UNIVERSITÄT PADERBORN

Fachtagung
Mechatronik 2019
Paderborn, 27.-28.03.2019

VDI VDE

Veranstaltungsort:
Heinz Nixdorf MuseumsForum
Fürstenallee 7, 33102 Paderborn

- **Mechatronische Produkte**
  (neue Funktionalitäten, Industrie 4.0, Kosten-effizienz, Zuverlässigkeit, ...)
- **Ressourceneffizienz**
  (Energie, Material, …)
- **Automatisierte Mobilität**
  (Konzepte, Modelle, Sensorik)
- **Systems Engineering und Entwicklungsmanagement**
  (Prozesse, Verfahren, Software, ...)
- **Innovative Konzepte und digitale Geschäftsmodelle**
  (Modelle, Regelung, Optimierung, Eco-Systeme)
- **Nutzerfreundlichkeit und Akzeptanz**
  (Assistenzsysteme, Schnittstellen, Interaktion, Gesetzgebung, ...)
- **Systemvernetzung und Systemintegration**
  (Konzepte, Verfahren, ...)
- **Smarte Aktoren**
  (Konzepte, Beispiele, ...)
- **Serienfertigung mechatronischer Produkte**
  (Fallbeispiele, Komponenten, Architektur, Qualitätsmanagement, ...)

www.VDI-Mechatroniktagung.de

Ein Beitrag zur Simulation vernetzter virtueller ECUs
A contribution to the simulation of networked virtual ECUs

Peter Baumann, Robert Bosch GmbH, 71272 Renningen, Deutschland, peter.baumann5@de.bosch.com

Dr. Roland Samlaus, Robert Bosch GmbH, 70442 Stuttgart, Deutschland, roland.samlaus@de.bosch.com

Prof. Lars Mikelsons, Universität Augsburg, 86159 Augsburg, Deutschland, lars.mikelsons@informatik.uni-augsburg.de

Dr. Thomas Kuhn, Fraunhofer IESE, 67663 Kaiserslautern, thomas.kuhn@iese.fraunhofer.de

Jasmin Jahic, Fraunhofer IESE, 67663 Kaiserslautern, jasmin.jahic@iese.fraunhofer.de

Prof. Dieter Schramm, Universität Duisburg-Essen, 47057 Duisburg, schramm@imech.de

Kurzfassung

Mobilität ist auf dem Weg sowohl autonomer als auch vernetzter zu werden. Eine Schlüsseltechnologie dafür ist die kontinuierliche Entwicklung unter Verwendung von Software-in-the-Loop (SiL)-Simulationen bis hin zur virtuellen Validierung oder Freigabe von Softwarefunktionen in einem rein virtuellen Setup. Aufgrund der Komplexität autonomer Fahrfunktionen befindet sich die zu untersuchende Funktion häufig nicht mehr auf einem Steuergerät (ECU), sondern ist über mehrere Steuergeräte verteilt. Daher wird eine Umgebung benötigt, mit der mehrere virtuelle ECUs (vECUs) zusammen mit entsprechenden physikalischen Modellen (Fahrdynamik, Antriebsstrang, usw.) simuliert werden können. In diesem Beitrag wird eine solche domänenübergreifende Fahrzeugsimulation realisiert. Dies beinhaltet eine Co-Simulationsumgebung, eine Werkzeugkette für die Generierung von vECUs als Functional Mock-up Units sowie eine modular aufgebaute Implementierung eines virtuellen Busses. Weiterhin ist eine Möglichkeit zur Handhabung er Restbussimulation realisiert. Die Simulationsumgebung wird in einer Fallstudie getestet bei der ein nahtloser Übergang von einer Model-in-the-loop zu einer Software-in-the-Loop (SiL) Simulation durchgeführt wird. Dabei ist eine Fahrfunktion auf zwei vECUs verteilt, die über einen virtuellen Bus miteinander kommunizieren.

Abstract

Mobility is on the way to be autonomous and connected. One key enabler for new technologies of this type is continuous development using software-in-the-loop (SiL) simulations up to virtual validation or release of software functions in a pure virtual setup. Due to the complexity of autonomous driving functions, the unit-under-test is often not encapsulated in one electronic contol unit (ECU), but distributed over multiple ECUs. Thus a simulation framework capable of simulating numerous connected virtual ECUs (vECUs) together with the corresponding physical models (vehicle dynamics, powertrain, etc.) is required. In this contribution such a cross domain vehicle simulation framework is realized consisting of a co-simulation environment, a toolchain for the generation of vECUs as Functional Mock-up Units (FMU) and a modular virtual bus implementation. A propsal to handle the remain bus simulation is also impelemented. This framework is tested in a case study to show a smooth transition from model-in-the-loop (MiL) to software in-the-loop (SiL) simulation of a function distributed over two vECUs communicating via a virtual controller area network (CAN) bus.

1 Einleitung

Je mehr automatisierte Fahrfunktionen ein Fahrzeug aufweist, desto stärker sind die einzelnen Fahrzeugdomänen (Fahrdynamik, Lenkung, Bremssystem, ...) aufgrund des Wegfalls des Fahrers gekoppelt. Folglich können die entsprechenden Teilsysteme und zugehörigen Funktionen nicht mehr unabhängig voneinander entwickelt werden. Stattdessen müssen Entwickler nicht nur ihre „Heimdomäne“ berücksichtigen, sondern auch alle anderen relevanten Domänen. Führte eine späte Integration der Teilsysteme früher nur zu suboptimalen Ergebnissen, ist die frühzeitige Integration für die Entwicklung zukünftiger Funktionen unabdingbar, da Testfahrten mit beispielsweise autonomen Fahrfunktionen aufgrund des Risikos und gesetzlicher Bestimmungen sehr schwierig durchzuführen sind. Dies lässt virtuelle Prototypen weiter an Wichtigkeit gewinnen und unterstreicht die Rolle der domänenübergreifenden Fahrzeugsimulation als essentielles Entwicklungswerkzeug.

In der domänenübergreifenden Fahrzeugsimulation werden die Modelle der beteiligten Fahrzeugdomänen zu einem domänenübergreifenden Modell integriert. In den einzelnen Fahrzeugdomänen werden meist spezielle Simulationswerkzeuge für den Aufbau der entsprechenden Modelle verwendet, welche dann mit Hilfe der Co-Simulation gekoppelt werden [4]. Dabei müssen neben den beteiligten Modellen der Physik auch die zugehörigen Software-Funktionen integriert werden. In frühen Entwicklungsphasen werden hier Modelle der Software (model-in-the-loop (MiL)) verwendet, bei denen viele Effekte wie zum Beispiel Quantisierung oder Kommunikationsverzögerungen vernachlässigt werden. Sollen diese Effekte in späteren Entwicklungsphasen berücksichtigt werden, so sind virtu-

elle Steuergeräte (vECUs) und virtuelle Busse notwendig (software-in-the-loop (SiL)).
Automatisierte und vernetzte Fahrfunktionen sind häufig auf mehrere ECUs verteilt. Für MiL-Simulationen, in denen Quantisierung und Kommunikationsverzögerungen typischerweise vernachlässigt werden, macht dies keinen großen Unterschied. Bei SiL-Simulationen mit denen verteilte Funktionen validiert werden sollen ist dies nicht der Fall [7]. Hier müssen die vECUs über virtuelle Busse mit realitätsnahem Timing-Verhalten miteinander kommunizieren. Eine entsprechende Simulationsarchitektur ist exemplarisch in Abbildung 1 dargestellt. Hier sind die Fahrzeugdomänen Fahrdynamik, Bremssystem, Lenkung und Antriebsstrang mit der Umwelt und Sensoren in einer Co-Simulationsumgebung miteinander gekoppelt. Während die Softwarefunktionen für Lenkung und Antriebsstrang als MiL-Modelle in die Co-Simulation integriert sind, sind der Fahrzeugcomputer für automatisierte Fahrfunktionen sowie zwei ECUs des Bremssystems virtualisiert und kommunizieren über einen virtuellen Controller Area Network (CAN) Bus miteinander. Verbindungen zwischen der Co-Simulationsumgebung und den vECUs existieren darüber hinaus für Signale, die nicht über den Bus ausgetauscht werden.
In diesem Beitrag wird eine Co-Simulationsumgebung vorgestellt, welche die dargestellte Architektur realisiert. Diese beinhaltet neben physikalischen Modellen zwei vECUs des Bremssystems sowie eine Restbussimulation.

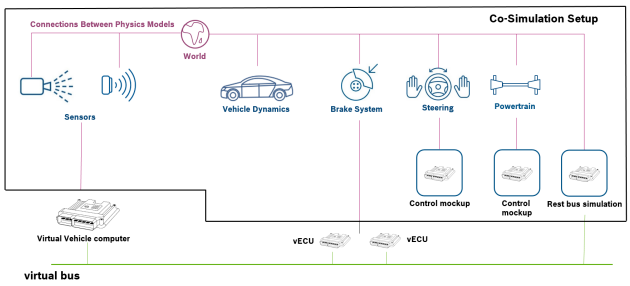

Abbildung 1 Exemplarisches Framework für die virtuelle Validierung mit zwei vECUs und einer Co-Simulationsumgebung

1.1 Themenbezogene Arbeiten

Für die effiziente Nutzung einer SiL-Umgebung sollte diese in die den Entwicklungsprozess begleitende Werkzeugkette integriert werden. In [9] wird eine solche Integration (in den Build-Prozess der Motorsteuerungssoftware) der SiL-Integrationsplattform Silver (QTronic) gezeigt. In [5] wird ein Beispiel für die Integration auf Basis von Continuous Integration mit Hilfe der Standards Functional Mock-up Interface (FMI) [1] und AUTOSAR vorgestellt.
Ein Framework für die Emulation der Kommunikation zwischen verschiedenen Fahrzeugen wird in [8] präsentiert. Dieses Framework kann um die Kommunikation innerhalb eines Fahrzeugs erweitert werden, bietet aber nicht die notwendige Methodik zur Orchestrierung der Co-Simulation der beteiligten physikalischen Domänen sowie die Unterstützung entsprechender Standards und Bereitstellung notwendiger Schnittstellen.

2 Virtuelle ECUs und Busse

Mit Hilfe von vECUs wird die komplette ECU-Software in Simulationen integriert. Dafür müssen hardwareabhängige Softwareteile virtualisiert werden, indem entweder Hardwaresimulatoren verwendet werden, um den Original-Code unverändert zu testen, oder hardwarespezifische Anteile ersetzt werden, um eine Kompilierung auf x86-Systemen zu ermöglichen. Mit Hardwaresimulationen sind äußerst akkurate Simulationen möglich [3], wobei die Modellbildung aufwändig sein kann und die Modelle aufgrund der hohen Detaillierung üblicherweise sehr langsam laufen. Für eine Simulation auf Fahrzeugebene sind die resultierenden Simulationszeiten oft nicht praktikabel. Das Ersetzen des hardwarespezifischen Codes ermöglicht im Allgemeinen eine schnellere Simulation, ist aber weniger akkurat und bildet nicht exakt das Verhalten der Originalsoftware wieder.

2.1 vECU Kategorien

Die AUTOSAR Softwarearchitektur ermöglicht die Definition von Softwareebenen für das Testen von ECU Software. Je nach Testziel der Simulation kann dadurch der Umfang der Software angepasst werden. Wenn beispielsweise lediglich Funktionen der Applikationssoftware (ASW) betrachtet werden sollen, ohne Limitierungen der Hardware zu testen, genügt es, lediglich diesen Teil der Software ohne Basissoftware (BSW) etc. zu integrieren. Für die verschiedenen Testszenarien lassen sich drei Kategorien definieren:

1. vECUs die lediglich ASW und Laufzeitumgebung (RTE) und optional ein Betriebssystem enthalten, um schnell Tests der ASW durchführen zu können.

2. vECUs die ASW, RTE, BSW, Betriebssystem und einen virtualisierten MCAL (Microcontroller Abstraction Layer) für x86 Systeme enthalten, um realistischere Test bezüglich Ablaufplanung, einfache Timinganalysen auf Task-Ebene sowie Busverhalten ausführen zu können.

3. ECU-code (HEX) emulieren. Hierbei wird der Originalcode auf Hardwaresimulationsmodellen ausgeführt. Dies ermöglicht detaillierte Analysen, inklusive Timing-Verhalten. Die Emulation ist allerdings üblicherweise viel langsamer als Echtzeit und der Aufwand zur Erstellung der benötigten Hardwaremodelle hoch.

2.2 FMU Generator für vECUs

Ein in die Software integrierter FMU Generator ermöglicht die vECUs in Co-Simulationen einzubinden. Für den Zugriff auf interne Variablen wird eine Schnittstelle entwickelt, die dem ECU-Code hinzugefügt werden muss. Da FMI die Simulation von Bussystemen nicht optimal abbildet, werden die Treiber der vECUs angepasst. Abbildung. 2 zeigt den Ablauf der FMU-Generierung:

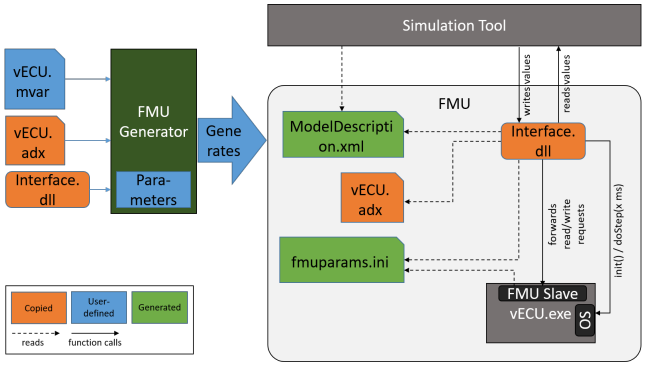

Abbildung 2 Der vECU FMU Generator

2.3 Virtuelle Busse

Virtuelle Busse bilden relevante Eigenschaften von Fahrzeugbussen in Simulationsmodelle ab. Sie ermöglichen eine realitätsnahe Kommunikation zwischen vECUs. Der vorgestellte virtuelle Bus realisiert eine modulare Architektur, bei der jedes Modul ein Teil des Kommunikationsstacks abdeckt. Dadurch können Module schnell ersetzt werden, so dass Simulationen mit verschiedenen Detaillierungsgraden des virtuellen Busses auf einfache Art und Weise realisiert werden können. Bei der Kopplung der Module werden analog zur Co-Simulation die internen Uhren der einzelnen Module synchronisiert [6]. Abbildung 3 illustriert die Simulationsarchitektur des Busses. Dabei wird die Verwendung dreier unterschiedlicher Abstraktionsebenen des Busses gezeigt.
Die gezeigte Netzwerkschnittstelle besteht aus der Warteschlange und dem Medienzugriffsprotokoll (MAC). Beides existiert mindestens einmal pro gekoppelter vECU. Die Warteschlange simuliert das Verhalten der vECU bezüglich wartender, noch nicht gesendeter Nachrichten und entscheidet beispielsweise ob sich Nachrichten überschreiben oder nacheinander über den Bus gesendet werden. Die MACs verbinden sich alle mit dem gleichen Kommunikationsmedium (CAN, Ethernet, Flexray, ...).

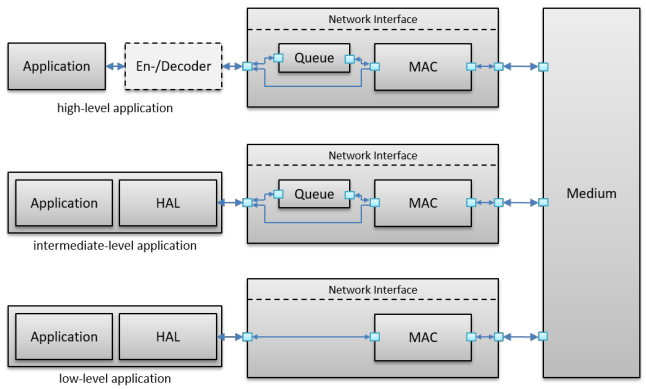

Abbildung 3 Architektur der Bussimulation

Werden nun Komponenten mit unterschiedlichen Abstraktionsebenen an den gleichen Bus angeschlossen, muss sichergestellt werden, dass die Schnittstellen und die Simulationssemantik dieser Komponenten in Bezug auf die Kommunikation gleich sind. Folgende Arten von Komponenten können an einen Bus angeschlossen werden:

1. **Funktionale Ebene (High-Level Application)**, An den Bus angeschlossene Komponenten, die intern keinerlei Kommunkationseffekte berücksichten, z.B. FMUs, werden als High-Level Application bezeichnet. Sie übertragen nur funktionale Daten und keine Protokollrahmen mit Headerinformationen. Die Quantisierung der funktionalen Daten muss daher in dem zusätzlichen Modul En-/Decoder erfolgen. Diese Abstraktionsebene wird beispielsweise verwendet, wenn physikalische Signale aus der Co-Simulation auf dem Bus verfügbar sein sollen. Die Kopplungsarchitektur implementiert die Funktionen transmitPDU/receivePDU um die Kopplung von funktionalen Modellen an den simulierten Bus zu ermöglichen. Die Funktion transmitPDU(payload:byte[], size:int, id:String) überträgt die Nutzdaten eines funktionalen Modells und fügt protokollspezifische Headerdaten hinzu. Der Parameter id beschreibt zum Beispiel die CAN-Bus ID oder die MAC Adresse in Ethernetnetzwerken. Die Funktion receivePDU entfernt Headerdaten und extrahiert die Nutzdaten für den Empfänger.

2. **Protokollebene (Intermediate-Level Application)** Komponenten, die an den Bus angeschlossen sind und Protokollinformationen mit dem simulierten Bus austauschen, werden Intermediate-Level Application genannt. Diese enthalten bereits spezifische Headerdaten für die simulierte Netzwerktechnologie. Anwendungen auf Protokollebene simulieren jedoch nicht die Ebene der Basissoftware (BWS) und die Treiberebene - sie kommunizieren daher mit der Warteschlange des simulierten Busses und nutzen bestehende Simulationskomponenten um das Warteschlangenverhalten zu simulieren. Die vorgsetellte Bussimulation unterstützt die Protokollebene durch die add(frame: byte[]) Funktion, die einen netzwerkspezifischen Rahmen in die Ausgangswarteschlange einer ECU platziert.

3. **Treiberebene (Low-Level Application)** Komponenten, die an den Bus angeschlossen sind und auch die Warteschlange selbst implementieren, werden Low-Level Application genannt. Sie benötigen daher eine hardwarenahe Schnittstelle zur Kommunikation mit dem virtuellen Bus. Die vorgestellte Bussimulation unterstützt dies zum Beispiel mit der getNextMessage(id:String) Callbackfunktion, die aufgerufen wird, wenn die Medienzugriffskomponente den nächsten zu übertragenden Rahmen benötigt. Der Parameter gibt an ob ein bestimmter Protokollrahmen angefragt wird, dies ist zum Beispiel in zeitsynchronisierten Netzen mit reservierten Zeitslots der Fall, oder ob die Auswahl des nächsten Rahmens der Treiberebene obliegt.

Für die Simulation des CAN-Bus, welcher in der nachfolgenden Fallstudie verwendet wird, wird eine ereignisba-

sierte Simulation als Ausführmodell (Model of Computation and Communication (MOCC)) genutzt. Hierbei werden Zeiträume in denen keine Ereignisse auftreten übersprungen. Ereignisse, die in unterschiedlichen Komponenten auftreten, werden global sortiert und in der richtigen Reihenfolge ausgeführt. Wie oben beschrieben übernimmt der virtuelle Bus im Falle von Intermediate- und High-Level Applications selbst die Simulation der Warteschlange, die CAN-Nachrichten absteigend anhand ihrer ID sortiert. Hierbei repräsentieren niedrigere Nachrichten-IDs eine höhere Priorität. Werden mehrere Nachrichten mit der gleichen ID in die gleiche Warteschlange eingegeben kann konfiguriert werden, ob die Nachrichten nacheinander einsortiert werden, oder ob sich diese überschreiben. Die Warteschlange entkoppelt die zeitkritische Kommunikation zwischen dem Medienzugriff und der Anwendung. Werden beispielsweise FMUs genutzt, um Applikationen zu realisieren, können diese aufgrund dieser Entkopplung längere Schrittweiten ausführen. Dies führt zu einer verbesserten Performanz der Simulation, da die Ausführung der Komponenten seltener synchronisiert werden muss.
Das MAC ist für jedes Kommunikationsmedium unterschiedlich. Im der nachfolgenden Fallstudie wird ein CAN-Bus und dementsprechend ein CAN-MAC verwendet. Diese Komponente simuliert das Verhalten des CAN-Bus Controllers, der den Medienzugriff und die Synchronisation der Busteilnehmer auf Bit-Ebene realisiert. Er überwacht das Medium und prüft die Warteschlange sobald das Medium frei wird auf zu übertragende Nachrichten. Danach beteiligt sich jeder CAN-Controller, der eine Nachrichten übertragen möchte, aktiv an der Arbitrierungssequenz. Das Medium simuliert aus Geschwindigkeitsgründen nur die ersten beiden Bits einer Übertragung. Diese sind notwendig für die Synchronisation der CAN-Controller. Die folgende Arbitrierung wird nicht bitweise durchgeführt, sondern für die wartenden Nachrichten anhand ihrer IDs zentral durch die Mediumskomponente berechnet. Diese informiert die angeschlossenen CAN-Controller über die Nachricht, welche die Arbitrierung gewonnen hat und übertragen wird, sowie über den erfolgreichen Abschluss der Übertragung.

3 Simulation vernetzter vECUs

In dieser Fallstudie wird der Übergang einer MiL- zu einer SiL-Simulation mit mehreren vECUs gezeigt, wobei die Funktionalität in beiden Varianten exakt übereinstimmt. Auf diese Weise können unter Wiederverwendung der MiL-Simulation busbezogene Effekte wie Kommunikationverzögerungen und Quantisierungsfehler untersucht werden, welches ein notwendiger Schritt für die virtuelle Freigabe von domänenübergreifenden Funktionen ist.

3.1 Domänenübergreifendes MiL Fahrzeugmodell

Die Integration von vECUs und virtuellem CAN-Bus in eine MiL-Simulation wird anhand eines domänenübergreifenden Co-Simulationsmodells eines elektrischen Fahrzeugs gezeigt, das sich in einem städtischen Verkehrsszenario bewegt [2]. Diese Simulationsumgebung ermöglicht Untersuchungen von Rekuperationsstrategien unter dem Einfluss von Umgebungsparametern wie beispielsweise Verkehrsdichte oder Ampelschaltungen. Die Architektur der MiL-Simulation ist in Abbildung 4 mit den blau umrahmten Kästen dargestellt. Da die unterschiedlichen Komponenten des Fahrzeugs in vier verschiedenen, jeweils domänenspezifischen Simulationswerkzeugen, modelliert sind, werden diese zu einer Co-Simulation miteinander gekoppelt. Dazu wird die Middleware Model.CONNECT von AVL verwendet, die sich zur Simulation mit allen Teilmodellen verbindet und den Signalaustausch zwischen diesen koordiniert. Für die Simulation der Fahrdynamik und des Fahrers wird das Simulationswerkzeug CarMaker (CM) von IPG verwendet. Somit kann das Fahrzeug einer vorgegebenen Route durch ein städtischen Verkehrsszenario folgen und dabei auf andere Verkehrsteilnehmer zu achten. Deren Verhalten wird mithilfe des mikroskopischen Verkehrssimulationspakets „Simulation of Urban Mobility“ (SUMO) vom DLR basierend auf gemessenen Verkehrsflüssen dargestellt. Der Antriebsstrang und der Elektromotor des Fahrzeugs sind als detailliertes, physikalisches Modell in GT-SUITE von Gamma Technologies dargestellt. Sowohl für die Modellierung der Batterie als auch für systemübergreifende Regelungsfunktionen, wie beispielsweise des Batteriemanagements, wird Simulink (SL) von MathWorks verwendet.

3.2 Übergang von MiL zu SiL

Der Übergang von MiL zu SiL wird anhand zweier stark gekoppelter Fahrzeugfunktionen, der Rekuperationskontrolle und der Funktion für die Ansteuerung der Reibbremsen durchgeführt. Anderen Regelfunktionen, wie beispielsweise die Antriebsstrangregelung, bleiben Teil des entsprechenden Simulink-Modells, da es zwischen ihnen und den beiden ausgewählten Funktionen keine direkten Abhängigkeiten gibt. Im Allgemeinen obliegt es dem Ingenieur zu entscheiden welche ECUs virtualisiert werden müssen und für welche MiL-Modelle ausreichend sind. Es handelt sich dabei um ein Kompromiss aus Genauigkeit und Komplexität der Co-Simulation.
Für jede der beiden ausgewählten Funktionen wird eine vECU der Kategorie 2, wie in Kapitel 2 beschrieben, als FMU generiert. Bei der BSW, der RTE und dem Betriebssystem handelt es sich dabei um Seriencode echter ECUs. Die ASW beinhaltet genau dieselbe Funktonalität wie die des entsprechenden Mil-Modells, welche vereinfacht ist und daher nicht Seriencode entspricht. Trotzdem kann der Übergang von MiL zu SiL damit gezeigt werden, da die Komplexität der ASW keinen Einfluss auf die vorgestellte Werkzeugkette hat. Das Einbinden einer Seriencode ASW ist Teil aktueller Arbeiten.
Die beiden vECUs sind über den in Abschnitt 2.3 vorgestellten virtuellen CAN-Bus verbunden, wodurch ein Datenaustausch unter Berücksichtigung busbezogener Effekte ermöglicht wird. Zusätzlich zu den vECUs ist das Restbussimulations- und Analysewerkzeug CANoe von Vector an den virtuellen CAN-Bus unter Verwendung des

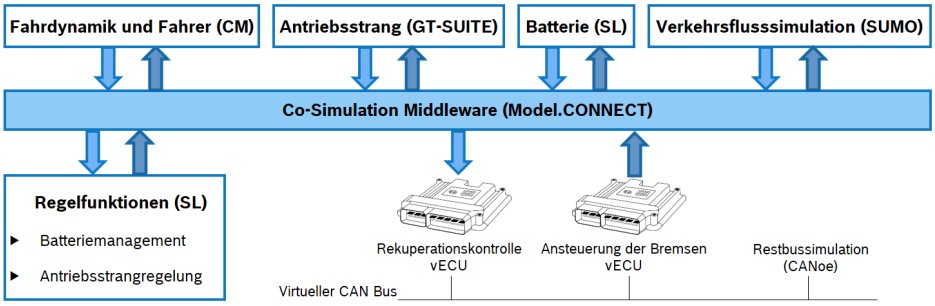

Abbildung 4 Architektur der vorgestellten Simulationsumgebung. Komponenten der MiL-Simulation sind blau umrahmt. Durch den Übergang zur SiL-Simulation kommen die Komponenten hinzu, welche an den virtuellen Bus angeschlossen sind.

CANoe „Fast Data eXchange“ Protokolls angeschlossen. CAN Nachrichten können dadurch in CANoe erstellt und über den virtuellen CAN-Bus an die vECUs gesendet werden. Aufgrund der vereinfachten ASW wird zur Zeit lediglich ein Parameter einer vECU aus CANoe vorgegeben. Sobald vECUs mit Serien-ASW verwendet werden, wird die Restbussimulation mittels CANoe allerdings unerlässlich sein, da sich die Anzahl an ausgetauschten Signalen erhöhen wird und Abhängigkeiten zu nicht modellierten ECU-Funktionen entstehen werden. Die endgültige Architektur der domänenübergreifenden Co-Simulationsumgebung zusammen mit vECUs, virtuellem Bus und Restbussimulation ist in Abbildung 4 gezeigt.

3.3 Diskussion der Simulationsergebnisse

3.3.1 Fahraufgabe

Bei der untersuchten Fahraufgabe handelt es sich um einen Linksabbiegevorgang an einer Ampelkreuzung unter Berücksichtigung des Gegenverkehrs. Das elektrische Fahrzeug nähert sich der Ampel mit einer Geschwindigkeit von 60 km/h, bremst aufgrund eines roten Ampelsignals in den Stillstand und wartet in einer Schlange von anderen Verkehrsteilnehmern auf ein grünes Ampelsignal. Während des Bremsvorgangs berechnet die Rekuperationskontrolle, dass eine reine Rekuperation nicht ausreichend ist, um das Fahrzeug rechtzeitig in den Stillstand zu bringen und beauftragt daher einen Eingriff der Reibbremsen mit einem berechneten Bremsmoment. Dieses Bremsmoment wird von der zweiten untersuchten Funktion auf die vier Reibbremsen verteilt. Sobald das grüne Ampelsignal erscheint, biegt das Fahrzeug unter Berücksichtigung des Gegenverkehrs links ab.

3.3.2 Vergleich MiL und SiL

Damit die Unterschiede zwischen der MiL und der SiL-Simulation sichtbar werden, wurde die Fahraufgabe mit beiden Modellen absolviert. In Abbildung 5 sind zwei Signalverläufe der beiden Simulationen zum Zeitpunkt des Bremsens vor der Ampelkreuzung gegeneinander aufgetragen. Im oberen Teil ist die Fahrzeuggeschwindigkeit v dargestellt. Da die Fahraufgabe in beiden Fällen exakt gleich ist und die Fahrzeuggeschwindigkeit vom Fahrdynamikmodell und nicht von den vECUs berechnet wird, sind in der Abbildung keine Unterschiede zwischen MiL- und SiL-Simulation zu erkennen. Bei der Betrachtung des Verlaufs der erforderlichen Bremskraft des vorderen rechten Rads trq_{FL}, die direkt von einer der beiden gekoppelten vECUs berechnet wird, werden dagegen kleine Abweichungen sichtbar. Diese sind auf busbezogene Effekte wie Kommunikationsverzögerungen und Quantisierungsfehler zurückzuführen, da die Funktionalität der vECUs in der SiL-Simulation exakt mit denen der ursprünglichen Funktionen in der MiL-Simulation übereinstimmt und der Einsatz eines virtuellen CAN-Bus zur Kopplung der vECUs in der SiL-Simulation der einzige Unterschied zwischen den Modellen ist.

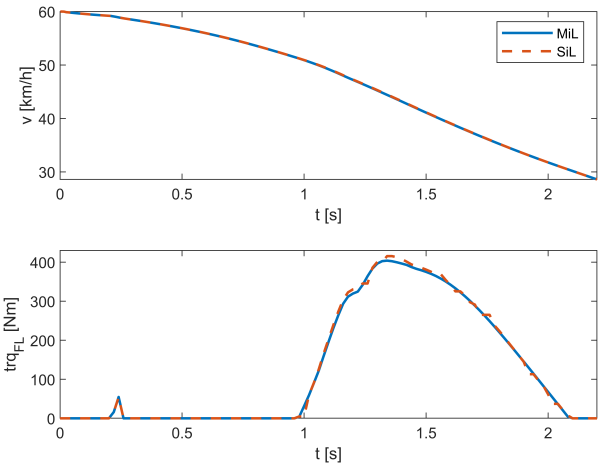

Abbildung 5 Vergleich von Simulationsergebnissen in der MiL- und SiL-Simulation

Kommunikationsverzögerungen Es muss zwischen zwei Arten von Verzögerungen unterschieden werden. Zum einen treten bei einer Co-Simulation ungewollte, künstliche Verzögerungen abhängig der Architektur und der Ausführungsreihenfolge der Teilmodelle auf und zum anderen werden in der SiL-Simulation reale Kommunikationsverzögerungen eines CAN-Bus simuliert.
In der MiL-Simulation der vorgestellten Fallstudie sind die beiden untersuchten Funktionen in zwei verschiedenen Simulink-Modellen implementiert, die parallel ausgeführt werden. Dabei ergibt beim Datenaustausch zwischen

den Funktionen eine Verzögerung von einem Makrozeitschritt h.
In der SiL-Simulation sind die untersuchten Funktionen in vECUs integriert, die ihr eigenes Zeitmanagement besitzen. Das Auslesen des Busses, Durchführen von Berechnungen und Schreiben von CAN-Nachrichten findet periodisch alle h Sekunden statt. Dies ist die kleinstmögliche Zeitverzögerung, die bei der Kommunikation zwischen den vECUs auftritt, da diese parallel ausgeführt und über Model.CONNECT miteinander synchronisiert werden. Zusätzlich muss die Kommunikationsverzögerung des CAN-Busses berücksichtigt werden. In einem CAN-Bus hängt die Kommunikationsverzögerung von der Bitrate des Busses sowie der Buslast ab, die wiederum mit der Anzahl der über den Bus übertragenen Nachrichten zunimmt [10]. In der vorgestellten SiL-Simulation werden nur zwei Nachrichten über den Bus übertragen (eine zwischen den vECUs und eine für die Restbussimulation via CANoe). Die unter diesen Voraussetzungen berechnete maximale Kommunikationsverzögerung des Busses ist deutlich geringer als die Ausführungsschrittweite der vECUs, wodurch der Bus hier im Vergleich zur MiL-Simulation keine zusätzliche Verzögerung in das System einbringen sollte. Da auch bei der Synchronisation der Simulationszeit des Busses und der vECUs über Model.CONNECT nicht garantiert werden kann, dass alle CAN-Nachrichten die vECUs während eines Zeitschrittes erreichen, können in der SiL-Simulation, z.B. durch hohe Systemauslastung, weitere Verzögerungen entstehen. Dies könnte verhindert werden, wenn ein neuer Simulationsschritt der vECUs erst beginnen darf, nachdem der Bus alle CAN-Nachrichten für diesen Simulationsschritt versendet hat. Dies ist Teil aktueller Arbeiten.

Quantisierungsfehler In jeder CAN-Nachricht können bis zu 64 Bit an Daten übertragen werden, welches genau einer Gleitkommazahl mit doppelter Genauigkeit entspricht. Um die Buslast eines CAN-Netzwerks gering zu halten, werden daher Signale in der Regel so quantisiert, dass sie möglichst wenig Daten verbrauchen und somit mehrere Signale in derselben CAN-Nachricht übertragen werden können. Für die Bremsmomentanforderung, die in der vorgestellten SiL-Simulation über den virtuellen CAN-Bus übertragen wird, werden 16 Bit Daten einer CAN-Nachricht verwendet. Dies ermöglicht unter Berücksichtigung des zulässigen Wertebereichs eine Genauigkeit des Signals von ca. $0,2\ Nm$. Dadurch wird verglichen mit der MiL-Simulation, bei der die Bremsmomentanforderug als deutlich genauere Gleitkommazahl zwischen den Funktionen ausgetauscht wird, ein kontinuierlicher Quantisierungsfehler induziert, der zu den angesprochenen Abweichungen in Abbildung 5 führt.

4 Zusammenfassung und Ausblick

In diesem Beitrag wird eine Simulationsumgebung vorgestellt, die einen nahtlosen Übergang von der MiL- zur SiL-Simulation einschließlich einer Bussimulation sowie einer Restbussimulation ermöglicht. Die Integration der vECUs erfolgt dabei über den offenen Standard FMI. Der Ansatz wurde mit vECUs mit Serienbasissoftware, aber vereinfachter Anwendungssoftware validiert.
In einem nächsten Schritt wird dieser Ansatz in einem industriellen Projekt getestet. Da dabei Serienapplikatonssoftware verwendet wird, wird sich die Buslast aufgrund der zusätzlichen Kopplungssignale erhöhen, wodurch mit größeren Unterschieden in den Simulationsergebnissen zwischen MiL und SiL zu rechnen ist.
Des Weiteren ergibt sich die Fragestellung wie sich die Co-Simulationsverzögerungen zwischen den Teilsystemen und die Kommunikationsverzögerungen im virtuellen Bus harmonisieren lassen. Es ist dabei zu beachten, das erstere ungewollte Effekte sind, die sich abhängig von Hyperparametern wie die Kopplungsschrittweite und Ausführungsreihenfolge der Teilsysteme beeinflussen lassen. Letztere hingegen bewusst modelliert sind, da sie reale Effekte abbilden, die für die virtuelle Freigabe von Fahrfunktionen berücksichtigt werden müssen.

5 Literatur

[1] Torsten Blochwitz et al. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proceedings of the 9th International MODELICA Conference; Munich; Germany*, 2012.

[2] Marcus Boumans et al. Consistent application of systems engineering and simulation for cross-domain function integration. 19. Stuttgarter Symposium, Robert Bosch GmbH, 2019.

[3] Ròbert Lajos Bücs et al. Virtual hardware-in-the-loop co-simulation for multi-domain automotive systems via the functional mock-up interface. In *Languages, Design Methods, and Tools for Electronic System Design*. Springer, 2016.

[4] Cláudio Gomes et al. Co-simulation: A survey. *ACM Computing Surveys (CSUR)*, 2018.

[5] Henrik Kaijser et al. Towards simulation-based verification for continuous integration and delivery.

[6] Thomas Kuhr et al. Feral - framework for simulator coupling on requirements and architecture level. In *11th IEEE/ACMInternational Conference on Formal Methods and Models for Codesign*, 2013.

[7] Lars Mikelsons and Roland Samlaus. Towards virtual validation of ecu software using fmi. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic*, 2017.

[8] Manuel Schiller and Alois Knoll. Emulating vehicular ad hoc networks for evaluation and testing of automotive embedded systems. In *SimuTools*, 2015.

[9] Dirk Von Wissel et al. Full virtualization of renault's engine management software and application to system development. *arXiv preprint*, 2018.

[10] Werner Zimmermann and Ralf Schmidgall. *Bussysteme in der Fahrzeugtechnik*. Springer, 2006.