpassungsbeispiel eingegangen und das Wissen um diese Simulation auf das nötige Maß erweitert. Wichtig ist: Die Simulation ist eventbasiert, es gibt zeitlich festgelegte Nachrichten zwischen den Modulen und es gibt von der CoRE Gruppe entwickelte Frameworks für Onmet++.

3. Analyse der Problemstellung

Untersucht wird das Problem der Simulationskopplung in genereller Form , so dass die Ergebnisse in der Konzeption auf den speziell vorliegenden Fall übertragbar sind. Dieser beinhaltet die Simulation VEOS, in der Steuergeräte über einen CAN-Bus kommunizieren. Es soll für die Kommunikation eine Datenkopplung vorgenommen werden, die sie über die Netzwerksimulation Omnet++ umleitet, in der ein CAN-Bus simuliert wird. Der Nutzen liegt in der Verbindung beider Simulationen, so dass z.B. in der Netzwerksimulation Omnet++ reale Steuergerätekommunikation untersucht werden kann und umgekehrt Auswirkungen der Kommunikation auf die Steuergeräte erfassbar werden (Abb.1.1 S.2).
Die Analyse der zu lösenden Aufgabe, wird mit der Zerlegung des Ziels in Teilpunkte und nötige Voraussetzungen geführt. Ziel in abstrakter Form ist eine Gesamtsimulation, die aus zwei Simulationen zusammengesetzt ist. Es werden zwei eventbasierte Simulationen gekoppelt. Von außen betrachtet, liefert eine Simulation eine Folge von Messdaten beziehungsweise Zuständen mit zugehörigem Zeitpunkt. Beim Blick auf den Akt der Simulation von innen, stellt man zwei Dinge fest: Es gibt Simulationszeit die in einer zeitlichen Ordnung voranschreitet und parallel gibt es den Simulationsinhalt, die Daten die sich im Simulationszyklus wandeln, beide sind einander zugeordnet. Diese Eckpunkte, Daten und Simulationszeit, der eventbasierten Simulation werden als Basis der Analyse verstanden. Separiert in Daten und Zeit sind die Probleme klar in ihrem Teilbereich fassbar. Man kann das Problem auf eine Frage reduzieren: “Wie können Zeit und Daten simulationsübergreifend gekoppelt werden, so dass es eine Gesamtsimulation ergibt?“.
Der folgende Abschnitt behandelt diese Frage. Zum einen ist die Synchronisation zwischen den Simulationen betrachtet, zum anderen ist der nötige Datenaustausch zwischen den Simulationen untersucht, um die Kopplung zu erreichen.

3.1. Anforderungen an die Synchronisation

In diesem Abschnitt wird die Simulationszeit in eventbasierten Simulationen untersucht, um das Problem der zeitlichen Kopplung der Simulationen zu zeigen und abstrakte Lösungen herzuleiten, die in der Konzeption konkretisiert werden.
Dabei ist die Einheit der Zeit für eine Simulation unerheblich. Wichtig ist, das was als Zeitmaßstab dient diskret kontinuierlich in eine Richtung strebt. Simulationszeit bietet einen großen Vorteil gegenüber der Zeit im konventionellen Sinne, es ist möglich sie zu pausieren. Der Nutzen, der hieraus gezogen wird, erlaubt es die Synchronisation der beiden Simulationen vorzunehmen.
Doch wie kann sich diese Synchronisation vorgestellt werden? Die beiden eventbasierten Simulationssysteme müssen dieselbe Zeitbasis verwenden oder zumindest eine, die in die andere eindeutig übersetzbar ist. Es gibt keine globale Uhr für die gekoppelten Simulationen, diese sind unabhängig. Die Aufgabe der Synchronisation ist eine Relation zwischen den Uhren herzustellen und zu erhalten. Synchron laufen sie wenn in Simulation A derselbe Zeitpunkt wie in B vorherrscht und das während des Voranschreitens der Simulation über alle Zeitpunkte hinweg. Abbildung 3.1 dient zur Visualisierung des Sachverhalts. Der Begriff Zeitpunkt nimmt hierbei eine wichtige Einschränkung vorweg, es wird nicht die ganze Zeit synchronisiert sondern Zeitpunkte(Events). Diese

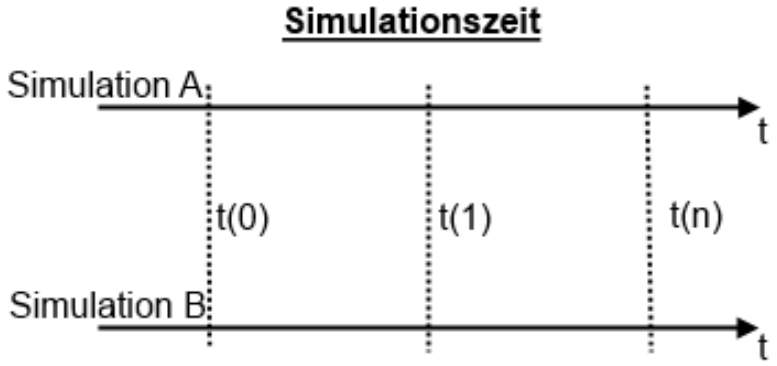

Abbildung 3.1.: Veranschaulichung der Simulationszeit mit Blick auf die Synchronisation in den Punkten t

sind eine grundlegende Eigenschaft von eventbasierten Simulationen, folglich stellen gekoppelte Synchronisationsevents den synchronen Ablauf sicher. Der Abstand der Synchronisationsevents ist die Auflösung der Synchronisation. Im Intervall zwischen diesen kann die Simulation in unbeeinflusster Form stattfinden, das schließt auch den Datenaustausch ein. Ein Beispiel zur besseren Vorstellung unter Zuhilfenahme der Kommunikation: Simulation A sendet an B, kurz vor Erreichen des Synchronisationspunktes t(1), eine Nachricht. Bei t(1) sind beide Simulationen synchron, A hat eine Nachricht gesendet und B hat eine Nachricht empfangen. Die Daten sind konsistent im Synchronisationspunkt. Auf diesem Ablaufgedanken, mit der Trennung von Daten und Synchronisation, gründen die weiteren Schritte. Es gibt auch andere Ansätze, diese sind in Abschnitt 3.1.1 gegenüber gestellt, in vielen Fällen setzen sie eine größere Flexibilität im Anpassen der Simulationsabläufe voraus.

Welche feste Synchronisationsauflösung muss mindestens gewählt werden? Die Simulationen müssen einen höheren Grad der Synchronisation aufweisen als die zeitliche Kommunikationsdichte auf einem Kanal, ist dies nicht gewährleistet, verliert man an Aussagekraft der Simulation. Es ist dann nicht festzustellen, ob zwei aufeinanderfolgende Events zeitgleich aufgetreten sind oder wie weit sie zeitlich auseinanderliegen. Der Versatz von Events, zwischen den gekoppelten Simulationen aufgrund der Synchronisation, ist als Drift bezeichnet. Der Drift ist abhängig von den genutzten Algorithmen in der Kopplung. Ein minimaler Drift spricht für eine hohe Güte der Simulationskopplung.
Zur Klassifizierung, eine eventbasierte Simulation arbeitet sequenziell. Im Kontext der Simulationskopplung arbeiten beide Simulationen verteilt und parallel zueinander. Für die synchronisierte Zusammenarbeit gibt es zwei Umsetzungskonzepte, unterteilt in konservative und optimistische Algorithmen. Konservativ bezeichnet einen Ansatz bei dem strikt vermieden wird, dass die Ordnung der Events verletzt wird. Der optimistische Ansatz lässt Nebenläufigkeit und das damit verbundene Konfliktpotential zu, setzt aber Fehlererkennung und Korrekturverfahren dagegen [13, S. 51-52].
Eine feste Auflösung der Synchronisation fügt viele auch unnötige Synchronisationsevents dem Simulationsablauf hinzu. Eine dynamische bedarfsorientierte Synchronisation ist genauer und ressourcensparender, ein Beispiel ist der konservative Null Message Algorithmus [13, S. 54-58] dieser wird in Abschnitt 3.1.1 vorgestellt.

Als nächstes wird untersucht, wie man die Simulation von einem synchronen Punkt zum nächsten bringen kann. Zwei Möglichkeiten werden erläutert, sequentielle und parallele Synchronisation. Es wird hierbei untersucht wie konservative oder optimistische Synchronisationsprinzipien einsetzbar sind.

3.1.1. Formen der Synchronisation

Synchronisation zweier eventbasierter Simulationen ist Ziel der Analyse. Zur Vereinfachung der Situation kann man die Simulationen als eindimensionaler Ablauf sehen und diese Dimension ist die Simulationszeit. Die Ereignisse beider Simulationen folgen je ihrer Zeitstempel-Ordnung. Bei zwei zu koppelnden Simulationen ist diese gemeinsame Ordnung erst zu schaffen. Die Notwendigkeit ist somit, durch Synchronisation die Ereignisse beider Simulationen in gemeinsamer Ordnung ablaufen zu lassen. Der erste konservative Gedankenansatz lässt beide Simulationen schrittweise aufeinanderfolgend arbeiten, dies wird erreicht durch Barrieren, die die Simulationen alternierend blocken. Der zweite optimistische Gedankenansatz lässt Parallelität zwischen den Simulationsabläufen zu, unter dem Aspekt, dass die kausale Ordnung der unabhängigen Kommunikation nicht gefährdet ist.
Wichtigste Voraussetzung für Synchronisation ist, dass die Simulationen unterbrechbar sind und das zu einem wählbaren Zeitpunkt. Ist das gegeben, kann mit Wissen über die Austauschpunkte der Simulationen die Synchronisationsauflösung gewählt werden, ggf. macht eine dynamische Synchronisationspunktverteilung dies obsolet. Dies festzulegen ist Teil der Konzeption 4.4. Es folgt eine Betrachtung des sequentiell konservativen und des parallelen Ansatzes. Die Datenkommunikation wird weitestmöglich von der Synchronisation gekapselt, sie ist in Abschnitt 3.2 gesondert behandelt.

Sequenzielle Synchronisation

Beim sequentiellen Gedanken übernimmt eine Simulation (A) die Vorreiterrolle, sie erreicht den Zeitpunkt t(n), wartet und gibt Simulation B, die bei t(n-1) wartet, das Signal nach t(n) aufzuholen. Ist t(n) erreicht, wird gewartet und Simulation A benachrichtigt, die dann weiter nach t(n+1) voranschreitet. So entsteht ein sequenzieller Simulationsablauf über die Synchronisationspunkte hinweg, der in Abbildung 3.2

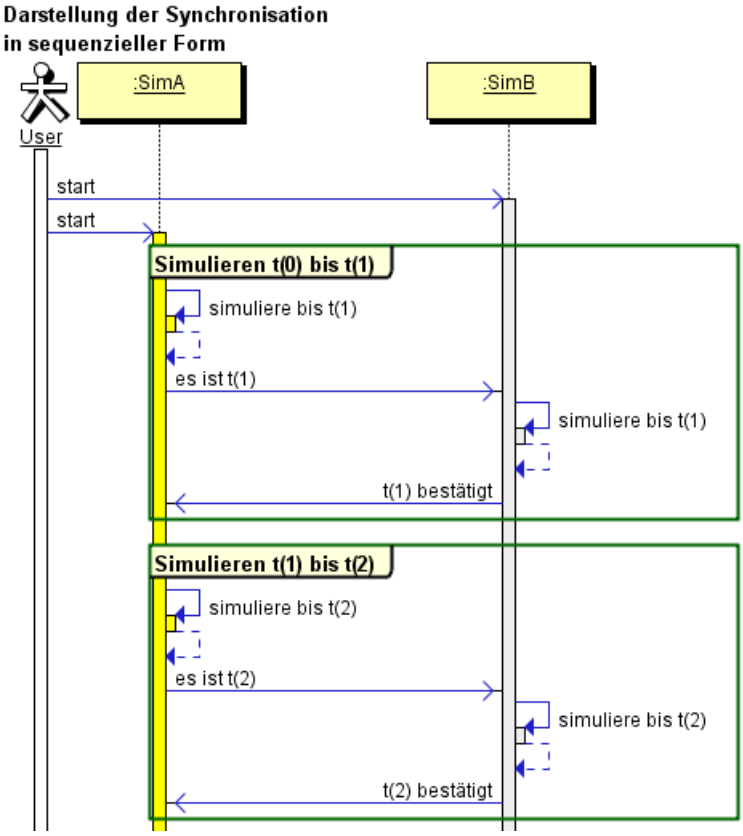

Abbildung 3.2.: Vereinfachte Darstellung der sequentiellen Synchronisation

veranschaulicht ist. Die Folge dieses Prinzips, es herrscht eine gemeinsame Ordnung, die ein Hybrid aus kausaler und Zeitstempel-Ordnung ist. Die Zeiträume zwischen den Synchronisationspunkten sind nicht synchron, was die Kommunikation zwischen den Simulationen betrifft. Daraus resultiert ein Drift, der maximal dem zeitlichen Abstand der Synchronisationspunkte entspricht. Dieser Drift tritt zwangsweise durch die Trennung der Kommunikation in der Kopplung von der Synchronisation auf und äußert sich durch den Versatz von Synchronisation zur Datenübertragung, die zur laufenden Simulationszeit stattfindet. Eine elegante Lösung zum Minimieren des Drifts stellt hier der Null Message Algoritmus dar.

“Whenever a process finishes processing an event, it sends a null message on each of its output ports indicating this bound; the receiver of the null message can then

compute new bounds on its outgoing links, send this information on to its neighbors, and so on. It is typically up to the application programmer to determine the time stamps assigned to null messages. This is the essential idea behind the "null message" or Chandy/Misra/Bryant algorithm (named after its inventors)." [13, S.57 oben] Dieses Zitat beschreibt den Kern des Algorithmus, in adaptierter Form wird er auf das vorliegende Problem übertragen und erläutert. Mit Abschluss jeder Recheneinheit wird eine Null Message versandt. Im Falle dieser Arbeit ist die "Recheneinheit" die Rechenzeit bis zu einem Kommunikationsereignis und die "NULL Message" die Synchronisationsnachricht. Anders als beim festen Intervall von Synchronisationspunkten erfolgt die Synchronisation direkt beim Sende-Event. Damit ist der Drift durch den Versatz von Synchronisationspunkt und Datenkommunikation minimal. Bei einem Blick auf die realen Gegebenheiten von VEOS und Omnet++ muss der Algorithmus noch weiter abgewandelt werden. Bei VEOS ist die Synchronisation nicht frei im Ablauf platzierbar. Im Original gibt es beim Null Message Algorithmus keine Führungsrolle, wenn man sich nach VEOS Möglichkeiten richtet und ihm eine Führungsrolle zugesteht, kann der Algoritmus mit Synchronisation nach jedem Senden und vor jedem Empfangen (VEOS seitig) umgesetzt werden.
Beide Optionen erhalten die Ordnung und die Simulation liefert reproduzierbare Ergebnisse. Ein Nachteil beim konservativen Ansatz ist, dass ein rein sequentieller Ablauf z.B. mittels Barrieren angestrebt wird, dieser macht in der Realzeit langwierige Wechsel notwendig. Die Einsparung von Realzeit führt zum nächsten Gedanken bei dem die Simulationen weitestgehend optimistisch nebenläufig arbeiten sollen.

Parallele Synchronisation

Ziel, beim Gedanken der parallelen Synchronisation, ist minimale Wartezeiten an den Barrieren während der Synchronisation zu haben und so die Realzeit optimal zu nutzen.
Es gibt Verfahren wie Time Warp, bei dem die Simulationen parallelisiert voranschreiten und bei Konflikten Rollbacks ausgeführt werden. Ein Konflikt ist eine Verletzung bzw. Unvereinbarkeit eines Events mit der zugrundeliegenden gekoppelten Simulationsordnung in kausaler oder zeitlicher Hinsicht. Der vorliegende Fall unterscheidet sich von den Voraussetzungen für das Verfahren, die Simulationen erlauben im Falle

eines Konflikts kein Rollback, somit müssen diese konservativ vermieden werden oder lösbar in akzeptablen Toleranzen bleiben. Konflikte sind in diesem konservativen Szenario der Arbeit nicht vorhanden, da die Synchronisationspunkte mit ihrer Zeitstempel-Ordnung nicht in eine Konfliktsituation geraten können und die Datensynchronisation in den Synchronisationspunkten gegeben ist.
Stattdessen wird eine parallele Variation der Synchronisation mittels Barrieren erprobt. Anders als im sequentiellen Fall, können beide Simulationen A und B bei t(n-1) zeitgleich starten, ist t(n) erreicht wird die jeweils andere Simulation benachrichtigt, anschließend wartet jede Simulation auf die Nachricht von der jeweiligen Gegensimulation. Bei Erhalt der Benachrichtigung ist sicher, dass die andere Simulation auch bei t(n) ist und die Simulation kann bis t(n+1) voranschreiten. Externe Einflüsse, z.B. Laufzeiten und Berechnungsdauer, können das Zeitverhalten der Simulationen zueinander zwischen den Synchronisationspunkten verändern, bei einem sequenziellen Ablauf ist dies nicht relevant, anders ist das bei Parallelität, wo sich dies in einem zeitlichen Drift zwischen den Synchronisationspunkten äußert. Der konsistente Zustand im Synchronisationspunkt bleibt unverändert erhalten.
Das Sequenzdiagramm 3.3 veranschaulicht den beschriebenen Ablauf. Die Reihenfolge, wann die Simulationen den Synchronisationspunkt in der Realzeit erreichen, ist nicht bestimmt und variabel, dies zeigt der Versatz beim Simulieren im Diagramm. Der Idealfall ist, wenn beide Simulationen in der Realzeit gleichzeitig ihre Ankunft am Synchronisationspunkt bekannt geben und weiterarbeiten. Das Diagramm stellt einen nicht Idealfall bei dem die Synchronisationsmeldungen versetzt sind dar. Der zeitliche Versatz im Simulieren ist das was als Simulationsdrift bezeichnet ist. In der Abbildung variiert er, anhand der Berechnungsdauer des Simulationsabschnitts, zwischen den Synchronisationspunkten. Beim Erreichen eines Synchronisationspunktes wird die Synchronisationsmeldung gesendet und auf Erhalt einer solchen von der Gegenstelle gewartet, ggf. ist diese bereits eingetroffen. Erst dann wird die Simulation bis zum nächsten Synchronisationspunkt fortgesetzt. Bei einer parallelisierten Simulationskopplung, in der es unterschiedliche Berechnungs- und Laufzeiten zwischen den Simulationen gibt, kann es zu einem maximalen Drift von einem Synchronisationspunktabstand für die Datenkommunikation kommen, je nach Perspektive kann dieser positiv oder negativ sein. Es kann sich eine zeitliche Führungsrolle abzeichnen

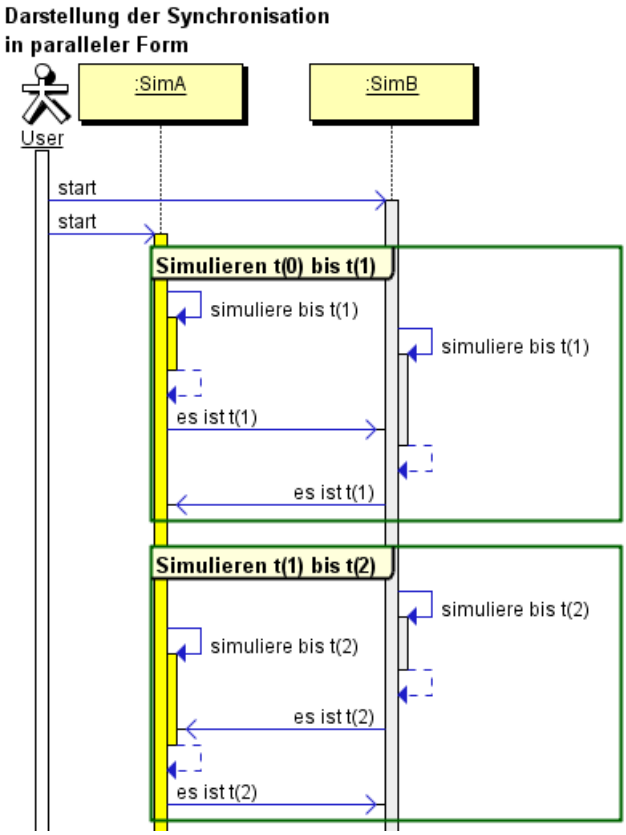

Abbildung 3.3.: Vereinfachte Darstellung der parallelen Synchronisation

bei der sich einem sequenziellen Ablauf angenähert wird und zum anderen kann diese Führungsrolle noch wechseln. Dies stellt das “worst case Scenario“ dar. Mit einer Erhöhung der Synchronisationspunktdichte können Drifts reduziert werden.
Abschließende Bewertung: Anders als beim sequentiellen Ansatz existiert ein dynamischer Drift zwischen den Simulationen. Dieser gefährdet nicht die zeitliche Integrität der Synchronisationspunkte und auch nicht ihren konsistenten Zustand, hinsichtlich der Daten. Dennoch stellt er eine Toleranz dar. Mit Verdopplung der Synchronisationsauflösung sind der sequentielle und parallele Ansatz diesbezüglich gleichwertig. Eine dynamische bedarfsorientierte Synchronisation wie z.B. beim konservativen Null Message Algorithmus liefert die besten Ergebnisse. Diese Denkansätze werden in der

Arbeit praktisch erprobt. Der Schwerpunkt liegt auf der sequentiellen Synchronisation mit fester und dynamischer Synchronisationsauflösung.

Anforderungen an die Implementation der Synchronisation

Es gibt allgemeine Anforderungen die übergeordnet, zu dem Konzept und der Implementierung, zu behandeln sind. Die Synchronisation in der Kopplung muss über ein allgemein eindeutiges Zeitformat erfolgen, welches mindestens die Simulationsauflösung unterstützt und nicht unterschiedlich von den Simulationsplattformen interpretiert werden kann. Um ein Beispiel zu nennen, wird ein Integer als Zeitformat an ein anderes System übertragen ist sicherzustellen, dass das “Most significant bit’"weiterhin an der korrekten Stelle interpretiert wird, bei der VECU ist dies z.B. frei konfigurierbar. Beide Simulationen müssen über ein Modul zur Synchronisation verfügen, das in den jeweiligen Ablauf der Simulation integriert ist. Der optimale Fall ist wenn die Synchronisation dynamisch zur Laufzeit der Simulation, Synchronisationspunkte im jeweiligen Ablaufplan anlegen kann, anderenfalls sind die Synchronisationspunkte statisch bei der Erstellung einzuplanen. Dies führt zu einer festen Synchronisationsauflösung, optional kann die Synchronisationsroutine auch an die Austauschpunkte der Kommunikation gebunden werden, was zu einer dynamischen bedarfsorientierten Synchronisation führt. Dies ist und muss Teil der Konzeption sein, da es von den Gegebenheiten und Möglichkeiten der Simulation abhängt.
Die Sendung, der für die Kopplung nötigen Synchronisationsmeldungen, muss über ein Medium erfolgen, welches ermöglicht die andere Simulation zu erreichen, idealerweise mit möglichst geringer Einschränkung, so ist Interprozesskommunikation möglich, schränkt aber den Ort der beiden Simulationen auf den selben Rechner ein.

3.2. Anforderungen an den Datenaustausch

Dieser Teil beschäftigt sich mit dem Datenaustausch der Simulationen in abstrakter Form, so dass die Ergebnisse in der Konzeption 4 aufgegriffen werden können, um die Kopplung der CAN-Daten der VEOS Simulation mit Omnet++, über ein im Konzept festzulegendes Medium, bidirektional herzustellen.

Produkt der Kommunikation der Simulationen sind Daten, diese werden in diesem abstrakten Teil als formlos betrachtet. Getrennt davon gibt es die Synchronisation, ihre Meldungen sind nicht Teil der Betrachtung. Unter Daten sind alle Übertragungen zu verstehen, die als Teil der Simulation zwischen den simulierten Objekten ausgetauscht werden. Oder aus Sicht des Simulationsobjekts, alle Nachrichten und Werte die von außen, also dem Raum der Simulationsumgebung in das Objekt eingespeist werden. Als Objekt wird z.B eine VECU gesehen, die mit den anderen VECUs oder der Netzwerksimulation Omnet++ in kommunikativer Interaktion steht.
Kommunikation fordert mindestens zwei Kommunikationsakteure, Sender und Empfänger. In einem unidirektionalen Datenkanal erfolgt die Kommunikation immer in die Richtung Sender-Empfänger, daher werden zunächst die Eigenarten eines solchen Senders untersucht und danach, die des Empfängers.

Sender

Ein Sender in der abstrakten Sicht, muss Daten aus der Simulation, die auf einem beliebig gearteten Kommunikationspfad anfallen, nach außerhalb der Simulation übertragen und sie an seinem Ziel verfügbar machen.
Im Folgenden wird diese Aussage aufgeschlüsselt und in Teilaufgaben beziehungsweise Anforderungen untergliedert. Die Datenkopplung kann an einer beliebigen Stelle innerhalb der Simulationen vorgenommen werden, daher muss ein Sender in Codeform an jeder Stelle platzierbar sein, dies wird im Folgenden als Ortsunabhängigkeit des Senders oder Ortstransparenz verstanden. Was sind diese Daten? Es herrscht bezüglich der Daten Transparenz, folglich ist Form und Art der Daten bis zum Punkt des konkreten Einsatzortes unbekannt, daher muss ein Sender mit jeder Datenform arbeiten können oder mit etwas, in das jede Datenform übersetzt werden kann, möglich ist eine Byte-Repräsentation. Das Ziel des Senders soll außerhalb der Simulation liegen, unterschieden wird in Rechner intern und extern. Der Sender muss so gestaltet sein, dass er Prozessgrenzen, optional auch Rechnergrenzen, überschreiten kann. Das Sendemedium muss entsprechend dazu in der Lage sein und das Ziel eindeutig adressieren können. Um den Zeitaspekt zu behandeln, seien zwei Möglichkeiten gegeben. Entweder die Sendung findet direkt beim Anfallen der zu übertragenden Daten statt oder es

werden Daten gesammelt und gebündelt zu einem definierten Simulationszeitpunkt abgeschickt. Die zweite Variante bringt viele Besonderheiten mit sich und eine nicht gewollte Kopplung an die Zeit. Diese Option bleibt gedanklich erhalten, aber Fokus dieser Arbeit liegt auf dem ersten Fall, dem Senden der Daten unmittelbar beim Auftreten. Die Tätigkeit des Senders erzeugt zwangsweise ein Duplikat der Objektdatenkommunikation über sein Medium. Zusätzlich zum Originalen, existiert ein alternativer Datenpfad der Kommunikation über eine andere Simulation. Die konzeptionelle Phase muss entscheiden, wie mit der originalen Übertragung umgegangen werden soll, immer mit dem Bewusstsein welche Effekte Änderungen im Simulationsablauf haben.

Empfänger

Der Empfänger ist die Gegenstelle der Datenkopplung und es liegt wie beim Sender Ortstransparenz vor, es muss folglich möglich sein den Empfänger überall im Code zu platzieren. Jeder Empfänger muss eindeutig identifizierbar und adressierbar sein. Diese Adresse muss mittels eines Mediums, welches in der Konzeption festgelegt wird, erreichbar sein. Die Übertragungen der Datenkopplung sind nicht eingegrenzt und werden konform zum Sender behandelt, sie müssen abschließend von ihrer Übertragungsform in ihre Nutzform übersetzt werden. Das Kernproblem der Datenübertragung liegt in dem Punkt wann die empfangenen Daten in die jeweilige Simulation integriert werden. Der Empfang von Nachrichten muss umgehend passieren, die Simulation selbst kann damit nicht beansprucht werden, um ihren Ablauf nicht zu verändern. Es muss eine abgespaltene asynchrone Ausführungseinheit dafür benutzt werden Nachrichten entgegen zu nehmen. Es ist nötig die eingehenden Daten in den Simulationsablauf zu integrieren. Aufrufe haben nur vom Simulationsscheduler zu erfolgen, damit dies gegeben ist, muss das Entgegennehmen der Daten mit in den zeitlichen Ablauf der Simulation eingeplant werden und kann nicht asynchron stattfinden. Gibt es eine bestehende Routine, die zur regelmäßigen Datenabfrage genutzt wird, ist sie für den Empfänger zu verwenden, andernfalls muss die Konzeption eine Lösung finden, wie dies zu behandeln ist. Das Einspeisen von Daten ist ein Problem, das nicht losgelöst von der Simulationszeit und der Synchronisation betrachtet werden kann, die Konzeption muss die Verbindung vom genutzten Synchronisationsalgorithmus

und dem Empfang in der Datenkopplung herstellen.
Der allgemeine Fall ist, dass Daten vom Sender kommen und vom Empfänger entgegen genommen werden, optional können sie in einem Buffer vorgehalten werden. Ein Sender ist bei diesem Gedankengang nicht zwingend nötig, ebenso ist es möglich einen synchronisierten, simulationsfremden Stimuli-Generator an den Empfänger senden zu lassen. In dieser Arbeit werden Simulationen miteinander gekoppelt, wobei die eine , Stimuli für die andere liefert, daher ist dieser Gedanke gar nicht so weit von den Zielen der Arbeit entfernt.

3.3. Zusammenfassung der Analyse

Es werden die Ergebnisse der Analyse zusammengefasst, so dass die Fragen aus der Eröffnung dieses Kapitels beantwortet und Stichpunkte zusammengetragen werden, an denen sich der Erfolgsgrad der Konzeption festmachen lässt.
Die Eingangsfrage ist: “Wie ist es möglich, Simulationen zu koppeln?“ Die abstrakte Betrachtung der Analyse zeigt, dass Simulationen zeitlich und auf Datenebene gekoppelt werden müssen. Der zeitliche Ablauf muss synchronisiert erfolgen, hierfür gibt es Denkansätze die ein konkretes Lösungskonzept erlauben. Die Kopplung der Simulationsinhalte, muss Orts- und Datenform- unabhängig sein, es herrscht diesbezüglich Transparenz. In der Konzeption muss das Problem des asynchronen Einspeisens von Datenübertragungen mit dem jeweiligen Ablauf der Simulation vereinbart werden. Werden alle aufgezeigten Charakteristika und Probleme gelöst, sind die Simulationen in der Lage einen gekoppelten Simulationslauf auszuführen, bei dem wählbare Datenkopplungen zwischen den Simulationen gebildet werden. Die Tabellen 3.1 und 3.2 listen die Eckpunkte der Analyse in zusammenfassender Form auf, unterteilt in Voraussetzungen und Aufgaben. Sie werden ein begleitender Bezugspunkt für die Konzeption, Implementierung und Evaluation sein.
Diese Analyse hat allgemeingültige Forderungen zu Problemlösung geliefert, in der folgenden Konzeption werden diese auf den vorliegenden Fall, übertragen und angewandt.

Nr	Voraussetzung
1	Die zu koppelnden Simulationen müssen von innen unterbrechbar sein.
2	Der Ablauf der Simulationen ist dynamisch oder statisch anpassbar.
3	Der Simulationsinhalt ist einsehbar und manipulierbar
4	Optional - Es müssen Informationen über den zeitlichen Verlauf der zu koppelnden Kommunikation herleitbar sein

Tabelle 3.1.: Voraussetzungen einer Simulationskopplung

Nr	Aufgabe
1	Festlegung der zu koppelnden Simulationen und deren Austauschpunkte
2	Festlegung des Mediums welches für alle Übertragungen zwischen den Simulationen genutzt wird
3	Festlegen eines Formats für eine gemeinsame Zeitbasis
4	Festlegung des Synchronisationsalgorithmus
5	Einbettung der Synchronisation in den Simulationsablauf
6	Beim Verwenden einer festen Synchronisationsauflösung muss diese angemessen gewählt werden
7	Festlegen wo Datenkommunikation zwischen den Simulationen nötig ist
8	Sicherstellen, dass Sender und Empfänger ortsunabhängig in die Simulationen einzubinden sind
9	Sicherstellen, dass Daten unabhängig von ihrem Format übertragbar sind
10	Sicherstellen, dass der Sender Ziele eindeutig adressieren kann
11	Sicherstellen, dass eine Datenkopplung keine Seiteneffekte zur Folge hat
12	Sicherstellen, dass der Empfänger Nachrichten immer entgegen nehmen kann und diese ggf. vorhält
13	Sicherstellen, dass der Empfänger eindeutig adressierbar ist
14	Sicherstellen, dass die asynchron eingehenden Nachrichten in Simulationsablauf integriert werden

Tabelle 3.2.: Aufgaben zur Realisierung einer Simulationskopplung

4. Konzeption

Dieses Kapitel stellt ein Konzept für den speziellen Fall der Kopplung von Omnet++ und VEOS auf. Das Konzept basiert auf den Ergebnissen der Analyse und den realen Gegebenheiten der verwendeten Simulationen. Nachdem die Analyse Wissen über den allgemeinen Fall geliefert hat, ist zunächst der spezielle Fall in Abschnitt 4.1 beschrieben. Im Anschluss in Abschnitt 4.2 ist AUTOSAR untersucht, um festzustellen welche Optionen für die Umsetzung der Analyse bestehen und ob die festgestellten Voraussetzungen erfüllt sind. Basierend auf Analyse und den vorherigen Abschnitten sind konzeptionelle Grundsatzentscheidungen in Abschnitt 4.3 behandelt, die bestimmend auf das Nachfolgende wirken. Die Konzeption der Implementierung ist unterteilt in die Hauptzweige der Analyse, Synchronisation und Datenübertragung. Abschließend in Abschnitt 4.8 ist das aufgestellte Konzept zusammengefasst dargestellt.

4.1. Charakteristiken der Simulationen

Die Ergebnisse der Analyse beziehen sich auf den allgemeinen Kopplungsfall. Im Konzept ist dies auf VEOS und Omnet++ übertragen, die stellvertretend für andere Simulationen stehen. Die Plattform VEOS simuliert ein Steuergeräteverbund in dem drei VECUs mit einem CAN-Bus verbunden sind. Die Funktion, die die Steuergeräte ausführen, ist die eines System zur Fahrtrichtungsanzeige in einem Automobil, siehe Abbildung 4.1. Die Netzwerksimulation Omnet++ beinhaltet einen virtuellen CAN-Bus, der für die Aufnahme der VECUs als Teilnehmer in Form von CAN-Knoten präpariert wird. Ziel der Kopplung ist es, dass die in Omnet++ arbeitende Netzwerksimulation die Datenübertragung zwischen den VECUs übernimmt, die original in VEOS stattfindet. Als Folge überträgt die Center ECU in Abbildung 4.1 ihre Kommunikation nicht auf dem virtuellen CAN-Bus in VEOS, sondern an die Netzwerksimulation Omnet++,

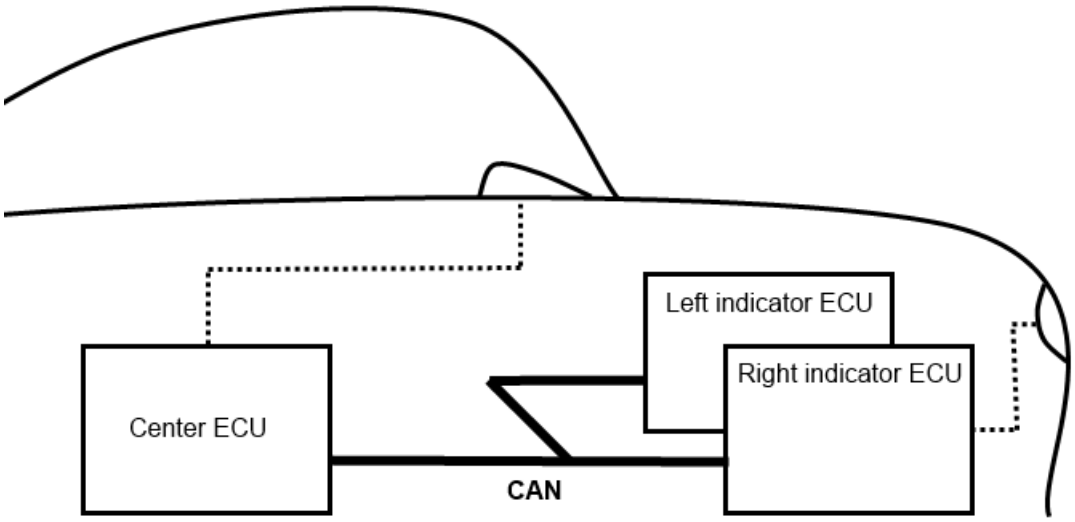

Abbildung 4.1.: VEOS Systemdarstellung im Automobilkontext

welche die Übertragungsstrecke simuliert, in diesem Fall aber nicht notwendigerweise einen CAN-Bus. Die Kopplung muss auch für andere simulierte Netzwerke funktionieren, zum Beispiel die von Realtime Ethernet. Sind die Übertragungen durch die Netzwerksimulation vorgenommen, müssen sie an die entsprechende VECU in VEOS zugestellt werden und der Kommunikationsbypass über Omnet++ ist abgeschlossen. Das System welches in VEOS simuliert wird, entspringt gedanklich der Realwelt, daher die Bezeichnung ECU in der Systemdarstellung Abb.4.1. Im Folgenden ist die ECU im Simulationskontext gesehen und als VECU bezeichnet. Ein Hauptpunkt der Arbeit, liegt im Anpassen der VEOS Simulation mit ihren AUTOSAR Inhalten. Omnet++ ist gesondert behandelt. AUTOSAR, die VECUs und VEOS sind im Folgenden untersucht um festzustellen, welche Möglichkeiten für das Konzept bestehen.

4.2. Konzeptionelle Eingriffsmöglichkeiten in eine AUTOSAR VECU

Dieser Abschnitt zeigt auf, welche Eingriffsmöglichkeiten in die AUTOSAR VECU bestehen um die Simulation koppelbar zu machen, denn die VEOS Simulationsumgebung selbst zu verändern, liegt außerhalb der Möglichkeiten. Veränderungen an VEOS sind über den gestaltbaren Simulationsinhalt der VECUs zu realisieren. Durch praktische und theoretische Untersuchungen konnten Zugriffsmöglichkeiten gefunden werden. Es ist möglich, zur Generierung der VECU, weitere ausführbare Codeeinheiten einzubringen und in den Ablaufplan aufzunehmen. Dies erfordert nur minimale Anpassungen an der VECU, zieht aber eine neue Generierung nach sich. Nach der Generierung stehen viele Teile der VECU als lesbare C-Dateien zur Verfügung. Mit Blick auf Abbildung 2.1, kann der Application Layer in Quellcode der VECU wiedergefunden werden. Die RTE steht als C-Datei zur Verfügung und mit ihr die Interfaces der Services und die zu übertragenden Signale. Des Weiteren sind große Teile des Service Layers wiederzufinden. Einblicke auf Codeebene bis hinunter zum Microcontroller Abstraction Layer, wurden nicht in geeigneter Form gefunden. Mit dem vollen Zugriff auf die RTE sind viele Optionen verfügbar, da es eine zentrale Rolle im System übernimmt und Einfluss auf alle angeschlossenen Komponenten erlaubt , siehe Abb.2.2.
Die Frage ist, ob diese Möglichkeiten des Zugriffs reichen, Tabelle 3.1 der Analyse listet notwendige Punkte die zur Bewertung genutzt sind:
1) Es ist möglich die eventbasierte VEOS Simulation von innen heraus zu unterbrechen.
2) Man kann statisch zur Erstellung der VECU mittels SystemDesk den Ablaufplan bestimmen.
3) Der Code der AUTOSAR Interna ist in einem Maß verfügbar, der eingeschränkte Manipulationen erlaubt.
4) In SystemDesk sind Daten zum Zeit- und Kommunikations-Verhalten der VECUs einsehbar.
Die festgestellten Voraussetzungen werden in einem hohen Grad erfüllt, so dass eine Kopplung vorgenommen werden kann. Omnet++ erfüllt ebenfalls diese Voraussetzungen in einem Maße, das eine gesonderte Darlegung unnötig macht. Die Voraus-

setzungen bilden einen Übergang zu den Aufgaben, die die Konzeption zu lösen hat, beginnend mit den Grundsatzentscheidungen.

4.3. Konzeptionelle Grundsatzentscheidungen

Der letzte Aspekt der zu behandeln ist, bevor die Konzepte zur Implementierung folgen, ist das Treffen der verbleibenden Grundsatzentscheidungen, die in der Analyse aufgezeigt wurden, siehe Tabelle 3.2 S.28. Die Punkte sind einzeln untersucht:
Punkt 1) Die Simulationen VEOS und Omnet++ sind festgelegt, es werden die für CAN bestimmten Signale der VECU über die Omnet++ Datenkopplung umgelenkt.
Punkt 2) Die Festlegung des allgemeinen Übertragungsmediums. Im Rahmen der Erprobung und Entwicklung der Arbeit befinden sich beide Simulationen auf dem selben Rechner, daher bieten sich alle Möglichkeiten der Interprozesskommunikation an, z.B. die der Pipes. Diese erfüllen die notwendigen Eigenschaften, aber es liegt eine Begrenzung auf den lokalen Rechner vor. In der Analyse zeigte sich, dass eine möglichst große Ortsunabhängigkeit, bzw. Ortstransparenz für die Simulationskopplung optimal ist. Ein erprobtes Mittel, Rechnergrenzen zu überwinden, ist das Local Area Network. LAN ist gut geeignet, da es über Lokalhost rechnerintern nutzbar ist und durch Adressanpassung auch externe Rechner erreichen kann. So ist eine Verwendung in weiterführenden Projekten gegeben ohne, dass Einschränkungen oder komplexer Änderungsbedarf anzutreffen ist. Mit Blick auf die Analyse muss festgestellt werden, ob LAN die notwendigen Kriterien erfüllt. Das LAN erlaubt intern und extern eine exakte Adressierung über die IP Adresse und den Port, ebenfalls können Daten ohne Einschränkungen übertragen werden. Damit sind die Punkte 9), 10), 13) der Tabelle 3.2 erfüllt.
Die nächste Grundsatzentscheidung ist rein konzeptioneller Natur und behandelt die Frage: Nach welchem Prinzip sollen die neuen Codeteile in die Simulation eingebracht werden? Ziel dieser Arbeit ist, Simulationen zu koppeln und nicht sie zu verändern! Als Folge dessen wird versucht, den neuen Quellcode und die Ergänzungen weitestmöglich vom Original und seiner Funktionalität getrennt zu halten. Für Code, der die VECU ergänzt, heißt dies, dass alle neuen Funktionalitäten in einer gesonderten C-Datei bzw. H-Datei gesammelt werden und nur die Aufrufe im originalen Quelltext ergänzt

werden. Hierbei ist darauf zu achten, dass der originale Ablauf unbeeinflusst bleibt und ob mögliche Seiteneffekte neu entstehen oder ausbleiben, Punkt 11).
Des Weiteren erlaubt das minimalistische Prinzip des Funktionsaufrufplatzierens eine große Ortsunabhängigkeit und geringe Codewiederholung bei den Kopplungsteilen.
In Omnet++ wird nach dem gleichen Prinzip verfahren, die Funktionalität ist in einzelne Knoten gekapselt, z.B. werden Knoten zur Datenkopplung und zur Synchronisation benötigt, sowie Knoten die als Gateway in der jeweiligen Simulation fungieren. Das Gateway ist eine Implementierung für den jeweiligen Simulationsfall, so dass es für jede Netzwerksimulation einer angepassten Neuentwicklung bedarf.
Ein weiterer Punkt der Konzeption ist die Frage: Wo die Datenkommunikation in der AUTOSAR VECU umgelenkt wird?
In der Analyse ist dies nicht von Belang, es ging um die Datenkommunikation in abstrakter Form. In diesem realen Fall sind die CAN-Nachrichten Ziel der Datenkopplung. Der erste mögliche Punkt, zum Umlenken der CAN-Nachrichten, ist auf dem virtuellen CAN-Bus innerhalb von VEOS zwischen den VECUs, siehe Abb.4.1. In der Erprobung ergab sich, dass SystemDesk hierzu Eingriffsmöglichkeiten unter dem Namen “Interventions“ anbietet, diese erlauben das Manipulieren und Ersetzen des Inhalts einer CAN-Nachricht. Diese Technik bringt zwei Probleme mit sich, zum einen bleibt die CAN-Nachricht unabhängig von ihrer Botschaft existent und wird in jedem Fall zugestellt. Zum anderen findet eine zwangsweise Dopplung der Nachricht statt, indem der reguläre Weg erhalten bleibt und gleichzeitig der Bypass der Nachricht arbeitet. Dieses Problem mag durch trickreiche Implementierung zu lösen sein. Das entscheidende Problem tritt beim Empfänger auf, denn einen Intervention Point für den Empfang gibt es nicht. Ebenso bleibt die Frage, wie man mit der Nachrichtendopplung umgehen soll und den möglichen Seiteneffekten. Diese Stelle erwies sich für den Bypass als problembehaftet. Der nächste Ansatzpunkt ist im Datenpfad vor dem virtuellen CAN-Bus anzunehmen. Somit ist dieser Punkt vorher im Bereich AUTOSAR Communication zu suchen, der sich über drei Layer erstreckt (Überblick Abb.2.1 S.6). Hier muss weiter ins Detail gegangen werden, um zu zeigen welche Möglichkeiten und Einschränkungen einzelne Punkte für die Datenkopplung bieten, bis eine Lösung für die Konzeption vorliegt. Auf Abbildung A.1 Seite 73 kann man den Datenpfad zum CAN-Bus gut verfolgen. Ein Bypass unter Veränderung des CAN-Treibers ist keine

Option, da diese Lösung, Zugriff auf dessen Quellcode verlangt und sehr speziell auf den CAN-Controller zugeschnitten ist (Roter Layer unten “Communication Drivers“). Der übergeordnete Communication HW Abstraction Layer ist da die bessere Option, da über ihn die Benutzung des Treibers standardisiert abläuft. Diese Lösung ist ausreichend, es würde nur für das Medium CAN funktionieren und müsste für z.B. Ethernet wieder speziell eingefügt werden. Es zeigt sich, dass mit steigendem Layer, die Verallgemeinerung und die Zugänglichkeit ansteigt. Es folgt der PDU Router, wo anhand der Routingtabelle in die Übertragungsmedien unterschieden wird. Dieser liegt im Service Layer, grundsätzlich ist dieser eine Option mit großer Abdeckung, aber voreilig gewählt in dem Sinne, dass er die aufbereiteten I-PDU vom AUTOSAR COM nutzt, dem Signal Gateway. Warum nicht diesen für maximale Abdeckung des Bypass verwenden? Bestimmend hierfür sind die gegebenen Zugriffsmöglichkeiten auf die AUTOSAR VECU, die im vorherigen Abschnitt benannt wurden. Es bleibt den Bypass direkt am Übergabepunkt der RTE an den Service Layer (AUTOSAR COM) zu platzieren. Dies ist die einfachste, vielfältigste und sicherste Methode. Es ist möglich jedes Signal unabhängig vom Medium zu behandeln und unbekannte Seiteneffekte, die die tieferen Programmschichten mit sich bringen zu vermeiden. Der Betrachtungsweg rückwärts durch die Layer ausgehend vom CAN-Bus wurde gewählt, um die Möglichkeiten des Kopplungsorts aufzuzeigen. Der Punkt für die Datenkopplung ist somit in der RTE auf der Signalebene und bietet die Möglichkeit alle Signale selektiv in die Datenkopplung einzubeziehen.

Im Fehlerfall und für Rückmeldungen vom Betrieb, sind Meldungen auf der jeweiligen Simulationskonsole auszugeben. Bei Fehlern sollen Ort und Art des Fehlers klar auszumachen sein, falls vorhanden ist der Systemfehlercode mitzuliefern.

4.4. Konzeption der Synchronisation

In der Analyse Abs.3.1.1 werden zwei grundsätzliche Verfahren betrachtet, sequenzielle und parallele, diese sind im Folgenden in ein Konzept umgesetzt. Beide Verfahren brauchen einen Initiator, im vorliegenden Fall ist es die VEOS Simulation, da die VECU mit ihrem festen Ablaufplan bestimmend für die möglichen Synchronisationsabläufe im System ist. Omnet++ kann sich optimal an VEOS mit seinem flexiblen Ablauf aus-

richten lassen. Das Konzept soll eine Kopplung liefern, die weitestmöglich unabhängig von den Simulationen ist, die einzige Forderung besteht darin, dass der Synchronisationsintervall in den Ablaufplan mindestens fest, besser flexibel aufgenommen werden kann. Ein Entwurf für die Implementierung ist formuliert und die Arbeitsweise der Synchronisationsalgorithmen dargelegt. VEOS hat immer die Führungsrolle beim Simulationsstart inne. Auf Seite von Omnet++ ist die Implementierung in das Klassengerüst eines Knoten eingebettet, um das Konfigurieren von außen zu erleichtern. Für den Entwurf ist der VEOS-Anteil hervorgehoben, da der Schwierigkeitsgrad und die Art der Anpassung dies notwendig machen.

Entwurf

Dieser Entwurf beinhaltet die Konzepte der Implementierung von zwei Synchronisationsfunktionen, die den sequentiellen und parallelen Gedanken aus der Analyse umsetzen. Die nötigen Initialisierungen sind übergreifend mit der Datenübertragung in einem gesonderten Abschnitt 4.6 behandelt. Die Festlegung der Synchronisationsauflösung erfolgt durch die Betrachtung der Simulationsgegebenheiten. Omnet++ arbeitet mit seiner Netzwerksimulation reaktiv zu den von VEOS generierten Übertragungen. Omnet++ hat folglich keine ableitbaren Forderungen an die zeitliche Auflösungen, da kein aktiver Nachrichtengenerator außer VEOS existiert. VEOS Nachrichtenaktivität ist feststellbar aus der Konfiguration der VECUs. Alle zehn Millisekunden werden Meldungen über den virtuellen CAN-Bus von der Center VECU zu den beiden indicator VECUs übertragen (Systemübersicht Abb.4.1 S.30). Die Auflösung der Synchronisation muss höher sein. Es wurde eine Millisekunde festgelegt, wobei die Auflösung höher gewählt werden kann. Sie ist begrenzt durch die Auflösung des jeweiligen Ablaufs z.B. der VECU und dem zu akzeptierenden, real zeitlichen Mehraufwand einer höheren Synchronisationsdichte.
Wie in der Analyse festgestellt, muss die Synchronisationsroutine in den Ablauf eingebunden werden, hier im Intervall von einer Millisekunde. Abschnitt 4.4 behandelt das detailliert. Ein alternativer Ansatz umgeht die feste Simulationsauflösung und platziert die Synchronisation nach jedem Sende-Event, bzw. vor jedem Empfangs-Event. Die Zeit wird in der VEOS Simulation in seiner Repräsentation float64 festgehalten und

in Omnet++ in double. Im Rahmen der Synchronisation wird diese Zeit zwischen den Simulationen ausgetauscht. Das Austauschformat muss den Gengegebenheiten entsprechen, die Übergabe als 64 Bit Fließkommazahl bietet sich an. Es wird aber eine Alternative genutzt, die Übergabe als 64 Bit Integer mit der Basiseinheit Mikrosekunden. Diese Konzeptentscheidung erfüllt weiterhin die für die Synchronisation notwendigen Eckpunkte mit ganzzahligen Werten, diese Entscheidung ist optional. Das Nassi Schneidermann Diagramm 4.2 bildet den prinzipiellen Ablauf der Synchronisationsroutine ab. Die Implementierung der Synchronisation ist abhängig vom Typ und der Simulation. Die im Folgenden dargestellten Algorithmen arbeiten für VEOS und Omnet++ übergreifend, so dass die Synchronisation der Kopplung ersichtlich wird.

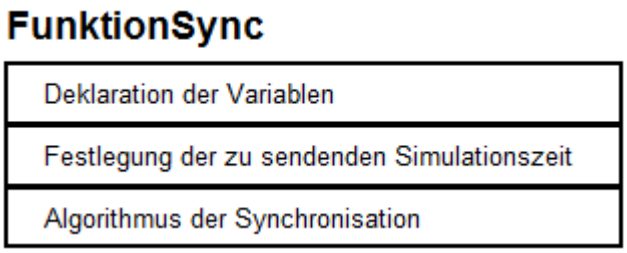

Abbildung 4.2.: Diagramm zur Struktur der Synchronisationsfunktion

Arbeitsweise und Algorithmus (sequentiell)

Dieser Abschnitt stellt das Konzept der sequentiellen Synchronisation dar. In der Analyse ist der Grundgedanke aufgezeigt. Dieser ist in ein vollständiges Konzept weiterentwickelt worden. Hierzu ist das Sequenzdiagramm 3.2 der Analyse erweitert und auf den Detailgrad eines Konzepts komplettiert worden, siehe Abb.4.3 s.45. Es sind im Folgenden die Punkte behandelt, die charakterisieren und die indirekt Teil des Diagramms sind aber der Klärung bedürfen.
Die Startreihenfolge ist nicht willkürlich. Omnet++ beginnt nicht den gekoppelten Simulationslauf und ist zuerst zu starten. Omnet++ wartet zu Beginn bei Simulationszeit bei t(0) auf ein Zeitsignal, bzw. Startsignal von VEOS. VEOS wartet nicht und führt die

Simulation bis zum ersten Synchronisationspunkt t(1) aus.
Bei t(1) sendet VEOS die Synchronisationsmeldung mit der Simulationszeit an Omnet++, dies erfolgt über ein Socket. Die Zieladresse ist hier Lokalhost und die Port-Adresse von der Omnet++ Synchronisation. Nach dem Senden wartet VEOS auf eine Bestätigung von Omnet++, welches zum Synchronisationspunkt t(1) aufholt und die Bestätigung im Anschluss absetzt. Durch den gegenseitigen Wechsel entsteht ein sequentieller Ablauf. Die Frage, wie die Simulationen dazu gebracht werden passend zum Synchronisationspunkt die Synchronisationsroutine aufzurufen, ist in der Implementierung individuell für Omnet++ und VEOS beantwortet. Dieser Konzeptteil lässt offen, wo und wann im Programmcode synchronisiert wird, in der VECU kann die Synchronisation frei platziert werden, ob in einem getriggerten Task oder im Anschluss an einen Sendeaufruf. Die Synchronisation ist transparent in Bezug auf ihren Einsatzort.
Die Laufzeit zwischen den Sockets ist bei diesem Verfahren unerheblich, das sequenzielle wechselseitige Warten stellt sicher, dass Realzeit keinen Einfluss auf das Verfahren haben kann. Die parallele Synchronisation, mit der zu erprobenden Nebenläufigkeit der Simulationen, verhält sich hingegen anders.

Arbeitsweise und Algorithmus (parallel)

Die Grundfunktionsweise ähnelt der sequentiellen Synchronisation aus dem vorherigen Abschnitt. Das Konzept basiert auf dem Diagramm 3.3 der Analyse, welches eine mögliche parallele Synchronisation darstellt. Das Sequenzdiagramm 4.4 S. 46 ist das darauf basierende Konzept. Wie bei der vorherigen Variante werden Sockets verwendet und VEOS ist verantwortlich für den Simulationsstart. Ab diesem Punkt unterscheiden sich die Abläufe. Beim Start sendet VEOS sofort das Zeitsignal zur Freigabe des Simulationslaufs an Omnet++. Danach beginnt VEOS mit dem Voranschreiten seiner Simulation bis t(1). Bei einer beispielhaft angenommenen Nullzeit Übertragung, beginnt die Omnet++ Simulation zeitgleich mit der Simulation bis t(1). Die angenommene Nullzeit existiert nicht, im Falle einer bekannten Übertragungszeit ist es möglich sie durch einseitiges Warten auszugleichen.
Die Simulationszeit auf beiden Seiten der Kopplung ist zwischen den Synchronisati-

onspunkten asynchron, der hierbei entstehende Simulationsdrift hängt unter anderem von der jeweiligen Berechnungszeit des Simulationsschrittes ab. Ist eine Simulation bis zum nächsten Synchronisationspunkt vorangeschritten, wird der Gegenpart mit einer Zeitmeldung verständigt und es wird gewartet, dass eine gleichartige Meldung vom Gegenpart empfangen wird. Erwartet ist ein Verhalten wie im Diagramm 4.4, wo Parallelität möglich ist, aber es zu einem zeitlichen Drift zwischen den Simulationsfortschritten kommt. Die Simulation auf einer Seite wird erst dann fortgesetzt, wenn das eigene Signal gesendet ist und das von der Gegenstelle empfangen wurde. An dieser Stelle kann der maximale zeitliche Drift der Daten, bei dieser Synchronisationsvariante, vorkommen. Der Abstand der Synchronisationspunkte ist so zu wählen, dass die gewollte Genauigkeit bzw. der maximale Drift immer eingehalten wird, für den Prototypen ist es 1ms.
Abschließend ist das “worst case scenario“ behandelt, als irreales Beispiel: Omnet++ braucht viel mehr Realzeit zum Simulieren als VEOS und in Ausnahmefällen ist Omnet++ deutlich schneller. VEOS wartet folglich i.d.R. lange auf Omnet++ bis die Synchronisationsmeldung eintrifft und hat seinerseits die Synchronisationsmeldung schon abgesetzt. Omnet++ erreicht den Synchronisationspunkt und kann ohne Wartezeit mit der Simulation nach Senden seiner Sync.-Meldung fortfahren, da die Synchronisationsmeldung von VEOS bereits vorliegt. Aus internen Gründen geschieht die weitere Omnet++ Simulation viel schneller als bei VEOS, so dass Omnet++ am nächsten Synchronistionspunkt warten muss. So kann sich der Drift durch zeitlichen Führungswechsel maximieren, abhängig von Übertragungs- oder Berechnungs- Zeit. Ist der zeitliche Ablauf konstant ungleich, stellt sich ein Verhalten ähnlich der sequentiellen Synchronisation ein, mit einem geringen parallelisierten Anteil. Der Drift ist ein Störfaktor für die zeitliche Exaktheit der Datenkopplung, deren Arbeit von der Qualität der Synchronisation abhängt.

Platzierung und Umgebungsvorbereitung

Um die Synchronisationsroutine in Betrieb zu nehmen, muss sie in den jeweiligen Ablauf eingebunden sein. Dies zu erreichen ist individuell für jede Simulation vorzunehmen und ist eine Grundvoraussetzung. Es muss eine Stelle im Programmcode

erzeugt oder gefunden werden, die periodisch fest oder dynamisch angesteuert wird. Für VEOS heißt das, dass im Ablaufplan einer beliebigen VECU die Synchronisation periodisch zeit- oder bedarfs-orientiert aufgerufen wird, je nach Synchronisationsprinzip. Die Anpassung des VECU Ablaufplans für die Taskausführung erfolgt vor der Generierung, es werden keine Ausführungszeitfenster festgelegt, sondern der ideale Intervall incl. einer Priorisierung der Tasks. Mit dem Werkzeug SystemDesk kann ein neuer Millisekunden-Task erstellt werden, an den ein Runable (standardisierte C-Funktion) gebunden ist. Nach der Generierung ist der Ort zum Einbinden der Routine existent und die Synchronisation kann als Funktionsaufruf in das Runable eingebracht werden oder dessen Funktionsaufruf ersetzen. Der Codeaufruf liegt in der RTE, das Runable selbst liegt in einer gesonderten Datei. Die Umsetzung dieses Konzeptteils wird im Abschnitt Anpassung bestehender Systeme 5.1 erläutert und erklärt die Implementierungsdetails.
In Omnet++ können Aufrufe durch Nachrichten (Messages) in die Wege geleitet werden. Diese werden von dem Synchronisationsknoten erzeugt und als Selbstnachricht in den Ablaufplan der Simulation eingebracht. Da die Selbstnachrichten immer neu generiert werden, kann die Synchronisation sich dynamisch den Gegebenheiten anpassen. Als Ergänzung für die Datenübertragung wird eine Erweiterung in Omnet++ an dem Sync-Knoten benötigt, jedes Mal wenn ein Synchronisationszyklus vollzogen wird, sendet der Knoten eine Meldung aus seinem Gate, die als Tickgenerator genutzt wird. Im Abschnitt zur Datenkopplung wird gezeigt, warum dies in Omnet++ erforderlich ist und in VEOS nicht.

4.5. Konzeption der Datenkopplung

Basis der Konzeption der Datenkopplung sind die Ergebnisse der Analyse3.2, die Charakteristiken der Simulationen4.1 und die Festlegung der Grundsatzentscheidungen4.3. Eine erste Idee die sich aus den AUTOSAR Grundlagen ergibt, ist die Möglichkeit die Routingtabelle des PDU Routers auszutauschen um zu erreichen, dass die Signale statt auf CAN, auf Ethernet gesendet werden. Die Folge wäre eine direkte Einspeisung der Daten einer realen ECU in Omnet++. Als Subjekt der Arbeit liegt eine VECU vor, diese legt die Grenzen des Datenaustausches auf die Simulationsumgebung VEOS fest. Es

wird ein eigens konzipiertes Modul gebraucht, das Daten über die Simulationsgrenzen hinaus transportiert. Es ist festgelegt, dass das Übertragungsmedium LAN ist. Wie bei der Synchronisation werden Sockets verwendet. Die zu koppelnden Daten sind die Nachrichten welche über CAN zwischen den VECUs ausgetauscht werden. Hierfür wird auf Signalebene die Datenkopplung in Form eines Bypass vorgenommen, die Bypass-Endpunkte sind der Sender und Empfänger. Omnet++ setzt die Signale der VECUs auf einen CAN-Bus um, der ähnlich dem originalen virtuellen CAN-Bus in VEOS spezifiziert ist. Startpunkt des Bypass zur Simulationskopplung ist der Sender.

Sender

Der Sender ist der Startpunkt einer unidirektionalen Übertragung. Es wird ein Socket genutzt, welches den Inhalt eines Speicherbereichs an eine Zieladresse übermittelt. Daraus resultiert, dass alle Datentypen gleichwertig als Bytefolge übertragen werden. Damit ist die Forderung nach Unabhängigkeit vom zu übertragenden Datentyp erfüllt. Der Empfänger kennt aus seinem Umgebungskontext den zu erwarteten Datentyp und eine Rückkonvertierung in den originalen Typ ist möglich, bzw. ein Puffer wird befüllt und kann beliebig interpretiert werden. Verwendet wird der Sender als frei platzierbarer Funktionsaufruf, der die Übertragung über den Socket gewährleistet. Die Details sind in dem entsprechenden Abschnitt 5.4 der Implementierung abgehandelt. Eine Erweiterung des Senders ist die Möglichkeit zur Signalbündelung. Der Sender kann Übertragungen puffern und als Sammlung von mehreren Übertragungen absetzen, dies kann nach Füllstand oder Zeitplan geschehen. Wenn Signalbündelung ein notwendiger Wunsch im Ablauf ist, dann muss es Teil der Netzwerksimulation Omnet++ sein und nicht Teil der Datenkopplung. So bleibt die Ordnung innerhalb der Kopplung unbeeinflusst und Untersuchungen zur Bündelung können in der Netzwerksimulation stattfinden.
Für die Sender von Omnet++ zu VEOS gilt, dass sie als eigenständige Knoten gekapselt sind. Sie verfügen über ein Eingangs-Gate für die Omnet++ Nachrichten und senden den Inhalt der Message an ihr Sendeziel. Dieses soll frei in der Konfigurationsdatei definierbar sein. Siehe Implementierungsabschnitt 5.5.

Empfänger

Der Empfänger ist der Abschlusspunkt einer unidirektionalen Verbindung, die in der Datenkopplung zum Einsatz kommt. Daten werden als Menge von Bytes empfangen und ggf. in ihr ursprüngliches Format konvertiert. In der Analyse ist festgestellt, dass empfangene Daten nicht asynchron in die Simulation gelangen können. Dieses Problem wird mit einem Puffer und einem Intervall zum Abruf der Daten von der Simulationsseite gelöst. Bei VEOS und Omnet++ ist hierbei unterschiedlich zu verfahren. Auf Abb.4.1 S.30 nimmt die Right indicator ECU die an sie gerichteten CAN-Botschaften VEOS intern entgegen. Dies ist nach der Datenkopplung nicht mehr der Fall, dennoch existiert die Routine zum zyklischen Abfragen des CAN-Busses weiter. Es kann das Abfragen des originalen CAN-Bus entfernt und durch den neuen Empfängeraufruf ersetzt werden. Im Ergebnis wird der Empfänger zyklisch, wie der originale CAN-Bus, abgefragt. Der Ort dieses Eingriffs zur Datenkopplung liegt in der RTE der jeweiligen VECU. Seiteneffekte werden durch das Ersetzen des "*empfangen Aufrufs*" vom originalen CAN, durch den eigenen Empfänger, ausgeschlossen. Der Aufruf des Communication Services und der folgenden angegliederten Elemente ist durch das Aufrufen des neuen Empfänger ersetzt worden. In Omnet++ muss ein derartiger Mechanismus zum Abrufen geschaffen werden. Die Simulation muss dazu gebracht werden im passenden Zyklus den Empfänger auf neue Daten abzufragen. Hierzu wird der Synchronisationsknoten von Omnet++ einbezogen (Abs.4.4). Im Konzept ist festgelegt, dass bei jedem Synchronisationszyklus eine Message gesendet wird. Diese Message wird als Tick für das Überprüfen des Empfängers genutzt, so dass die Daten von der Simulation eingelesen werden können. Der Empfänger in Omnet++ muss ein Input-Gate für diese Tick-Messages haben und ein Output-Gate zum Versenden der empfangenen Signale an die CAN-Simulation.

Entwurf

Zur Realisierung des Empfängers in der geplanten Form wird ein Socket, ein Port für den Empfang, Speicherplatz für den Puffer und ein Thread benötigt. Thread, Puffer und Socket bilden die Empfangseinheit, in der der Thread Übertragungen entgegennimmt und im Puffer archiviert. Die Simulation kann jederzeit auf diesen Puffer zugreifen

und Daten entnehmen. Dieser Zugriff muss Threadsafe erfolgen. Ist der Pufferplatz erschöpft werden alte Daten verworfen. In der Implementierung 5.4 wird dieses Konzept detailliert umgesetzt.

4.6. Initialisierung

Die Initialisierung ist ein vorbereitender Schritt, der alle Teile der Kopplung in gleicher Form betrifft. Alle verwendeten Ressourcen müssen bei Start der Simulationen verfügbar gemacht werden. Dies sind zum Großteil Sockets, Threads, Mutexe und Umgebungsvariablen. Hierzu ist speziell für VEOS eine Initialisierungsfunktion zu schreiben und in der Startsequenz der VECU zu platzieren. Diese ist in der RTE zu finden. Bei Omnet++ finden die Initialisierungen im Konstruktor der Klassen statt, dieser nutzt eine Konfigurationsdatei zur Festlegung vieler Variablen. Dadurch gewinnt Omnet++ sehr an Übersicht, sobald die Systeme größer werden.

4.7. Konzeption für Omnet++

Die Simulation in Omnet++ ist näher zu behandeln. Es werden die zu entwickelnden Knoten im Verbund dargestellt und konzipiert. Ebenso wird erläutert, wie diese Knoten ihren Weg in beliebige Omnet++ Simulationen finden können. Die Abbildung 4.5 S.47 zeigt die drei neu entwickelten Knoten, bezeichnet als sync, omnetIn, omnetOut und zwei mit Gateway-Funktionalität angepasste CAN-Knoten, sowie deren Anbindung zum virtuellen CAN-Bus. Die behandelten Teile sind von einer roten Kurve umfasst, die anderen Teilnehmer können den CAN-Bus parallel benutzen, sind aber nur exemplarisch vorhanden und nicht direkt Teil dieser Arbeit. Der Synchronisationsknoten entspricht der Beschreibung aus Abschnitt 4.4 zur Konzeption der Synchronisation. Die Abbildung 4.5 zeigt den Empfänger als omnetIn mit seiner Verbindung zum Knoten, der die Synchronisation beinhaltet(sync_veos). Auf dieser Verbindung wird er über einen neuen Synchronisationszyklus informiert, so dass der Eingangspuffer überprüft wird. Damit die Anzahl der Empfängerknoten skalierbar bleibt, ist die Breite des Ausgangs-Gates des Sync.-Knotens dynamisch über die Konfigurationsdatei festlegbar. Sender- und Empfänger- Knoten entsprechen den zuvor festgelegten Konzepten. Die CAN-

Simulation besteht aus einem Beispielprogramm, basierend auf FiCo4OMNeT(Fieldbus Communication for OMNeT++) (siehe http://core4inet.realmv6.org/trac/) [12]. Das Beispielprogramm ist ein CAN-Netzwerk, je ein Teilnehmer ist durch einen Knoten repräsentiert, der mit dem CAN-BUS Knoten verbunden ist. Diese können senden und empfangen in dem Umfang, wie es ein realer CAN-Teilnehmer könnte. Konzeptidee ist, einen CAN-Knoten für Teilnehmer so zu modifizieren, dass er mit Empfänger bzw. Sender der Kopplung verbunden werden kann. Nach der Modifikation sind die Knoten ein konfigurierbares CAN-Gateway für die Kopplung, das Signale in CAN-Frames verpackt, bzw. entpackt und je nach Richtung im Rahmen der Kopplung weiterreicht. Bei Betrachtung der CAN-Knoten ist festzustellen, dass sie sich aus mehreren Komponenten zusammensetzten. Für die Modifikation sind die Unterkomponenten für den Sende- und Empfangsteil von Bedeutung, diese sind unter den Klassennamen *FiCo4OMNeT_CanTrafficSourceAppBase* und *FiCo4OMNeT_CanTrafficSinkAppBase* zu finden[10] [11].
Im allgemeinen Fall, muss für jede individuelle Omnet++ Simulationen ein eigens angepasster Gateway-Knoten erstellt werden, der die Elemente der Kopplung mit denen der jeweiligen Simulation verbindet. Alle anderen Teile funktionieren unabhängig von der umgebenden Simulation, so dass sie ohne Anpassungen in beliebige Omnet++ Simulationen übernommen werden können. In der Implementation 5.5 sind die Details zur Umsetzung dieses Konzeptgedankens behandelt.

4.8. Zusammenfassung des Konzepts

Die Analyse zeigt die beiden Hauptgruppen auf: Synchronisation und Datenübertragung. Im Rahmen der Konzeption sind sie in einzelne Arbeitsbereiche mit Umsetzungskonzepten untergliedert worden. Diese sind auf Tabelle 4.1 gelistet. Mit Abschluss dieses Kapitels steht ein Konzept zur Verfügung, welches alle Forderungen der Analyse erfüllt und implementierbar ist. Viele Konzeptansätze können für andere Simulationen und Simulationsplattformen adaptiert werden, für diese Studie ist die beschriebene exemplarische Situation umgesetzt.

NR	Typ	Aufgabe nach Konzeptentwurf
1	VEOS	Erweiterung einer VECU um einen neuen Millisekunden-Task
2	VEOS	Synchronisation implementieren und platzieren
3	VEOS	Senderoutine implementieren und platzieren
4	VEOS	Empfangsroutine implementieren und platzieren
5	VEOS	Initialisierung aller neuen Ressourcen
6	Omnet++	Synchronisationsknoten analog zu VEOS implementieren
7	Omnet++	Empfangsknoten implementieren
8	Omnet++	Sendeknoten implementieren
9	Omnet++	Modifizieren eines CAN-Knoten auf Gateway-Funktionalität
10	Omnet++	Verbinden aller Teile zu einem Simulationsnetz

Tabelle 4.1.: Konzeptionelle Arbeitsschritte

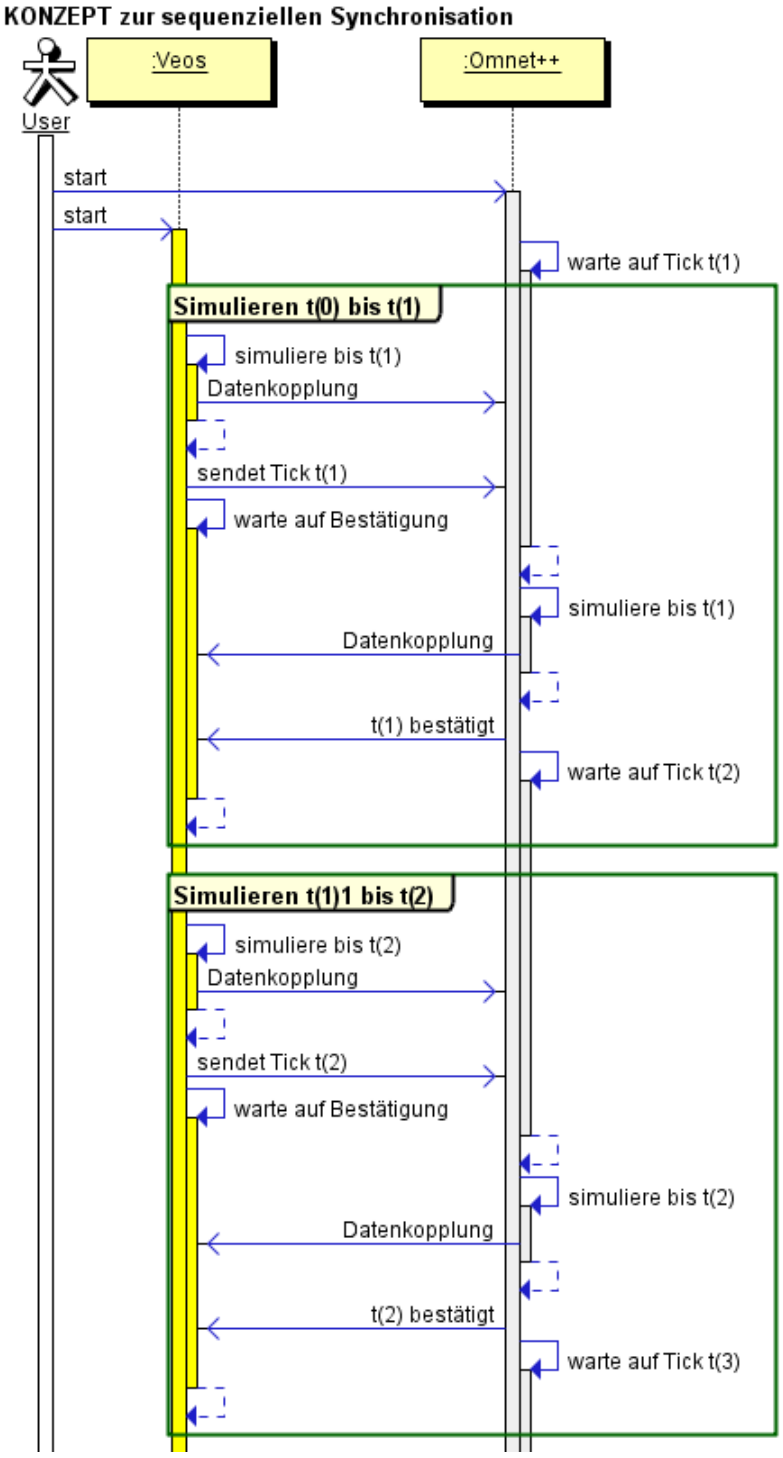

Abbildung 4.3.: Darstellung der sequentiellen Synchronisation

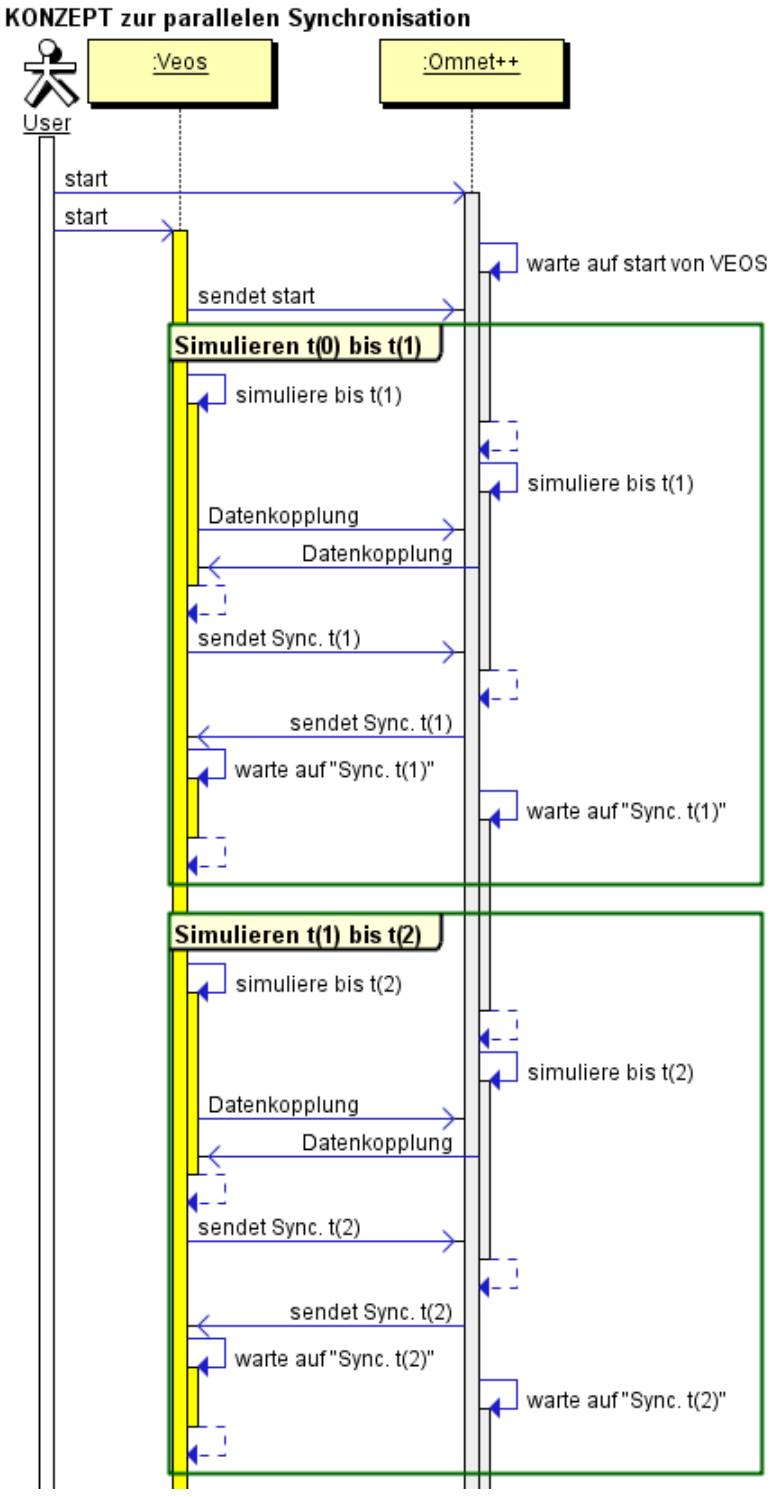

Abbildung 4.4.: Darstellung der parallelen Synchronisation

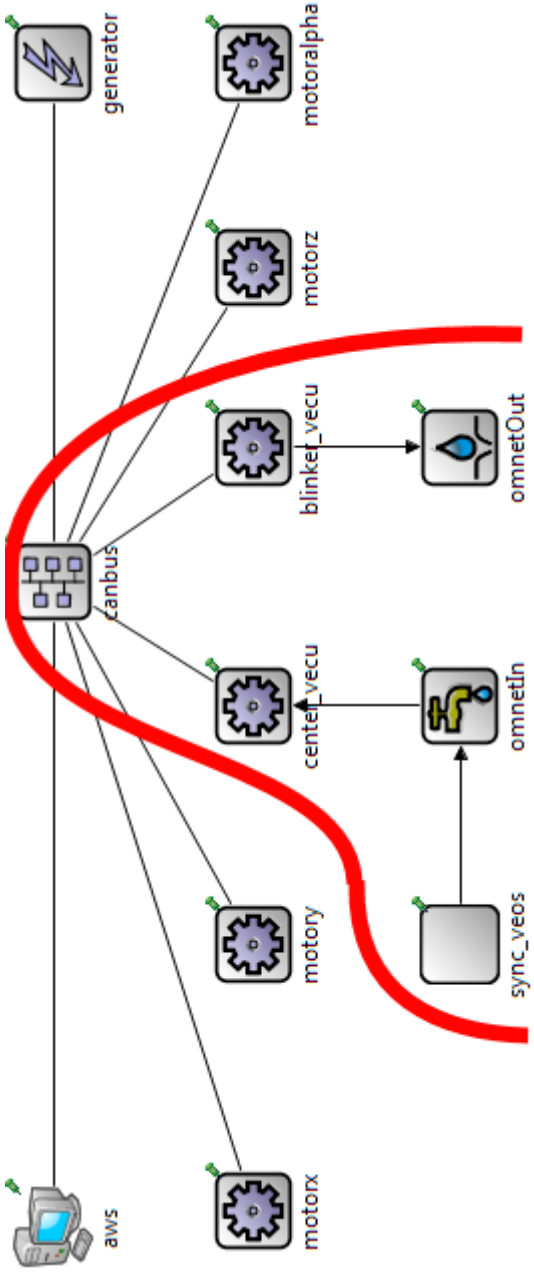

Abbildung 4.5.: Übersicht Omnet++ und der zur Kopplung wichtigen Module (in Rot abgetrennt)

5. Umsetzung und Implementierung

Ziel ist die Darstellung des Weges zum lauffähigen Prototypen, der das erworbene Wissen und die getroffenen Entscheidungen dieser Untersuchung, aus der Praxis heraus, bewertbar macht. Alle Teilaufgaben des Konzepts aus Tabelle 4.1 S.44 sind umgesetzt. Die Abarbeitung der Konzeptpunkte ist funktionell gegliedert. Es sind die in Abschnitt 2.4 vorgestellten Arbeitsmittel genutzt, sowie ergänzend Notepad++ und Eclipse zur Bearbeitung der generierten Code-Dateien der VECUs. Dieser Abschnitt ist keine Dokumentation des Programmcodes, das Erreichte soll in verständlicher und anschaulicher Form dargestellt sein. Hierfür finden Nassi-Shneiderman-Diagramme Anwendung, um Anschaulichkeit und einen Grad der Abstraktion zu bewahren. Vor der Implementierung liegt das Präparieren der VECUs für VEOS.

5.1. Anpassung bestehender Systeme

Damit die VEOS Simulation mit der Synchronisationsroutine, die eine feste Auflösung nutzt, arbeiten kann, muss eine VECU mit geeignetem Task vorbereitet werden. Dies geschieht analog zum ersten konzeptionellen Arbeitsschritt von Tabelle 4.1: “Erweiterung einer VECU um einen neuen Millisekunden-Task“. Um dies zu erreichen wird die Simulation als Projekt in SystemDesk editiert. Eine ECU ist auszuwählen, in diesem Fall die Center ECU (Systemübersicht Abb.4.1 S.30), es werden die Task & Runables konfiguriert. Ein neuer Task wird angelegt, dieser erhält den Namen SyncTask , der bei einer hohen Priorität im 1ms Intervall aufgerufen werden soll (Abb. 5.1). Parallel ist ein Runable Sync anzulegen, also ein ausführbarer Codeblock. Abschließend sind beide zu verknüpfen, so dass der SyncTask das Runable Sync regelmäßig aufruft, Abb.5.2 zeigt die fertige Anpassung. Der Abbildung ist ebenfalls zu entnehmen, dass “Task_10ms“ mit seinen Runables einen Intervall von 10ms hat, dieser ist unter anderem verant-

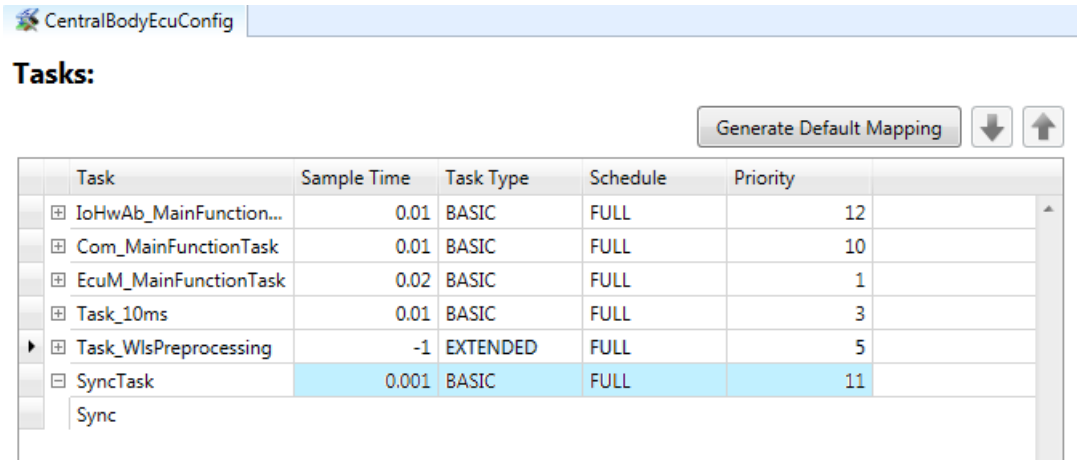

Abbildung 5.1.: Übersicht der Task in der CentralBodyEcu, ergänzend siehe Abb.5.2

wortlich für das Senden auf dem CAN-Bus. Die Information des Sende- und Empfangs-Intervalls ist wichtig, als Eckpunkt zur Festlegung der Synchronisationsauflösung. In anderen Konfigurationsfenstern ist das Festlegen aller Elemente und Metriken des Gesamtsystems möglich. Es können Informationen über die Konfiguration des originalen CAN-Busses eingesehen werden, so dass diese zur besseren Realitätsannäherung in der CAN-Simulation von Omnet++ genutzt werden. In Abschnitt 5.5 ist behandelt, wie die Konfigurationsdaten in Omnet++ genutzt werden. Die Veränderungen an der ECU sind minimal zu halten, so dass der neue Task die einzige Änderung in dieser Phase ist. Sind die Einstellungen der ECUs festgelegt, können sie generiert werden. Mit der Generierung sind sie noch nicht als VECU System in VEOS verwendbar. Dennoch existieren schon die relevanten Dateien in denen, im weiteren Verlauf, die Anpassungen zur Synchronisation vorgenommen werden. Abschließend nach allen Anpassungen, können die VECUs für VEOS kompiliert werden. Zuvor sind Änderungen entsprechend der vom Konzept festgelegten Eingriffe vorzunehmen.

5.2. VECU File Beschreibung

Im Konzept ist auf Basis eines geeigneten Anpassungspunktes und Erreichbarkeit für die Implementierung festgelegt, wo Eingriffe erfolgen. Die Erreichbarkeit des

Runnables:

All | Unmapped | Of Task_WlsPreprocessing

SWC Name

Runnable	Symbol Name	Trigger Events	Sample Time	Task Name
SWC Name: ComSwc				
Com_MainFunction...	Com_MainFunction...		-1	Com_MainFunctio
Com_MainFunction...	Com_MainFunction...		-1	Com_MainFunctio
SWC Name: EcuMSwc				
EcuM_MainFunction	EcuM_MainFunction		-1	EcuM_MainFunctic
SWC Name: IndicatorLogic				
TssPreprocessing	TssPreprocessing	TssCyclic10ms	0.01	Task_10ms
Logic	Logic	LogicCyclic10ms	0.01	Task_10ms
Toggle	Toggle	ToggleCyclic10ms	0.01	Task_10ms
WlsPreprocessing	WlsPreprocessing	WlsReceivedEvent	-1	Task_WlsPreproces
Sync	Sync	TimingEvent_for_S...	0.001	SyncTask
SWC Name: IoHwAbSwc				
IoHwAb_MainFunc...	IoHwAb_MainFunc...		-1	IoHwAb_MainFunc
TurnSwitchSensor_i...	IoHwAb_TurnSwitc...	Ev_TurnSwitchSens...	-1	
SWC Name: TurnSwitchSensor				
TssRunnable	TssRunnable	TssCyclic20ms	0.02	Task_10ms
SWC Name: WarnLightsSensor				
WlsRunnable	WlsRunnable	WlsReceivedEvent	-1	Task_WlsPreproces

Abbildung 5.2.: Zuordnung der Runables der CentralBodyEcu, ergänzend siehe Abb.5.1

Quellcodes ist bestimmend für die Lösungsmöglichkeiten und Arbeitsbereiche der Implementierung.
Für die nächsten Abschnitte ist vertieftes Wissen um den verfügbaren Quellcode und die Funktionen nötig. Jede VECU verfügt über ihren eigenen gesonderten Quellcode und ist separat in VEOS instanziiert. Ziel ist eine theoretische Vertrautheit mit dem Code zu schaffen, so dass eine Vorstellung von den Anpassungspunkten im AUTOSAR Kontext entsteht. Tabelle 5.1 listet die wichtigsten Punkte im Programmcode.
Die Arbeiten finden im hohen Layer der RTE statt, welcher sich aus einer Menge von

Funktionen zusammensetzt, die man als Tasks im Ablaufplan wiederfindet, in denen die Aufrufe der Runable-Funktionen zu finden sind. Es gibt Aufrufe, die bei Start und Stop der ECU getätigt werden, sie sind sich vorzustellen wie Konst- und De- struktoren. In den Grundlagen zur RTE (Abb.2.2 S.7), zeichnet sich bereits ein Aufrufcharakter, ausgehend von der RTE, ab. In der Form, dass Runables, dort Software Components, über die RTE aufgerufen werden und die RTE ihre Signale z.B. an AUTOSAR Communications weitervermittelt. Dies kann in Codeform wiedergefunden und manipuliert werden, wie es im Folgenden an der Synchronisation gezeigt ist.

Ort –> Datei	Funktionen	Beschreibung
Separat –> IndicatorLogic.c/.h	Diverse	Enthält Funktionskörper der Runables wie z.B. Sync & Toggle. Es gibt mehrere Dateien dieser Art, die nach Funktionen gegliedert sind.
Center Ecu –> Rte.c/.h	Task_10ms	**Bypass Start.** Startet Logic() und Toggle(). Sendet danach Ergebnissignal an das AUTOSAR COM.
	SyncTask	Vorbereiteter Aufrufpunkt der Synchronisation
	Rte_Start	Initialisierung
	Rte_Stop	Beenden Routine (ähnlich Destruktor)
Right indicator Ecu –> Rte.c/.h	Task_100ms	**Bypass Ende.** Ruft Signale vom Com ab. Steuert damit die Fahrtrichtungsanzeiger an.
	Rte_Start	Initialisierung
	Rte_Stop	Destruktor

Tabelle 5.1.: Anpassungsorte in der VECU

5.3. Synchronisation

Für die Synchronisation ist der Einsatzort im Rahmen der Generierung der ECU vorbereitet worden. In der generierten RTE.c ist eine entsprechende Task “SyncTask“ mit dem Runable “Sync“ zu finden. Eine Runable ist eine Funktion und so kann frei gewählt werden, ob der Aufruf der Implementation der Synchronisation in diesem Runable erfolgen soll oder ob damit der originale Aufruf überlagert wird, so dass die eigentliche Synchronisation anstatt des Runables aufgerufen wird. Letzteres wird für den Prototypen verwendet. Ist eine bedarfsorientierte Synchronisation ohne feste Auflösung gewollt, so ist sie z.B. direkt nach dem Senden zu platzieren, dies nähert sich dem Gedanken der Null Message Synchronisation an.
Der benötigte UDP Socket für die Synchronisation muss beim Start initialisiert werden, hierfür gibt es in der RTE.c mit *Rte_Start* eine Routine für die Basisinitialisierung der VECU, die mit den neuen Initialisierungen als Funktionsaufruf erweitert wird. Es wird von Aufrufen gesprochen, da eine Konzeptentscheidung zur minimalen Änderung der VECU Inhalte vorliegt, dies betrifft auch die Kapselung von neuem Code. Die Funktionskörper liegen in gesonderten .c bzw. .h Dateien, diese müssen dem Compiler von SystemDesk mitgeteilt und inkludiert werden.

Die Herangehensweise an die Implementation in Omnet++ ist anschaulicher in dem Sinne, dass die Synchronisation nicht in geschaffene Lücken eingebracht wird, sondern ein wirklicher Bestandteil der Simulation ist. Der Abschnitt 4.7 “Konzeption für Omnet++“ ist die Basis für die Implementierung und Abschnitt 4.4 ist Grundlage für die Synchronisation. Es wird ein Knoten erzeugt, der von der Klasse *cSimpleModule* abgeleitet ist, er enthält die unterschiedlichen Synchronisationsroutinen und ein UDP Socket. In der .ned-Datei sind die Parameter, für die externe Konfiguration mittels der in Omnet++ üblichen .ini-Datei vorbereitet. Folgendes ist einstellbar:
Adresse des VEOS Gegenstellenrechners, die Portadresse der Gegenstelle der Synchronisation in VEOS. Die eigene Adresse für die Omnet++ Synchronisation und der Synchronisationsmodus kann festgelegt werden, in parallel, sequentiell oder zukünftige Synchronisationsalgorithmen. Es kann die Breite des Ausgangsvektors, für die Skalierbarkeit der Anzahl der angegliederten Empfänger, festgelegt werden.
Die Initialisierung der Knoten erfolgt in Omnet++ bei Start der Simulation. Hierbei

wird im Knoten der Synchronisation eine Selfmessage für den Simulationszeitpunkt t(0) eingebracht, so dass die Simulation direkt in die Synchronisationsroutine läuft und dort am blockierenden Socket gefangen werden kann. Der weitere Ablauf ist die Implementierung des Synchronisationsalgorithmus über die Grenzen der einzelnen Simulationen hinaus.

Algorithmus in der Implementierung der sequentiellen Synchronisation

Kernelement der Synchronisation ist der eigentliche Algorithmus, diesem ist im Konzept 4.4 Form gegeben. Die Implementierung setzt das Sequenzdiagramm 4.3 von Seite 45 in Programmcode um, wobei für die Betrachtung der Code abstrahiert wird. Beide Simulationen, VEOS und Omnet++, arbeiten im Rahmen der Synchronisation in direkter Verbindung miteinander, deshalb wird auf eine Trennung nach Plattformen verzichtet, stattdessen wird dem Ablauf des Diagramms gefolgt. Zum Unterstützen des Verständnisses sind die folgenden Ausführungen in den Nassi-Shneiderman-Diagrammen 5.3 und 5.4 verkürzt und ohne Hintergrundinformationen dargestellt. Sie zeigen eine Möglichkeit, wie ein Synchronisationsalgorithmus aufgebaut sein kann.

Im Startzustand bei t(0) wartet Omnet++ aufgrund der Führungsrolle von VEOS. Warten findet immer in Empfangsbereitschaft am Socket statt. Die VEOS Simulation beginnt hingegen direkt mit der Arbeit bis zum Zeitpunkt t(1). Bei t(1) läuft die Simulation, bedingt durch den Ablauf in der VECU, in die Synchronisationsroutine. Die Freigabemeldung an Omnet++ wird konstruiert. Hierfür wird die Simulationszeit abgerufen, diese liegt in float64 vor. Die Zeit wird mittels des Sockets an Omnet++ übertragen und VEOS wartet seinerseits auf eine bestätigende Zeitmeldung von Omnet++.
In Omnet++ wird, mit Ankunft dieser Meldung aus VEOS, die Synchronisationsroutine fortgesetzt. Die Meldung wird ausgewertet, die Simulationszeit wird zurückgewonnen und als zukünftiger Synchronisationspunkt in den Ablauf von Omnet++ als Selbstnachricht eingebunden. Somit ist Omnet++ dynamisch im Zeitverhalten der Synchronisation und erfordert keine Einstellungen von Zeiten. Enthält die Nachricht der Gegensimulation keine Zeitsignatur, wird geprüft ob es ein Steuersignal ist, die-

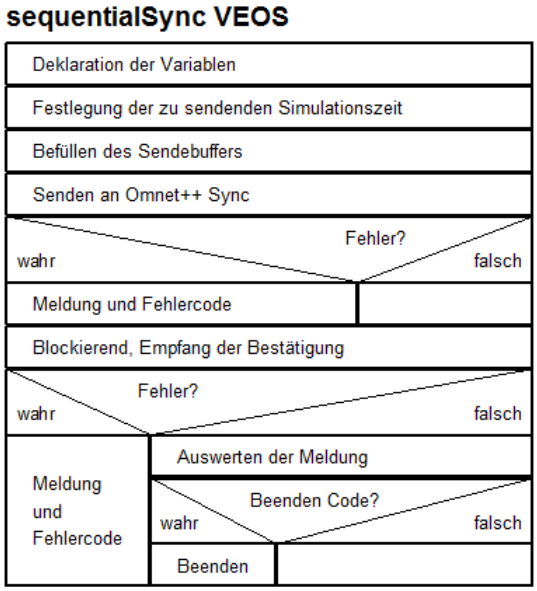

Abbildung 5.3.: Programmablauf VEOS

ses kann mitteilen, dass die Gegenstelle beendet wurde und diese Simulation auch zu beenden ist. Erweiterungen dieser Steuersignale sind denkbar und hängen von den Ansteuerungsmöglichkeiten der Simulationskörper ab. Abschließend wird die Synchronisationsroutine von Omnet++ verlassen, die Simulation bewegt sich zum Zeitpunkt t(1) um dort erneut durch die Selbstnachricht aufgerufen zu werden. Es wird die Simulationszeit als Bestätigung an VEOS gesendet. VEOS erhält die Freigabemeldung, wertet diese aus und die Synchronisationsroutine wird verlassen, um durch den Ablauf bei t(2) wieder aufgerufen zu werden. An dieser Stelle schließt sich der Kreis. Für die VEOS Synchronisationsfunktion ergibt sich das Bild von Diagramm 5.3.

Die Synchronisationsroutine wird in der VECU jede Millisekunde aufgerufen, optional nach jedem Sendeereignis. Für die Routine ist es nicht von Belang wo sie aufgerufen wird. In jeden Fall wird die Simulation unterbrochen und die Simulationszeit übertragen, so dass die Gegensimulation zu diesem aktuellen Synchronisationspunkt aufholen kann und die Bestätigung zurücksendet. In Omnet++ sieht man die Vorteile, die ein dynamischer Ablauf bringt (Abb.5.4). Ohne Konfiguration kann Omnet++ mit VEOS zusammenarbeiten, die Kopplung arbeitet so halb dynamisch. VEOS mit festem oder

schwankendem Intervall und Omnet++ kann reaktiv zu VEOS seinen Ablauf anpassen. Abbildung 5.4 zeigt den Omnet++ Code in abstrahierter Form.

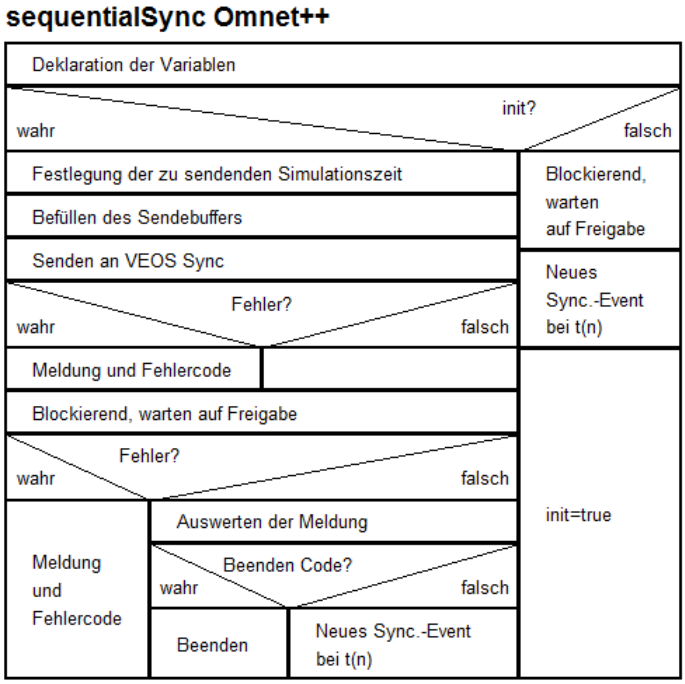

Abbildung 5.4.: Programmablauf Omnet++

Algorithmus in der Implementierung der parallelen Synchronisation

Die Technik aus dem vorherigen Abschnitt bleibt die gleiche, es werden nur die Unterschiede aufgezeigt. Der Algorithmus basiert auf dem Sequenzdiagramm 4.4 aus dem Konzept zur parallelen Synchronisation auf Seite 46. Der Startablauf ist ähnlich, VEOS erteilt die Freigabe für Omnet++. Der Unterschied liegt darin, dass VEOS die Freigabe das Startsignal, vor dem Simulieren sendet. Unter der Annahme einer nahe Nullzeitübertragung erhält Omnet++ das Startsignal zeitnah und beginnt, parallel zu VEOS, mit der Simulation bis t(1). Die Information wo t(1) liegt, kann Omnet++ von VEOS im Startsignal erhalten das einen Zeitpunkt enthält, der in der

Zukunft liegt oder es wird ein fester Wert aus der ini-Datei aufaddiert, die erste Variante ist vorzuziehen, um eine weitere Drift-Quelle auszuschließen. Die Simulation wird bei beiden Simulationssytemen in dem Maße fortgesetzt, dass der geblockte Ausführungsthread in der Synchronisationsroutine aufgeweckt wird und diese verlässt. Bei Erreichen des nächsten im Ablauf verzeichneten Synchronisationspunktes, wird die Routine wieder betreten und über das Socket wird eine Synchronisationsmeldung an den jeweils anderen verschickt, sowie anschließend auf eine Meldung des anderen gewartet. Der Kreis des Synchronisationszyklus schließt sich an der Stelle, wo sich die Übertragungen zwischen Senden und Warten auf Empfang überschneiden. Im Code ergeben sich Erweiterungen beim Start und Reihenfolgeänderungen, so dass sich zwei Ausführungslinien abzeichnen, die sich gegenseitig wecken und sich im Drift maximal +/- einen Synchronisationspunktabstand voneinander entfernen können. Damit kann der Drift doppelt so hoch werden, wie bei der sequenziellen Synchronisation mit gleicher Synchronisationsauflösung.

5.4. VEOS Datenkopplung

Das Konzept legt fest, dass Signale, die an den CAN-Bus gerichtet sind, über Omnet++ umgeleitet werden. Die Umleitung soll in der RTE beginnen. Im Code steht die Rte.c hierfür zur Verfügung. Nach der Untersuchung, welches Runable der VECU die CAN-Übertragung vornimmt, wird auch der zugehörige Task identifiziert. Im behandelten Phototypensystem nennt er sich Task_10ms, er ruft die Arbeitsroutinen auf und anschließend wird ihr Out-Buffer an AUTOSAR COM übertragen, dies geschieht im 10ms Intervall. Es wird der originale Sendeaufruf: *Com_SendSignal(0, (const void*) &Rte_Buffer_1);* und seine vorbereitenden Befehle entfernt. Abschließend wird der neue Sendebefehl eingebracht. Dies ist eine Funktion, die das *sendto()* vom Socket und Fehlerbehandlung umfasst. Signale werden direkt in Byte-Form an Omnet++ versandt. Das Ergebnis ist ein Sender in einer synchronisierten Simulationsumgebung, der aufgrund seiner Umgebungsunabhängigkeit überall platziert werden kann. Bedingt aus den Charakteristiken der VECU, gibt es einen Datenstrom mit einem 10ms Intervall an Omnet++. Wie Omnet++ diesen verarbeitet und weiter behandelt ist im Abschnitt 5.5 aufgezeigt.

Von Omnet++ erhält VEOS den simulierten Ausgangsdatenstrom zurück. Dieser ist an den Endpunkt des Bypass gerichtet, exemplarisch die Right indicator Ecu. Die Signalankunftszeit setzt sich zusammen aus der Startzeit plus die simulierte Omnet++ CAN-Laufzeit.
Das Signal wird folgendermaßen entgegengenommen, im Konzept ist der Empfang in der VECU Task eingeplant, die im Original die Daten vom virtuellen CAN-Bus entgegennimmt. Diese befindet sich in der RTE der Right indicator ECU. Der Einsprungpunkt heißt *Task_100ms* und wird im 100ms Intervall angesteuert. Es herrscht ein Polling-Verhalten vor, bei dem im Originalen mittels *Com_ReceiveSignal(2, (void *) &Rte_Signal_1);* das letzte Signal vom AUTOSAR COM abgerufen wird. Das Zeitverhalten ist demnach 10 zu 1, denn es werden im 10ms Intervall Signale gesendet und im 100ms Intervall entgegengenommen. Für die Implementierung ist das in dem Sinne von Belang, dass mindestens das letzte erhaltene Signal verfügbar sein muss. Wie beim Sender, wird der originale Empfangsaufruf entfernt und durch den entwickelten Empfänger ersetzt. Der Empfänger darf den normalen Ausführungsthread nicht mit seiner Empfangstätigkeit beanspruchen, dies übernimmt ein separater Thread, dessen empfangene Daten über eine Mutex geschützte Übergabestelle erreichbar sind. Das Diagramm 5.5 zeigt, wie der Empfangsthread gekapselt arbeitet.

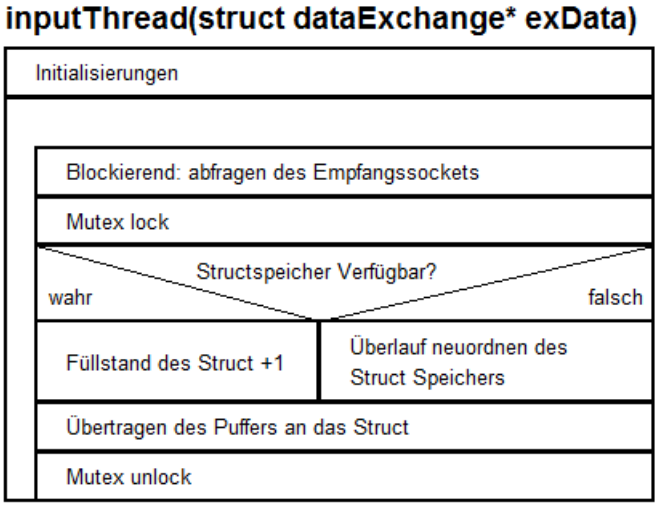

Abbildung 5.5.: Arbeitsweise des Empfangsthread

Dieser Empfänger wird zum Simulationsstart initialisiert und in Betrieb genommen. Vor der Anpassung wurde das originale AUTOSAR COM abgefragt, nach der Anpassung wird der Datenpuffer des neuen Empfängers abgerufen. Damit ändert sich der Ort der Abfrage, alle anderen Teile bleiben unverändert. In der Implementierung ist dieser Puffer ein Struct, mit einer Doppelbedeutung. Es enthält vier Teile: Einen Mutex, eine Zählstandvariable, ein Array für die eingehenden Übertragungen und die Konfigurationsinformationen des Empfangssocket. Das Struct wird einleitend zur Initialisierung benutzt und im Betrieb ist es mit seinem Mutex der Austauschpunkt der Datenhaltung. Aus dem originalen Programmkontext heraus kann hergeleitet werden, wie groß ein erwartetes Signal ist, in diesem Fall ein Byte. Zur besseren Generalisierung sollte der Puffer größer dimensioniert sein z.B. er soll bis zu zehn Signale fassen, die bis zu zehn Byte lang sein können. Um den Empfangspuffer abzufragen, wird in einer Funktion der interne Mutex des Austauschstructs gesperrt und es wird eine Kopie vom Struct, die auf Daten und Füllstand begrenzt ist, angelegt. Anschließend wird der Füllstand der Originals auf null gesetzt und der Mutex freigegeben. Alle weiteren Schritte werden auf der Kopie des Austauschstructs vorgenommen, wie z.B. das Auswerten des letzten Signals. Der Weg der Kommunikation zwischen den VECUs verläuft über Omnet++, welches die Simulation des CAN-Bus vornimmt. Zur Erprobung verbleibt die dritte VECU im originalen Zustand und bildet mit dem identischen Verhalten die Kontrollgruppe.

5.5. Integration in eine Omnet++ CAN-Simulation

Die Implementation der Kopplung, für die Omnet++ Simulationen, findet exemplarisch im Framework FiCo4OMNet statt. Ein bestehendes CAN-System wird angepasst, dieses ist mit den Anpassungen in Abb.4.5 S.47 zu sehen. Die Implementierungen und Anpassungen der Knoten sind nacheinander erläutert.
Um die Omnet++ Simulation den Realitäten des VEOS CAN-Bus näher zu bringen, wird der Omnet++ CAN-Bus entsprechend konfiguriert. Die Baudrate ist auf 125000Bit/s festgesetzt, CAN nutzt das Protocl 2.0B. Die CAN-ID für die Bypass-Nachrichten kann frei festgelegt werden, es ist hierbei sicherzustellen, dass es keinen Konflikt mit anderen Meldungen gibt. Die originalen CAN-Meldungen der Omnet++ Simulation sind für

die Testläufe deaktiviert, können aber ungehindert stattfinden, da jeder CAN-Knoten nur auf bestimmte CAN-ID lauscht. Dies ist eine Entscheidung zur Steigerung der Übersicht, um die Protokolldaten besser auswertbar zu machen. Alle aufgezeigten Anpassungen können in der Konfigurationsdatei der Simulation vorgenommen werden. Damit die allgemeine Koppelbarkeit mit Omnet++ Simulationen gegeben ist, wird der Empfang und das Senden an VEOS getrennt von der Simulation gehalten. Das Gatway zur eigentlichen Simulation stellt ein angepasster CAN-Knoten dar, der je nach Simulation neu zu entwerfen ist, z.B. für Real-time Ethernet. Der Weg eines VEOS Signals in Omnet++ beginnt beim Empfänger.

5.5.1. Empfänger

Die Bezeichnung des entwickelten Empfängers ist OmnetIn. Der Knoten OmnetIn verfügt über zwei Gates für In- und Output. Das Input-Gate ist für Eingangsmeldungen, die das Abrufen von neuen Daten auslösen. Die Planung sieht vor, dass dieser Eingang mit dem Ausgang des Synchronisationskonten verbunden wird, so dass bei jedem Zyklus der Synchronisation der Dateneingang überprüft wird. Das Output-Gate wird genutzt, um die empfangenen Daten an die Omnet++ Simulation als Message zu senden. Daten vom Eingangssocket des Knotens werden mit der selben Technik wie bei VEOS entgegengenommen (Abschnitt 5.4). Es wird ein Thread für den Empfang angelegt und der Datenaustausch findet über einen Mutex geschützten Speicher statt. Die Adresse des Empfängers kann in der Konfigurationsdatei festgelegt werden. Trifft eine Message von der Synchronisation über das Input-Gate ein, wird die Funktion *handleMessage* aufgerufen, die den Eingangspuffer in Messages übersetzt, es wird das gepufferte Signal an den ContextPointer der Message gebunden und diese über das Output-Gate versendet. Abbildung 5.6 zeigt diesen Programmablauf. Mit Absetzen der Botschaft an das CAN-Gateway sind Aufgaben des Empfängers abgeschlossen. Der Empfang verläuft mit einer Anbindung zur Synchronisation, so wird sichergestellt, dass ein konsistenter Datenzustand am Synchronisationspunkt vorherrscht.

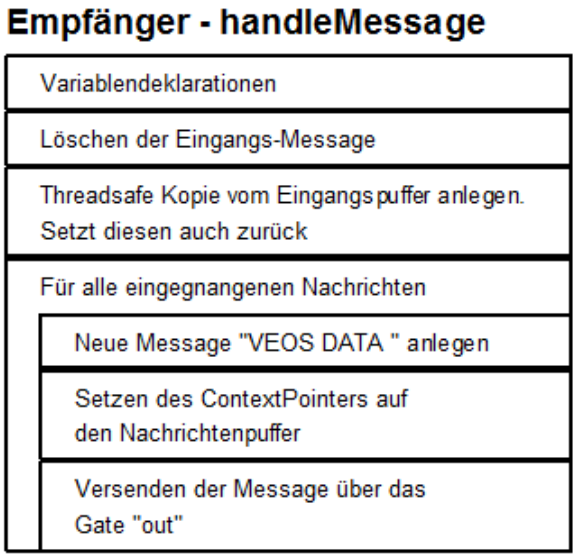

Abbildung 5.6.: Programmablauf Omnet++ Empfangsknoten

5.5.2. CAN-Knoten

Der CAN-Knoten, auch CAN-Teilnehmer genannt, ist ein bereits vorhandenes Element und stellt immer einen möglichen End- oder Startpunkt dar. Der Knoten setzt sich aus fünf Unterelementen zusammen. Ziel ist es, den Knoten um die nötige Gateway-Funktionalität zu erweitern. Im unmodifizierten Zustand kann dieser Knoten, anhand seiner Konfiguration, Nachrichten absetzen und empfangen. Hierbei setzt er Meldungen mit der vorkonfigurierten ID ab und reagiert nur auf diese. Abb.5.7 zeigt den Knoten im modifizierten Zustand. Im Original sind die beiden oberen Elemente *sourceApp* und *sinkApp* End- bzw. Startpunkte. Beide verfügen im Anpassungszustand über eine gerichtete Verbindung zur Umgebung, so dass über diese neuen Gates, Messages zwischen der CAN-Simulation und den neuen Empfangs- bzw. Sende- Knoten, ausgetauscht werden. Den angepassten Knoten gibt es in drei Ausführungen, als nur Input, nur Output und wie in der Abb.5.7, als vollwertiger I/O Knoten. Diese Varianten sind notwendig da “offene“ Gates, also unverbundene, in Omnet++ nicht erlaubt sind. Diese benötigen nicht immer vollen I/O, daher haben auch die anderen Ausführungen

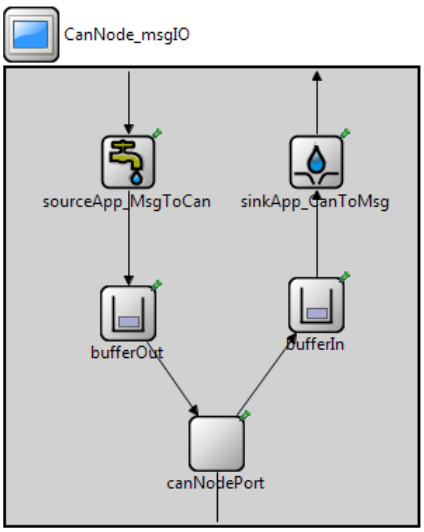

Abbildung 5.7.: CAN-Knoten mit Gateway-Erweiterung

ihre Existenzberechtigung. Es sind *sourceApp* und *sinkApp* mit den vorgenommenen Anpassungen dargestellt, um die Gateway-Funktionsweise zu erläutern.

SourceApp_MsgToCan

Die angepasste SourceApp hat die Funktion, eingehende Messages in das CAN-System zu übersetzen. Ein neuer CAN-Frame wird angelegt in den die Daten, die hinter dem Contextpointer der eingegangenen Message liegen, geladen werden. Das Übersetzen ist eine unabhängige Erweiterung der originalen Funktionalität, diese bleibt unverändert erhalten. Im Detail wird zunächst festgestellt, ob die eingegangene Message vom Input-Gate stammt. Ist dies so, setzt die Gatway-Erweiterung ein. Ein *CanDataFrame* wird angelegt und mit der gewollten ID und Daten versehen. Abschließend wird der CanDataFrame zum CAN-Bus Knoten versandt und ist von da an ein regulärer Teil der CAN-Simulation. In der Abb.5.7 ist die SourceApp oben links zu sehen.

SinkApp_CanToMsg

Die SinkApp ist für die Übersetzung in umgekehrter Richtung zuständig, von CAN zu Omnet++ Messages. Neu eingehende CAN-Frames werden anhand ihrer ID gefiltert,

so dass nur die angenommen werden, deren ID in der Konfiguration als akzeptierter Eingang verzeichnet ist. In der Abb.5.7 ist die SourceApp oben rechts zu sehen. Die App ist ein Endpunkt in der CAN-Simulation. Sie wird vom angeschlossenen Puffer, siehe Abb.5.7, über eingegangene Daten informiert. Diese werden nach einer virtuellen Bearbeitungszeit verarbeitet, hierfür werden Selfmessages genutzt. Die Gateway-Funktionalität befindet sich in der Verarbeitung, es ist eine Prüfung hinzugefügt, ob die Frame-ID der gesetzten Übertragungs-ID entspricht. Ist dies der Fall, wird der Datenteil des CAN-Frames in einen anderen Speicher kopiert und eine neue Message mit dem Contextpointer auf diesen Speicher zeigend, aus den Output-Gate an den Sender, verschickt. Die Änderungen sind eine Ergänzung zum Originalcode und lassen die Grundfunktion unverändert.

5.5.3. Virtueller CAN-Bus

Der virtuelle CAN-Bus ist nicht Teil der Arbeit, stellt aber die Mitte der CAN-Simulation dar, somit ist ein Verständnis der Arbeitsweise, speziell der Nachrichtenwege, notwendig. Die Übersichtsabbildung 4.5 S.47 zu Omnet++, zeigt die Vernetzung und die relative Position des Busses zu den CAN-Knoten der Teilnehmer. Folgende Beschreibung findet aus dem Kontext der Abbildung statt. Der Knoten namens Center_Vecu sendet die Omnet++ Repräsentation eines CAN-Frames an den Bus. Das ein Byte VEOS Signal befindet sich an nullter Stelle des CAN-Frame Datenteils, der maximal 8 Byte fasst. Zur Simulation des Busses, wird die Nachricht für alle angeschlossenen Teilnehmer dupliziert und an diese versandt, so dass ein Busverhalten vorgetäuscht wird. Der Frame ist mit einer ID versehen, die Knoten prüfen bei Erhalt, ob dieser Frame für sie von Belang ist. Im Beispiel akzeptiert die Blinker_Vecu den Frame und sendet ihn, nach dem im Abschnitt 5.5.2 beschriebenen Verfahren, umgesetzt an den Sender, OmnetOut.

5.5.4. Sender

Der Sender ist das letzte Element im Datenpfad der Omnet++ Simulation. Er überträgt das Signal an VEOS. VEOS verfügt über einen Empfänger, der in der Ziel-VECU platziert ist (Abs.5.4), dessen Adresse und Port, Ziel der Übertragung ist. Der Sender arbeitet unmittelbar, so dass eintreffende Daten ohne Verzögerung weitergeleitet werden. Die

Implementierung erfolgt ähnlich, der in VEOS genutzten Technik, einzig die Form ist anders. Der Omnet++ Sender ist ein Knoten mit einem Input-Gate. In Abb.4.5 ist er mit OmnetOut bezeichnet. Die Daten sind am Contextpointer der eingehenden Message hinterlegt und werden mittels des Sockets, an das in der Konfiguration festgelegte Ziel versandt.

5.6. Rückblick auf die Implementierung

Basierend auf den Konzepten ist die Implementierung umgesetzt. Für VEOS und Omnet++ ist ein System entwickelt worden, das die beiden Simulationen synchronisiert. Neben der Synchronisation wurde eine Datenkopplung von VEOS und Omnet++ implementiert. Der Zehnpunkteplan des Konzepts Tab.4.1 ist behandelt und umgesetzt worden. Der Hauptaufwand war die Anpassung der VECUs und das Auffinden geeigneter Zugriffspunkte für die Datenkopplung im AUTOSAR Standard. Die Änderungen am Quellcode der RTE sind ein irregulärer Eingriff in den Ablauf der VEOS Simulation einschließlich ihrer VECUs und vom Hersteller in der Form nicht vorgesehen. Für kommende Arbeiten ist ein anderer Ansatz zu wählen, z.B. in der Entwicklung einer neuen VECU, die einzig zum Zweck der Kopplung konzipiert ist, dieser Gedanke ist in Abschnitt 7 weiter verfolgt. Mithilfe der Grundlagen um AUTOSAR, musste im Rahmen der Erzeugung der VECU, die entsprechende Quellcoderepräsentation gefunden werden, so dass die notwendigen Änderungen vorgenommen werden konnten.
Das System verfügt als Prototyp über zwei mögliche Synchronisationsroutinen. Für die Datenkopplung wird das Signal, welches über CAN an die Right indicator Ecu geschickt wird, an die Omnet++ Simulation umgelenkt. Hierfür nutzt die Implementierung UDP-Sockets. Omnet++ nimmt die Signale unmittelbar entgegen und bringt sie über einen zum Gateway modifizierten Knoten in die CAN-Netzwerksimulation ein. Die Datenkonsistenz ist immer zum Synchronisationszeitpunkt gegeben. Nach der Verarbeitung der Daten in der Netzwerksimulation werden diese, über ein weiteres Gateway und den Sender, an die Ziel-VECU unmittelbar übertragen. Die Ziel-VECU ruft die eingegangenen Signale, bedingt durch ihre Arbeitsweise, im 100ms Intervall ab. Damit ist ein gekoppelter Ablauf erreicht bei dem ersichtlich ist, dass eine Skalierung der angewandten Methoden auf eine umfangreichere Simulation möglich ist.

6. Evaluation

Es ist festzustellen, wie weit die Implementierung die Kriterien der Analyse erfüllt und ob das Ziel der Simulationskopplung erreicht ist. Für das Kriterium der erfolgreichen Umsetzung der Analyse ist die zusammenfassende Tabelle 3.2 von Seite 28 herangezogen. Basierend auf ihr ist jeder Arbeitspunkt behandelt:

1) Die Simulationen sind festgelegt als VEOS und Omnet++. Deren Austauschpunkte sind die CAN-Schnittstellen.
2) Das zu nutzende Übertragungsmedium ist LAN.
3) Das Zeitformat ist als eine 64Bit Ganzzahl mit Mikrosekunden Basis festgelegt. Diese Festlegung ist problembehaftet, da der Wertebereich im Gegensatz zu Double oder Float64 eingegrenzter ist. Diese Entscheidung kann mit der Umstrukturierung auf eine Fließkomma-Repräsentation revidiert werden, ist also optional.
4) Es wird ein konservativer Algorithmus angewandt. Dieser arbeitet auf Basis einer festen Synchronisationsauflösung und kann auch zur dynamischen Synchronisation eingesetzt werden in Form einer Adaption, des Null Message Algorithmus. Ob fest oder dynamisch, hängt vom Ort des Einbindens ab. Weiter wird ein paralleler Algorithmus untersucht, es wird ein erhöhter Drift zugunsten von Parallelität erprobt.
5) Die Integration der Synchronisation erfolgt in Omnet++ als eigenständiger Knoten. In VEOS wird eine separierte Funktion platziert, entweder in einer Task mit statischem Intervall für eine feste Synchronisationsauflösung oder beim Senden und Empfangen für eine dynamische, bedarfsorientierte Synchronisation.
6) Die feste Synchronisationsauflösung wurde auf 1ms festgelegt. Aus den Simulationsmetriken muss sie kleiner gleich 10ms sein.

7) Es wird die CAN-Kommunikation der VECUs für die Datenkopplung, in Form eines Bypass, umgelenkt.
8) Die neuen Knoten sind in Omnet++ frei platzierbar. Die Funktionen in den VECUs sind im erreichbaren Quellcode frei platzierbar. Initialisierungen müssen für jede VECU separat vorgenommen werden.
9) Alle Datentypen werden in Byteform übertragen.
10) Der Sender adressiert mit IP und Port.
11) In der VECU wird der Bypass in der RTE vorgenommen, AUTOSAR COM und folgende Elemente sind nicht involviert. Die Folge des Bypass in dieser Form ist, dass der originale Communication Bereich nicht mehr Teil der Simulation ist.
12) Empfänger verfügen über einen eigenen Thread und Datenpuffer.
13) Beim dezentralen Modell hat jeder Empfänger der Kopplung eine eigene Port-Adresse. Ein zentralisiertes Modell ist ressourcensparender. Siehe Kapitel 7 Verbesserungen und Aussichten.
14) Omnet++ nimmt Daten an jedem Synchronisationspunkt in die Simulation auf. Die VECU fragt alle 100ms den CAN-Bus bzw. den Puffer des Empfängers ab.

Die Voraussetzungen für eine Kopplung sind gegeben. Es gibt dabei Einschränkungen in der Verfügbarkeit von Quellcode und der dynamischen Anpassung des Ablaufs der VECU. Für die Evaluation am Prototypensystem werden Log-Dateien genutzt, die von Beobachtungen aus den Probeläufen ergänzt werden. Das erhaltene Bild wird den Erwartungen an den Testlauf gegenübergestellt. Zuvor ist zwecks Vergleichbarkeit, die originale Funktion der AUTOSAR Simulation dargestellt.

Original- und Anpassungszustand

Das originale Simulationssystem in VEOS besteht aus drei VECU. Sie simuliert die Steuerung der Fahrtrichtungsanzeiger in einem Automobil. Die zentrale VECU nimmt Eingaben entgegen und steuert über einen CAN-Bus die VECU für den linken oder rechten Fahrtrichtungsanzeiger an. Für Aktivität in der Simulation sind die Fahrtrichtungsanzeiger auf stetigen Wechsel ihres binären Zustands eingestellt, so dass ein Warnblinklicht nachgebildet wird.
Im Anpassungszustand wird für die Kopplung ein Bypass der Nachrichten, von der zen-

Abbildung 6.1.: Simulationslauf in der Darstellung eines Zeitstrahls
s := Synchronisationspunkt
w:= Sendepunkt
r := Empfangspunkt

tralen VECU zur VECU des rechten Fahrtrichtungsanzeigers, eingebunden. Der Bypass erfolgt über die Simulation Omnet++. Die linke VECU bildet mit ihrem identischen Aufbau die Kontrollgruppe für die rechte VECU und wird im originalen Zustand belassen. Für die Simulationen liegt die Erwartung vor, dass sie entsprechend des 1ms Intervalls synchronisiert werden. Während der laufenden Simulation werden Nachrichten im 10ms Intervall in den Bypass gespeist. Diese werden in der Omnet++ CAN-Simulation verarbeitet und an die VECU für den Fahrtrichtungsanzeiger vorne rechts weitergereicht, dieser holt die letzte Nachricht alle 100ms ab. Basierend auf Log-Daten zeigt der Zeitstrahl 6.1 die Events der Kopplung. Die Events treten in den erwarteten Zeitpunkten ein, “s)“ kennzeichnet die Synchronisationspunkte im Millisekunden-Intervall, die in den jeweiligen Simulationen dem Algorithmus entsprechend eintreten.

Synchronisation - Bewertung

Im Simulationsdurchlauf wurden die verschiedenen Algorithmen erprobt. Im sequenziellen Fall, mit fester Synchronisationsauflösung, zeigt sich die Führungsrolle von VEOS beim Pausieren der Simulation. Je nachdem wo die Pause ausgeführt wird, ist VEOS maximal eine Millisekunde voraus. Wird die Synchronisation auf dynamisch nach dem Senden verlagert, reduziert sich die Synchronisationsrate. Zusätzlich ist der Drift, der zeitliche Abstand zwischen Senden und Synchronisationspunkt, minimal. Beim parallelen Ansatz zeigen sich die Probleme der Analyse. Beim externen Eingriff des Pausierens wartet die nicht pausierte Simulation bei t(n+1), der zeitliche Abstand

der Simulationen ist genau eine Millisekunde. Wechselseitige Unterbrechungen im parallelen Algorithmus kann den Drift maximal werden lassen. Die Erprobung bestätigte die getroffenen Annahmen zum Zeitverhalten der Algorithmen. Die Variation des NULL Message Algorithmus, mit der Synchronisation zu jedem Austauschevent, erweist sich als am günstigsten, da das Synchronisationsaufkommen abhängig vom Bedarf ist und der Drift minimal.

Datenübertragung - Bewertung

Die Datenübertragung arbeitet wie vorgesehen, dies wurde anhand der Log-Dateien, der Untersuchung von Puffern und Beobachtung der Simulation sichergestellt. Der Zeitstrahl 6.1 dient zur Verdeutlichung, “w“ und “r“ sind die relevanten Zeitpunkte der Datenübertragung. Die Daten werden zum richtigen Zeitpunkt versandt ggf. entgegengenommen. Der Inhalt der Übertragungen wurde durch Auslesen der Puffer, im Zustand vor und nach der Anpassung, überprüft. Die Signale sind ein Byte groß, die Einhaltung des Werts konnte auf der ganzen Bypass-Strecke verifiziert werden. In Omnet++ wird die Nachricht mit dem Synchronisationspunkt in die Simulation eingebracht. Der CAN-Bus verteilt die Nachrichten an alle Teilnehmer und speist sie am Knoten “omnetOut“ in den Empfangspuffer der Ziel-VECU ein. Tabelle 6.1 zeigt diesen Teilweg im Log von Omnet++, hervorgehoben sind die relevanten Knoteninteraktionen. Die Ausgabe der VECU ist ein Blinklicht mit einer Periode von ca. 1Hz. Die dezentrale Kapselung, der Sender und Empfänger mit ihren Sockets, ist nicht ressourcensparend und angemessen für kleine Kopplungen, bei umfangreichen Simulationen ist ein zentralisiertes Kopplungssystem geeignet, dies ist bei den Überlegungen zur Verbesserung in Abschnitt 7 behandelt.

Simulationszeit	Kommunikationsrichtung	Typus
0.009	sync_veos –> omnetIn	SyncTick
0.010	sync_veos –> omnetIn	SyncTick
0.010	**omnetIn –> center_vecu**	**VEOS DATA**
0.0101	**center_vecu –> canbus**	
0.0101	canbus –> motorx	
0.0101	canbus –> motory	
0.0101	canbus –> motorz	
0.0101	canbus –> center_vecu	
0.0101	**canbus –> blinker_vecu**	
0.0101	**blinker_vecu –> omnetOut**	**CAN DATA**
0.011	sync_veos –> omnetIn	SyncTick
0.012	sync_veos –> omnetIn	SyncTick

Tabelle 6.1.: Gezürztes Logfile Omnet++, basierend auf Abb. 4.5 S.47

7. Abschlussbetrachtung

Die Erkenntnis der Arbeit ist: Erstens die Möglichkeiten der Simulationskopplung, zweitens die Voraussetzungen die für eine Kopplung gegeben sein müssen und drittens was in Form von Teilbereichen generell zu realisieren ist, wie z.B. die Synchronisation innerhalb der Kopplung und der Datenaustausch. Es zeigt sich, dass das Wissen, welches im Rahmen der Arbeit zusammengetragen wurde, übertragbar auf das generelle Kopplungsproblem ist. In der Problemzerlegung sind Strategien zur Synchronisation entwickelt worden, sie sind Verfahren, die parallel zur praktischen Untersuchung der Simulationen entstanden sind und sich im Verlauf der Arbeit weiterentwickelten. Es wurde professionelle Software verwendet, um VECUs zu generieren und zu simulieren. Dadurch konnte man Einblicke in die Umsetzung von AUTOSAR gewinnen und in deren Ablauf eingreifen. Der Eingriff in den Quellcode erfolgte unkonventionell nach der Generierung und unterwanderte so den Erstellungsprozess. Die Kopplung zeigte, wie unterschiedliche Simulationen als verteiltes Simulationssystem arbeiten können und gemeinsam ein Bild von Steuergeräten und CAN-Bus liefern. Bei der Arbeit mit SystemDesk wurde ermittelt, wie und wo Steuergeräte anpassbar sind, um eine Kopplung vorzunehmen. Für die Netzwerksimulation sind allgemeine Knoten entwickelt worden, die für die Kopplung genutzt werden. Ebenso sind CAN-Knoten zu Gateways erweitert worden, die die Datenkopplung in die jeweilige spezielle Simulation herstellen. Es ist festgestellt, dass alle eventbasierten Simulationen koppelbar sind, wenn der Zugriff auf den Simulationskörper ausreicht. Die Kopplung ist vom allgemeinen Problem, zu einem konkreten Fall von VEOS und Omnet++ weiterentwickelt worden. Die allgemeine Betrachtung konnte vom speziellen Fall gestützt werden. Die Entwicklung einer Simulationskopplung ist sehr nahe an den Gegebenheiten der Simulationen gehalten, auch wenn die Kopplungsteile modular verwendbar sind, bleibt der Aufwand sie in die Simulationen einzubringen und die Kompatibilität sicherzustellen. Ein Nebenziel der

Arbeit ist es, Simulationskopplung in allgemeiner Form zu behandeln und Startpunkte bzw. Methoden aufzuzeigen.
Das untersuchte Verfahren der Kopplung ist noch in einer frühen Entwicklungsphase und lässt viele Erweiterungsmöglichkeiten zu.

Verbesserungen und Aussichten

Die bisherige Kopplung ist als dezentrales Modell konzipiert, das hat zur Folge, dass alle Elemente gekapselt über einen eigenen Ressourcensatz verfügen. Ein zentralisiertes Modell ist ressourcensparender, aber die gemeinsame Ressourcennutzung erfordert eine allgemeine Kopplungsverwaltung, die vorher für die gekapselten dezentralen Teile nicht nötig war. Für Kopplungen in größerem Ausmaß ist dies der notwendige nächste Schritt. Bleibt man beim Zentralisierungsgedanken und den VECUs, ist es möglich eine VECU zu entwickeln, die einzig als Gegenstelle der Kopplung fungiert. Diese würde dann mit der Ziel-VECU verbunden werden und so die Kopplung herstellen. Vorteil dieser Methode ist, dass keine Änderungen an der originalen VECU notwendig sind und so ihr Ablauf in keinster Weise beeinflusst wird. Nachteil ist, dass keine Möglichkeiten bestehen Interna der VECU zu überbrücken, die originalen Kommunikationsteile sind immer involviert.
Direkte Simulationskopplung wird schnell eingeschränkt durch die Simulationen selbst und ihre Gegebenheiten. Eine dritte unabhängige Anwendung, die als zentrale Kopplungsverwaltung mit den Simulationen in Interaktion steht, bietet einen optimalen Freiheitsgrad. Damit ist es möglich in einer sternförmigen Anordnung 2+n Simulationen zu koppeln und so eine Basis für die Anwendung komplexer Algorithmen, aus dem Bereich der parallelen und verteilten Simulationssysteme, zu schaffen [13].

In der Arbeit wurden verschiedene Synchronisationsarten erprobt, diese zu erweitern und andere zu ergänzen, ist ein Ziel für die Zukunft. Der konservative Null Message Algorithmus zeigte die besten Ergebnisse, die optimistischen Ansätze weisen das höchste Potential auf, wenn die Voraussetzungen für deren Einsatz herstellbar sind. Für umfassendere Simulationen sind Synchronisationsalgorithmen zu verwenden, bzw. zu entwickeln, die die Kopplung einer Vielzahl von Simulationen erlauben, damit diese in einer verteilten Gesamtsimulation komplexe Sachverhalte, über die

Simulationen hinweg, erfassbar machen. Die Kopplung kann z.B. genutzt werden, um reale Steuergerätekommunikation in Realtime Ethernet Netzwerken zu simulieren, dem Hauptforschungszweig der CoRE Gruppe.

Simulationskopplung ist ein interessantes Forschungsgebiet, welches in der Praxis sehr von den Gegebenheiten der Simulationen abhängt. Aber mit Kreativität und Ideenreichtum ist auch die unzugänglichste Simulation für eine Kopplung zu präparieren.

A. Anhang

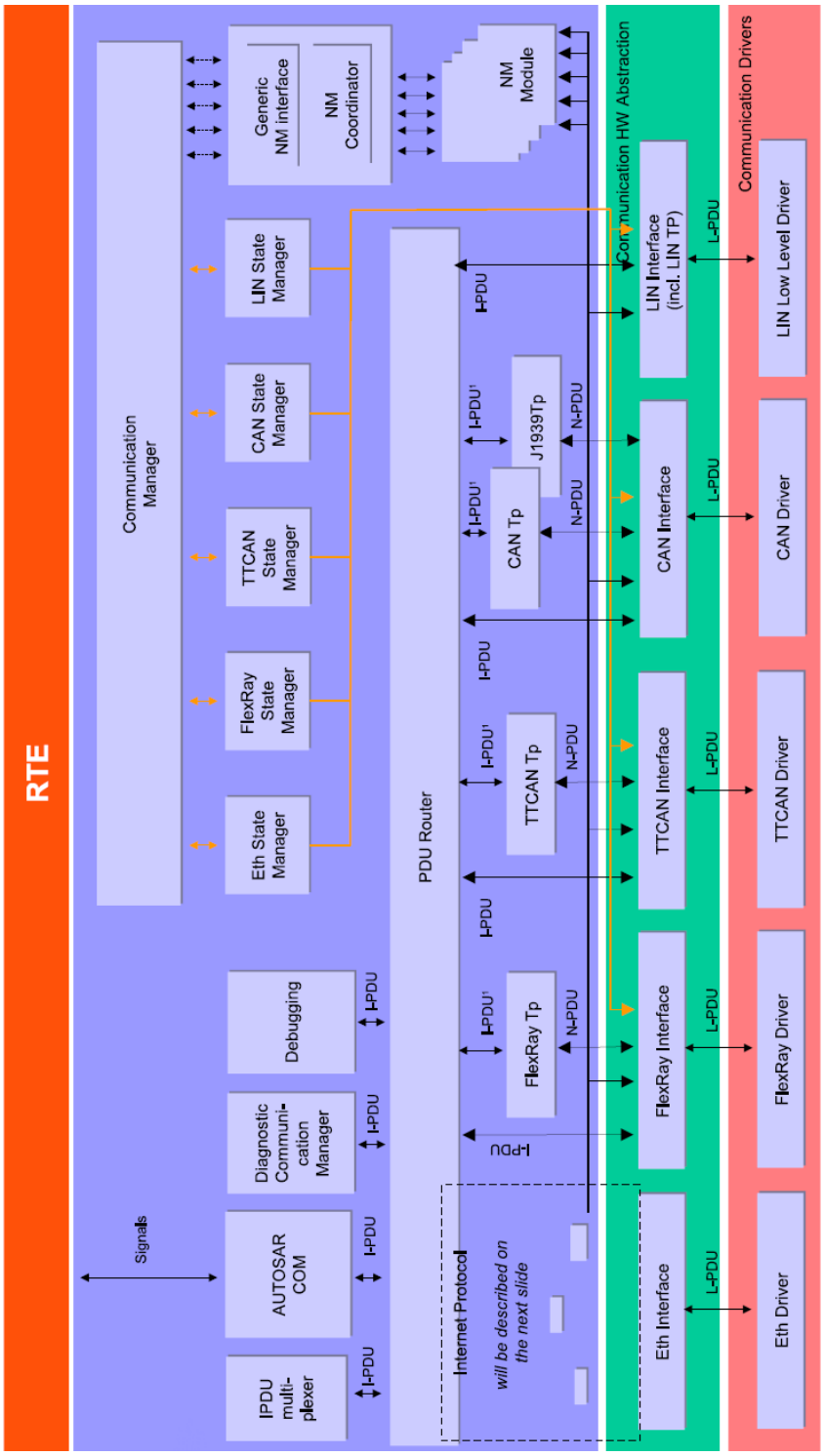

Abbildung A.1.: ECU Communication[3, Figure 3.10]

Abbildungsverzeichnis

Literaturverzeichnis

[1] AUTOSAR. Glossary. Part of AUTOSAR Release 4.2.1(055), 2014.

[2] AUTOSAR. Specification of Communication. Part of AUTOSAR Release 4.2.1(015), 2014.

[3] AUTOSAR. Specification of ECU Configuration. Part of AUTOSAR Release 4.2.1(087), 2014.

[4] AUTOSAR. Specification of Operating System. Part of AUTOSAR Release 4.2.1(034), 2014.

[5] AUTOSAR. Specification of PDU Router. Part of AUTOSAR Release 4.1 V4.2.0 R4.1 Rev 3(035), 03 2014.

[6] AUTOSAR. Specification of RTE. Part of AUTOSAR Release 4.2.1(084), 2014.

[7] AUTOSAR. 10 YEARS AUTOSAR INTRODUCTION. http://www.autosar.org/fileadmin/files/events/10yearsautosar/ATZextra_AUTOSAR_-_THE_WORLDWIDE_AUTOMOTIVE_STANDARD_FOR_EE_SYSTEMS.pdf Abruf:20.03.2015

[8] AUTOSAR. Autosarbasics. http://www.autosar.org/about/basics/ Abruf:20.03.2015

[9] AUTOSAR. OSEK VDX Portal. http://www.osek-vdx.org/ Abruf:10.01.2015

[10] CoRE. FiCo4OMNeT::CanTrafficSinkAppBase Class Reference. http://core4inet.realmv6.org/fico4omnet_documentation/doxy

/class_fi_co4_o_m_ne_t_1_1_can_traffic_sink_app_base.html
Abruf:02.02.2015

[11] CoRE. FiCo4OMNeT::CanTrafficSourceAppBase Class Reference.
http://core4inet.realmv6.org/fico4omnet_documentation/doxy
/class_fi_co4_o_m_ne_t_1_1_can_traffic_source_app_base.html
Abruf:02.02.2015

[12] CoRE. Sourcecode Documentation - Fieldbus Communication for OMNeT++
http://fico4omnet.realmv6.org/fico4omnet_documentation/doxy/
Abruf:02.02.2015

[13] Richard M Fujimoto. *Parallel and distributed simulation systems*, volume 300. Wiley New York, 2000.

[14] dSPACE GmbH. *SystemDesk 4.x - Tutorial.* Release 2014-A.

[15] dSPACE GmbH. *SystemDesk 4.x - Guide.* Release 2014-A.

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

Hamburg, 26. März 2015 Sebastian Schrade