Integrierte modell- und simulationsbasierte Entwicklung zur dynamischen Bewertung automobiler Elektrik/Elektronik-Architekturen

Zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEURS (Dr.-Ing.)

von der KIT-Fakultät für
Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)
genehmigte

DISSERTATION

von

M. Sc. Harald Bucher

geb. in Freiburg im Breisgau

Tag der mündlichen Prüfung:
28.07.2020

Hauptreferent: Prof. Dr.-Ing. Dr. h. c. Jürgen Becker
Korreferent: Prof. Dr. Bernhard Bauer

Integrierte modell- und simulationsbasierte Entwicklung zur dynamischen Bewertung automobiler Elektrik/Elektronik-Architekturen

1. Auflage: November 2020

Für meinen Vater Hermann.

Kurzfassung

Die Digitalisierung hat enormen Einfluss auf nahezu alle Industriezweige. Besonders die Automobilbranche befindet sich seit einigen Jahren im Wandel. Trends wie autonomes Fahren, Konnektivität, smarte Mobilität sowie die Elektrifizierung führen zu einer drastischen Erhöhung der Fahrzeugkomplexität. Diese Komplexität muss durch die zugrunde liegende Elektrik/Elektronik-Architektur (*E/E-Architektur*) und zunehmend durch Software beherrscht werden.

Während in klassischen E/E-Architekturen mehr Funktionalität durch neue Steuergeräte umgesetzt wurde, findet aktuell eine Konsolidierung der Funktionalitäten in einer zentralisierten Architektur statt. Die massive Änderung der E/E-Architektur und die immer stärkere, aber auch notwendige Entkopplung der Entwicklungsstränge rufen unmittelbar neue Herausforderungen an den Entwicklungsprozess hervor. Diese äußern sich einerseits in der Homogenisierung des Entwicklungsprozesses und der bisher sehr stark heterogen eingesetzten Methoden und Werkzeuge zur E/E-Architekturentwicklung. Andererseits ist die Einbindung von Simulation in frühen Entwicklungsphasen zentral.

Design-Entscheidungen der E/E-Architektur haben maßgeblichen Einfluss auf das Verhalten von Fahrzeugfunktionen und umgekehrt. Daher müssen sie möglichst frühzeitig analysiert und evaluiert werden, um kostspielige Fehlerkorrekturen in späten Entwicklungsphasen zu minimieren. Die modellbasierte Architekturentwicklung und die Simulation sind jedoch nach wie vor weitestgehend getrennt voneinander laufende Prozesse und führen oft zu langen und nicht handhabbaren Werkzeugketten. Dies erschwert bei Neu- oder Weiterentwicklungen eine effiziente Analyse und Bewertung der bidirektionalen Abhängigkeiten zwischen Architektur und Verhalten. Vor allem die Untersuchung mehrerer unterschiedlicher Architekturvarianten gestaltet sich dadurch problematisch.

In dieser Arbeit wird daher eine integrierte Methodik zur modell- und simulationsbasierten Entwicklung und Bewertung von E/E-Architekturen in frühen Entwicklungsphasen vorgestellt. Basierend auf den dafür abgeleiteten Anforderungen werden eine Reihe an Modellierungskonzepten zur integrierten, ausführbaren Verhaltensspezifikation in E/E-Architekturbeschreibungssprachen eingeführt. Die Konzepte stützen sich auf einen schichtenorientierten Ansatz, welcher die typischen Abstraktionsebenen einer E/E-Architektur ergänzt. Ebenenübergreifende Mechanismen, wie z. B. Mappings und AUTOSAR-basierte Schnittstellen, wer-

den eingeführt, um das Verhalten auf mehreren Abstraktionsebenen miteinander zu verknüpfen, sowie nicht-funktionale Architekturaspekte berücksichtigen zu können. Dabei kommen bewährte Paradigmen und Notationen wie *Actor*-orientierte Modelle, UML State Charts sowie elektrische Schaltungen zum Einsatz, die miteinander kombiniert werden können. Insbesondere die Actors können dabei aus einer wiederverwendbaren, ausführbaren Funktionsbibliothek instanziiert werden.

Ausgehend von der integrierten Architektur- und Verhaltensspezifikation wird komplementär dazu eine erweiterbare und portierbare Softwarearchitektur zur automatischen Simulationssynthese einer domänenübergreifenden und aspektorientierten Simulation entwickelt. Diese Synthese gestattet es, das zunächst realisierungsunabhängige Funktionsverhalten mit nicht-funktionalen und realisierungsnahen Aspekten der E/E-Architektur zu verschmelzen und zu untersuchen. Dazu zählen Netzwerkkommunikation, Ausführungszeiten und Scheduling von Funktionen oder Stromaufnahmen von Hardwarekomponenten. Zur Integration werden dazu geeignete Abbildungs- und Synthesevorschriften zwischen Architektur- und Simulationsmodell abgeleitet.

Die Konzepte werden in das E/E-Architekturwerkzeug PREEvision® integriert, umgesetzt und anhand mehrerer Anwendungsfälle evaluiert. Dieses in der Automobilindustrie etablierte Werkzeug war zuvor nicht in der Lage, als ausführbare Architekturspezifikation in Form einer ganzheitlichen Simulation über alle relevanten Abstraktionsebenen zu fungieren. Zur Simulation wird dabei das quelloffene, heterogene Modellierungs- und Simulationswerkzeug Ptolemy II verwendet. Basierend auf der integrierten Simulation wird zudem ein generisches Konzept namens *Architecture-Model-in-the-Loop (AMiL)* zur iterativen und metrikbasierten Evaluation von Architekturvarianten, insb. von simulationsbasierten, dynamischen Metriken, entwickelt. Realisiert wird dies durch zwei Ergänzungen, einer variantensensitiven Simulationssynthese sowie ein Konzept zur transparenten Rückführung der Simulationsdaten zur Assoziation mit dem ursprünglichen Architekturmodell. Evaluiert wird der Ansatz durch die Bewertung zweier Architekturvarianten hinsichtlich statischer Buslast und dynamischen Kommunikationslatenzen von funktionalen Wirkketten.

Hinsichtlich der verteilten Entwicklung und Wiederverwendung von Simulationsmodellen wird im Rahmen der Evaluation ein weiterer Anwendungsfall demonstriert. Dieser diskutiert die modellbasierte und iterative Einbindung domänenspezifischer Modelle mittels integrierter Co-Simulation von *Functional Mock-up Units (FMUs)* und rundet die Arbeit ab.

Abstract

Digitization has an enormous impact on almost all branches of industry. The automotive industry, in particular, has been in a state of flux for several years now. Trends such as autonomous driving, connectivity, smart mobility services, and the electrification of powertrains are leading to a drastic increase in vehicle complexity. This complexity must be mastered by the underlying electrical/electronic-architecture (*e/e-architecture*) and increasingly by software.

While in classical e/e-architectures, more functionality was achieved by new electronic control units, a consolidation of the functional features is currently taking place in a centralized architecture. The massive change in the e/e-architecture and the continuously intensifying, but also necessary decoupling of the developmental strands directly call for new challenges to the development process. On the one hand, these manifest in the homogenization of the development process and the methods and tools for e/e-architecture development. The latter have been used very heterogeneously to date. On the other hand, the incorporation of simulation in early development phases is crucial.

Design decisions of the e/e-architecture have a significant influence on the behavior of vehicle functions and vice versa. Hence, they must be analyzed and evaluated as early as possible to minimize costly error corrections in late development phases. However, model-based architecture development and simulation are still largely separately running processes and often lead to long and unmanageable toolchains. This separation makes it more difficult for new or further developments, to perform an efficient analysis and evaluation of the bidirectional dependencies between architecture and behavior. In particular, the investigation of several different architecture variants is thus problematic.

Therefore, this dissertation presents an integrated methodology for model- and simulation-based development and evaluation of e/e-architectures in early development phases. Based on the derived requirements, a series of modeling concepts for the integrated, executable behavior specification in e/e-architecture description languages will be introduced. These are based on a layered approach that complements the typical abstraction layers of an e/e-architecture. Cross-layer mechanisms, such as mappings and AUTOSAR-based interfaces, are introduced to link behavior at multiple abstraction layers and to consider non-functional architecture aspects. Proven paradigms and notations such as Actor-oriented

models, UML state charts and electrical circuits are used, which can be combined with each other. Especially the actors can be instantiated from a reusable, executable function library.

Based on the integrated architecture and behavior specification, an extensible and portable software architecture for the automatic simulation synthesis of a cross-domain and aspect-oriented simulation will be developed. This synthesis allows to merge and investigate the functional behavior, which is initially independent of the implementation, with aspects of the e/e-architecture that are non-functional and close to the implementation. These include network communication, execution times, and scheduling of functions or power consumption of hardware components. For integration, suitable mapping and synthesis rules between architecture and simulation models are derived.

The concepts are integrated into the e/e-architecture tool PREEvision®, implemented and evaluated based on several use cases. This tool, well-established in the automotive industry, was previously not able to function as an executable architecture specification in terms of a holistic simulation across all relevant abstraction layers. The open-source heterogeneous modeling and simulation tool Ptolemy II is used for the simulation. Based on the integrated simulation, a generic concept called Architecture-Model-in-the-Loop (AMiL) is developed for the iterative and metric-based evaluation of architecture variants, especially of simulation-based, dynamic metrics. This concept is introduced by two supplementing extensions that are a variant-sensitive simulation synthesis and a concept for the transparent feedback of simulation data together with the association to the original architecture model. The approach is examined by evaluating two architecture variants concerning static bus load and dynamic communication latencies of functional activity chains.

Regarding the distributed development and reuse of simulation models, a further use case will be demonstrated within the scope of the evaluation. It discusses the model-based and iterative integration of domain-specific models utilizing integrated co-simulation of FMUs which rounds off the work.

Vorwort

Die vorliegende Dissertation ist im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Technik der Informationsverarbeitung (ITIV) des Karlsruher Instituts für Technologie (KIT) entstanden. Die langjährige Zeit hat mich sowohl beruflich als auch privat sehr geprägt, in der ich viele wertvolle Erfahrungen sammeln und neue Kontakte knüpfen durfte. Viele Menschen haben mich auf diesem Weg begleitet, bei denen ich mich bedanken möchte.

Zunächst möchte ich mich besonders bei meinem Doktorvater Herrn Prof. Jürgen Becker für die Betreuung und die Möglichkeit bedanken, in seiner Arbeitsgruppe am ITIV über ein von mir gewähltes Forschungsthema zu promovieren. Des Weiteren bedanke ich mich bei Herrn Prof. Bernhard Bauer der Universität Augsburg für die Übernahme des Korreferats. Darüber hinaus möchte ich mich bei Prof. Becker dafür bedanken, in verschiedenen Forschungsprojekten und -vorhabensbeschreibungen mitarbeiten und Verantwortung übernehmen zu dürfen, welche über das Promotionsthema hinausgingen und ich so wertvolle Einblicke in weitere Themengebiete erlangen konnte. Dazu zählten auch selbstständig akquirierte Industriekooperationen, in denen ich insb. Kontakt zur Vector Informatik GmbH herstellen konnte. An dieser Stelle möchte ich mich ganz herzlich bei den Herren Dr. Clemens Reichmann und Mathias Supp für intensive Diskussionen und Anregungen rund um das Promotionsthema bedanken, welche zum Gelingen dieser Arbeit beigetragen haben.

Weiterhin möchte ich mich bei allen ehemaligen und aktuellen Arbeitskollegen am ITIV für die angenehme und stets abwechslungsreiche Arbeitsatmosphäre bedanken, in der der Spaß und einige Unternehmungen nach Feierabend nie zu kurz kamen. Ein Dank gilt Dr. Christoph Roth als ehemaliger Bürokollege, Simon Reder als geschätzter Kollege in einem gemeinsamen Forschungsprojekt sowie Steffen Bähr, der in der Endphase der Promotion mitgelitten hat. Zusätzlich möchte ich mich bei meinen betreuten Studierenden bedanken, die mit ihren Abschlussarbeiten einen Beitrag zur Dissertation geleistet haben.

Ein großes Dankeschön möchte ich meinen Freunden in der Heimat und in Karlsruhe aussprechen, die für die nötige Ablenkung und Abwechslung abseits des Alltags gesorgt haben. Hervorheben möchte ich an dieser Stelle Herrn Dr. Florian Salah als langjähriger Wegbegleiter, Berater und Freund. Bei wöchentlichen Grill-

und Fußballabenden mit feinsten Beef Cuts und einem kühlen Blonden wurde höchst kompetent über Sport- und Finanzwissenschaften diskutiert.

Nicht zuletzt möchte ich mich ganz besonders bei meiner Familie, aber auch bei Familie Schlenker und Limberger bedanken, die in schwierigen Zeiten stets hinter mir standen. Mein größter Dank gilt jedoch meiner Freundin Selina für ihre unermüdliche Geduld und ihr Verständnis. Ohne sie wäre ich heute nicht der, der ich bin und darf mich außerordentlich glücklich schätzen sie an meiner Seite zu haben.

Karlsruhe, im November 2020
Harald Bucher

Inhaltsverzeichnis

1 Einleitung

Die Automobilindustrie befindet sich seit einigen Jahren im Wandel. Neben Komfort und Infotainment halten immer mehr aktive Sicherheits- und Fahrerassistenzsysteme, sog. *Advanced Driver Assistance System (ADAS)*-Systeme, Einzug ins Automobil, z. B. Notfallbremsassistent, Spurhalteassistent oder adaptiver Abstandsregelautomat. Der Trend zielt auf das zukünftig autonome Fahren ab. Parallel dazu hat eine Umstellung auf Elektro- und Hybridantriebe stattgefunden sowie eine vermehrte Konnektivität mit der Umgebung und Einbindung von Mobilitäts- und IT-Diensten. Letztere umfassen eine Anbindung an die Cloud und Over-the-Air-Updates von Software. Dieser Wandel wird oft auch als *Autonomous Driving, Connectivity, Electrification and Smart Mobility (ACES)*-Trend bezeichnet und ist der Haupttreiber neuer Innovationen für Alleinstellungsmerkmale von Fahrzeugherstellern [25].

Der ACES-Trend und die Integration von immer mehr Funktionalität sowie neuer Technologien ins Fahrzeug führen zu einer starken Erhöhung der Fahrzeugkomplexität, die zunehmend durch Software und der zugrunde liegenden E/E-Architektur beherrscht werden muss [163]. Tatsächlich wird erwartet, dass vor diesem Hintergrund der Weltmarkt für Automotive Software und *Elektrik/Elektronik (EE)*-Systeme zwischen 2020 und 2030 von ca. 238 Milliarden auf ca. 469 Milliarden US-Dollar anwächst. Der Softwareanteil am gesamten Fahrzeug wird etwa 30 % betragen [26, S. 1] [25, S. 12].

Während in klassischen, verteilten Architekturen neue Funktionalität noch durch die Einführung neuer Steuergeräte sowie dazugehöriger Sensorik/Aktorik realisiert wurde, führt der Trend aktuell zur funktionalen Vernetzung mittels Konsolidierung und Integration von mehr Funktionalität in einem Steuergerät. Vor dem Hintergrund der funktionalen Sicherheit und des autonomen Fahrens werden zudem die Daten redundanter Sensoren in einem Steuergerät fusioniert. Voraussetzung dafür sind u. a. leistungsfähigere Rechenplattformen, breitbandige Buskommunikation wie bspw. Ethernet und intelligente Sensoren/Aktuatoren [5, 26, 163].

Abbildung 1.1 zeigt schematisch die Evolution der E/E-Architektur hin zu einer hierarchischen Domänen-Architektur (vierte Generation), bestehend aus mehreren sog. zentralen Domänen-Steuergeräten, z. B. für Infotainment und Fahrerassistenzsysteme, und einem zentralen Kommunikations-Gateway [25, S. 11].

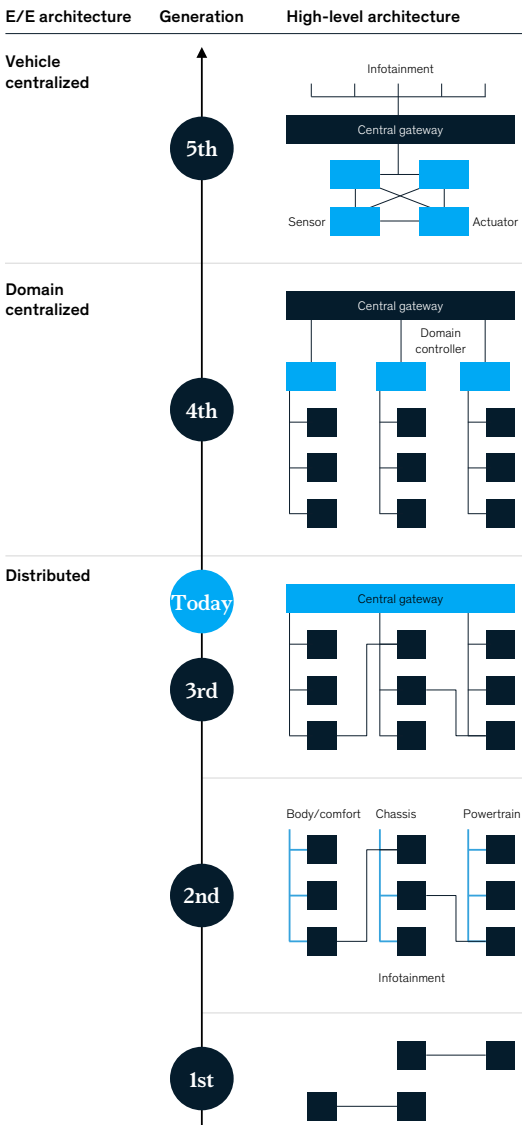


Abbildung 1.1: Evolution der E/E-Architektur von unabhängigen, funktionsspezifischen *Electronic Control Units (ECUs)* zu einer zentralisierten Architektur (Quelle: [25, S. 11]).

Diese Domänen-Architektur wird bereits von einigen Herstellern verfolgt [163]. Schließlich soll in der fünften Generation eine vollständig zentralisierte Architektur entstehen. Es wird erwartet, dass diese aus leistungsfähigen, intelligenten Sensoren/Aktuatoren und wenigen hochperformanten Rechenplattformen besteht, die mehrere Domänen virtuell integrieren. Ethernet wird sich als zentraler Kommunikations-Backbone etabliert haben.

Die verteilte Entwicklung spielt in diesem Zusammenhang zudem eine wichtigere Rolle, z. B. die Trennung von Hardware- und Software-Entwicklungszyklen, damit sowohl Fahrzeughersteller als auch Zulieferfirmen unabhängiger agieren können [5, 26]. Der Entwicklungsprozess besteht in dieser Parallelisierung aus einer sehr heterogenen Landschaft aus Methoden und Werkzeugen, Entwurfsrichtlinien und Arbeitsabläufen. Dies trifft auf alle Entwicklungsphasen zu, von der Analyse- über Konzept-, Design- bis zur Integrations- und Verifikationsphase [163].

1.1 Motivation

Die drastische Änderung der zugrunde liegenden EEA erfordert jedoch eine Homogenisierung dieses heterogenen Prozesses. Dazu zählt sowohl die transparente und durchgängige Modellierung der Anforderungen und der gesamten E/E-Architektur als auch die frühzeitige Einbindung von virtuellen Tests bzw. Simulationen [163].

Seit einigen Jahren hat sich die modellbasierte E/E-Architekturentwicklung besonders in der Automobilindustrie etabliert und wird im Zuge der Homogenisierung weiter an Bedeutung gewinnen. Zum einen, da durch die Formalisierung eine konsistente, gemeinsame Beschreibung zur Nachvollziehbarkeit über alle Abstraktionsebenen vorliegt. Andererseits, da durch die Formalisierung automatische Synthesen, Bewertungen und Optimierungen ermöglicht werden. Hinzu kommt die Variantenvielfalt, die sich ohne modellbasierte Ansätze und Werkzeugunterstützung nicht mehr handhaben lässt. Zum Beispiel lassen sich für einen "BMW 318i (E90) mit Komponenten wie beispielsweise Karosserieform, Motor, Getriebe, Abtrieb, Farbe, Klima, Infotainment etwa $160 \cdot 2^{70}$ Varianten" ableiten [180, S. 446]. Hinsichtlich der verteilten Entwicklung ist dabei die grafische Darstellung und die Kollaboration auf unterschiedlichen Abstraktionsebenen mit einer benutzerfreundlichen Oberfläche sowie die Zusammenführung als *single-source* Modell[1] in einer Datenbank von hoher Bedeutung. Diese beinhaltet auch die Versionierung

[1] Unter single-source ist zu verstehen, dass sämtliche Informationen eines Systems, Prozesses etc. auf ein und derselben Grundlage erstellt wurden und das zugrunde liegende Modell die einzige Quelle dieser Informationen darstellt.

und das Änderungsmanagement, um iterative Überarbeitungen integriert und synchronisiert zu unterstützen.

Auf der anderen Seite steht die Simulation bzw. das virtuelle Testen als weiterer, starker Trend im Entwicklungsprozess. Ralph Sundermeier, Leiter für CAE-Entwicklungsmethoden bei VW, prognostiziert: "bis 2025 wird jeder Ingenieur virtuelle Berechnungen durchführen" [135]. So ist sie eine komplementäre Testmethode, um kostenintensive, reale Testfahrten zu reduzieren und gefährliche oder unnachahmbare Grenzsituationen ohne Risiko simulieren zu können. Nach [180, S. 446] wurden für eine E-Klasse vor der Freigabe 36 Mio. Testkilometer absolviert; "allein die Freigabe eines aktuellen Fahrerassistenzsystems [bedarf] bis zu zwei Millionen Testkilometer" [180, S. 446]. Für ein *Electronic Stability Program (ESP)*-System ist heute schon die Simulation ein zugelassenes Hilfsmittel zur Freigabe [180, S. 447] [134]. Mittel- bis langfristig soll so eine digitale Homologation von automatisierten Fahrfunktionen als Teil der Freigabe stattfinden, um sowohl Kosten zu sparen als auch den Test- und Freigabeprozess zu verkürzen. Eine notwendige Voraussetzung dafür ist jedoch, dass die eingesetzten Simulationsmodelle gegen repräsentative Realszenarien nach einem genormten Regelwerk validiert werden. Dies stellt aktuell noch eine große Herausforderung dar und ist Gegenstand aktueller Forschung und Regulierungsgremien wie bspw. das Forum zur Harmonisierung von Fahrzeugregulierungen der *United Nations Economic Commission for Europe (UNECE)*. Eine generische und standardisierte Vorgehensweise inkl. vordefinierter Szenarien existieren noch nicht [189, 190].

Nicht nur vor dem Hintergrund der Freigabe ist Simulation eine sinnvolle Ergänzung. Hinsichtlich der Variantenvielfalt eines Fahrzeugs bzw. der zugrunde liegenden E/E-Architektur ist sie besonders wertvoll, um u. a. reproduzierbare Ergebnisse zum Vergleich von verschiedenen Systemvarianten zu erhalten. Gerade in frühen Entwicklungsphasen sind Simulationen und Architekturevaluationen als Entscheidungshilfe maßgebend bei der Neu- oder Weiterentwicklung von E/E-Architekturen und deren Komponenten. Dies reduziert sowohl das Entwicklungsrisiko als auch die daraus entstehenden Kosten in späteren Entwicklungsphasen bei nicht entdeckten Designfehlern und beschleunigt den Entwicklungsprozess [2, 153, 179].

Um die verschiedenen Domänen wie Funktionsverhalten, Umgebung, Kommunikation etc. in der Simulation zu berücksichtigen, kommen auch hier nach [163] sehr heterogene, domänenspezifische Modelle und dazugehörige Simulationswerkzeuge zum Einsatz. Soll das Gesamtsystem ganzheitlich simuliert werden, ist eine Kopplung der Modelle bzw. der Werkzeuge und entsprechende Schnittstellen zwischen diesen notwendig. Zur Wiederverwendung von bestehenden Modellen sind dazu "u. U. werkzeugspezifische Adaptionen oder gar Neuentwicklungen notwendig" [142] und es entstehen oft sehr fragile und schwer wartbare Werk-

zeugketten, die zudem zu nicht-deterministischem Verhalten führen können [80]. Alternativ dazu existieren formale Modellierungs- und Simulationswerkzeuge, die implizit eine Kopplung bzw. Co-Simulation von heterogenen Modellen gestatten, aber die Wiederverwendung von bestehenden Modellen erschweren. Zudem haben sich in den letzten Jahren standardisierte Schnittstellen zum Austausch und zur Co-Simulation von Modellen etabliert, besonders die *Functional Mock-up Interface (FMI)*-Schnittstelle [116] hat vermehrt Zuspruch gefunden, um den o. g. Herausforderungen der Werkzeugkopplung entgegenzuwirken und die Wiederverwendung von Modellen zu erhöhen. Die Entkopplung der Entwicklungsstränge war dabei analog zum Trend der verteilten Entwicklung von Hardware und Software ebenso ein Haupttreiber, um bspw. unabhängig von spezifischen Modellzulieferern eine Systemintegration vornehmen zu können und umgekehrt die Zulieferer lediglich gegen die spezifizierte Schnittstelle testen müssen.

Aktuell sind die modellbasierte Architekturentwicklung sowie die simulative Evaluation von Funktionen und Architekturaspekten jedoch weitestgehend getrennt voneinander laufende Prozesse. Dabei hat die E/E-Architektur maßgeblichen Einfluss auf das Verhalten von Funktionen und umgekehrt [153]. Daher bedarf es einer stärkeren Einbettung der modell- und simulationsbasierten Entwicklung in den Entwicklungsprozess sowie eine Vereinfachung der Handhabung beider Teilprozesse. Dafür ist eine integrierte, werkzeuggestützte Spezifikation eines architekturzentrierten Verhaltens als auch eine integrierte Ausführung der Simulation zur Wiederverwendung der Ergebnisse essenziell, um diese zu visualisieren oder hinsichtlich wichtiger Kennzahlen weiter verarbeiten zu können.

Durch einen derart integrierten Ansatz werden sowohl E/E-Architektur, Verhalten als auch Simulations- und Evaluationsergebnisse in sich konsistent abgelegt. Gleichzeitig werden bewährte Abstraktionsebenen, Perspektiven und Modellierungskonzepte beibehalten und um neue Sichten der Simulation im Kontext einer ausführbaren Architekturspezifikation erweitert. Die Einbettung in ein und dieselbe modellbasierte Entwicklungsumgebung eröffnet zudem das Potenzial zur Wiederverwendung bestehender Designs und deren flexible, iterative Verfeinerung durch eine kollaborative, verteilte Entwicklung am single-source Modell auf unterschiedlichen Abstraktionsebenen. Zusätzlich gestattet die verzahnte modell- und simulationsbasierte Entwicklung in einer solchen Umgebung neue Möglichkeiten, verschiedene Architekturvarianten gegenüberzustellen, welche nicht nur statische Attribute und Metriken berücksichtigt, sondern auch auf den Simulationsdaten basierende, dynamische Metriken sowie die Kombination aus beiden.

Mithilfe der Verknüpfung mehrerer Abstraktionsebenen der Architektur und des Verhaltens, kann bereits in frühen Phasen eine automatisch synthetisierte und do-

mänenübergreifende Simulation bereitgestellt werden, die die einzelnen Aspekte auf der jeweiligen Abstraktionsebene sowie nicht-funktionale Eigenschaften berücksichtigt. In Kombination mit dem modellbasierten Variantenmanagement stellt dies eine erhebliche Produktivitätssteigerung und Fehlervermeidung bei der iterativen, verteilten Entwicklung von Architektur und ausführbarem Verhalten dar und berücksichtigt zudem die Kollaboration an ein und demselben Modell. Damit lassen sich bspw. folgende Fragestellungen untersuchen, wobei diese durch iterative Einbindung von weiteren Abstraktionsebenen und nicht-funktionalen Architektureigenschaften zu unterschiedlichen Entwicklungsständen verfeinert werden können:

- Wie verhält sich die logische Funktionsarchitektur?
- Partitionierung der Funktionen auf die vernetzte Hardwarearchitektur
  - Welche Bussysteme treten zwischen Funktionen auf und welchen Einfluss hat die Kommunikation auf das Funktionsverhalten?
  - Wie wirkt sich die Interaktion der Funktionen mit verschiedenen Betriebsmodi des Steuergeräts auf das Verhalten aus?
  - Wie sieht die applikationsspezifische Stromaufnahme der involvierten Hardwarekomponenten in den dynamischen Betriebsmodi aus?
  - Wie können geteilte Ressourcen, bspw. in Form von Rechenkernen auf einem Mikroprozessor, berücksichtigt werden und welchen Einfluss hat z. B. das Scheduling der Funktionen auf das Verhalten?
- Detaillierte elektrische und leitungssatzsensitive Simulation
  - Welchen Einfluss haben detaillierte elektrische Schaltungen auf das Verhalten einer dazugehörigen Funktion auf logischer Ebene?
  - Welche (ohmschen) Leitungsverluste treten abhängig von den Verlegewegen durch die Topologie über Versorgungs- und Masseleitungen bei statischen und dynamischen Stromverbrauchern auf?
- Iterative Bewertung von Architekturvarianten bzgl. (quasi-)statischer und dynamischer Metriken
  - Welche Buslasten treten in Abhängigkeit der Funktionspartitionierung auf den involvierten Bussystemen auf?
  - Welche Kommunikationslatenzen ergeben sich für funktionale Wirkketten in Abhängigkeit von ihrer Partitionierung und wie unterscheiden sich verschiedene Architekturvarianten bzgl. dieser Metriken?

1.2 Zielsetzung und Abgrenzung der Arbeit

Ziel dieser Arbeit ist es Fragestellungen, wie in Abschnitt 1.1 skizziert, möglichst frühzeitig in der Konzept- und Designphase der E/E-Architekturentwicklung in einer iterativen und kollaborativen Herangehensweise unter Berücksichtigung bewährter Konzepte zu eruieren. Dafür muss ein modellbasierter Ansatz zur Beschreibung einer E/E-Architektur gewählt werden, der innerhalb dieser Arbeit so erweitert werden soll, dass eine verzahnte modell- und simulationsbasierte Entwicklung entsteht und die Machbarkeit der Integration zeigt. Somit werden lange Werkzeugketten oder -brüche gezielt vermieden und zur Homogenisierung des Entwicklungsprozesses beigetragen. Ein weiteres Hauptziel ist dabei die Ankopplung an einer möglichst vollständigen E/E-Architekturbeschreibungssprache sowie die prototypische Umsetzung und Integration der Methoden und Konzepte in ein E/E-Architekturwerkzeug. Mithilfe der integrierten Betrachtungsweise kann eine durchgängige und nahtlose Untersuchung über alle für die Fragestellungen relevanten Abstraktionsebenen gewährleistet werden.

Voraussetzung dafür ist eine integrierte und ausführbare Verhaltensspezifikation. Zur Steigerung der Wiederverwendung und zur Erleichterung der Modellierung sollen bewährte Paradigmen und Notationen unterstützt sowie verknüpft werden können. Um die verschieden involvierten Domänen und auch nicht-funktionalen Architekturaspekte der obigen Fragestellungen berücksichtigen zu können, sind abstraktionsebenenübergreifende Mechanismen notwendig. Diese Mechanismen erlauben es das Verhalten in Bezug zur Architektur zu setzen und umfassen zusätzlich Automatismen, welche den Anwender beim Modellieren unterstützen und die verfügbaren Modellinformationen weitestgehend versteckt in ein ausführbares Simulationsmodell überführen. Eine detaillierte Diskussion der notwendigen Anforderungen wird in Kapitel 4 vorgestellt.

Konkret beschäftigt sich die vorliegende Dissertation mit folgenden Forschungsfragen:

- Wie können ausführbare Verhaltensspezifikationen in E/E-Architekturbeschreibungssprachen integriert modelliert und darin bewährte Paradigmen und Notationen für Verhalten miteinander verknüpft werden? Wie können die Modellierungskonzepte übertragbar und erweiterbar gehalten werden?
- Welche Möglichkeiten bestehen, um Architekturaspekte und nicht-funktionale Eigenschaften mit Verhalten auf unterschiedlichen Abstraktionsebenen miteinander zu verknüpfen? Wie kann dies in ein einheitliches Simulationsmodell überführt und die Heterogenität der Berechnungsmodelle zusammengeführt werden?

- Welche Charakteristika muss ein Simulator aufweisen, um gleichzeitig eine domänenübergreifende Simulation der in Abschnitt 1.1 genannten Aspekte sowie Anforderungen wie integrierte Co-Simulation, Wiederverwendbarkeit und Erweiterbarkeit erfüllen zu können?
- Wie sieht eine portierbare, erweiterbare und konfigurierbare Softwarearchitektur sowie der architekturzentrierte Modellierungsprozess zur automatischen Synthese eines solchen Simulationsmodells aus?
- Wie können die Simulationsergebnisse in der Entwicklungsumgebung visualisiert und mit den Architekturartefakten assoziiert werden?
- Wie kann eine iterative und automatisierte Evaluation von Architekturvarianten bzgl. statischen und dynamischen Metriken erreicht werden?

Diesen und den in Abschnitt 1.1 präsentierten Fragestellungen werden im Rahmen dieser Arbeit durch eine ganzheitliche, integrierte Methodik und dazugehörigen Werkzeugen gegenübergetreten. Der daraus entstehende Forschungsprototyp dient als Erweiterung und Framework im industriell etablierten E/E-Architekturwerkzeug PREEvision® [2] [175].

Die Verwendung als industriell produktreifes Werkzeug ist dabei explizit nicht Ziel. Weiterführende Phasen im Entwicklungsprozess betreffend die Umsetzung konkreter Hardware/Software-Prototypen und Integrationstests sowie die eingangs erwähnte Validierung bzw. Kalibrierung der Simulationsmodelle durch *Proving Ground* Realszenarien sind explizit nicht Ziel und Teil dieser Arbeit. Auch die Lösung aller Herausforderungen bei der modell- und simulationsbasierten E/E-Architekturentwicklung in den einzelnen Themenfeldern ist im Rahmen einer Dissertation nicht möglich.

1.3 Aufbau der Arbeit und Beiträge

Im folgenden Kapitel 2 werden zunächst die für diese Dissertation wichtigsten Grundlagen kurz beschrieben. Diese umfassen im Wesentlichen Themen zum Fahrzeugentwicklungsprozess und zum Entwurf von automobilen E/E-Architekturen, die modellbasierte Entwicklung und Simulation von heterogenen eingebetteten Systemen sowie den in dieser Arbeit eingesetzten Simulator Ptolemy II.

In Abb. 1.2 ist eine Übersicht der weiteren Inhalte und Beiträge dieser Dissertation dargestellt.

[2] https://www.vector.com/de/de/produkte/produkte-a-z/software/preevision, zuletzt aufgerufen am 12.12.2019.

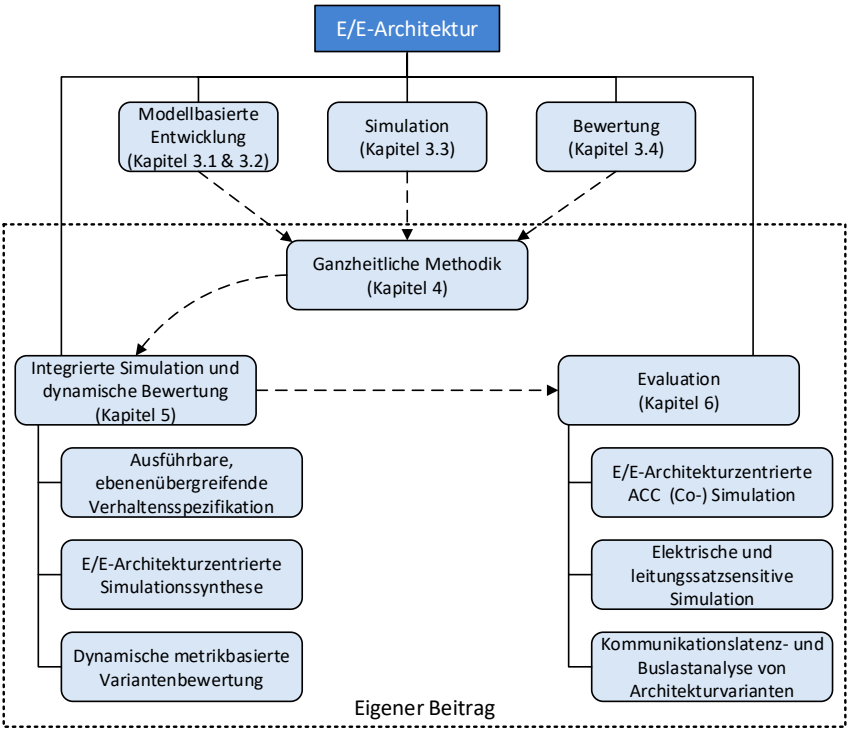


Abbildung 1.2: Übersicht der Arbeit und Beiträge.

Kapitel 3 beleuchtet und diskutiert den aktuellen Stand der Technik und Forschung und gliedert sich in drei Teile: (1) Methoden, Sprachen und Werkzeuge zur modellbasierten Entwicklung von eingebetteten Systemen mit Fokus auf E/E-Architekturen, (2) Simulations- und Testmethoden von E/E-Architekturen, insb. in frühen Entwicklungsphasen und (3) die Bewertung von E/E-Architekturvarianten. Aufgrund der prototypischen Umsetzung sämtlicher in dieser Arbeit entwickelten Methoden und Konzepte im etablierten E/E-Architekturwerkzeug PREEvision®, wird dieses in einem separaten Abschnitt ausführlich behandelt. Dies beinhaltet vor allem die zugrunde liegenden Metamodelle der Architekturbeschreibungssprache, die für die abgeleiteten Abbildungs- und Synthesevorschriften in Kapitel 5 herangezogen werden.

In Kapitel 4 werden basierend auf den drei behandelten Themenfeldern in Kapitel 3 eine Gesamtmethodik vorgestellt. Diese adressiert sowohl die identifi-

zierten Lücken in den einzelnen Bereichen, aber insb. vereint sie diese zur einer integrierten, verzahnten modell- und simulationsbasierten Entwicklung von E/E-Architekturen sowie zu deren dynamischen Bewertung. Schließlich werden systematisch die daran gestellten Anforderungen abgeleitet.

Kapitel 5 stellt zunächst ein Gesamtkonzept vor, das den aufgestellten Anforderungen an eine integrierte, verzahnte modell- und simulationsbasierte E/E-Architekturentwicklung gerecht wird. Dazu wird zunächst eine Reihe an Modellierungskonzepten zur ebenenübergreifenden, ausführbaren Verhaltensspezifikation entwickelt und der Bezug zum Architekturmodell hergestellt. Die Konzepte werden prototypisch in der etablierten modellbasierten Entwicklungsumgebung PREEvision® umgesetzt und integriert. Darauf aufbauend werden Konzepte zur Interpretation des zugrunde liegenden E/E-Architekturmodells und zur Synthese eines aspektorientierten, heterogenen Simulationsmodells vorgestellt, welches das spezifizierte Verhalten und die Architekturaspekte abstraktionsebenenübergreifend miteinander verschmilzt. Insbesondere die Abbildungs- und Synthesevorschriften werden detailliert erläutert. Basierend auf der integrierten Simulation wird anschließend ein generisches Konzept zur iterativen und metrikbasierten Evaluation von Architekturvarianten präsentiert. Dazu zählt auch ein Konzept zur transparenten Rückführung und die Assoziation der Simulationsdaten mit den ursprünglichen Architekturartefakten. Auch diese Konzepte werden anhand einer ausführbaren Simulations- und Analysemetrik innerhalb des Werkzeugs PREEvision® prototypisch umgesetzt.

Die Evaluation der Ansätze und Ergebnisse dieser Arbeit werden in Kapitel 6 anhand mehrerer Anwendungsfälle demonstriert und diskutiert. Hinsichtlich der Steigerung der verteilten Entwicklung und Wiederverwendung von Modellen, wird zudem ein Anwendungsfall demonstriert, welcher die modellbasierte und iterative Einbindung domänenspezifischer Modelle mittels integrierter Co-Simulation von FMUs diskutiert. Das Kapitel schließt mit einer ganzheitlichen Betrachtung bzgl. der abgeleiteten Anforderungen, einer Diskussion der Übertragbar- und Erweiterbarkeit sowie einer Einordnung in verwandte Arbeiten.

Kapitel 7 fasst die Ergebnisse und Schlussfolgerungen dieser Dissertation zusammen und gibt einen Ausblick auf mögliche aufbauende Arbeiten.

2 Grundlagen

2.1 Entwicklung automobiler E/E-Architekturen

Bedingt durch die immer weiter steigende Komplexität von *E/E-Architekturen*, hervorgerufen durch den ACES-Trend, den wachsenden Anforderungen an funktionale Sicherheit und der extremen Variantenvielfalt, haben sich Vorgehensweisen und Entwicklungsmethoden in der Automobilindustrie etabliert, um diese Komplexität zu beherrschen. Vor diesem Hintergrund werden im Folgenden einige wichtige Grundlagen dargelegt.

2.1.1 Entwicklungsprozesse und Vorgehensweisen

2.1.1.1 Phasen einer Fahrzeugentwicklung

Die Entwicklung eines Fahrzeugs und ihre zugrunde liegende E/E-Architektur durchläuft wie jedes Produkt einen gewissen Produktentstehungsprozess, der sich nach [159, S. 7 ff.] in vier Phasen aufteilen lässt und vereinfacht dargestellt für jede Baureihe zeitlich überlappend zueinander startet:

1. Die **Produktstrategie** bestimmt basierend auf Marktforschungen potenzielle Kunden, definiert die kundenerlebbaren Funktionen, die das Fahrzeug enthalten soll und berücksichtigt gesetzliche Rahmenbedingungen und Standards.
2. Während der **Technologieentwicklung** wird die E/E-Architektur abgeleitet, die sowohl alle definierten funktionalen Innovationen realisieren kann und zudem nicht-funktionale Eigenschaften wie Kosten erfüllt. Die E/E-Architektur wird entweder von bestehenden Architekturen abgeleitet oder als neue Plattform aufgebaut, die in beiden Fällen für Fahrzeuge im hohen Preissegment angesiedelt sind, um im Falle von Rückrufaktionen die Gesamtkosten durch die kleinere Stückzahl zu reduzieren.
3. Die Phase der **Fahrzeugentwicklung** beschäftigt sich mit dem Aufbau erster Prototypen bzw. Piloten, um einzelne Komponenten und am Ende das

Fahrzeug in Erprobungsfahrten zu validieren. Die Werkzeugentwicklung für die spätere Serienproduktion beginnt ebenfalls in dieser Phase.

4. Der **Verkaufsstart** ist die letzte Phase und beschäftigt sich mit sämtlichen zur Serienproduktion notwendigen Aktivitäten wie der Planung der Fließbandproduktion sowie der Verifizierung und Validierung sämtlicher Systeme und Funktionalitäten und der finalen Validierung des Fahrzeugs mit Erprobungsfahrten. Die Phase startet meist parallel zur Fahrzeugentwicklung.

2.1.1.2 V-Modell

Um die Komplexität während der Fahrzeugentwicklung zu beherrschen, werden Entwicklungsprozesse zur Systementwicklung (engl. Systems Engineering) eingesetzt. In der Automobilbranche hat sich das *Vorgehensmodell (V-Modell)* etabliert [151, 154, 159], das die Systementwicklung in mehrere Entwicklungs- und dazugehörige Testschritte unterteilt, siehe Abb. 2.1. Automotive *Software Process Improvement and Capability Determination (SPICE)* dient dabei als Prozessreferenz und Richtlinie zur Prozessbewertung. Ein guter Überblick über alternative Entwicklungsprozesse und Prozessbewertungsmodelle sowie die Berücksichtigung von funktionaler Sicherheit nach ISO 26262 ist in [1] zu finden.

Im linken Ast befinden sich die einzelnen Entwicklungsschritte, deren Detailgrad von oben nach unten immer weiter zunimmt, bei der Analyse der Systemanforderungen beginnt und über die Spezifikationen des Systems und deren Komponenten mit der Implementierung bzw. Umsetzung letzterer endet. In jedem Schritt werden dabei zusätzlich Testfälle festgelegt, die im rechten Ast von unten nach oben die Erfüllung der Spezifikation im jeweiligen Schritt abprüfen. Die Wechselwirkung zwischen Spezifikation und Test kann iterativ erfolgen, bis alle Anforderungen im jeweiligen Schritt erfüllt und alle Tests positiv ausgefallen sind [154].

Der erste Entwicklungsschritt der Analyse von Systemanforderungen beinhaltet die Identifikation und Analyse von kundenerlebbaren Funktionen, die das Fahrzeug enthalten soll sowie die dazugehörigen funktionalen und nicht-funktionalen Anforderungen, die sie erfüllen sollen/müssen. Auf Basis dieser Analyse wird eine Systemspezifikation erstellt, die nach [154] auch als Spezifikation der *logischen Systemarchitektur* bezeichnet wird. Hierbei werden noch keine konkreten technischen Realisierungsentscheidungen getroffen, sondern ein logisches Funktionsnetzwerk aufgebaut sowie deren Schnittstellen und auszutauschenden Signale für das gesamte Fahrzeug oder ein Subsystem spezifiziert. Die logische Architektur bildet die Grundlage zur Zerlegung des Systems in Hardware- und Softwarekomponenten sowie deren Spezifikation. Dieser Schritt wird nach [154]

auch als Spezifikation der *technischen Systemarchitektur* bezeichnet und beinhaltet sowohl die Zuweisung, welche Funktion in Software und auf welchem Steuergerät ausgeführt wird als auch den Vergleich von verschiedenen technischen Realisierungsalternativen.

Die Spezifikation der technischen Systemarchitektur beinhaltet somit die Anforderungen an den nachfolgenden Entwurfsschritt und bildet bedingt durch die verteilte Wertschöpfungskette im Fahrzeugentwicklungsprozess typischerweise die Schnittstelle zwischen Fahrzeughersteller (engl. *Original Equipment Manufacturer (OEM)*) und Zulieferer.[1] Die Spezifikation kann dabei bereits ausführbare Spezifikationen bzw. Simulationsmodelle für den Zulieferer enthalten, auf deren Basis die Anforderungen an den *Hardware/Software Architekturentwurf* analysiert werden. Nach der Spezifikation des Architekturentwurfs folgt die Spezifikation der einzelnen Komponenten (*Hardware/Software Komponentenentwurf*) ohne Berücksichtigung von Implementierungsdetails, welche schließlich im letzten Schritt *Hardware/Software Implementierung* zum Tragen kommen (bspw. die Berechnung in Ganzzahlarithmetik).

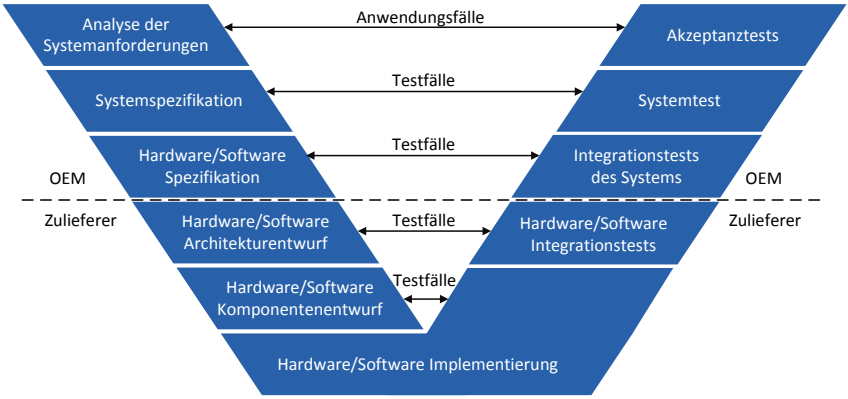


Abbildung 2.1: Entwicklungs- und Testschritte zum Entwurf von E/E-Architekturen nach dem V-Modell (Quelle: eigene Darstellung nach [159, S. 10] und [154, S. 26]).

Anschließend folgt der Übergang zur Verifikation/Validierung bzw. zum Test, beginnend beim Komponententest von Hard- und Software. Hard- und Softwarekomponenten werden anschließend integriert und zusammen auf einem elektronischen Steuergerät mithilfe von Integrationstests getestet. Der Hersteller

[1] Für eine Übersicht über verschiedene Zuliefererebenen (engl. *tier*) vgl. [159, S. 1 ff.].

erhält die getesteten Komponenten vom Zulieferer zurück, integriert diese mit den übrigen Komponenten wie Sensoren/Aktuatoren, die zunächst auch virtuell vorliegen können, und testet diese mit Integrationstests des Systems gegen die technische Systemarchitektur. Es folgt der Systemtest gegen die Systemspezifikation bzw. der logischen Systemarchitektur und schließlich der Akzeptanztest gegenüber der Systemanforderungen mithilfe von Anwendungsfällen, die zur Abnahme des Fahrzeugs dienen.

2.1.2 Abstraktionsebenen von E/E-Architekturen

Zur strukturierten Beschreibung und Entwicklung einer E/E-Architektur entlang des Entwicklungsprozesses hat sich ein Schichtenmodell etabliert. Das Modell erlaubt es die unterschiedlich involvierten, aber verzahnten Fachdisziplinen vom Produktmanager über Entwickler in ihren Spezialbereichen (z. B. Systemarchitekt, Softwareentwickler etc.) bis hin zum Kunden strukturiert in einzelnen Ebenen zu beschreiben. Jede Ebene stellt dabei eine eigene Perspektive bzw. Sicht auf die E/E-Architektur dar und erleichtert damit das Verständnis innerhalb der jeweiligen Disziplin. Darüber hinaus werden Abhängigkeiten zwischen den Ebenen modelliert, um eine durchgängige Nachvollziehbarkeit der Artefakte auf den einzelnen Ebenen zu erlangen. Nach [159] werden vier Systemebenen unterschieden und sind in Abb. 2.2 illustriert:

1. Der **Funktionsumfang** spezifiziert und listet alle kundenerlebbaren Funktionen, die das Fahrzeug enthalten soll und besteht i. d. R. aus mehreren Merkmalen. Funktionen können untereinander Abhängigkeiten aufweisen, um Fahrzeugvarianten wie Basis- und Sonderausstattung zu berücksichtigen. Weiterhin ist jede Funktion typischerweise mit technischen und/oder nicht-technischen Anforderungen (z. B. gesetzliche Rahmenbedingungen) gebunden.
2. Auf der **Funktions-/Softwarearchitektur** werden die einzelnen Funktionen durch ein Funktionsnetzwerk typischerweise bestehend aus Sensoren, Funktionen, Aktoren sowie deren Datenaustausch modelliert. Die Funktionsblöcke werden zunächst realisierungsunabhängig aus logischer Sicht modelliert und über Abbildungskanten mit den Funktionen der ersten Ebene verknüpft, um die realisierende Komponente festzulegen.
3. Auf der **Vernetzungsarchitektur** werden die logischen Funktionen auf die zur Verfügung stehenden Hardwarekomponenten (Sensoren, ECUs, Aktoren) über weitere Abbildungskanten zugewiesen. Die logischen Datenabhängigkeiten werden über konventionelle Verbindungen oder Bussysteme in einem Kommunikationsnetzwerk realisiert. Des Weiteren müssen die

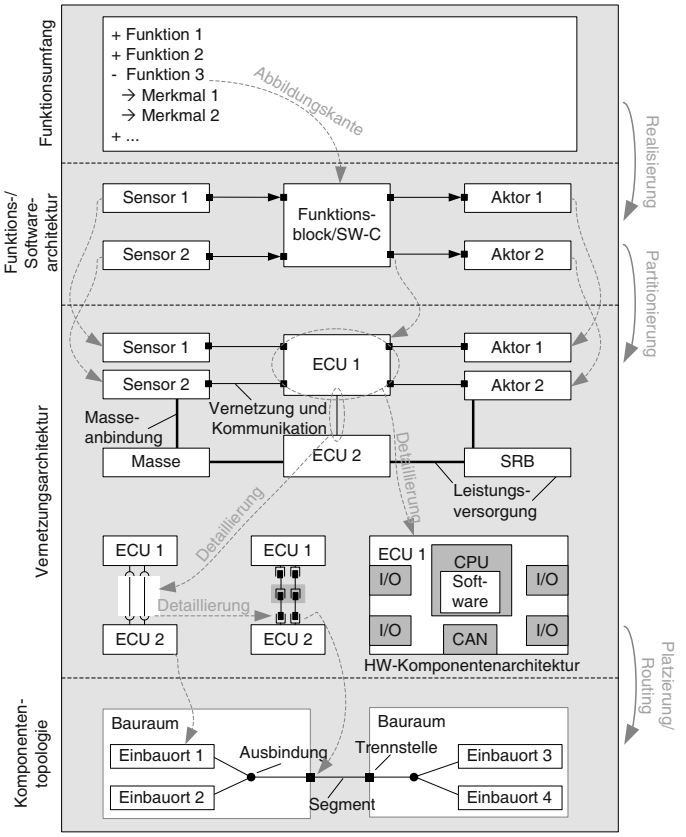


Abbildung 2.2: Abstraktionsebenen einer E/E-Architektur (Quelle: [159, S. 16].

Hardwarekomponenten mit elektrischer Leistung versorgt werden. Dies wird in einem Modell zur *Leistungsversorgung* beschrieben. Die logischen Vernetzungsverbindungen werden weiterhin in elektrische Verbindungen und physikalische Leitungen im *Leitungssatz* detailliert. Das Innenleben von Hardwarekomponenten bzw. Steuergeräten, das typischerweise Mikrocontroller, Speicherelemente sowie analoge Anteile und Treiberbausteine für Bus- oder sonstige Schnittstellen enthält, wird über eine separate *Hardwarekomponentenarchitektur* beschrieben.

4. Die **Komponententopologie** beschreibt die Geometrie des Fahrzeugs, in der „Hardwarekomponenten der Vernetzungsarchitektur auf Bauräume abgebildet und die Leitungen für die Vernetzung, Kommunikation und Leistungsversorgung entlang vorgegebener Verlegewege geroutet“ werden [159, S. 17].

Eine detaillierte Beschreibung der einzelnen Ebenen und ebenenübergreifender Mechanismen werden in Abschnitt 3.2 anhand des Entwurfswerkzeugs PREEvision® gegeben.

2.1.3 Softwareentwicklung mit AUTOSAR

Automotive Open System Architecture (AUTOSAR)[2] ist ein Konsortium bestehend aus Automobilherstellern, Zulieferern und weiteren Unternehmen, das die Entwicklung von Steuergeräte-Software mittels einer standardisierten *Softwarearchitektur* und *Entwurfsmethodik* sowie *Applikationsschnittstellen* verbreiteter Anwendungsdomänen wie bspw. Komfortfunktionen vorantreibt [188, Kap. 7]. Das Konsortium wurde 2003 gegründet und besteht heute aus den Kernpartnern Bosch, der BMW Group, Continental, Daimler, Ford, General Motors, PSA Group, Toyota und Volkswagen. Ziele von AUTOSAR sind u. a. die Beherrschung der Komplexität in Software und der E/E-Architektur durch die steigende Anzahl an Funktionsintegration im Fahrzeug, die Skalierbarkeit und Übertragbarkeit von Softwarefunktionen auf verschiedene Fahrzeugplattformen, die Wiederverwendbarkeit von Software, Verkürzung der Entwicklungszyklen und Senkung des Entwicklungsrisikos sowie der Kosten [8].

2.1.3.1 Konzept

Das zentrale Konzept von AUTOSAR ist die Entwicklung der Anwendungssoftware unabhängig von der zugrunde liegenden Steuergeräte-Infrastruktur. Eine Anwendung wird dabei über formale Softwarekomponenten (engl. *Software Components (SWCs)*) spezifiziert. Die virtuelle Integration von SWCs auf System- bzw. Fahrzeugebene erfolgt durch den sog. *Virtual Function Bus (VFB)*, über den alle Anwendungen miteinander über eindeutige Schnittstellen kommunizieren, sowohl auf einem Steuergerät als auch über Steuergeräte hinweg. Die Schnittstellen der SWCs zum VFB sind dedizierte Ports, die sowohl die Kommunikation der Softwarekomponenten untereinander als auch mit der Basissoftware des Steuergeräts abwickeln. Der VFB entkoppelt somit Anwendungen von ihrer späteren Realisierung auf unterschiedlichen Steuergeräten. Dies erhöht massiv die

[2] https://www.autosar.org/, zuletzt aufgerufen am 12.06.2018.

Wiederverwendung von Anwendungssoftware z. B. in unterschiedlichen Fahrzeugproduktlinien. In einem späteren Entwicklungsschritt, wenn die SWCs auf Steuergeräte verteilt werden, wird der VFB durch eine für ein Steuergerät spezifisch konfigurierte Laufzeitumgebung und Basissoftware realisiert [188, Kap. 7] [172].

2.1.3.2 Softwarearchitektur

Die Softwarearchitektur von AUTOSAR unterteilt sich in drei Schichten: eine hardwareunabhängige *Applikationsschicht*, einer *Laufzeitumgebung* (*Runtime Environment (RTE)*) und die hardwareabhängige *Basissoftware* des Mikrocontrollers (*Basic Software (BSW)*). Die Schichten sind schematisch in Abb. 2.3 illustriert.

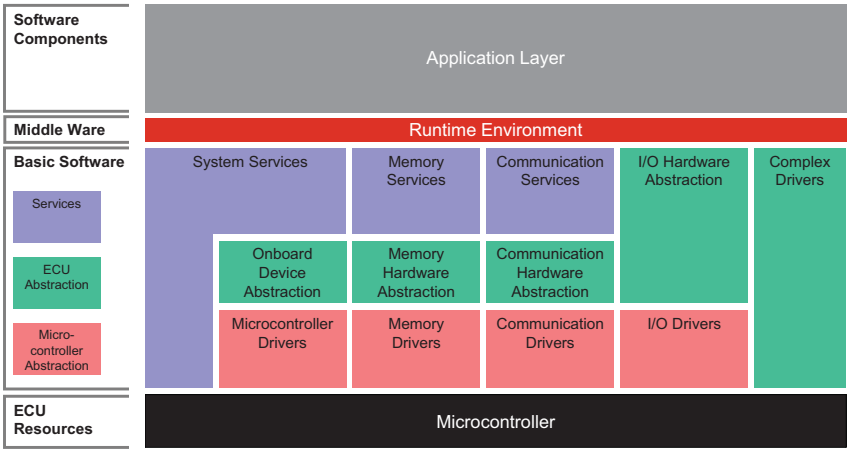


Abbildung 2.3: AUTOSAR Softwarearchitektur (Quelle: [188, S. 110]).

Die Applikationsschicht besteht aus einer Menge an Softwarekomponenten, die die Anwendungsfunktionalität des Steuergeräts definieren und ist weitestgehend unabhängig von der zugrunde liegenden Hardware.

Die Entkopplung der SWCs von der Hardware bzw. der Basissoftware geschieht über die Laufzeitumgebung, die standardisierte Schnittstellen zu Services der Basissoftware bereitstellt und für die Kommunikation zwischen SWCs sorgt. Sämtliche Kommunikation der Applikationen erfolgt über die RTE, unabhängig davon, ob sich diese auf demselben Steuergerät oder auf einem anderen befinden.

Die Laufzeitumgebung wird oft auch als Middleware bezeichnet [188, Kap. 7] [9, 173].

Die Basissoftware unterteilt sich in drei weitere Abstraktionsebenen und stellt eine Menge an standardisierten Modulen zur Verfügung um eine standardisierte Steuergerätesoftware zwischen OEM und Zulieferern bereitzustellen und so die Applikationen unabhängig und austauschbar entwickeln zu können [9, 173] [188, Kap. 7]:

1. Mikrocontroller Abstraktionsebene: abstrahiert von der Mikrocontroller-Hardware und stellt über standardisierte Schnittstellen Dienste für die ECU-Abstraktionsebene bereit, bspw. „Treiber für den Zugriff auf Speicher, Kommunikation und Input/Output (IO) des Mikrocontrollers“ [173].
2. ECU Abstraktionsebene: abstrahiert von den zur Verfügung stehenden Ressourcen auf dem Steuergerät wie Peripherie zur Kommunikation oder Speicher und deren Verbindung zu Mikrocontrollern. Sie bietet einen einheitlichen Zugriff auf diese Ressourcen aus oberen Schichten. Gleichzeitig ist diese Schicht unabhängig vom Mikrocontroller.
3. Service Abstraktionsebene: stellt Kommunikationsdienste, Systemdienste, wie z. B. Betriebsmodi der ECU, und Speicherdienste für die Anwendungsschicht direkt über die RTE zur Verfügung. Das Betriebssystem ist ebenfalls Bestandteil dieser Ebene.

Des Weiteren können komplexe Gerätetreiber für Funktionen existieren, die außerhalb der Standard-Basissoftware-Module liegen und bspw. applikationsspezifische Hardwareeinheiten ansprechen müssen.

AUTOSAR stellt für die Kommunikation zwischen SWCs und RTE *AUTOSAR Interfaces* bereit, die Datentypen, Skalierungsfaktoren, Einheiten sowie die Semantik für verschiedenste Anwendungsdomänen wie Chassis, Innenraum und Komfort, Antriebsstrang, Fahrwerk, sowie Insassen- und Fußgängerschutz spezifizieren [188, Kap. 7]. Darüber hinaus existieren zwei weitere standardisierte Interface-Typen: (1) das *Standardized AUTOSAR Interface* zur Kommunikation zwischen SWCs und AUTOSAR Services, „die von BSW-Modulen des Service Layers wie dem ECU State Manager oder dem Diagnostic Event Manager bereitgestellt werden [174]“; (2) das *Standardized Interface* als C-*Application Programming Interface (API)*, z. B. zur Kommunikation zwischen Basissoftware-Modulen untereinander oder zwischen RTE und Betriebssystem. [174]

2.1.3.3 Entwurfsmethodik

Aufgrund der verteilten Entwicklung und Wertschöpfungskette zwischen OEMs und Zulieferern wurde neben der Softwarearchitektur eine standardisierte Entwurfsmethodik spezifiziert. Diese definiert sowohl für die Softwarekomponenten als auch deren Integration in Steuergeräte sowie für die Konfiguration von Steuergeräten und deren Vernetzung über verschiedene Bussysteme formale AUTOSAR Templates auf *eXtensible Markup Language (XML)*-Basis. Die Templates enthalten alle notwendigen Informationen um diese maschinell verarbeiten und zwischen Werkzeugen austauschen zu können und werden in den verschiedenen Entwicklungssträngen der Anwendersoftware, RTE, BSW sowie zur ECU- und Systemkonfiguration angewandt. Die Templates werden auch dazu verwendet, um auf Basis der SWC-Beschreibungen den vom Steuergerät abhängigen Code für die RTE und die BSW zu generieren. Ein Überblick über die beschriebene Methodik ist in Abb. 2.4 zu finden.

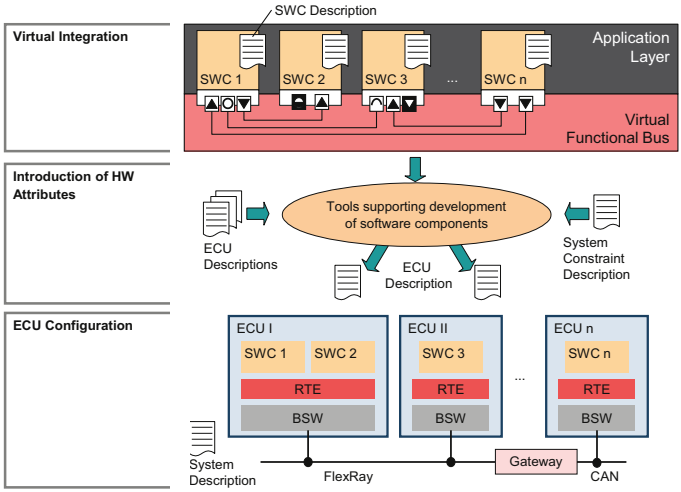


Abbildung 2.4: AUTOSAR Entwurfsmethodik (Quelle: [188, S. 112]).

2.1.3.4 AUTOSAR Adaptive

Bei der klassischen Vorgehensweise von AUTOSAR wurden Softwarekomponenten statisch auf Steuergeräte verteilt und deren Vernetzung zur Entwurfszeit

mittels Konfigurationen festgelegt. Im Bereich von echtzeitkritischen Anwendungen ist dies auch sinnvoll und wird weiterhin praktiziert. Bedingt durch den Paradigmenwechsel eines abgeschlossenen Systems hin zu einem offenen Fahrzeugsystem, welches mit der Umwelt interagiert, steht eine dynamische Anpassung bzw. die Update-Fähigkeit von Softwarekomponenten im Vordergrund. Treiber dieser Entwicklung sind im Wesentlichen der ACES-Trend und Over-the-Air-Aktualisierung von Software [177].

Aus diesem Grund wurde ein neuer, mit der klassischen Plattform kompatibler Standard für eine *Adaptive Platform*[3] spezifiziert. Dieser bietet einen objektorientierten Ansatz sowie eine serviceorientierte Kommunikation und ein auf POSIX basiertes Betriebssystem. Dies erlaubt es, Anwendungen zur Laufzeit auf beliebige Steuergeräte zu migrieren, sog. *Adaptive Applications* [177].

2.1.4 Modellbasierte Sprachen zum Systementwurf

Zur Beherrschung der Komplexität von heutigen E/E-Architektur werden diese wie in Unterabschnitt 2.1.2 beschrieben in mehrere Abstraktionsebenen unterteilt. Dieses Vorgehen folgt zum einen dem Prinzip „teile und herrsche“ und ist andererseits Bestandteil des etablierten modellbasierten Systems Engineering (engl. *Model-Based Systems Engineering (MBSE)*) Ansatzes [32] zur Entwicklung von komplexen Systemen. Dabei wird das System auf jeder Ebene mit formalen Modellen beschrieben und jede Ebene stellt eine spezifische Sicht auf das System dar. Je nach Systemebene wird in verschiedenen Granularitätsstufen und Umfang modelliert.

Im Kontext von E/E-Architekturen existieren zwei Möglichkeiten zur Beschreibung: *universelle Modellierungssprachen* oder *domänenspezifische Sprachen* (engl. *Domain-Specific Languages (DSLs)*) [38]. Zu den universellen Modellierungssprachen zählt u. a. die *Systems Modeling Language (SysML)* [32, 126], welche ein Derivat der *Unified Modeling Language (UML)* [127] und der Standard zur Modellierung von beliebigen Systemen ist. Ein Vorteil der SysML ist, dass bestehende Modellierungswerkzeuge für die UML wiederverwendet werden können, da sie durch UML-Profile bzw. Stereotypen erweiterbar sind. Dies bedeutet im Gegenzug, dass die erweiterten Konstrukte in UML-Diagramme abgebildet werden müssen. Gleichzeitig ist das ein Nachteil, weil domänenspezifische Elemente zunächst in softwareorientierte UML-Konstrukte übersetzt werden müssen [167]. Zudem sind durch die grafische Annotation der Stereotypen bei größeren Modellen sehr textlastige Diagramme zu erwarten, was schnell zur Unübersichtlichkeit führt. Diese Art von grafischer Darstellung ist somit nicht optimal zur Visualisierung von architekturspezifischen Komponenten und verhindert weitestgehend auch die

[3] https://www.autosar.org/standards/adaptive-platform/, zuletzt aufgerufen am 14.06.2018.

unterschiedliche Sichtweise auf diese in den verschiedenen Abstraktionsebenen sowie eine ebenenübergreifende E/E-Architekturmodellierung und -darstellung nach Abb. 2.2 [102].

Eine DSL unterscheidet sich dadurch, dass die Sprache auf eine spezielle Domäne eines Systems zugeschnitten ist und die für ihre Fachdomäne verwendeten Begriffe und Komponenten verwendet. Nach Fowler ist eine DSL „a computer programming language of limited expressiveness focused on a particular domain" [38]. Die Darstellung kann dabei entweder textuell oder grafisch erfolgen.

Der Vorteil einer DSL gegenüber universellen Sprachen ist, dass sie eine höhere Abstraktion als die universellen Sprachen bieten, indem das Problem mit spezifischen Editoren (textuell oder grafisch) und Sprachelementen direkt abgebildet werden kann, ohne dass das Problem vorher in die universelle Sprache übersetzt werden muss. Die Einschränkung der Sprachelemente erhöht zudem die Konsistenz eines Modells inhärent. Oft reicht zur Modellerstellung schon das Domänenwissen des Anwenders aus, ohne Programmier- oder Modellierungsexperte sein zu müssen. Die Modelle sind daher übersichtlich, leicht verständlich und sind zum großen Teil selbst dokumentierend [167]. Aus den genannten Gründen folgt eine viel höhere Produktivität als es mit universellen Sprachen der Fall wäre [162].

Als Nachteil ist anzusehen, dass der initiale Entwicklungs- und Einarbeitungsaufwand höher ist als bei universellen Sprachen, jedoch wird die Produktivität insgesamt gesteigert. Die Pflege der domänenspezifischen Sprache ist meist Fachleuten vorbehalten und nur schwer von Entwicklern außerhalb der Domäne zu warten [167].

Bei der EAST-ADL (siehe Unterabschnitt 3.1.1) oder der EEA-ADL (siehe Abschnitt 3.2), für welche mit PREEvision® ein Werkzeug zur grafischen Editierung zur Verfügung steht, handelt es sich um DSLs.

2.2 Modellbasierte Entwicklung und Simulation heterogener Systeme

Der Einsatz von Modellen bzw. die modellbasierte Entwicklung (engl. *Model-Based Design (MBD)* [161, 60]) und dessen Simulation ist heutzutage ein etablierter Ansatz zur frühzeitigen Entwicklung und zum Test von eingebetteten Systemen. Modelle spielen eine zentrale Rolle und bilden die Grundlage zur Spezifikation, Simulation, Synthese bzw. Code-Generierung und Verifikation von Systemen. Die frühzeitige Anwendung im Entwicklungsprozess kann zur Aufdeckung von Entwurfsfehlern führen, bevor kostspielige Prototypen aufgebaut werden. Dies

steigert zum einen die Produktqualität und führt außerdem zu einer signifikanten Reduktion von Kosten und Entwicklungszeit und damit die time-to-market [140, 22].

Heterogene, eingebettete Systeme und deren Weiterentwicklung, die als *Cyber-Physical Systems (CPSs)* bezeichnet werden, beziehen eine Reihe an Domänen mit in die Entwicklung dieser Systeme ein. Ein CPS besteht aus mehreren eingebetteten Systemen mit Hardware-/Softwarekomponenten, die über Sensoren und Aktuatoren physikalische Prozesse steuern und untereinander über ein Netzwerk kommunizieren [82]. Eine E/E-Architektur kann demnach als eine Art CPS aufgefasst werden.

Ein CPS integriert verschiedene Disziplinen wie Regelungstechnik, Mechanik, Sensorik/Aktorik, Netzwerkkommunikation sowie Elektronik und Software in die Entwicklung. In jeder Disziplin werden typischerweise domänenspezifische Werkzeuge angewandt, deren individuelle Integration meist nicht effektiv ist [80] (siehe auch Unterabschnitt 3.3.2.1 für eine weitere Diskussion hinsichtlich heterogener Simulation).

Die modellbasierte Entwicklung hat sich in einigen dieser Disziplinen etabliert. Dazu gehören bspw. Werkzeuge wie Modelica [114] zur Modellierung und Simulation von komplexen, physikalischen Systemen, MATLAB/Simulink [104] zur Entwicklung, Simulation und Code-Generierung von Regelungssystemen, LabVIEW von National Instruments für den Entwurf von Messtechnik-Systemen oder die Modellierung und Simulation von Kommunikationsnetzwerken, z. B. mit OMNeT++ [169] oder ns-3[4]. Viele modellbasierte Werkzeuge folgen dabei einem Actor-orientieren Ansatz [86] [136, S. 18]. Actors sind ausführbare, nebenläufige Softwarekomponenten, die über *Ports* miteinander kommunizieren.

In den folgenden Abschnitten werden einige Grundlagen zum Verständnis der modellbasierten Entwicklung und Simulation gegeben. Eine detaillierte Behandlung von Modellbildung und Simulation ist im Rahmen dieser Arbeit nicht möglich, für weiterführende Informationen, insb. im Automotive Kontext, sei an dieser Stelle auf einschlägige Literatur wie [65] verwiesen.

2.2.1 Begriffe in der Modellierung und Simulation

In der Literatur existieren mehrere Definitionen der Begriffe *Modell*, *Simulation* als auch *Modellierung und Simulation*. Letzteres ist nach [94] eine eigenständige Disziplin zur Entwicklung und/oder Anwendung von Modellen und Simulationen.

[4] http://www.nsnam.org, zuletzt aufgerufen am 04.12.2018.

Nach [94] ist ein *Modell* eine „physikalische, mathematische oder anderweitig logische Repräsentation eines Systems, Entität, Phänomens oder Prozesses“. Die *Modellierung* beschreibt dabei den Prozess zur Erstellung eines Modells und dient zur Abstraktion bzw. Vereinfachung des real zu betrachtenden Systems, um ein tieferes Verständnis darüber zu erlangen. Das resultierende Modell soll dabei für eine gute Approximation möglichst viele relevante Eigenschaften des realen Systems abbilden, gleichzeitig aber nicht zu komplex sein, damit es für den Anwender verständlich und analysierbar bleibt. Weiterhin kann zwischen *statischen* (z. B. 3D-Modelle einer Maschine) und *dynamischen* Modellen, d. h. wie sich ein System über die Zeit verhält, unterschieden werden [94, 100, 136].

Unter einer *Simulation* versteht man die Ausführung und Experimentierfähigkeit eines Modells über die Zeit mit dem Ziel, das Verhalten des realen Systems zu imitieren [100]. Simulationen werden typischerweise durch computergestützte Werkzeuge realisiert und dienen zur dynamischen bzw. simulativen Verifikation von Systemen und ihren Attributen. Sie werden angewandt, bevor die realen Systeme, bspw. ein Steuergerät, zur Verfügung stehen oder neue Funktionen in ein existierendes Steuergerät integriert werden sollen. Die Simulationsmodelle werden in einem Simulationswerkzeug mit einer Modellierungssprache spezifiziert, die durch drei Komponenten charakterisiert werden kann [136, S. 5]:

1. **Syntax**: In welcher Notation wird ein Modell dargestellt?
2. **Semantik**: Welche Bedeutung hat das Modell und wie funktioniert es?
3. **Pragmatik**: Wie wird das Modell angewandt, editiert, analysiert etc.?

Insbesondere die Semantik ist wichtiger Bestandteil eines Modells, sowohl von statischen als auch dynamischen. Sie sorgt für eine eindeutige, unmissverständliche Interpretation eines Modells innerhalb einer Domäne und wird deshalb auch als *Semantische Domäne* bezeichnet. Dabei darf eine Semantik nicht mit der Spezifikation von Verhalten gleichgesetzt werden [47]. Die strukturelle Sicht auf ein System ist nicht weniger wichtig wie dessen Verhalten und beide Sichten benötigen eine Semantik. Im Bereich der objektorientierten Softwareentwicklung benötigt auch ein statisches Klassendiagramm der UML [127] eine Semantik, um zu verstehen, was jedes syntaktische Element bedeutet. In diesem konkreten Fall z. B. eine Klasse mit ihren Attributen und Methoden und ihre Beziehung zu anderen Klassen. Diese Abbildung von syntaktischen Elementen zu dessen Bedeutung (engl. *Semantic Mapping*) wird daher als *statische Semantik* angesehen ist essenziell bei der Spezifikation einer Modellierungssprache [47].

Lee et al. unterscheiden in [136] weiter zwischen *starker* und *weicher* Semantik (engl. *strong semantics*, *weak semantics*). Modelle mit starker Semantik weisen eine klare, unmissverständliche Bedeutung auf, wobei Modelle mit weicher Semantik Raum zur Interpretation lassen. Eine starke Semantik ist insbesondere bei der

automatisierten Analyse von Modellen oder bei Simulationsmodellen wichtig, die deterministisch ausführbar sein oder mit Modellen aus anderen Domänen gekoppelt werden sollen (sog. heterogene Modelle, siehe Unterabschnitt 3.3.2.1). Ein Beispiel für eine weiche Semantik sind Blockdiagramme, ohne den Verbindungen zwischen den Blöcken eine konkrete Bedeutung zu geben, d. h. ohne zu spezifizieren, welche Art von Interaktion zwischen den Blöcken konkret stattfindet. Beispielsweise eine synchrone/asynchrone Kommunikation oder eine diskrete/-kontinuierliche Interaktion. Dazu zählt die Modellierungssprache SysML [126], ein Derivat der UML zur Systemmodellierung. Der Mehrwert von Notationen mit weicher Semantik liegt daher eher im informellen Austausch von Entwürfen oder Konzepten zwischen Entwicklern innerhalb einer Domäne. Eine weiche Semantik ist daher vergleichbar mit einer statischen Semantik.

Der formale Regelsatz einer starken Semantik wird nach [86] als *Berechnungsmodell* (engl. *Model of Computation (MoC)*) bezeichnet.

2.2.2 Berechnungsmodelle

Ein Berechnungsmodell oder MoC spezifiziert einen formalen Regelsatz, der einem Modell eine klare, unmissverständliche Bedeutung verleiht. Es kann von einer bestimmten oder einer Klasse von Modellierungssprachen repräsentiert werden. Unterschiedliche Berechnungsmodelle können dabei mehr oder weniger gut für die Abbildung und Analyse bestimmter Charakteristika eines Systems geeignet sein und werden deshalb auch oft als Domäne oder semantische Domäne bezeichnet (siehe Unterabschnitt 2.2.1). Der formale Regelsatz lässt sich in drei Kategorien einteilen, die folgende Fragen beantworten [136, S. 7]:

1. Was ist die Bedeutung einer Komponente?
2. Welcher Ausführungs- und Nebenläufigkeitsmechanismus existiert zwischen Komponenten?
3. Welcher Kommunikationsmechanismus existiert zwischen Komponenten?

Ein Simulationsmodell wird von einem Simulator nach diesen Regeln ausgeführt, die für alle erstellten Modelle in dieser Domäne gleichermaßen gelten. Aufgrund der verschiedenen Domänen, die sie repräsentieren können, wird im folgenden Abschnitt eine mögliche Klassifikation dargestellt.

2.2.2.1 Klassifikation

Wie in Abschnitt 2.2 eingeführt ist ein heterogenes, eingebettetes System durch die Verzahnung der Domänen digitale Hardware/Software, (Netzwerk-) Kommu-

nikation sowie ihre Wechselwirkung mit physikalischen Prozessen charakterisiert. Diese können jeweils mit ein oder mehreren MoCs mehr oder weniger präzise beschrieben werden. Zu den bekanntesten und gängigsten MoCs zählen hierbei z. B. *Synchronous Data Flow (SDF)* [84] für digitale Signalverarbeitung und Streaming-Applikationen, *Discrete Event (DE)* für komplexe Systeme wie digitale Hardwarearchitekturen oder Kommunikationsnetzwerke [198], *Finite State Machines (FSMs)* für sequenzielle Softwarebeschreibungen und *Continuous-Time (CT)* für physikalische Prozesse. Diese sind auch in Abb. 2.5 zu finden, die eine mögliche Klassifikation von Berechnungsmodellen darstellt. Dabei ist zu sehen, dass diese unterschiedlichen Kategorien zugeordnet werden können, dessen Abstraktion von oben nach unten immer weiter abnimmt, d. h. ihre Regeln immer weiter verfeinert und somit spezifischer einer Domäne zugeschnitten werden.

Nach Abb. 2.5 kann z. B. zunächst unterschieden werden, ob die Ausführung von und Kommunikation zwischen Komponenten *sequenziell* (engl. sequential) oder *nebenläufig* (engl. concurrent) abläuft. Nebenläufige MoCs können entweder *zeitbehaftet* oder *zeitlos* sein oder eine Mischform darstellen. Das SDF Datenflussberechnungsmodell ist typischerweise zeitlos, kann aber bspw. für eine periodische Ausführung erweitert werden [136], siehe Abb. 2.5.

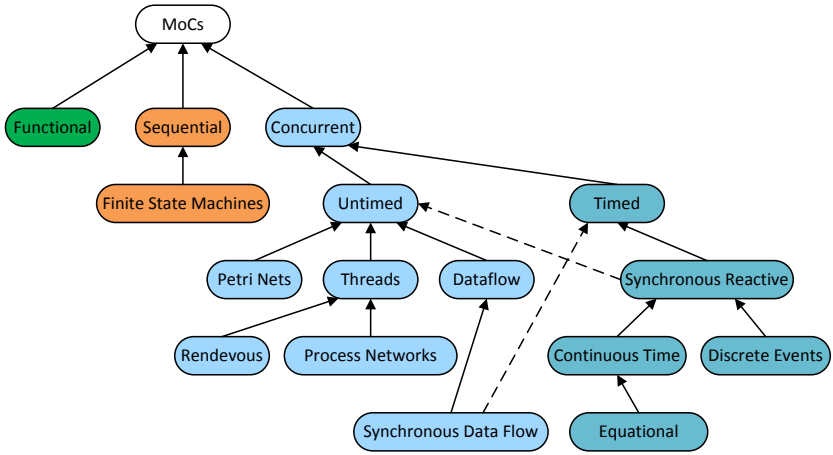


Abbildung 2.5: Klassifikation von MoCs (Quelle: eigene Darstellung nach [136, S. 24]).

Zeitbehaftete Berechnungsmodelle sind insb. zur Modellierung und Simulation von heterogenen eingebetteten Systemen wichtig, um eine Analyse hinsichtlich ihrer Reaktionszeit bzgl. einer Echtzeitanforderung auf einen bestimmten Ein-

gangsvektor durchführen zu können. Die Grundlage zur Spezifikation von Zeit ist ein entsprechendes Zeitmodell, das zusammen mit dem Ausführungs- und Kommunikationsmechanismus beschreibt, wie die Nebenläufigkeit dargestellt wird und wie die Ausführungsreihenfolge von Komponenten definiert ist. Vor allem für die eindeutige Darstellung von Gleichzeitigkeit (engl. simultaneity) bzw. von gleichzeitigen Aktionen während der Ausführung ist ein kohärentes Zeitmodell notwendig und ist daher die Grundlage für eine deterministische Ausführung [136]. Ein solches Zeitmodell ist bspw. die *Superdense Time* [99], die auch vom Ptolemy II Simulator verwendet und auf die in Unterabschnitt 2.3.2.3 genauer eingegangen wird.

2.2.2.2 Discrete-Event Systeme

Diskrete, ereignisbasierte Berechnungsmodelle (DE) zählen zu den ältesten und weitestverbreiteten zur Simulation von komplexen Systemen in unterschiedlichen Anwendungsdisziplinen [198]. Über die Jahre sind unterschiedliche Varianten in einer Vielzahl von Werkzeugen und Sprachen zur Modellierung und Simulation von technischen Systemen entstanden. Dazu zählen bspw. auch die Systembeschreibungssprache SystemC [53] oder die Hardwarebeschreibungssprache VHDL [51].

Ein erster DE Formalismus wurde von Zeigler entwickelt und als *Discrete Event System Specification (DEVS)* [197] bezeichnet. Dieser Formalismus erweitert im Grunde Moore Automaten zu zeitbehafteten Moore Automaten, in dem jeder Zustand eine gewisse, diskrete Zeitspanne verstreichen lässt, bevor die Reaktion des Zustands auf seine Eingänge für andere Zustände sichtbar wird und somit der Automat fortschreitet [88].

Das Grundprinzip von DE MoCs sind zeitdiskrete Interaktionen zwischen Komponenten mithilfe von Ereignissen (engl. *Events*). Events tauchen nur zu einem bestimmten, diskreten Zeitpunkt auf und signalisieren eine Änderung des Zustands einer Komponente. Sie werden als Reaktion auf eintreffende Events erzeugt und verbundenen Komponenten oder derselben Komponente zur Verfügung gestellt. Jedes Event ist dabei mit einem Zeitstempel ausgestattet, der einem bestimmten Zeitmodell folgt (bspw. der Superdense Time) und werden in zeitlicher Reihenfolge abgearbeitet. Die Simulation eines DE Modells entspricht also der Abarbeitung einer Sequenz an zeitdiskreten Events.

Für die Ausführung eines DE Modells werden die Zeitstempel chronologisch entsprechend dem zugrunde liegenden Zeitmodell sortiert und in einer *Event Queue* abgelegt. Dazu sind zunächst initiale Events notwendig, die von jeder Komponente in der Event Queue platziert werden können und für andere Komponenten oder sich selbst bestimmt sind. Ein Simulator wählt das Event mit dem kleinsten

Zeitstempel, extrahiert die auf dieses Event sensitiven Komponenten und stellt das Event an den Eingängen dieser Komponenten zur Verfügung. Als Reaktion der Zielkomponenten können weitere Events an deren Ausgängen erzeugt, die entweder denselben oder einen größeren Zeitstempel haben und in der Event Queue platziert.

Erst wenn alle Events mit demselben Zeitstempel abgearbeitet sind, schreitet die Simulationszeit zum Event mit dem nächstgrößeren Zeitstempel in der Queue voran. Die Zeitspanne zwischen zwei Events in der Queue ist dabei direkt vom modellierten Netzwerk abhängig und kann theoretisch beliebig groß sein[5]. Diese Zeitspanne wird dann einfach übersprungen und reduziert somit Scheduling- und Kommunikationsaufwand. Die Simulation terminiert, sobald der Zeitstempel mit der maximal spezifizierten Simulationszeit erreicht wurde oder keine Events mehr in der Queue vorhanden sind.

Die Art der Reaktion der Komponenten auf ihre Eingänge variiert zwischen DE MoCs. DEVS fordert demnach wie oben beschrieben, dass jeder Zustand Events an seinen Ausgängen erzeugt, die einen größeren Zeitstempel als die empfangenen Events an den Eingängen aufweisen. Andere Varianten erlauben eine sofortige Reaktion (engl. *instantaneous*), d. h. sie produzieren Events ohne Zeitverzögerung und weisen somit denselben Zeitstempel wie die empfangenen Events auf [88, 136]. Dies erfordert jedoch zusätzliche Mechanismen, um ein deterministisch ausführbares Modell zu erhalten. Siehe dazu die DE Variante des Ptolemy II Simulators in Unterabschnitt 2.3.3.1.

2.2.2.3 Continuous-Time Systeme

Continuous-Time Berechnungsmodelle werden verwendet, um - im Gegensatz zu DE Systemen - kontinuierliche Signalverläufe über der Zeit darzustellen und dienen als Basis zur Modellierung und Simulation von dynamischen, physikalischen Prozessen. Daher sind sie besonders geeignet, um die physikalische Komponente von heterogenen, eingebetteten Systemen bzw. CPSs abzubilden.

Die Simulation von kontinuierlichen Problemen ist ein komplexes und lang studiertes Feld [27] und würde den Rahmen dieser Arbeit sprengen. Jedoch sollen nachfolgend einige wichtige Grundlagen aufgezeigt werden.

Physikalische Prozesse werden oft durch ein System aus gewöhnlichen Differentialgleichungen (engl. *Ordinary Differential Equations (ODEs)*) modelliert. ODEs können wie folgt ausgedrückt werden:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t), \tag{2.1}$$

[5] Begrenzt durch die numerische Repräsentation des Zeitmodells.

wobei $\mathbf{x}(t)$ ein Vektor an bekannten differentiellen Variablen, auch Zustandsvektor (engl. *state vector*) genannt, und t die unabhängige, skalare Zeitvariable sind sowie $\dot{\mathbf{x}}$ der Vektor der unbekannten Ableitungen von $\mathbf{x}$ nach t. Eine äquivalente Darstellung ist die Integralform mit einer gegebenen Startbedingung an Konstanten $\mathbf{x}(t_0) = \mathbf{x}_0$:

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \dot{\mathbf{x}}(\tau) d\tau = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{f}(\mathbf{x}(\tau), \tau) d\tau. \tag{2.2}$$

Angenommen x ist ein Skalar, dann entspricht $x(t)$ der Fläche zwischen t_0 und t unter der Funktion $f(x(\tau), \tau)$ plus einer Konstanten x_0.

Kausale Modelle ODEs sind in expliziter Form ausgedrückt, d. h. $\dot{x}(t)$ ist die Unbekannte, die von $x(t)$ abhängt, aber nicht umgekehrt $x(t)$ direkt vom Wert von $\dot{x}$ zum Zeitpunkt t. Aus diesem Grund können ODEs als kausale Modelle angesehen werden, d. h. auf Basis von Ein- und Ausgängen modelliert und daher in Actor-orientierten Simulationswerkzeugen umgesetzt werden, bspw. Ptolemy II (siehe Abschnitt 2.3), MATLAB/Simulink [104] oder LabVIEW von National Instruments. Ein generisches Blockschaltbild von Gl. (2.2) ist in Abb. 2.6 zu sehen. Dabei ist $\dot{x}(t)$ der Eingang und $x(t)$ der Ausgang des Integrators, wobei der Block der Funktion zur Berechnung von $\dot{x}(t)$ mithilfe von $x(t)$ im Rückkopplungspfad liegt. Durch die Kausalität muss der Wert von $\dot{x}$ zum Zeitpunkt t zur Berechnung von $x(t)$ nicht bekannt sein. Sobald $\dot{x}(t)$ bekannt ist, beeinflusst dieser lediglich zukünftige Werte von x [87, 88].

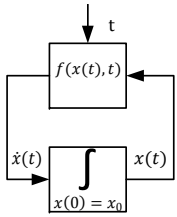


Abbildung 2.6: Blockdiagramm einer ODE (Quelle: modifizierte Darstellung nach [87]).

Zur Simulation von CT Modellen bzw. zur Lösung von ODEs auf Rechnern sind numerische Methoden zur Integration notwendig, die diese diskretisieren und an diskreten Zeitpunkten t_n lösen. Eine voraussetzende Eigenschaft zur numerischen Integration ist, dass die Signale stückweise kontinuierlich sind. Ein Algorithmus zur Lösung wird auch als *Solver* bezeichnet. Diese lassen sich in Solver mit *festen*

Schrittweiten (engl. fixed-step) und *variablen Schrittweiten* (engl. variable-step) klassifizieren. Zu den einfachsten Solvern gehört das *Vorwärts-Euler-Verfahren* (engl. forward Euler), das mit einer festen Schrittweite h

$$x(t_{n+1}) = x(t_n) + h\dot{x}(t_n) \tag{2.3}$$

berechnet, mit $h = t_{n+1} - t_n$. Die Berechnung zählt zu den *expliziten* Verfahren, da die Berechnung des aktuellen Wertes von $x(t_{n+1})$ nur von vorangehenden Werten von $\dot{x}$ abhängt und damit die Kausalität einhält. Im Gegensatz dazu existieren *implizite* Verfahren, die zur Berechnung von $x(t_{n+1})$ die aktuellen Werte der Ableitung an $\dot{x}(t_{n+1})$ benötigen. In Modellen die Abb. 2.6 folgen, kann dies zu zyklischen Abhängigkeiten und zur Verletzung der Kausalität führen. Eine Möglichkeit diese dennoch zu verwenden ist, die Ableitung zu schätzen und diese iterativ zu verfeinern. Je nach Modell kann jedoch keine Garantie zur akkuraten Bestimmung des Wertes gegeben werden und es können nicht eindeutige Ergebnisse auftreten. Allerdings werden sie i. d. R. zur Simulation von *steifen* (engl. stiff) Systemen verwendet oder welchen, die durch differentielle, algebraische Gleichungen (engl. *Differential Algebraic Equations (DAEs)*) ausgedrückt werden. Für solche komplexen Systeme weisen sie bessere Fehler-, Konvergenz- und Stabilitätseigenschaften auf. [27, 67, 87, 136].

Solver mit festen Schrittweiten haben den Nachteil, dass die Schrittweite individuell der Applikation zugeschnitten sein muss. Eine zu große Schrittweite führt zu einer schlechten Approximation, eine zu klein gewählte dagegen zwar zu genauen Ergebnissen, aber mit langen, unpraktikablen Berechnungszeiten. Außerdem sind sie nicht dafür geeignet, Nichtkontinuitäten in Signalen bzw. diskrete Events exakt zu detektieren, was bspw. eine Voraussetzung zur Modellierung von sog. *hybriden* Systemen [31] ist. Hybride Systeme kombinieren diskrete und kontinuierliche Berechnungsmodelle. Hybride Systeme beschreiben kontinuierliche Prozesse, die von digitalen Komponenten gesteuert werden und somit auf diskrete Ereignisse reagieren können müssen.

Daher werden i. d. R. Solver mit variablen Schrittweiten eingesetzt, die ihre Schrittweite zur Laufzeit automatisch an das Verhalten des Systems anpassen. Dies ist exemplarisch in Abb. 2.7 illustriert. Wenn schnelle Änderungen im kontinuierlichen Signal $f(x(t), t)$ auftreten, wird zur genauen Approximation eine kleine Schrittweite h_n gewählt, bei langsamen Änderungen hingegen eine typischerweise maximal spezifizierte Schrittweite h_{max}, um Berechnungszeit zu sparen. Die Bestimmung der Schrittweite h_n basiert auf der Schätzung des lokalen Fehlers der numerischen Integration. Es wird zunächst eine vorläufige Schrittweite gewählt, dann die numerische Integration durchgeführt und anschließend der Fehler geschätzt. Falls dieser größer als eine spezifizierte Toleranz ist, wird die Schrittweite

abgelehnt und eine neue Iteration mit einer kleineren Schrittweite durchgeführt, bis der Fehler innerhalb der Toleranz liegt.

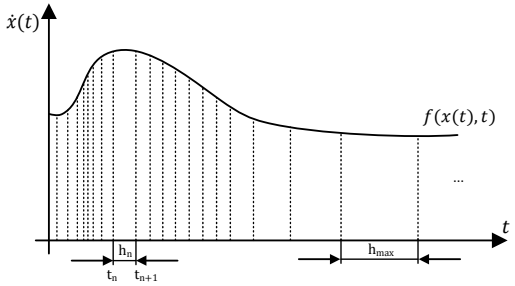


Abbildung 2.7: Numerische Integration mit variablen Schrittweiten (Quelle: eigene Darstellung nach [88, S. 171]).

Das gleiche Prinzip wird verwendet, um Nichtkontinuitäten, sog. *zero crossings* oder *state events*, zu detektieren, die vom kontinuierlichen System selbst ausgelöst werden (nicht von externen diskreten Events). Um die Solver zur Simulation auch solcher Situationen einsetzen zu können, sind zusätzliche Mechanismen bzgl. der Modellierung der Zeit bzw. des Zeitfortschritts notwendig [27]. Eine solche Möglichkeit bietet das bereits erwähnte und in Ptolemy II eingesetzte Modell der Superdense Time [136].

Zu den weitestverbreiteten Solvern gehören die *Runge-Kutta* [27] Verfahren, wobei das Vorwärts-Euler-Verfahren ein Spezialfall davon ist. Diese verbessern die Genauigkeit signifikant durch Evaluation von $f(x(t), t)$ an mehrfachen Punkten zwischen t_n und t_{n+1}. Zu bemerken ist abschließend, dass Simulationswerkzeuge mit Solvern variabler Schrittweiten eine spekulative numerische Integration ausführen müssen, um evtl. mit einer kleineren Schrittweite zum vorigen akzeptierten Schritt zurückzukehren, bis der Fehler in der aktuellen Iteration zur Toleranz konvergiert.

Akausale Modelle Einige kontinuierliche Systeme können nicht mit ODEs dargestellt werden, wenn sie bestimmten physikalischen Einschränkungen folgen müssen wie z. B. den Kirchhoff'schen Gesetzen. Solche Systeme enthalten algebraische Einschränkungen, die implizit mit den Differentialgleichungen definiert sind, werden DAE genannt und können wie folgt ausgedrückt werden [67]:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{z}(t), t), \\ 0 &= \mathbf{g}(\mathbf{x}_r(t), \mathbf{u}_r(t), \mathbf{z}(t)), \end{aligned} \tag{2.4}$$

wobei $\mathbf{u}(t)$ ein Vektor an Eingangssignalen, $\mathbf{z}(t)$ ein Vektor an algebraischen Variablen und $\mathbf{x}_r(t)$, $\mathbf{u}_r(t)$ Vektoren mit einer Dimension kleiner oder gleich von $\mathbf{x}(t)$, $\mathbf{u}(t)$ sind.

DAEs entstammen typischerweise aus akausalen Modellen, wie z. B. elektrischen Schaltungen in *Simulation Program with Integrated Circuit Emphasis (SPICE)* [120], mechatronischen Systemen der gleichungs- und objektorientierten Sprache Modelica [114], VHDL-AMS [55] oder MATLAB/Simscape Power Systems [103]. Generell basieren die meisten modernen Solver zur Lösung der impliziten Gleichungen auf der Methode von Gear [40], die eine Kombination aus den klassischen ODE Solvern und Techniken zur symbolischen Manipulation mit Iterationen darstellt [67]. Heutige Werkzeuge für akausale Modelle wie Modelica wandeln DAEs zunächst automatisch in ODEs um, da diese effizienter zu lösen sind. Jedoch können auch Actor-orientierte Werkzeuge DAEs modellieren, die dann aber algebraische Schleifen aufweisen und durch weitere implizite Iterationen zu lösen sind [87].

Akausale Modelle unterscheiden sich grundsätzlich von kausalen Modellen, als dass sie nicht durch Ein-/Ausgänge modelliert werden können. Das hat aber den Vorteil, dass sie sich in ihrer natürlichen Form darstellen lassen und den Fokus darauf legen, *was* modelliert und simuliert wird und nicht auf das *wie*, z. B. elektrische Schaltpläne.

2.2.2.4 Quantisierte Zustandssysteme

Die vorgestellten Methoden zur numerischen Integration von ODEs/DAEs haben alle eine gemeinsame Eigenschaft: sie diskretisieren die Zeit zur numerischen Lösung. Im Vergleich dazu hat sich Ende der 90er Jahre ein neuer, orthogonaler Ansatz zur numerischen Integration von ODEs angebahnt und wurde erstmals von Zeigler [199] als *quantisierte Systeme* (engl. quantized systems) eingeführt. Kofman und Junco erweiterten den Ansatz um eine Hysterese, sodass sich der Ansatz zur DE Simulation von generischen ODEs eignete. Diese Art von Systemen bezeichneten sie als *quantisierte Zustandssysteme* (engl. *Quantized-State Systems (QSSs)* [71]).

Im Gegensatz zu den klassischen, zeitdiskreten Methoden quantisieren die QSS Methoden den kontinuierlichen Zustand. Betrachtet werde erneut ein ODE System mit

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t), \tag{2.5}$$

wobei $\mathbf{x}(t)$ ein analytisch lösbarer Zustandsvektor und $\mathbf{u}(t)$ ein bekannter Eingangsvektor sind. Klassische Solver mit variablen Schrittweiten wie die Runge-Kutte Verfahren bestimmen die Abtastpunkte, die für das gesamte System, d. h. für *alle* Zustände gelten. In einem Actor-orientiertem Modell bspw., würden dem-

nach *alle* Actors zu einem bestimmten Zeitpunkt ausgeführt werden. Die QSS Methode erster Ordnung (QSS1) approximiert das ODE System nach Gl. (2.5) mit einem *quantisierten Zustandsvektor* (engl. quantized state vector) $\mathbf{q}(t)$, sodass

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t), t) \tag{2.6}$$

resultiert. Eine sog. *quantisierte Hysteresefunktion* (engl. hysteretic quantization function) verbindet jede quantisierte Zustandsvariable $q_i(t)$ mit der zugehörigen Zustandsvariable $x_i(t)$ mit:

$$q_i(t) = \begin{cases} x_i(t) & \text{wenn } \left|x_i(t) - q_i(t^-)\right| = \Delta Q_i, \\ q_i(t^-) & \text{sonst,} \end{cases} \tag{2.7}$$

wobei ΔQ_i als *Quantum* bezeichnet wird und $q_i(t^-)$ der Wert des quantisierten Zustands zum letzten Quantisierungs-Zeitpunkt ist.

Daraus folgt, dass $q_i(t)$ seinen Wert nur ändert, wenn es von $x_i(t)$ um mehr als das Quantum abweicht oder gleich jenem ist. Unmittelbar nach einer Wertänderung gilt dann $q_i(t) = x_i(t)$. Aus Gl. (2.7) folgt, dass $\mathbf{q}(t)$ stückweise konstant ist und, unter der Annahme, dass $\mathbf{u}(t)$ ebenfalls stückweise konstant ist, dass $\mathbf{x}(t)$ stückweise linearen Trajektorien folgt [71]. Dies zeigt einen wichtigen Vorteil gegenüber klassischen, zeitdiskreten Solvern, als dass jedes q_i individuell und *asynchron* zu unterschiedlichen Abtastpunkten aktualisiert wird, abhängig von dessen spezifizierten Quantum und der Zeitdauer, mit der es überschritten wird.

Da $\mathbf{x}(t)$ stückweise linear ist, können die Zeitpunkte, an denen das Quantum erreicht wird, analytisch berechnet werden ohne die Notwendigkeit, Iterationen zur Bestimmung durchführen zu müssen (wie im Falle der variablen Schrittweite in klassischen Solvern). Daher sind QSS Methoden besonders dafür geeignet, sehr effizient hybride Modelle mit Nichtkontinuitäten zu simulieren, da zero crossings bzw. State Events analytisch vorhergesagt werden können [68]. Darüber hinaus weisen QSS Verfahren gegenüber klassischen Solvern weitere wichtige Eigenschaften auf, wie z. B. starke numerische Stabilität, *globale* Fehlerbegrenzung (für LTI Systeme bewiesen) und relative Fehlerkontrolle [71, 68, 27, 70, 37].

QSS höherer Ordnung Ein Nachteil von QSS ist allerdings, dass es lediglich eine Approximation erster Ordnung darstellt (*Quantized-State System 1. Ordnung (QSS1)*) und für eine gute Genauigkeit das Quantum sehr klein gewählt werden muss. Dies hat eine hohe Anzahl an Quantisierungs-Ereignissen (engl. *quantization events*) zur Folge, die invers proportional zum Quantum sind [27]. Aus diesem Grund wurden QSS Methoden höherer Ordnung entwickelt. *Quantized-State System 2. Ordnung (QSS2)* [66] und *Quantized-State System 3. Ordnung (QSS3)* [69]

teilen dieselben Eigenschaften und Vorteile wie QSS1, basieren aber auf Quantisierungsfunktionen höherer Ordnung (1. Ordnung für QSS2 bzw. 2. Ordnung für QSS3). Das bedeutet die quantisierten Zustände q_i folgen stückweise linearen bzw. quadratischen Trajektorien und die Zustände x_i dementsprechend stückweise quadratischen bzw. kubischen.

Ein Quantisierer erster Ordnung ist exemplarisch in Abb. 2.8 gezeigt. Die quantisierten Zustände folgen demnach stückweise definierten Geraden, deren Wert und Steigung sich nur ändern, sobald eine Gerade sich vom Zustand x_i um das Quantum ΔQ_i unterscheidet. Die neue Gerade nimmt dann, analog zu QSS1, den aktuellen Wert des Zustands und zusätzlich dessen Steigung zu diesem Zeitpunkt an, bis sich die Gerade erneut von x_i um das Quantum unterscheidet.

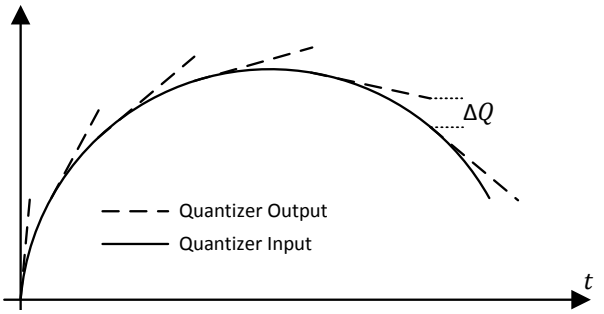


Abbildung 2.8: Trajektorien eines Quantisierers erster Ordnung in einem QSS2 System (Quelle: eigene Darstellung nach [68]).

Die Verfahren höherer Ordnung weisen eine höhere Genauigkeit auf, ohne dabei signifikant die Anzahl an Berechnungen bzw. Quantisierungen zu erhöhen. Der Zeitschritt zwischen zwei Quantisierungen ist bei QSS2 durch die lineare Approximation nun nur noch proportional zur Wurzel des Quantums, bei QSS3 proportional zur dritten Wurzel [69].

Die bisherigen QSS Methoden sind explizite Verfahren, sodass sie selten mit *steifen* Systemen[6] (engl. *stiff*) zurechtkommen. Aus diesem Grund wurde die Familie um sog. *linear implizite* (engl. *linear implicit*) QSS Verfahren erweitert (*Linear Implicit Quantized-State System (LIQSS)* 1. - 3. Ordnung) [111] als auch um ein Derivat für grenzstabile Systeme (*Centered QSS (CQSS)*) [113]. Die expliziten QSS Verfahren produzieren hochfrequente Oszillationen mit den meisten steifen Systemen, die eine hohe Anzahl an Events nach sich ziehen und somit unverhältnismäßig

[6]Steife Systeme weisen gleichzeitig langsame und schnelle Dynamiken auf.

berechnungsintensiv sind [113]. Trotz ihrer Anlehnung an impliziten Verfahren sind die LIQSS Methoden jedoch explizit [111].

DE Implementierung Eine weitere wichtige Eigenschaft von QSS Verfahren ist ihre einfache Integration in DE Berechnungsmodelle, worin die quantisierten Zustände und das Eingangssignal der Integratoren als eine Sequenz an diskreten Events $q_i(\tau_k)$ bzw. $\dot{x}_i(\tau_k)$ angesehen werden können [69]. Demnach ermöglichen sie die Simulation von hybriden Modellen in einem einzigen DE-basierten Modell. Durch die DE-basierte Modellierung und durch die Kausalität von ODEs können sie analog zu klassischen Solvern als Blockdiagramme modelliert werden (siehe Abb. 2.6). Aber auch DAEs können mit QSS Verfahren DE-basiert mit einer signifikanten Reduktion an Berechnungen im Vergleich zu klassischen Lösungen simuliert werden [67].

Ein generisches Blockschaltbild einer QSS Implementierung von Gl. (2.6) ist in Abb. 2.9 dargestellt.

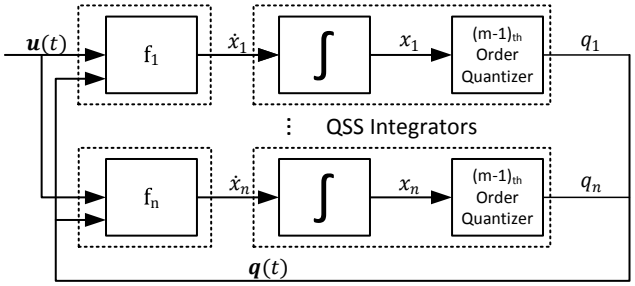


Abbildung 2.9: Blockdiagramm eines ODE Systems in QSSm Repräsentation (Quelle: eigene Darstellung nach [69]).

Dabei wird das Modell in n sog. *statische Funktionen* (engl. static functions) zur Berechnung von $\mathbf{f}(\mathbf{q}(t), \mathbf{u}(t), t)$, n *quantisierte Integratoren* (engl. QSS Integrators) und optional in k Eventquellen zur Repräsentation des Eingangsvektors $\mathbf{u}(t)$ aufgeteilt. Ein quantisierter Integrator realisiert den individuell spezifizierten QSS Solver m. Ordnung mit der dazugehörigen Quantisierungsfunktion der Ordnung $m-1$. Bei QSS Verfahren höherer Ordnung ist abschließend zu bemerken, dass die Events nicht nur den eigentlichen Wert transportieren, sondern zusätzlich die Steigung (QSS2) und die zweite Ableitung (QSS3) des aktuellen Segments.

2.3 Ptolemy II

Ptolemy II (PtII) [34] ist ein in Java implementiertes, quelloffenes Entwurfswerkzeug zur Modellierung und Simulation heterogener Systeme und zählt zu den sog. formalen Frameworks (vgl. Unterabschnitt 3.3.2.1). PtII Modelle folgen dabei einem Actor-orientierten Ansatz [86], vergleichbar mit etablierten modellbasierten Simulationswerkzeugen wie MATLAB/Simulink. PtII legt den Fokus auf die deterministische und hierarchische Komposition von Modellen mit verschiedenen MoCs. Letztere regeln die Ausführung von und Interaktionen zwischen den Actors und werden durch sog. *Director* Attribute implementiert. Ein Director in PtII wird auch als *Domäne* (engl. *domain*) bezeichnet. Wegen der Wichtigkeit des Aufbaus eines PtII Modells in Kapitel 5, werden im Folgenden die Kernkomponenten des PtII Metamodells, der Semantik und einige PtII Domänen detaillierter erläutert.

2.3.1 Metamodell und Syntax

Die Syntax eines Modells legt generell dessen Darstellung fest, wobei die *abstrakte Syntax* die Struktur und die *konkrete Syntax* eine spezifische, lesbare Notation zur Repräsentation der spezifiziert. Prinzipiell kann die abstrakte Syntax von PtII als ein hierarchisch verschachtelter Graph aufgefasst werden. Dabei stellt die Baumstruktur die Hierarchie in einem Modell und ein Graph ein Submodell auf jeder dieser Hierarchieebenen dar [136]. Eine konkrete Syntax ist bspw. die auf der XML basierte Sprache *Modeling Markup Language (MoML)* [85] oder die visuelle Syntax *Vergil* zur grafischen Editierung von Modellen.

Ein *Metamodell* ist ein Modell einer Modellierungssprache oder abstrakten Syntax und wird oft durch ein UML Klassendiagramm beschrieben [59]. Das Metamodell der abstrakten Syntax von PtII ist in Abb. 2.10 zu sehen, das die Beziehungen zwischen den Basisklassen und deren wichtigsten Attribute und Methoden zeigt. Instanzen dieser objektorientierten Basisklassen bilden schließlich ein Ptolemy II Modell.

Die Basisklasse `NamedObj` ist dabei als Wurzel zu verstehen, die einen Namen und einen Container, zur Realisierung der Hierarchie, besitzt und von der alle zur Darstellung eines PtII Modells relevanten Basisklassen erben. Letztere sind die vier Klassen `Attribute`, `Entity`, `Relation` und `Port`. Demnach besteht jedes Modell aus einer top-level `CompositeEntity` Komponente, deren Container `null` ist und die wiederum beliebig viele Instanzen von `Entity` mit Attributen und Ports enthalten kann. Die Ports sind dabei über sog. *links* mit Relationen assoziiert, die die Verbindungen zwischen den Ports herstellen. Attribute sind i. d. R. Parameter,

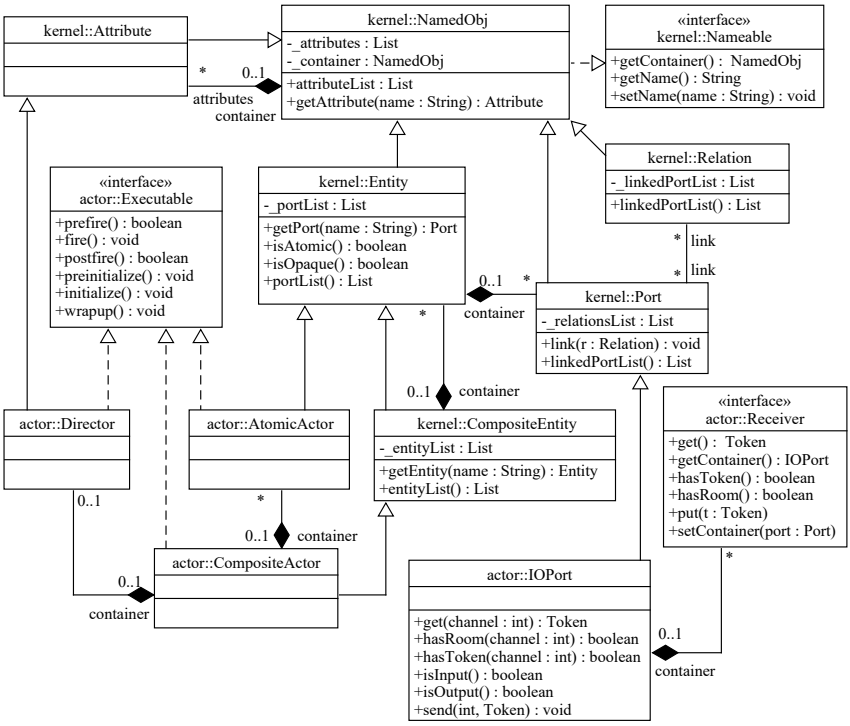


Abbildung 2.10: Metamodell der abstrakten Syntax von PtII (Quelle: erweiterte Darstellung nach [136, S. 425]).

die im gesamten Modell zugreifbar sind und auch zur Laufzeit verändert werden können.

Um nun ein ausführbares Modell mit den eingangs erwähnten Actors zu erhalten, existieren die beiden Klassen `AtomicActor` und `CompositeActor`, die jeweils von `Entity` und `CompositeEntity` abgeleitet sind und die Schnittstelle `Executable` implementieren. Atomare Actors können dabei nicht weiter verfeinert werden, wobei zusammengesetzte (engl. *composite*) Actors weitere hierarchisch geschachtelte PtII Modelle enthalten können.

Die Schnittstelle `Executable` stellt die notwendigen Methoden zur Spezifikation des Ausführungsverhaltens eines Modells bereit, sowohl für die Actors als auch für Directors. Eine `Director` Instanz ist daher ein speziell ausführbares Attribut, was durch die Ableitung von der Klasse `Attribute` und die Implementierung

der Schnittstelle `Executable` in Abb. 2.10 verdeutlicht wird. Eine Instanz von `CompositeActor` kann dabei entweder genau einen oder keinen Director besitzen, wobei die top-Level Instanz immer genau einen Director aufweisen muss, damit das Modell ausführbar wird. [136]

Darüber hinaus existiert eine weitere Schnittstelle `Decorator`, die von der Schnittstelle `Nameable` erbt (nicht dargestellt in Abb. 2.10). Es realisiert das bekannte Dekorierer-Entwurfsmuster aus der Softwareentwicklung und bietet die Möglichkeit, beliebige Objekte von `NamedObj` mit spezifischen, zusätzlichen Attributen vom Typ `DecoratorAttributes` anzureichern bzw. zu dekorieren. Beispielsweise können Ports mit sog. *Kommunikationsaspekten* (engl. *communication aspects*) dekoriert werden, wonach der Datenaustausch zwischen Actors einem spezifischen Kommunikationsprotokoll folgen kann (mehr dazu in Kapitel 5).

2.3.2 Semantik

Die bereits erwähnte `Executable` Schnittstelle ist Teil der Ausführungssemantik von PtII. Analog zur Syntax lässt sich die Semantik von PtII in eine *abstrakte Semantik* und *konkrete Semantik* unterteilen.

Die abstrakte Semantik stellt eine Generalisierung von MoCs dar und wird durch einen formalen Regelsatz zur (nebenläufigen) Ausführung und Kommunikation sowie durch ein Zeitmodell spezifiziert. Sie wurde erstmals in [86] eingeführt und wird auch als *Actor Abstract Semantics* bezeichnet. Dieser formale Regelsatz ist grundlegend für die deterministische Interaktion verschiedener, heterogener Domänen über Hierarchieebenen hinweg. Daher ist sie eine Voraussetzung zur Konstruktion von heterogenen Modellen [81, 86, 136].

Eine konkrete Semantik definiert dagegen ein spezifisches MoC und beschreibt, *wie* die Regeln zur Ausführung, Nebenläufigkeit und Kommunikation umgesetzt werden, z. B., dass eine asynchrone Kommunikation zwischen Actors stattfindet. Ein Director implementiert daher eine konkrete Semantik. Durch die objektorientierte Struktur können sowohl Actors als auch Directors jeweils voneinander erben. Das heißt insb. für Directors, dass eine abstrakte Domäne durch einen abgeleiteten Director spezialisiert werden kann und damit auch die konkrete Domäne. Aufgrund der Vererbung der Eigenschaften der abstrakteren Domäne wird damit auch die Kompatibilität zu anderen Domänen gewahrt, die bereits mit der abstrakten Domäne kompatibel sind.

Die Tatsache, dass jede Instanz eines `CompositeActor` seinen eigenen Director besitzen kann, ermöglicht eine hierarchische Heterogenität bestehend aus verschiedenen Domänen, welche durch die abstrakte Semantik deterministisch miteinander interagieren können. Ein zusammengesetzter Actor, welcher einen Di-

rector besitzt, wird auch als *opaque* (dt. *undurchlässig*) bezeichnet, der durch den Director in sich selbst ausführbar ist und von außen betrachtet eine Blackbox darstellt. Seine Struktur und Semantik sind im umgebenden Modell nicht sichtbar. Im Gegensatz dazu ist ein zusammengesetzter Actor ohne Director als *transparent* anzusehen, welcher nach außen hin sichtbar und nicht selbst ausführbar ist. Die konkrete Semantik wird dann durch den Director des umgebenden Modells gesteuert [136].

Ein exemplarisches, heterogenes PtII Modell ist in Abb. 2.11 zu sehen. Es realisiert ein vereinfachtes Modell eines Generators, welcher über einen PID Regler geregelt wird und über einen Überspannungsschutz verfügt. Auf oberster Ebene wird ein DE Director (siehe Unterabschnitt 2.3.3.1) eingesetzt, der die zeitliche Ausführung von und Kommunikation zwischen Actors über diskrete Ereignisse (engl. *discrete events*) steuert. Die drei zusammengesetzten Actors besitzen alle ihren eigenen Director, wobei hier beispielhaft die Verfeinerungen des *Controller* und des *Generator* Actors gezeigt sind.

Der Controller realisiert den PID Regler des Generators und wird über einen typischerweise zeitlosen Datenfluss SDF Director gesteuert, der für Streaming Applikationen und digitale Signalverarbeitung geeignet ist [31, 136]. Der *Supervisor* Actor realisiert die Überwachung der Überspannung mit einer FSM und der *DiscreteGenerator* Actor realisiert intern das zeitlich kontinuierliche Modell des Generators mit einem CT Director [90].

Der Generator stellt dabei diskrete Schnittstellen nach außen, zur digitalen Eingabe und Weiterverarbeitung, bereit. Letztere sind für die Einbettung von kontinuierlichen Modellen in ein diskretes Modell notwendig und werden in PtII über die Actors *ZeroOrderHold* am Eingang und *PeriodicSampler* am Ausgang realisiert (siehe verfeinerter Generator in Abb. 2.11). ZeroOrderHold Actors wandeln ein diskretes Signal aus der DE Domäne in ein konstantes, kontinuierliches Signal um. PeriodicSampler Actors erzeugen für die umgebende DE Domäne aus einem kontinuierlichen Signal periodisch diskrete Events am Ausgang. Das Beispiel verdeutlicht die Möglichkeit zur hierarchischen Komposition sehr heterogener MoCs (zeitlos, zeitlich diskret und kontinuierlich), die durch die abstrakte Semantik innerhalb eines PtII Modells deterministisch miteinander interagieren können.

Im Folgenden werden auf die drei Kernaspekte der abstrakten Semantik, (1) Ausführungssemantik (2) Kommunikation und (3) Zeitmodell, genauer eingegangen [81, 136].

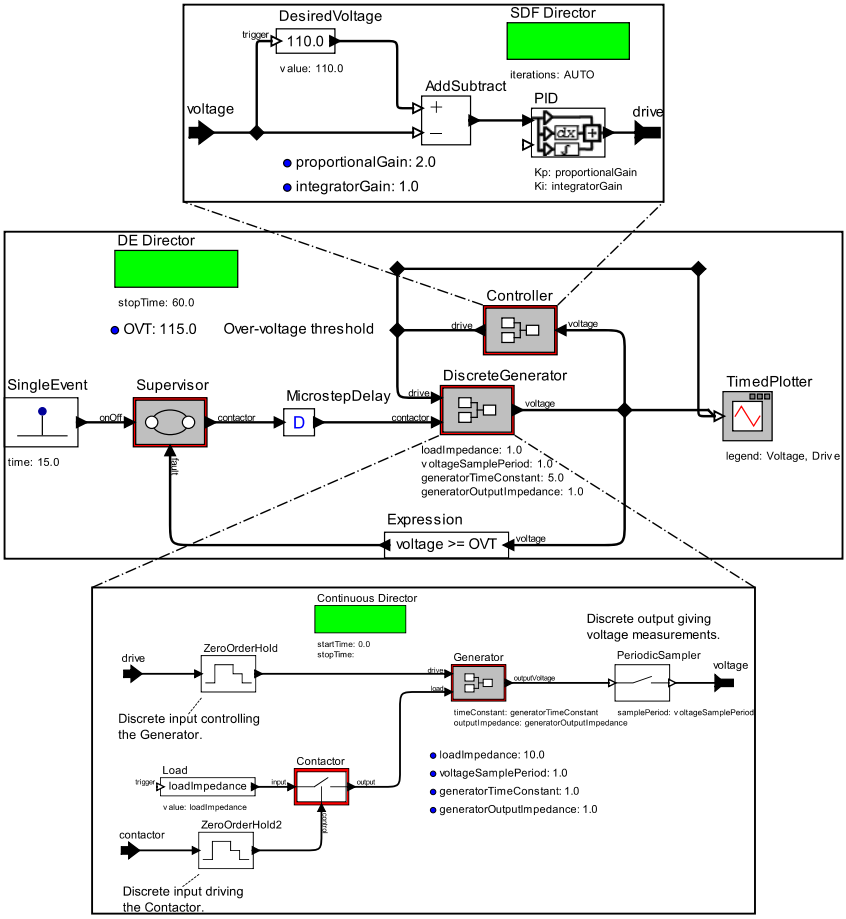


Abbildung 2.11: PtII Beispielmodell eines elektrischen Generators mit PID Regler und Überspannungsschutz (Quelle: modifizierte Darstellung nach [136, Kap. 1.9]).

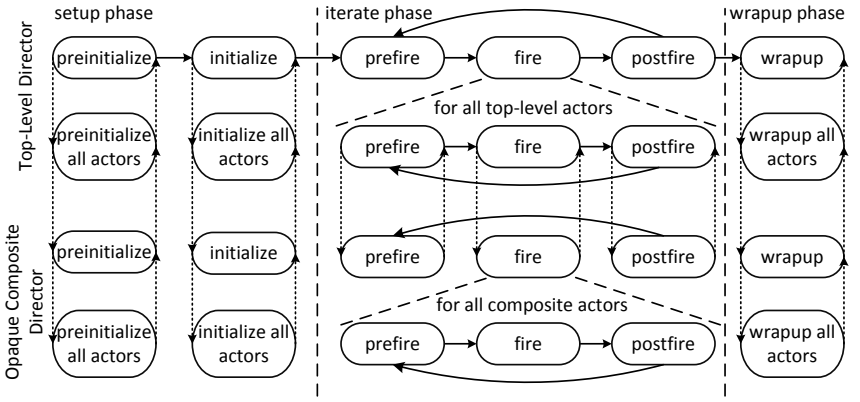


Abbildung 2.12: Schematische Darstellung der Ausführungssemantik eines hierarchischen PtII Modells mit einem undurchlässigen Actor (Quelle: eigene Darstellung nach [136, S. 431]).

2.3.2.1 Ausführungssemantik

Die konkrete Semantik einer spezifischen Domäne wird durch einen Director implementiert, der einen zusammengesetzten Actor ausführbar macht und dessen Ausführung und Kommunikation mit anderen Actors steuert. Die Operationen der ausführbaren Actors und Directors werden dabei durch die Executable Schnittstelle vorgegeben, welche auch atomare Actors implementieren. Abb. 2.12 zeigt den generellen Ablauf einer PtII Modellausführung anhand eines Beispiels mit einem undurchlässigen Actor.

Die Ausführung besteht dabei aus drei Phasen, der *Setup Phase*, *Iterate Phase* und *Wrapup Phase*, die sich wiederum in Subphasen unterteilen lassen. Diese Subphasen entsprechen den Operationen der `Executable` Schnittstelle.

Die Operationen `preinitialize()` und `initialize()` sind für die Initialisierung aller Actors zuständig. Die Aufgabe von `preinitialize()` ist dabei die statische Analyse des Modells, bspw. die Bestimmung der Ausführungsreihenfolge von Actors oder Datentyp-Checks, und wird genau einmal vor allen anderen Operationen ausgeführt. Zur Initialisierung von Modellparametern und zum Zurücksetzen lokaler Zustände von Actors dient die `initialize()` Operation. Sie kann ein oder mehrmals während einer Modellausführung aufgerufen werden (abh. von der Domäne).

Das Kernstück der Modellausführung ist die Iterate Phase, in der jeder Actor (und Director) eine Sequenz an Iterationen über die Operationen `prefire()`, `fire()` und `postfire()`abarbeitet. Erstere prüft ggf. Vorbedingungen zum weiteren Aufruf von `fire()`. Die `fire()` Operation ist für das eigentliche Verhalten eines Actors zuständig: Daten von Eingangsports einlesen, Datenverarbeitung, senden der Daten über Ausgangsports. Die letzte Operation der Iterate Phase, `postfire()`, aktualisiert lokale Zustände eines Actors. Zur Aktualisierung des Zustands eines Actors ist die `postfire()` Operation verantwortlich. Sie ist ein wichtiger Bestandteil der abstrakten Semantik und Voraussetzung für sog. *domänenpolymorphe Actors*, welche in beliebigen Domänen eingesetzt werden können. Analoges gilt für einen Director.

Die finale Wrapup Phase besteht aus der gleichnamigen Operation `wrapup()`, die auch im Fehlerfall garantiert aufgerufen wird und das Ende des Lebenszyklus eines Actors oder Directors signalisiert.

2.3.2.2 Kommunikation

Actors verschicken und empfangen über deren Ports sog. *Tokens*. Die Art des Austauschs dieser Tokens wird wiederum durch den Director bestimmt. Dieser platziert entsprechend seiner Domäne Instanzen sog. `Receiver` in den Eingangsports von Actors (vgl. Metamodell in Abb. 2.10), wobei jeder Eingangsport aus mehreren Kanälen bestehen kann und jeder Kanal genau einen Receiver besitzt. Der Receiver bestimmt den tatsächlich stattfindenden Kommunikationsmechanismus, wie z. B. *FIFOs*, *Mailboxes* oder *Rendevous* [136]. Dies wird dadurch realisiert, dass die Operation `send()` über den Ausgangsport einen Token an den Receiver innerhalb der verbundenen Empfängerports p_{in} delegiert. Ein Token wird dabei durch die `put()` Operation eines Receivers in diesem abgelegt. Ein empfangener Token wird in der `fire()` Subphase über die `get()` Operation des zugehörigen Ports p_{in} extrahiert.

2.3.2.3 Zeitmodell

Einige Berechnungsmodelle besitzen einen Zeitaspekt, d. h. Actors folgen bei deren Ausführung einer logischen Zeit. Ein konsistentes Zeitmodell für diese Zeitsteuerung ist der letzte Kernpunkt der abstrakten Semantik, das die deterministische und hierarchische Kombination verschiedener Domänen gewährleistet. Das Zeitmodell wird dabei in zwei zentrale Mechanismen aufgeteilt: (1) hierarchische Zeit und (2) die *Superdense Time* [99, 136].

Die hierarchische Zeit wird dabei durch die Modellhierarchie bestimmt, indem nur der top-Level Director einen Zeitfortschritt durchführt. Weitere Directors in darunterliegenden Ebenen erhalten den aktuellen Zeitwert vom Director der höheren Hierarchieebene. Das ermöglicht die Einbettung von zeitlosen Domänen wie Datenflussmodelle in welchen, die ein strenges, zeitliches Verhalten aufweisen, bspw. DE und CT Modelle. Dabei werden zeitbezogene Operationen von zeitlosen Domänen an die zeitbehafteten Domänen auf höherer Ebene delegiert. Falls der top-Level Director kein zeitbehaftetes Berechnungsmodell implementiert, existiert kein Zeitfortschritt im Modell.

Das eigentliche Zeitmodell, die *Superdense Time*, besteht aus einem Zeitstempel-Wertepaar (τ, n), wobei τ die sog. Modellzeit (engl. *model time*) und n der sog. *Mikroschritt* (engl. *microstep*) ist. Die Modellzeit gibt an, wann ein Event auftritt und der Mikroschritt stellt eine Reihe an Events dar, die zur gleichen Modellzeit τ auftreten können. Zwei Zeitstempel (τ_1, n_1) und (τ_2, n_2) sind *schwach gleichzeitig* (engl. *weak simultaneous*), wenn $\tau_1 = \tau_2$ und *stark gleichzeitig* (engl. *strong simultaneous*), wenn zusätzlich $n_1 = n_2$ gilt.

Dieses Modell erlaubt eine eindeutige Beschreibung von diskreten Ereignissen, Unstetigkeiten in kontinuierlichen Signalen und eine Sequenz an Events von unendlich kurzer Zeitdauer in diskreten Signalen. Es ermöglicht damit insb. die Interoperabilität von DE und CT Domänen. Zudem bildet es die Basis zur Bestimmung einer eindeutigen Ausführungsreihenfolge von Actors. Darüber hinaus können Actors eine Anfrage stellen, zu einem bestimmten, späteren Zeitpunkt erneut gefeuert zu werden, indem sie die `fireAt()` Methode ihres Directors aufrufen. Der Director prüft, ob die Anfrage legitim ist und liefert ggf. einen alternativen Zeitstempel zurück.

2.3.3 Domänen

In den folgenden Abschnitten werden nähere Informationen zu einer Auswahl an für diese Arbeit relevanten Domänen gegeben.

2.3.3.1 Discrete-Event Domäne

Wie in den meisten Domänen kommunizieren die Actors innerhalb der DE Domäne über ihre Ports und tauschen sog. Events aus. Ein Event ist ein Paket bestehend aus einem Token, das die Daten kapselt und einem Zeitstempel nach Unterabschnitt 2.3.2.3. Prinzipiell reagiert jeder Actor in der `fire()` Operation auf Events an Eingangsports in zeitlicher Reihenfolge.

Der DE Director verwaltet die Events in einer globalen Event-Warteschlange, die chronologisch nach ihren Zeitstempeln sortiert ist. Ein Event wird dabei beim Versand eines Tokens über die verknüpften DE Receiver in den Empfängerports (vgl. Unterabschnitt 2.3.2.2) in der Warteschlange platziert. Der DE Director sorgt nun dafür, dass *alle* Events mit dem kleinsten Zeitstempel ausgewählt und an ihre adressierten Ziel-Actors bzw. deren Empfangsports weitergeleitet worden sind. Erst dann werden die entsprechenden Actors gefeuert. Dieser Mechanismus garantiert den Determinismus in der DE Domäne bzgl. der Superdense Time.

Allerdings garantiert dies keinen Determinismus, falls zwei Events stark gleichzeitig auftreten (vgl. Unterabschnitt 2.3.2.3). Dies kann passieren, da eine Vielzahl von Actors i. d. R. ohne Zeitverzögerung, sowohl bzgl. τ als auch bzgl. n, auf Events an ihren Eingängen reagieren. Die Reihenfolge der Auswahl der Events und deren Actors wäre damit nicht mehr eindeutig und somit die Simulation abhängig von der Reihenfolge der aktuell gewählten Events. Um dies zu vermeiden, wird zusätzlich ein sog. *Level* von Actors berücksichtigt, sodass sich das Zeitmodell in der DE Domäne zu einem Tupel (τ, n, m) ergänzt, wobei m den Level darstellt. Der Level wird statisch durch eine topologische Sortierung der Actors, mithilfe eines gerichteten, azyklischen Graphen (*Directed Acyclic Graph (DAG)*), bestimmt. Diese Sortierung hat zur Folge, dass ein Actor, welcher ein Event sendet, immer vor den Actors ausgeführt wird, die dieses Event empfangen.

Falls kein azyklischer Graph aus dem DE Modell konstruiert werden kann, verbietet PtII generell dessen Ausführung. Der Grund dafür ist, dass aufgrund der vorhandenen Zyklen (sog. *Zero-Delay Loops*) die Levels nicht bestimmt und somit kein Determinismus garantiert werden kann. Um dennoch ein ausführbares Modell zu erhalten, muss in jeder Zero-Delay Schleife im Modell ein Actor platziert werden, der mind. eine Verzögerung bzgl. n aufweist, bspw. ein `TimeDelay`, `Register` oder `Microstep` Actor (vgl. das Beispielmodell in Abb. 2.11). Weitere Details zur DE Domäne sind in [21, 136] zu finden.

2.3.3.2 QSS Domäne

Eine relativ neue und noch experimentelle Domäne sind die in Unterabschnitt 2.2.2.4 eingeführten QSS Methoden zur numerischen Lösung von gewöhnlichen Differentialgleichungen. Die Domäne ist durch drei Hauptkomponenten charakterisiert [87]:

1. Ein *QSS Director*, der einen DE Director erweitert, stellt die korrekte Verarbeitung von Events in chronologischer Reihenfolge sicher und bietet Standardparameter, die für alle QSS Integratoren in einem QSS Modell gültig sind.

2. Die *QSS Integratoren* realisieren die in Unterabschnitt 2.2.2.4 eingeführten, quantisierten Integratoren inkl. des spezifischen QSS Solvers. Sie bieten u. a. Parameter zur individuellen Einstellung des QSS Solvers und des absoluten und relativen Quantums. Diese Parameter überschreiben die des Directors.
3. Ein spezieller Token-Typ namens `SmoothToken`, der eine Erweiterung eines primitiven `DoubleToken` darstellt.

Das effektive Quantum aus dem absoluten und relativen Quantum ist bestimmt durch den größeren Wert des absoluten Quantums und dem Produkt aus dem relativen Quantum und dem aktuellen Wert des kontinuierlichen Zustands x_i.

Ein SmoothToken ist ein Event, das nicht nur den eigentlichen `double`-Wert enthält, sondern zusätzlich ein oder mehrere Ableitungswerte[7]. mithilfe der *Expression Language* [20] von PtII lässt sich ein SmoothToken mit z. B. zwei Ableitungswerten folgendermaßen definieren: `smoothToken(<value>, {<deriv1>, <deriv2>})`. Auf diese Art können stückweise stetige Signale zwischen Actors ausgetauscht werden, die entweder nur den eigentlichen Wert oder auch die Ableitungen verwenden können. Dies ist eine Voraussetzung für QSS Integratoren höherer Ordnung und Actors, die als statische Funktionen dienen sollen (siehe Unterabschnitt 2.2.2.4). Actors, die SmoothTokens empfangen, aber Werte zwischen zwei aufeinanderfolgenden Events benötigen, können mithilfe den zuletzt empfangenen Ableitungswerten den Token auf die aktuelle Zeit extrapolieren[8].

Dieses Verhalten ist im Token selbst gekapselt, daher muss das Verhalten der einzelnen Actors nicht zur Verarbeitung von SmoothTokens angepasst werden. Auch Operationen wie Addition und Multiplikation von SmoothTokens ist als polymorphe Operation realisiert und kann mit DoubleToken kompatiblen Actors direkt verwendet werden. Bei diesen Operationen werden zusätzlich die Ableitungen addiert bzw. bei Multiplikation die Produktregel von Ableitungen angewandt. Zu beachten ist, dass dies natürlich zusätzliche Rechenzeit im Vergleich zu regulären Operationen hervorruft. Momentan werden in PtII die QSS Verfahren QSS1-QSS3 sowie LIQSS1 und LIQSS2 unterstützt.

2.3.3.3 Modal Models

Zusätzlich zu den nebenläufigen Domänen bietet PtII auch Domänen mit sequenzieller Semantik, darunter insb. FSMs, eine weit verbreitete Domäne zur Modellierung von zustandsbasiertem Verhalten in komplexen Systemen. Eine FSM in PtII wird dabei in einem speziellen Typ eines zusammengesetzten Ac-

[7] Aktuell begrenzt auf bis zu drei Ableitungswerte.
[8] Zwei aufeinanderfolgende SmoothTokens signalisieren eine signifikante Änderung im stückweise stetigen Signal, d. h. eine Quantisierung des Zustands.

tor eingebettet, wird von einem FSM Director kontrolliert und ist deshalb ein ausführbarer, undurchlässiger Actor.

Innerhalb sind Actors nicht mehr länger nebenläufige Komponenten, sondern Zustände (engl. *states*), die über Transitionen (engl. *transitions*) miteinander verknüpft sind und ggf. Übergangsbedingungen (engl. *conditions* oder *guards*) aufweisen. Events an Eingangsports des zusammengesetzten Actors können u. a. auch als Übergangsbedingung dienen. Transitionen können beim Übergang gewisse Aktionen (engl. *Actions*) auslösen, die bspw. ein Token über einen Ausgangsport des zusammengesetzten Actors nach außen senden können (*Output Actions)*. Darüber hinaus können während der Transition interne Variablen bzw. Parameter über sog. *Set Actions* gesetzt bzw. manipuliert werden, was in erweiterten (engl. *extended*) FSM resultiert. Die grundlegende Ausführungssemantik des FSM Directors in den `fire()` und `postfire()` Operationen durchläuft folgende Schritte [136]:

- `fire()`
    1. Eingangsports einlesen.
    2. Übergangsbedingungen an ausgehenden Transitionen des aktuellen Zustands auswerten.
    3. Aktive Übergangsbedingung auswählen, welche `true` zurückliefert.
    4. Ausführung der Transition und dessen Output Actions.
- `postfire()`
    1. Ausführung der Set Actions der gewählten Transition.
    2. Aktualisierung des aktuellen Zustands auf den Zielzustand der gewählten Transition.

Die Aktualisierung des Zustands innerhalb der `postfire()` Operation ist wichtiger Bestandteil der abstrakten Semantik, um Domänenpolymorphie zu erhalten (vgl. Unterabschnitt 2.3.2.1).

Eine einfache FSM ist z. B. der in Abb. 2.11 enthaltene Supervisor Actor, dessen internes Verhalten in Abb. 2.13 illustriert ist. Die FSM befindet sich im initial definierten Zustand *off*, sodass die Last vom Generator getrennt ist. Der Automat wechselt in den Zustand *on*, sobald ein boolesches Event am Eingangsport *onOff* eintrifft, welches `true` ist. Während der Transition wird der Ausgangsport *contactor* auf `true` gesetzt, welcher den Generator mit der Last verbinden würde. Tritt aber gleichzeitig auch ein auf `true` gesetztes Event am Eingangsport *fault* auf, wechselt der Automat sofort in den sicheren und finalen Zustand *fault* und setzt den Ausgangsport während der Transition wieder auf `false`. Dies wird durch eine sog. sofortige (engl. *immediate*) Transition realisiert, welche den ursprünglichen

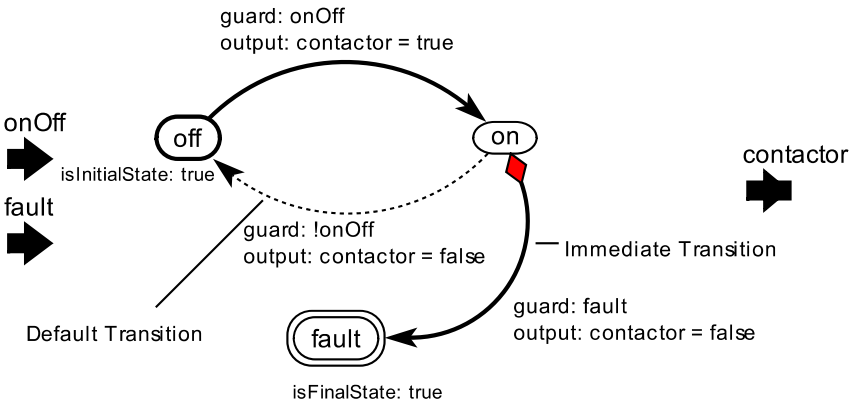


Abbildung 2.13: FSM Modell des Supervisor Actors aus Abb. 2.11 (Quelle: modifizierte Darstellung nach [136, S. 35]).

Zielzustand (hier: *on*) überspringt, sofern die Übergangsbedingung zum nächsten Zustand erfüllt ist (hier: *fault*). Die zu sehende, gestrichelte Transition ist eine *Default Transition*, die wie eine gewöhnliche Transition agiert, aber nur aktiv wird, wenn keine Übergangsbedingungen anderer ausgehenden Transitionen erfüllt sind.

Eine Besonderheit der FSM Domäne ist, dass zusätzlich hierarchische Zustände deterministisch modelliert und simuliert werden können. Diese Art von FSM Modellen werden in PtII *Modal Models* [89, 136] genannt. Innerhalb dieser Modelle können Zustände nicht nur durch weitere FSMs verfeinert werden, sondern durch beliebige Submodelle anderer Domänen. Dies gestattet die deterministische Konstruktion von Modellen hybrider bzw. cyber-physischer Systeme, die diskretes Verhalten mit kontinuierlichen, physikalischen Prozessen koppeln.

Die Ausführungssemantik von hierarchischen FSM Modellen bzw. Modal Models folgt dabei einem feineren, aber dennoch einfachen Muster:

- Wird ein hierarchischer Zustand innerhalb eines Modal Model gefeuert, werden zuerst seine Verfeinerungen (Submodelle oder weitere FSMs) gefeuert.
- Übergangsbedingungen an ausgehenden Transitionen der verfeinerten FSM auf niedrigerer Hierarchieebene werden ausgewertet, eine aktive Transition ausgewählt und Output Actions ausgelöst.

- Übergangsbedingungen an ausgehenden Transitionen des hierarchischen Zustands werden ausgewertet. Diese können Werte von Ausgangsports referenzieren, die von der verfeinerten FSM geschrieben worden sind.
- Eine aktive Transition des hierarchischen Zustands wird ausgewählt und Output Actions ausgelöst.
  - Falls die Transition des hierarchischen Zustands und der verfeinerte Zustand eine Output Action am selben Ausgangsport auslösen, überschreibt die Transition des hierarchischen Zustands die seines verfeinerten Zustands, da die Ausführung streng sequenziell erfolgt.

Analoges gilt für die `postfire()` Operation, bei der zunächst die Submodelle ausgeführt werden und dann die Transition, die den entsprechenden Zielzustand aktualisiert. Gegebenenfalls werden interne Variablen überschrieben, die bereits intern von Submodellen geschrieben worden sind.

Eine Ausnahme vom obigen Verhalten erfolgt, falls eine sog. verdrängende (engl. *preemptive)* Transition auf einem verfeinerten Zustand S_r modelliert wird. Falls S_r der aktuelle Zustand ist und die Übergangsbedingung einer solchen ausgehenden Transition von S_r erfüllt ist, wird das Submodell *nicht* vorher ausgeführt. Ein weiterer, wichtiger Transitions-Typ im Kontext von hierarchischen FSMs, wie er in UML State Charts [127] in Form von Pseudozuständen vorkommt, ist die *History Transition*. Diese erweitern gewöhnliche Transitionen, indem beim Übergang in den verfeinerten Zielzustand S_r dessen internes Submodell und ggf. weitere Submodelle *nicht* durch die `initialize()` Operation in den initialen Zustand zurückgesetzt werden. Stattdessen werden das Modell und seine Submodelle ausgehend vom zuletzt aktiven Zustand fortgesetzt, vorausgesetzt der Zustand S_r wird nicht erstmals betreten. Die Semantik entspricht demnach der des *Deep History* Pseudozustands von UML. Weitere Transitions-Typen und Details zu Modal Models sind in [89, 136] zu finden.

3 Stand der Technik und Forschung

3.1 Modellbasierter Entwurf von E/E-Architekturen

3.1.1 EAST-ADL

Die *Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL)* [33] ist eine Architekturbeschreibungssprache (engl. *Architecture Description Language (ADL)*) für E/E-Architekturen und wurde ursprünglich im EAST-EEA[1] Projekt entwickelt, die durch die Ergebnisse der ATTEST und ATTEST2 Projekte zur EAST-ADL2 weiterentwickelt wurde. Durch den Einfluss weiterer Projekte wie TIMMO[2], TIMMO-2-USE[3] und MAENAD[4] liegt sie in der Version 2.1.12 vor. Für einen guten Überblick über EAST-ADL bezogene Projekte sei auf [1, S. 23 ff.] verwiesen.

3.1.1.1 Aufbau

Die EAST-ADL bietet ein *Meta Object Facility (MOF)* basiertes [125, 128] Metamodell und wurde im Zuge ihrer Entwicklung stark am AUTOSAR [9] Standard orientiert. Mithilfe dieses Informationsmodells werden E/E-Architekturen in ihren Abstraktionsebenen mit unterschiedlicher Granularität beschrieben. Orthogonale Aspekte wie Anforderungen, Variabilität, Timing und funktionale Sicherheit werden ebenfalls berücksichtigt. [14, 33] Der Aufbau der EAST-ADL ist in Abb. 3.1 dargestellt.

Das Systemmodell (*System Model*) beschreibt die E/E-Architektur eines Fahrzeugs auf vier unterschiedlichen Abstraktionsebenen. Die Systembeschreibung ist dabei auf jeder Ebene vollständig entsprechend ihrer Granularität [14]. Auf der Fahrzeugebene *(Vehicle Level)* wird das Fahrzeug mit einer Menge von erlebbaren Features anhand Feature-Modellen in einer Baumstruktur modelliert, die den Funktionsumfang des Fahrzeugs lösungsunabhängig beschreiben (*Technical*

[1] https://itea3.org/project/east-eea.html, zuletzt aufgerufen am 28.05.2018.
[2] https://itea3.org/project/timmo.html, zuletzt aufgerufen am 28.05.2018
[3] https://itea3.org/project/timmo-2-use.html, zuletzt aufgerufen am 28.05.2018.
[4] http://www.maenad.eu/index.html, zuletzt aufgerufen am 28.05.2018.

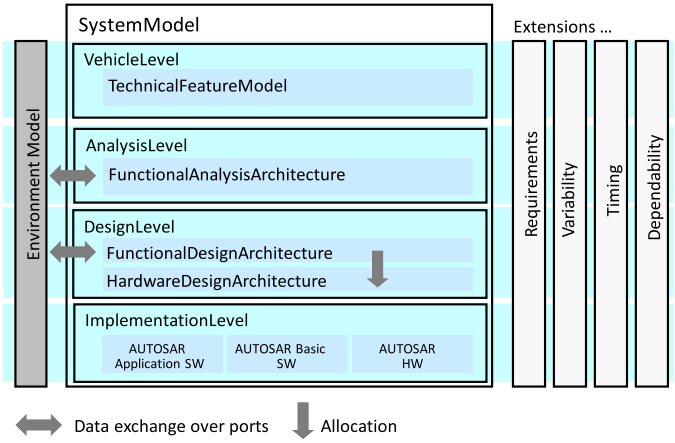


Abbildung 3.1: Aufbau der EAST-ADL (Quelle: [14, S. 7]).

Feature Model). Features können mit Anforderungen, Use Cases etc. verknüpft werden und Relationen untereinander aufweisen, um Variantenmanagement zu betreiben.

Die Analyseebene (*Analysis Level*) enthält die *Functional Analysis Architecture (FAA)*, die die E/E-Architektur des Fahrzeugs aus funktionaler Sicht mittels einer abstrakten Funktionsarchitektur beschreibt. Diese definiert die logische Funktionalität und Dekomposition mithilfe hierarchischer und über Ports verbundene Funktionsblöcke, welche die Features der Fahrzeugebene realisieren. Funktionen werden auf dieser Ebene über *AnalysisFunction* und *FunctionDevice*, welche die Schnittstelle zur Umwelt bilden, modelliert. [14] [33, S. 21]

Die Designebene (*Design Level*) beinhaltet die *Functional Design Architecture (FDA)* und die *Hardware Design Architecture (HDA)*. Die FDA ist eine implementierungsorientierte Dekomposition von Funktionen der FAA, indem nicht-funktionale Eigenschaften wie Effizienz, Hardware-Allokation, Änderungsmodi und funktionale Sicherheit mit berücksichtigt werden, um den Einfluss auf das Applikationsverhalten abschätzen zu können [33, S. 21] [14].

Funktionen auf dieser Ebene können bspw. *DesignFunction* und *LocalDeviceManager* sein, welche im Sinne von AUTOSAR Applikationssoftware realisieren, während *BasicSoftwareFunction* sich auf Middleware bezieht. *HardwareFunction* Funktionen repräsentieren das logische Verhalten von Hardwarekomponenten und bilden die Schnittstelle zur Umwelt. Funktionen werden sowohl auf der FAA als auch auf der FDA typisiert instanziiert, welche dann in Anlehnung an

AUTOSAR Prototypen genannt werden. Die HDA repräsentiert die Hardwarearchitektur der eingebetteten Systeme, wie diese miteinander verbunden sind (bspw. über Bussysteme) und welche Funktionen der FDA darauf abgebildet werden. Sie beinhaltet auch die Darstellung von elektrischen Verbindungen, Konnektoren und Pins.

Die Implementierungsebene (*Implementation Level*) repräsentiert die Softwarearchitektur und deren Abbildung auf die Hardware, wird aber nicht durch die EAST-ADL vorgegeben, sondern durch AUTOSAR Elemente beschrieben [14].

Komplementär zum Systemmodell existiert ein Umgebungsmodell (*Environment*), das die Interaktion der E/E-Architektur mit der Umwelt beschreibt. Eine Funktion wird hier als kontinuierliches Fahrzeugverhalten interpretiert oder kann Teil der Umgebung sein, etwa ein anderes Fahrzeug. Zusätzlich zur Umgebung existieren weitere orthogonale Erweiterungen zur Betrachtung von funktionaler Sicherheit nach ISO 26262 [56] und Timing – welche wesentlich durch das MAENAD Projekt getrieben wurden – sowie Variantenmanagement und Verhalten. Artefakte dieser vertikalen Ebenen können Artefakte des Systemmodells auf allen Ebenen über Relationen referenzieren [14]. Die eigentliche Analyse, wie bspw. FMEA, Fehlerbaumanalyse, Scheduling-Analyse bzgl. Timing etc., ist externen Tools vorbehalten, zu denen spezifische Gateways entwickelt werden müssen [14].

Die einzelnen Ebenen sind ISO 26262 konform und orientieren sich an der Vorgehensweise des V-Modells, da auf jeder Ebene dedizierte Verifikations- und Validierungsbeschreibungen in Form von *Cases* vorgesehen sind [14]. Zudem wird die Nachvollziehbarkeit innerhalb des Entwicklungsprozesses über Relationen von Artefakten einer Ebene zu ihrer nächst höheren und zu vertikalen Ebenen adressiert.

3.1.1.2 Verhaltensmodellierung

Zur Modellierung von Verhalten existieren in der EAST-ADL zwei Ansätze. Zum einen können Funktionstypen der Analyse- und Designebene mit sog. *FunctionBehavior* Blöcken assoziiert werden, welche zur Spezifikation von Blackbox-Verhalten herangezogen werden. Das heißt das eigentliche interne Verhalten der Funktion wird nicht modelliert, sondern es wird lediglich auf ein externes, sich außerhalb des EAST-ADL-Modells befindlichen Verhaltensmodell referenziert, welches in einer werkzeugspezifischen Sprache vorliegt. Die Transparenz und Konsistenz zwischen Architektur- und Verhaltensmodell wird somit erschwert, insb. bei verteilter Kollaboration am Architekturmodell.

Um diese Nachteile zu umgehen ist andererseits ein Behavioral Annex [33, 14] entstanden, der es erlaubt sog. *Behavioral Constraints* an Funktionsblöcke zu annotieren. Die *Behavioral Constraints* folgen dabei analog zu den Funktionsblöcken dem Typ/Instanz-Prinzip. Dabei wird unterschieden zwischen *Temporal Constraints*, zur Assoziation von Funktionen mit Verhalten in Form von FSMs, *Quantification Constraints*, zur Deklaration von Datentypen und Übergangsbedingungen (engl. guards) sowie *Computation Constraints*, zur Modellierung von logischen Pfaden, Datentransformationen oder Zustandsübergängen in FSMs.

Insbesondere mithilfe der Temporal Constraints ist es möglich, Funktionen innerhalb des E/E-Architekturmodells mit einfachem FSM Verhalten zu annotieren. Die EAST-ADL legt dabei fest, dass Funktionen mit einem Behavioral Constraint eine fixe, synchrone Ausführungssemantik nach dem *run-to-completion* Prinzip besitzen [96, 97], vergleichbar mit dem SDF Berechnungsmodell. Die Schnittstellen zwischen den FSMs und Funktionen werden mittels sog. *Binding Relationen* hergestellt. Prototypen zum Export von Simulink-Modellen mittels der Referenzierung von Funktionen der Designebene auf externe Verhaltensmodelle sowie von Stateflow-Modellen aus Temporal Constraints sind im Zuge des MAENAD[5] Projektes entstanden (siehe auch Unterabschnitt 3.4.3).

3.1.2 AADL

Die *Architecture Analysis and Design Language (AADL)* [145] ist eine Architekturbeschreibungssprache zur Modellierung und Analyse von eingebetteten sicherheitskritischen Echtzeitsystemen und wird primär in der Avionik eingesetzt. Ein verbreitetes Werkzeug zur Erstellung von AADL Modellen ist OSATE[6].

Die AADL bietet im Wesentlichen ein Framework zur statischen Beschreibung einer modularen Softwarearchitektur, der Rechenplattform, auf der die Software ausgeführt werden soll, sowie Hardwarebausteine wie Busse und andere physikalische Bausteine, mit denen das System interagiert. Ein System ist dabei ein übergeordnetes Element, besteht aus ein oder mehreren Subkomponenten und kann selbst in anderen Systemen verwendet werden. Die Dynamik wird in Form von kommunizierenden Tasks und nicht-funktionalen Eigenschaften wie Ausführungszeit und Deadlines von Tasks sowie Prozessor- und Speicherressourcen beschrieben [36, S. xvi & Kap. 3]. Zudem können Ende-zu-Ende Wirkketten spezifiziert werden.

Die primär eingesetzten Elemente zur Beschreibung von Software sind *Process*, *Threads*, *Data* und *Subprogram*. Ein Prozess beinhaltet Threads und (geteilte) Daten,

[5] http://www.maenad.eu/index.html, zuletzt aufgerufen am 28.05.2018.
[6] http://osate.org/, zuletzt aufgerufen am 28.05.2018.

wobei Threads sequenzielle Abläufe darstellen, parallel zu anderen Threads existieren können und auf den Daten arbeiten. Subprogramme können von Threads oder Subprogrammen aufgerufen werden. Die Art der Kommunikation zwischen Hardware-/Softwarekomponenten wird über Schnittstellen beschrieben, Features genannt, und die Komponenten werden über Connections miteinander verbunden werden. Schnittstellen sind i. d. R. Event- oder Daten-*Ports*. Über sog. Modi-Deklarationen kann spezifiziert werden, dass Threads und die dazugehörigen Schnittstellen nur in bestimmten Modi aktiv sind. Modi-Übergänge werden typischerweise durch eingehende Event-Ports ausgelöst [36, Kap. 3].

Die Hardwarekomponenten werden im Wesentlichen durch *Processor*, *Device* und *Bus* beschrieben, die für die Ausführung der Threads zuständig sind. Devices sind Sensoren, Aktoren o. Ä., die die Schnittstelle zur Umwelt darstellen, Busse verbinden die Hardwarekomponenten miteinander.

Zusätzlich zur eigentlichen AADL sind zwei Annexe entstanden, die einige Modellierungserweiterungen bereitstellen. Annex 1 [144] wurde 2006 veröffentlicht und beinhaltet Erweiterungen zur grafischen Notation der AADL, eine Spezifikation des Metamodells und XMI Austauschformats, Richtlinien für AADL konformen Quelltext in C und ADA sowie ein Error Model Annex zur Modellierung von Fehlertypen, Fehlverhalten und Fehlerpropagation zur Steigerung der Zuverlässigkeit eines Systems. Annex 1 wurde 2015 überarbeitet und Möglichkeiten zur Code-Generierung von Laufzeitsystemen aus AADL-Modellen hinzugefügt [146].

Annex 2 [148] wurde ursprünglich 2011 publiziert und umfasst Erweiterungen zur Spezifikation von Richtlinien zur Assoziation der AADL mit anderen Modellierungssprachen, ein Annex zur verfeinerten Modellierung von Verhalten von Komponenten und einen ARINC653 Annex. Nach einer Überarbeitung der Kernsprache bzgl. einiger Errata und geringfügigen Verbesserungen von Version 2.1 auf 2.2 im Jahre 2017 [147], wurde Annex 2 im Februar 2019 entsprechend darauf angepasst.

Der *Behavioral Annex* beinhaltet Erweiterungen zur verfeinerten Spezifikation von Verhalten sowohl von Komponenten als auch Interaktionen, um das Systemverhalten speziell sicherheitskritischer Applikationen präziser analysieren zu können. Das Verhalten wird mithilfe von einfachen FSMs beschrieben und umfasst fünf Subsprachen, um diese innerhalb einer AADL Komponente, z. B. in einem Thread, zu spezifizieren [79]. Im Zuge der Entwicklung dieses Annexes entstanden zudem Arbeiten wie [46, 95, 195], die sich mit der Definition oder Interpretation der Semantik beschäftigen. Das Werkzeug OSATE[7] unterstützt die Modellierung des Verhaltensannexes und erlaubt zudem den Import von Simulink-Modellen, die in ein entsprechendes AADL-Modell übersetzt werden.

[7] http://osate.org/about-osate.html, zuletzt zugegriffen am 28.05.2018.

Ein Export oder die Simulation des importierten Modells werden allerdings nicht erwähnt.

Im Kontext der Automotive-Domäne und verglichen mit anderen Architekturbeschreibungssprachen, wie z. B. der EAST-ADL oder *Electric/Electronic-Architecture - Analysis Design Language (EEA-ADL)* (siehe Abschnitt 3.2), bietet die AADL einen limitierten Umfang. Die modellbasierte Anforderungsentwicklung wird zwar aktuell in einem weiteren *Requirements Definition and Analysis Annex* entwickelt [36, S. xvi] und prototypisch im Werkzeug OSATE umgesetzt, ist aber noch nicht offiziell publiziert. Darüber hinaus werden weitere Aspekte wie Variantenmanagement oder die Nachvollziehbarkeit über alle Ebenen nicht oder nicht ausreichend unterstützt. Automotive-spezifische Bussysteme werden ebenso wenig unterstützt wie die elektrische und physikalische Repräsentation von Verbindungen oder die Topologie. Die in der Automobilindustrie etablierte Softwareentwicklung nach dem AUTOSAR Standard wird auch nicht betrachtet. [14]

3.1.3 SPES 2020 / SPES_XT

Software Platform Embedded Systems (SPES) 2020 [133] war ein von 2009 bis 2012 vom BMBF gefördertes Forschungsprojekt „zur Entwicklung einer Methodik zur durchgängig modellbasierten Entwicklung von eingebetteten Systemen“[8] mit Fokus auf eingebetteter Software.

Das Ergebnis ist ein Entwicklungsprozess mit vier Sichtweisen (engl. view points) und je mehreren Abstraktionsebenen zur Verfeinerung bzw. Hierarchisierung der einzelnen Ansichten. Die Ansichten sind teilweise mit den Abstraktionsebenen einer E/E-Architektur aus Abb. 2.2 vergleichbar und werden in *Anforderungssicht*, *funktionale Sicht*, *logische Sicht* und *technische Sicht* unterteilt. Artefakte können über Sichten und Abstraktionsebenen hinweg über Relationen aufeinander abgebildet werden, um die durchgängige Nachvollziehbarkeit im Entwicklungsprozess zu adressieren. Eine Sicht zur Berücksichtigung der Topologie wird jedoch nicht betrachtet [133, Kap. 3].

Verglichen mit den Abstraktionsebenen der EEA-ADL entspricht die Anforderungssicht den Produktzielen, die funktionale Sicht der logischen Architektur, die logische Sicht der System-Softwarearchitektur und die technische Sicht der Hardware-Vernetzung (vgl. Unterabschnitt 3.2.2.1 bis 3.2.2.4). In der funktionalen Sicht wird jedoch im Gegensatz zur logischen Architektur der EEA-ADL keine Spezifikation der Kommunikation mit Signalen vorgenommen [133, Kap. 12, S. 170 ff.].

[8] http://spes2020.informatik.tu-muenchen.de/, zuletzt aufgerufen am 23.05.2018.

Funktionen werden dabei in Blackbox- und Whitebox-Funktionen hierarchisiert, wobei letztere die interne Struktur einer Blackbox repräsentieren und als atomar betrachtet werden. In Summe müssen diese dasselbe Verhalten aufweisen wie die umgebende Blackbox an ihren Schnittstellen, daher werden Port-Mappings in einer $m : n$-Abbildung eingeführt [133, Kap. 5.4, S. 76]. Um Whitebox-Funktionen wiederverwenden zu können, müssen daher zusätzliche Konsistenz-Checks an den Schnittstellen durchgeführt werden [133, S. 77].

Gleichzeitig werden Qualitätsaspekte in Form von funktionaler Sicherheit [133, Kap. 8] und Echtzeitanforderungen gemeinsam als querschnittliche Eigenschaften über alle Sichten betrachtet. Dabei werden plattformunabhängige und -spezifische Anforderungen wie bspw. Aktivierungsrichtlinien und Laufzeiten von Software-Tasks oder Hardware- und Kommunikationsressourcen mithilfe von domänenspezifischen Sprachen modelliert. Die letzteren Konstrukte sollen untereinander und mit der Anforderungssicht verknüpft werden können, um eine Verifizierung zu ermöglichen. Basierend auf den formalen Modellen können statische Ausführungspläne der Software generiert und mittels Modellchecks verifiziert werden [133, Kap. 9].

SPES_XT[9] ist das Ergebnis des Folgeprojekts und hatte zum Ziel, die erarbeiteten Konzepte zu vertiefen als auch die praxistaugliche Umsetzung in der Industrie zu untersuchen. Es erweitert die SPES 2020 Methodik um drei Kernpunkte [132, Kap. 3]: (1) ein *Context Framework* zur Modellierung von weiteren Sichten auf Systemkomponenten (strukturell, funktional, operativ), (2) ein *Prozessbeschreibungs-Framework* zur modellbasierten Beschreibung von Entwicklungsprozessen und (3) Erweiterungen zur Einbettung des SPES-Frameworks in einen *generischen Systems-Engineering-Prozess* unterschiedlicher Disziplinen.

Des Weiteren werden Konzepte zur Modellierung von physikalischen Komponenten mittels hybriden Automaten [139], von Funktionsnetzwerken mittels Sequenzdiagrammen, von funktionaler Sicherheit mittels Kontextdiagrammen sowie von Varianten mittels Ontologien vorgeschlagen. Zusätzlich wird ein generischer Ansatz zur Entwurfsraumexploration von eingebetteten Systemen eingeführt und eine Implementierung auf Basis von Eclipse skizziert. Letztere fokussiert auf eine Werkzeugintegration, eine konkrete Realisierung wird jedoch nicht diskutiert [132, Kap. 3].

3.1.4 Volcano™ Vehicle Systems Architect

Vehicle Systems Architect (VSA) [108] ist ein Werkzeug für den Systementwurf von AUTOSAR-basierten Systemen und Teil von Mentor Graphics' Volcano™ Tool-

[9] http://spes2020.informatik.tu-muenchen.de/spes_xt-home.html, zuletzt aufgerufen am 23.05.2018.

Suite. Die Stärken von VSA sind das Design der Hardware-Vernetzung bestehend aus ECUs und deren Kommunikationsnetzwerk sowie der Softwarearchitektur mit AUTOSAR Softwarekomponenten und deren Verbindungen mit Ports (siehe Unterabschnitt 2.1.3).

Softwarekomponenten können auf die ECUs abgebildet und die entsprechenden Systemsignale definiert werden [108, 184]. Unterstützt wird auch das Netzwerkdesign in Form von auszutauschenden Datenelementen und deren Signalen. Aufbau und Definition von Frames sowie eine Timing-Analyse für spezifische Bussysteme müssen allerdings im separaten Werkzeug *VSA COM Designer* der Volcano™ Familie stattfinden. Die E/E-Architektur kann mithilfe der bereits vorgestellten EAST-ADL beschrieben werden und erlaubt damit auch die Spezifikation von Timing-Anforderungen auf Systemebene (siehe Unterabschnitt 3.1.1). Es unterstützt zudem die Generierung von AUTOSAR-konformen Konfigurationsdateien für ECUs und eine proprietäre Skriptsprache zur Implementierung von Konsistenz-Checks oder benutzerdefinierten Operationen [108].

Anforderungen, Funktionsumfänge und detailliertere Ebenen nach Abb. 2.2 werden jedoch nicht betrachtet, wie z. B. die elektrische Ebene, der Leitungssatz und die Topologie. Eine Möglichkeit zur Handhabung von Varianten wird ebenso wenig adressiert wie eine integrierte Verhaltensmodellierung.

3.1.5 Capital® Tool-Suite

Die Capital® Tool-Suite[10] ist Mentor Graphics' Lösung für das elektrische Design des Bordnetzes. Aus dem logischen und elektrischen Design können physikalische Verbindungen und Bauteile des Leitungssatzes durch benutzerdefinierte Regeln synthetisiert sowie das Routing in der Topologie vorgenommen werden [107]. Quasi-statische Strom- und Spannungsanalysen von Subsystemen und Systemdiagrammen sind im Werkzeug Capital Analysis möglich, ähnlich zu den statischen Stromanalysen in PREEvision® (siehe Unterabschnitt 3.2.5). Im Juni 2018 wurde der Entwurf für SAE J1939-basierte CAN-Netzwerke mit dem zusätzlichen Werkzeug Capital Systems Networks mit integriert[11], das bereits geroutete Signale aus Capital Systems Architect auf Netzwerkträger synthetisiert.

Trotz dieser Integration eines weiteren Werkzeugs, ist das Design jedoch über mehrere Produktfamilien verteilt, was eine integrierte und ganzheitliche Betrachtungsweise einer E/E-Architektur zu höheren Abstraktionsebenen durch ein fehlendes, durchgängiges Metamodell erschwert. Aus demselben Grund gestaltet sich insb. die Aufrechterhaltung der Konsistenz der E/E-Architektur über Ebenen

[10] https://www.mentor.com/products/electrical-design-software/capital, zuletzt aufgerufen am 19.06.2018.
[11] https://www.mentor.com/products/electrical-design-software/networks, zuletzt aufgerufen am 19.06.2018.

hinweg bei verteilter Kollaboration als schwierig, da direkt abhängige Wechselwirkungen bei Änderungen auf einer dieser Ebenen folgen können. Dazu zählen z. B. Anforderungen an Funktionen und Softwarekomponenten, deren Abbildung auf Hardwarekomponenten, die auszutauschenden Signale sowie die Verbauorte der Steuergeräte und die Verlegewege des Leitungssatzes in der Topologie.

Die Modellierung von Verhalten in den verschiedenen Werkzeugen, vor allem ein ebenenübergreifendes Verhalten in Verbindung mit dem Netzwerk- und elektrischen Design in Capital Systems Network bzw. Analysis, wird nicht näher diskutiert. Eine Verfeinerung von logischem Funktionsverhalten mit elektrischen Schaltungen aus dem Architekturmodell oder die Simulation von Funktionen unter Einfluss von Netzwerkkommunikation ist somit bspw. nicht möglich.

3.1.6 Weitere Ansätze

Ein Überblick und qualitativer Vergleich von Werkzeugen entlang des V-Modells, insb. auf Systemebene ist in [184] zu finden. Ein weit verbreitetes zur universellen Modellierung von eingebetteter Software basierend auf SysML ist *Rhapsody Designer* von IBM. Es unterstützt die Erstellung und Analyse von Anforderungen sowie die Verhaltensmodellierung von Softwarekomponenten mittels State Charts, Sequenz- und Aktivitätsdiagrammen. Letztere können simuliert und Code dafür generiert werden. Zudem existiert ein UML-Profil für AUTOSAR, mit welchem *arxml*-Konfigurationen der Software exportiert werden können, analog zu Volcano™ VSA und PREEvision®. Eine Verknüpfung zu detaillierteren *Elektrik/Elektronik-Architektur (EEA)*-Abstraktionsebenen nach Abb. 2.2 und Kommunikationsnetzwerke werden jedoch nicht adressiert.

Ein weiterer Ansatz basierend auf SysML und dem UML-Profil *Modeling and Analysis of Real-Time and Embedded Systems (MARTE)* [124] wird in [191, 192] beschrieben. Der Ansatz orientiert sich an der EAST-ADL und erweitert diese um Aktivitätsdiagramme aus der SysML zur Verhaltensmodellierung sowie um ein Replikations-Profil zur Abbildung von Redundanzen. MARTE-Konstrukte werden verwendet, um ECUs und Busse zu spezifizieren und um Timing-Informationen wie Deadlines, Aktivierungsperioden und Ausführungszeit von Aktivitäten zu annotieren. Der Ansatz ist im Rahmen des MAENAD Projekts entstanden (siehe Unterabschnitt 3.1.1) und ist darüber hinaus in der Lage, ausgehend von einer Spezifikation in der EAST-ADL Sprache, über ein Gateway automatisiert limitierte AUTOSAR-Modelle in Form von arxml-Konfigurationen zu generieren. Dabei werden simple Abbildungsstrategien verwendet. Zur Durchführung von Timing-Analysen existiert ein Konverter, um die angereicherten EAST-ADL-Modelle in das Format des Analysewerkzeugs *Qompass* zu übersetzen.

Ein ähnlicher Ansatz basierend auf SysML und MARTE wird auch in [109] und [133, Kap. 12] vorgestellt, die Timing-Simulation und Verifikation der Timing-Anforderungen finden aber im Werkzeug chronSim [122] statt. Ein alternativer Ansatz wird in [39] vorgestellt. In diesem werden SysML Parameter-Diagramme verwendet, die auf MATLAB/Simulink-Befehle verweisen und von einem Modul des Werkzeugs Cameo Systems Modeler getriggert werden. Über dessen bestehende Schnittstelle zu MATLAB/Simulink werden die simulierten Daten mit den in SysML modellierten Anforderungen abgeglichen. Ein ähnlicher Ansatz wird in [157] präsentiert, welcher eine Schnittstelle zwischen UML/SysML basierten Werkzeugen und MATLAB/Simulink beschreibt. Über einen Import/Export-Prozess können SysML-Modelle in Simulink-Modelle übersetzt und umgekehrt wieder importiert werden. Das eigentliche Verhalten eines SysML-Blocks wird jedoch nicht integriert im Systemmodell modelliert, sondern in Simulink.

3.1.7 Diskussion

Die vorgestellten Ansätze und Werkzeuge bieten alle umfangreiche Beschreibungsmöglichkeiten von E/E-Architekturen als Ganzes als auch von eingebetteten Systemen im Einzelnen. Dazu zählen insb. die EAST-ADL, nicht nur zur Beschreibung von E/E-Architekturen, sondern auch von nicht-funktionalen, vertikalen Eigenschaften wie Abhängigkeiten, Varianten, Timings oder funktionale Sicherheit. Kommerzielle Werkzeuge, die sie bereits mit einbinden, etwa das vorgestellte Vehicle Systems Architect, unterstreichen die Akzeptanz. Die AADL hingegen zielt mehr auf die detaillierte Beschreibung von eingebetteten Hardware/Softwarekomponenten sowie deren Vernetzung ab. Nicht-funktionale Eigenschaften und Verhaltensspezifikation werden durch zusätzliche Annexe adressiert. Die SPES Methodik beschreibt die grundlegenden Ebenen und Sichten, die bei der Entwicklung von verteilten, eingebetteten Systemen berücksichtigt werden sollen und spiegeln sich in etablierten Werkzeugen wie PREEvision® und Capital als auch in Beschreibungssprachen wie der EAST-ADL wider.

Detaillierte Ebenen einer E/E-Architektur nach Abb. 2.2 werden jedoch nicht berücksichtigt. Gleiches gilt auch für die AADL. Die EAST-ADL greift hier etwas tiefer, indem die Modellierung von elektrischen Verbindungen zwar unterstützt wird[12], physikalische Leitungen und Kabel sowie deren Verlegewege durch die Fahrzeugtopologie werden durch das Metamodell der Hardware-Architektur jedoch nicht abgedeckt [33, S. 55 ff.]. Im Gegensatz dazu spezialisiert sich die diskutierte Capital® Tool-Suite genau auf diese Bordnetz-relevanten Details, lässt aber Verknüpfungen zu höheren Ebenen vermissen und werden nicht durch ein

[12] „Hardware connectors represent wires that **electrically** connect the hardware components through its pins." [33, S. 59]

durchgängiges Metamodell abgedeckt. Zusammenfassend lässt sich feststellen, dass keiner der diskutierten Ansätze und Werkzeuge ein einheitlich zugrunde liegendes Metamodell zur durchgängigen Beschreibung einer kompletten E/E-Architektur nach Abb. 2.2 bereitstellt und gleichzeitig abstraktionsebenenübergreifende Mechanismen wie Abhängigkeiten, Varianten, Timing-Anforderungen und funktionale Sicherheit etc. berücksichtigt. Hier setzt die im nächsten Abschnitt präsentierte EEA-ADL an, die sich im Werkzeug PREEvision® zu einem industrieweiten de-facto Standard etabliert hat.

Die *architekturzentrierte Verhaltensmodellierung* von dedizierten E/E-Architekturkomponenten auf den höheren Ebenen findet vermehrt Zuspruch. Zum einen um das Systemverhalten bereits in frühen Entwicklungsphasen präziser analysieren zu können, zum anderen um eine formale Modellierung und Verknüpfung des Verhaltens mit der Architektur zu erreichen. Der Trend wird durch die bereits erläuterten Verhaltensannexe der EAST-ADL und AADL untermauert.

Der erste Ansatz der EAST-ADL referenziert lediglich Funktionstypen der Analyse- und Designebene auf externe Verhaltensmodelle und ist für eine integrierte Betrachtungsweise nicht geeignet und erschwert daher die Transparenz zwischen Architekturmodell und Verhalten, aber auch die verteilte und konsistente Kollaboration mit mehreren Teams. Der zweite Ansatz der Behavioral Constraints erlaubt eine Annotation von Verhalten direkt im Architekturmodell in Form von einfachen Zustandsmaschinen. Zusätzliche Constraints wie Berechnungsvorschriften oder zur Verfügung stehende Übergangsbedingungen müssen in separaten Quantification und Computation Constraints Diagrammen spezifiziert werden [96, Kap. 3], was schnell zu schwer lesbaren und wartbaren Spezifikationen führen kann. Die synchrone Ausführungssemantik jener Funktionen, die durch Behavioral Constraints annotiert sind, ist fix vorgegeben. Dies ist jedoch u. U. nicht immer anwendbar und auch nicht effizient, da für unterschiedliche Domänen geeignetere Berechnungsmodelle zur Verfügung stehen können, wie z. B. für Netzwerkkommunikation eine asynchrone DE Semantik (siehe Unterabschnitt 2.2.2). Hinsichtlich der Spezifikation von Verhalten gilt analoges für die AADL, die es mittels einfachen FSMs, allerdings bestehend aus nunmehr fünf Subsprachen, spezifiziert.

Ein Vorteil dieses Vorgehens ist, dass das Architektur- und Verhaltensmodell zu jeder Zeit konsistent und zunächst auch werkzeugunabhängig ist. Komplexeres Verhalten lässt sich mit flachen Zustandsmaschinen allerdings nur noch schwer und unübersichtlich darstellen und führt zur sog. *Zustands- und Transitions-Explosion*. Hauptgrund dafür ist die fehlende Hierarchisierung. Zudem können keine parallelen oder History-Zustände, wie es bspw. mit UML State Charts möglich ist, modelliert werden. Xu et al. [194] schlagen eine Metamodellerweiterung des Verhaltensannexes der AADL um diese Art von Zuständen vor und imple-

mentieren die Erweiterungen im Werkzeug OSATE. Die Transformation in ein ausführbares Modell bzw. eine Simulation ist jedoch nicht möglich.

Die Kombination mit weiteren Verhaltensformalismen und Berechnungsmodellen zur detaillierten Modellierung kann nicht integriert stattfinden. Des Weiteren ist eine integrierte Verfeinerung der Komponenten durch die Anreicherung mit direkt ausführbaren Blöcken aus einer Bibliothek, wie bspw. Ptolemy II, Simulink oder gar werkzeugunabhängige FMUs (vgl. Unterabschnitt 3.3.2.1) nicht möglich. Stattdessen muss ein detaillierteres Verhalten in externen Werkzeugen modelliert und ausgeführt werden. Ein Austausch einer bestimmten Funktionalität durch alternative Realisierungen des Verhaltens wird somit auch unterbunden und die Konsistenz zwischen Verhaltens- und Architekturmodell ist nicht mehr sichergestellt.

Auch in der SPES Methodik ist eine Verhaltensmodellierung der sog. Whitebox-Funktionen in der funktionalen Sicht mittels FSMs vorgesehen als auch die Modellierung von kontinuierlichem Verhalten mittels hybriden Automaten. Jedoch wird die Modellierung und Simulation von ausführbarem Verhalten methodisch nicht integriert betrachtet, sondern in SPES_XT durch sog. *Analysis* Blöcke innerhalb einer Prozessbeschreibung modelliert, die die Schnittstelle zu externen Modellen bilden und die Nachverfolgbarkeit gewährleisten sollen. Die Realisierung dessen sowie die Rückführung der Simulationsdaten und die Assoziation mit dem Systemmodell werden jedoch nicht näher diskutiert [132, Kap. 3].

Die in Unterabschnitt 3.1.6 vorgestellten SysML-Ansätze bieten den Vorteil, zusätzlich zu Zustandsdiagrammen auch Aktivitäts- und Sequenzdiagramme abzubilden. Auch hier kann allerdings kein verfeinertes und ausführbares Verhalten in Form von funktionalem Verhalten hinterlegt werden. Darüber hinaus bestehen keine Verbindungen zu detaillierteren EEA-Abstraktionsebenen nach Abb. 2.2, wodurch ebenenübergreifende Aspekte wie architekturbedingte Netzwerkkommunikation nicht berücksichtigt werden können. Eine Behandlung von Architekturvarianten wird nicht adressiert. Zudem weist die SysML-Modellierung die bereits in Unterabschnitt 2.1.4 dargelegten Nachteile einer universellen Modellierungssprache auf.

Generell lässt sich zusammenfassend feststellen, dass eine ebenenübergreifende, vertikale Interaktion zwischen den Verhaltensmodellen sowohl bei den SysML-Ansätzen aber auch bei den architekturzentrierten Ansätzen nicht adressiert wird. Auch die Kombination von zustandsorientiertem Verhalten mit weiteren Paradigmen und Berechnungsmodellen, wie z. B. Actor-orientierte oder kontinuierliche Modelle, zur detaillierten Spezifikation von funktionalem Verhalten mit wiederverwendbaren Komponenten wird nicht ausreichend adressiert. Zudem werden keine Lösungen präsentiert, wie das Verhalten von abstrakten, realisierungsunabhängigen Funktionen auf höherer Ebene, wie der Analyseebene der EAST-ADL

oder der logischen Architektur von PREEvision®, mit Informationen aus realisierungsnahen EEA-Abstraktionsebenen wie der Hardware- und Vernetzungsebene oder gar Elektrik verknüpft werden kann. Dies ist aber ein notwendiges Kriterium, um eine durchgängige und iterativ verfeinerbare Analyse bzw. Simulation einer E/E-Architektur beginnend in frühen Entwicklungsphasen durchführen zu können.

Eine Verhaltensmodellierung war in PREEvision® bis Version 8.5 nicht möglich und wurde deshalb zum Zeitpunkt des Verfassens von Abschnitt 3.2 nicht behandelt. Mit der Version 9.0 wurde die Modellierung von UML-konformen State Charts eingeführt, die in Kapitel 5 aufgegriffen werden und in das dortige Konzept mit einfließen.

3.2 PREEvision®

3.2.1 Überblick

Die EEA-ADL ist eine domänenspezifische Sprache (siehe Unterabschnitt 2.1.4) zur abstraktionsebenenübergreifenden Modellierung von E/E-Architekturen in Fahrzeugen [102]. Sie basiert auf einem MOF [125, 128] Metamodell und bildet im Entwurfs- und Analysewerkzeug PREEvision® den Kern als komplexe Datenstruktur.

PREEvision® ist eine eclipse-basierte Anwendung und daher vollständig in Java implementiert. Durch den Plugin Mechanismus von eclipse[13] und ein generisches Diagramm-Framework [41] ist es einfach erweiterbar, z. B. um benutzerdefinierte Metrikblöcke, Ansichten oder grafische Editoren. Aber auch generische Modellierungsmechanismen innerhalb des Datenmodells erlauben weitreichende benutzerspezifische Anpassungen und Erweiterungen, bspw. durch sog. *Custom Attributes* [170]. Darüber hinaus steht ein umfangreiches Metrik-Framework zur Bewertung und zum Vergleich von E/E-Architekturvarianten zur Verfügung [41] und erlaubt sowohl ein nach ISO 26262 [56] konformes Design und eine Analyse bzgl. funktionaler Sicherheit [1] als auch eine AUTOSAR [8] konforme Software-Architekturentwicklung. Eine Kollaborationsplattform sowie ein integriertes Änderungs- und Release-Management realisieren die standortübergreifende Zusammenarbeit mit unterschiedlichen Rollen [170].

Da die zugrunde liegende EEA-ADL das umfangreichste Metamodell zur durchgängigen, detaillierten Beschreibung auf allen Ebenen bietet (von Anforderungen

[13] Für eine detaillierte Beschreibung von eclipse, den Plugin Mechanismus und den Aufbau im Kontext von PREEvision® sei an dieser Stelle auf [102] und [41] verwiesen.

bis zur physikalischen Topologie), dient sie als zentrale Datenstruktur für die in Kapitel 5 entwickelte integrierte Simulation und dynamische Bewertung von E/E-Architekturen. PREEvision® stellt dabei den programmatischen Rahmen für die prototypische Umsetzung dar. Aus diesem Grund wird das Werkzeug im Folgenden ausführlicher dargestellt.

3.2.2 Abstraktionsebenen

Abb. 3.2 zeigt den Aufbau von PREEvision®, welcher, dem Aufbau nach Abb. 2.2 folgend, zunächst in mehrere Abstraktionsebenen aufgeteilt werden kann und eine spezifische Sicht auf die EEA mit zunehmendem Detailgrad darstellt. Jede Ebene stellt dabei ihre eigenen auf die Sicht zugeschnittene Diagramme zur Verfügung. Diese Ebenen werden in den folgenden Abschnitten beschrieben, anschließend werden ebenenübergreifende Mechanismen der EEA-ADL sowie weitere modellbasierte Entwicklungsunterstützungen beschrieben.

3.2.2.1 Produktziele

Auf der ersten Ebene werden die Produktziele beschrieben, welche sich in drei Kategorien einteilen lassen [170]:

1. *Anforderungen und Kundenfunktionen*: Dieser Bereich stellt eine strukturierte und systematische Anforderungsentwicklung und -management zur Spezifikation einer EEA zur Verfügung, um ein gemeinsames Verständnis zwischen Auftraggeber und -nehmer herzustellen. Eine Kundenfunktion ist dabei als erlebbare Funktion wie etwa eine automatisch geregelte Klimaanlage zu verstehen, wohingegen eine Anforderung ein spezifisches, technisches Merkmal ist, z. B. eine zulässige Betriebstemperatur eines Steuergeräts. Anforderungen werden textuell beschrieben und können über Pakete hierarchisiert sowie automatisch mit einer ID versehen werden und Querverweise untereinander aufweisen. Tabelleneditoren vereinfachen dabei die Bearbeitung. Kundenfunktionen können analog erstellt und gewartet werden, stellen aber im Hinblick auf Fahrzeugvarianten ein Feature-Modell und dazugehörige Diagramme zur Verfügung, mit denen die Kundenfunktionen über Attribute in Beziehung zueinander gesetzt werden können. Beispielsweise kann ein Feature einer Variante ein anderes ausschließen.
2. Auf der UML basierte *Use Case Diagramme* erlauben die Modellierung von Systemfunktionen, z. B. Fahrersitzeinstellung, aus Nutzersicht.

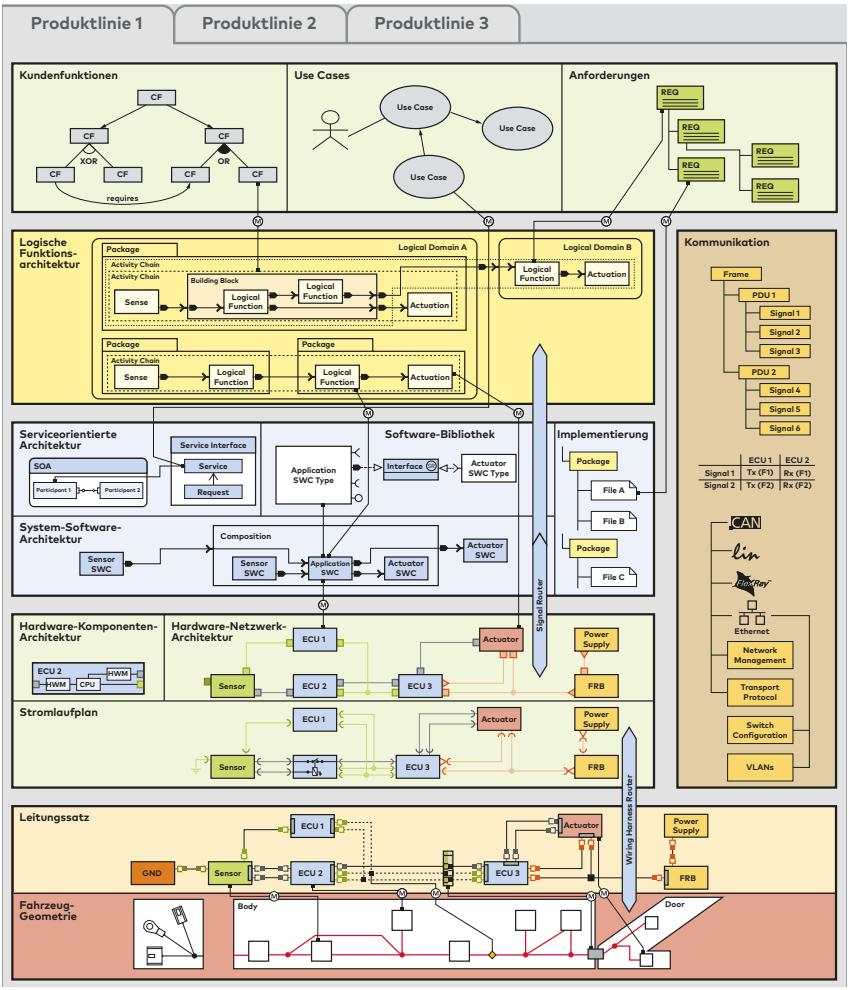


Abbildung 3.2: Abstraktionsebenen der in PREEvision® v8.5 umgesetzten EEA-ADL (Quelle: [170, S. 127]).

3. *Funktionalitätsnetzwerke* (nicht dargestellt in Abb. 3.2) dienen zur grafischen Modellierung von Systemfunktionen als eine kausale Kette aus sog. *Funktionsanforderer*, *Funktionalität* und *Funktionserfüller*.

3.2.2.2 Logische Funktionsarchitektur

Die logische Funktionsarchitektur oder schlicht *Logische Architektur (LA)* repräsentiert die logische Systemsicht auf das Netzwerk an verteilten Fahrzeugfunktionen und abstrahiert von deren späteren, konkreten Realisierung in Hardware oder Software.

Die LA wird durch Blockdiagramme modelliert und erlaubt durch atomare Funktionen sowie zusammengesetzte Blöcke (sog. *Building Blocks*, siehe Unterabschnitt 3.2.6) und Domänen zur Hierarchisierung eine System-Dekomposition zur Beherrschung der Komplexität. Domänen oder Subsysteme können keine weiteren Domänen enthalten und können nur als Ganzes wiederverwendet oder mit anderen Blöcken über ihre Schnittstellen verbunden werden. Funktionen können zudem in Paketen und Subpaketen organisiert werden wobei über *Wirkketten* (engl. *Activity Chains*) Funktionen, die eine bestimmte Kundenfunktion realisieren, ohne ihre Hierarchie dargestellt werden können.

Die Funktionen können mit Software- oder Hardwareartefakten verknüpft werden, wobei die Schnittstellenspezifikation die Grundlage zur Definition von auszutauschenden Signalen zwischen den Funktionen bzw. zwischen den physikalischen Steuergeräten bilden (siehe Unterabschnitt 3.2.3). Zusätzlich können sie mit Kundenfunktionen und/oder Anforderungen verknüpft werden, um nachvollziehen zu können, welche Funktion jene verknüpften Artefakte implementiert.

Ein wesentliches Merkmal der LA ist, dass alle Funktionen aus einer Bibliothek an *Funktionstypen* explizit typisiert instanziiert werden. Analog werden die Schnittstellen der Funktionen über *Port-Prototypen* spezifiziert, die für alle Instanzen einer Funktion gleich sind. Ports werden über *Assemblies* verbunden und können dabei durch die dem Port-Prototyp konkret zugewiesenen Schnittstelle unterschiedliche Kommunikationsarten aufweisen. Dazu zählen Daten- und Kontrollfluss sowie serviceorientierter Informationsaustausch. Ports eines Building Blocks sind Delegationsports zu den intern verknüpften atomaren Funktionen und benötigen keine weitere Schnittstellenzuweisung.

3.2.2.3 System-Softwarearchitektur

Die System-Softwarearchitektur beschreibt die involvierten Softwarekomponenten sowie deren Interaktion und erlaubt ein AUTOSAR konformes Softwaredesign. Die Spezifikation des Designs folgt analog zur LA mithilfe einer Funktionsbibliothek an *Softwaretypen* sowie deren *Port-Typen* und die dazugehörigen Schnittstellenzuweisungen. Sie unterscheiden sich lediglich durch ein feiner unterteiltes *Typ-Prototyp-Instanz* Konzept zur Trennung der Typisierung von Softwarekomponenten und Ports sowie deren Instanziierung und Zuweisung zu Hardwareartefakten [170, 41]. Zusätzlich zur LA gibt es einige software- bzw. AUTOSAR-spezifische Schnittstellentypen für Ports. Zur Dekomposition werden analog zur LA sog. *Kompositionsblöcke* (engl. *compositions*) eingesetzt. Die Softwarekomponenten können mit logischen Funktionen und den ausführenden Hardwareartefakten verknüpft werden.

Zusätzlich zu den Kommunikationsarten über die Schnittstellenspezifikation der Ports ermöglicht die Ebene auch die Modellierung von *serviceorientierten Softwarearchitekturen* mit dem Ziel, eine höhere Flexibilität von Diensten in einem verteilten Funktionsnetzwerk bereitzustellen, die Komplexität durch Abstraktion und intensive Wiederverwendung von Diensten sowie die Entwicklungskosten von Software zu reduzieren [170]. Die serviceorientierte Architektur stellt eigene Diagramme zur Modellierung zur Verfügung.

Die Subebene der *Implementierung* verknüpft die Softwarekomponenten mit realisierenden Artefakten. Dies können Quelltexte oder andere modellbasierte Quelldateien sein.

3.2.2.4 Hardwarearchitektur

Die Softwarearchitektur kann im Grunde in drei Subebenen mit zunehmender Granularität aufgeteilt werden, die jeweils eigene Ansichten und Diagrammtypen aufweisen:

1. die logische Ebene,
2. elektrische Ebene (Stromlaufplan) und
3. physikalische Ebene (Leitungssatz und Topologie).

Die *logische Ebene* beschreibt sämtliche Hardwarekomponenten wie ECUs, Sensoren und Aktuatoren sowie die Netzwerktopologie der involvierten Bussysteme, über die die Komponenten mit logischen Busverbindungen verknüpft sind. Zusätzlich können die Komponenten über konventionelle Verbindungen miteinander verbunden sein und Signale austauschen. Diese entsprechen ana-

logen Signalen wie Spannungspegeln oder Ströme. Schließlich stehen dedizierte Komponenten wie Sicherungsboxen (engl. *Fuse-Relay-Boxes*) sowie Generatoren, Batterien und Massepunkte (engl. *Ground Points*) und die dazugehörigen logischen Leistungsversorgungs- und Masseverbindungen zur Verfügung. Diese Komponenten und Verbindungen dienen zur Modellierung des Leistungsversorgungs- und Massekonzeptes. Die logische Sicht wird in einem Hardware-Vernetzungsdiagramm modelliert.

Der interne logische Aufbau der einzelnen Komponenten wird in der *Hardwarekomponenten-Architektur* modelliert. Hierzu stehen mehrere *interne* Komponenten wie Recheneinheiten (*Processing Units (PUs)*, *Central Processing Units (CPUs)*, *Digital Signal Processors (DSPs)* etc.) zur Ausführung von Software, Hardwaremodule zur analogen Umsetzung von Funktionen oder Speicherbausteine zur Verfügung. Die Verknüpfung der internen Komponenten geschieht mit *internen logischen* Verbindungen. Die dazugehörigen internen logischen Konnektoren können, im Gegensatz zu den externen, eine Richtung (in, out, in/out) aufweisen[14].

Durch die Tatsache, dass die Komponenten auf allen drei Subebenen nach außen hin identisch aufgebaut sind und lediglich die Verbindungen und Verbinder eine andere Bedeutung bekommen, können die elektrische Ebene und der Leitungssatz durch eine sog. *Electric Circuit Synthesis* automatisiert synthetisiert werden [170, 41].

Elektrische Ebene (Stromlaufplan) Die elektrische Subebene bzw. der Stromlaufplan verfeinert die logischen Verbindungen und Konnektoren durch ein oder mehrere dazugehörige elektrische Verbindungen, die die Komponenten über schematische Pins verknüpfen. So wird bspw. die logische Verbindung eines *Controller Area Network (CAN)* Busses durch je zwei elektrische Verbindungen und schematische Pins zwischen den verknüpften Komponenten verfeinert, welche die differentielle Signalübertragung des CAN-Bussystems über die CAN-High und CAN-Low Pegel realisieren. Eine elektrische Verbindung verbindet somit Komponenten mit demselben elektrischen Potential [41].

Leitungssatz Im Leitungssatz wird der Kabelbaum der E/E-Architektur modelliert, indem die elektrischen Verbindungen und schematischen Pins durch die physikalische Realisierung verfeinert werden. Elektrische Verbindungen sind nun die logische Repräsentation von ein oder mehreren physikalischen Leitungen oder Kabeln, die über Splices oder Trennstellen miteinander verbunden sind. Die Leitungen werden durch physikalische Leitungs-Pins mit komponentenseitigen

[14] mit Ausnahme der Leistungsversorgungsverbinder, welche gerichtet sind.

Pins an die Komponenten angeschlossen, welche typischerweise Steckern zugeordnet sind. Der Import des standardisierten Austauschformats *Kabelbaumliste (KBL)* wird darüber hinaus unterstützt.

Topologie Die Topologie beschreibt und visualisiert die Fahrzeuggeometrie mithilfe eines 2,5-dimensionalen Modells. Wesentlicher Bestandteil der Topologie ist die Aufteilung in Baubereiche, wie z. B. Motorraum oder Tür, und die darin befindlichen Einbauorte und geometrischen Ausmaße, an denen die Hardwarekomponenten verbaut werden können. Die Einbauorte werden über sog. Topologiesegmente miteinander verbunden, in denen die Leitungen verlegt bzw. geroutet werden können. Das Routing der Leitungen wird in PREEvision® durch einen *Wiring Harness Router* (dt. Leitungssatz-Router) [170, Kap. 10.4.9], der die kürzesten Verlegewege ermittelt, automatisiert unterstützt. Je nach Einbauort von Komponenten müssen die Leitungen durch topologische Trennstellen[15] oder aufgrund der Vernetzung durch Ausbindungen (engl. Branch-Offs) aufgetrennt werden. Letztere werden im Leitungssatz durch abstrakte Trennstellen bzw. Splices abstrahiert und über Mappings (siehe Unterabschnitt 3.2.3.1) miteinander in Beziehung gesetzt.

3.2.3 Ebenenübergreifende Mechanismen

In diesem Abschnitt werden nachfolgend ausgewählte ebenenübergreifende Zusammenhänge beschrieben, insb. der Kommunikationsentwurf sowie die Verknüpfung von Artefakten auf unterschiedlichen Abstraktionsebenen, die sog. *Mappings*.

3.2.3.1 Mappings

Mappings sind dedizierte Metamodell-Klassen, die die Artefakte auf den unterschiedlichen Ebenen und teilweise auch innerhalb einer Ebene miteinander verknüpfen können und sind in Abb. 3.2 mit einem (M) visualisiert. Generell erlauben die Mappings eine durchgängige und eindeutige Nachvollziehbarkeit über alle Ebenen hinweg. Aber nicht alle Artefakte können auf beliebige Artefakte abgebildet werden. Diese Einschränkungen werden durch dedizierte Mapping Metamodell-Klassen und deren Assoziation zu den spezifischen Quell- und Zielartefakten gewährleistet, was mit schlichten Relationen nicht möglich wäre. Die den Mappings zugehörigen Metamodelle sind in [41] zu finden.

[15] Siehe die Trennstelle zwischen *Door* und *Body* in Abb. 3.2.

Nachfolgend wird auf Basis der vorgestellten Mapping-Metamodelle in [41] ein Überblick über die wichtigsten Mappings zwischen den einzelnen Ebenen gegeben.

LA ⇔ Software-/Hardwarearchitektur Mappings von logischen Funktionen der LA auf Softwarekomponenten erlauben eine weitere Verfeinerung der Funktion in Software, die wiederum auf ausführende Hardwarekomponenten abgebildet werden können. Diese Art von Mappings werden dem Typ `LogSWMapping` zugeordnet. Andererseits kann eine logische Funktion direkt, ohne Umweg über die Softwareebene, mit der Klasse `LogSWNetMapping` auf Recheneinheiten der abstrakten Klasse `ProcessUnit` abgebildet werden. Die Abbildung auf eine abstrakte Recheneinheit kann bspw. dann der Fall sein, wenn in der frühen Konzeptphase die Implementierung auf einem konkreten Baustein wie Mikrocontroller, DSP etc. noch nicht festgelegt ist. Oder aber es handelt sich um eine Legacy-Funktion, die als Ganzes abgebildet werden muss. Wenn eine Funktion als analoge Komponente umgesetzt werden soll, wird sie über ein `LogHWMapping` mit einem Hardwaremodul verknüpft. Ein typisches Beispiel hierfür ist eine Ausgangsverstärkerstufe. Zusätzlich zu den Block-Mappings existieren Port- und Port-Prototyp-Mappings. Diese Art von Mappings dient hauptsächlich zur Spezifikation von Hardware/Software-Schnittstellen im Rahmen von Analysen bzgl. funktionaler Sicherheit nach ISO26262 [170].

System-Softwarearchitektur ⇔ Hardwarearchitektur Analog zur LA können Softwarekomponenten über die Klasse `SWComponentMapping` auf Hardwarekomponenten, wie Recheneinheiten und deren Spezialisierungen, oder Hardwaremodule abgebildet werden. Auch Softwareports (Instanzen) lassen sich über die Klasse `InterfaceSWHWMapping` mit logischen Hardware-Konnektoren sowie schematischen Pins verknüpfen.

Hardwarearchitektur ⇔ Topologie Zur Bestimmung des Einbauorts von Hardwarekomponenten wie ECUs, Sensoren, Aktuatoren etc., existieren Mappings vom Typ `PlacementComponentMapping`. Des Weiteren werden Splices über die Klasse `PlacementSpliceMapping` auf Ausbindungen oder auch Einbauorte abgebildet, abstrakte Trennstellen aus dem Leitungssatz werden über `PlacementInlineConnectorMapping`s mit topologischen Trennstellen verknüpft. Nicht zuletzt müssen Leitungen und Kabel entsprechenden Topologiesegmenten über `PathMapping`s zugewiesen werden, damit nach dem Routing die entsprechenden Leitungs- bzw. Kabellängen bestimmt werden können. Leitungen, die an Splices angeschlossen werden, weisen zudem die Besonderheit auf, dass sie je nach Realisierung der Splices nur einseitig angebunden werden können und somit

eine zusätzliche Leitungslänge aufweisen (siehe [41, S. 93, Abb. 3.34] für die Umsetzung im Metamodell). Die `PathMappings` werden während dem Routing automatisch angelegt. Während die Mappings vom Typ `PlacementComponentMapping` und `PlacementSpliceMapping` notwendige Voraussetzungen für die Ausführung des Leitungssatz-Routers sind, sind die Mappings von abstrakten Trennstellen optional, da diese vom Router erzeugt werden, falls in der Topologie eine geroutete Leitung eine topologische Trennstelle kreuzt. Bereits existierende abstrakte Trennstellen im Leitungssatz müssen jedoch ein Mapping zur Topologie aufweisen [175, Kap. 10.4.9, S. 1618 ff.].

3.2.3.2 Kommunikation

Die Kommunikation erstreckt sich über die Ebenen LA, Softwarearchitektur und Hardware-Vernetzungsebene und spezifiziert zunächst die Signale, die zwischen den Funktionen und letztlich zwischen den zur ausführenden Hardwarekomponenten ausgetauscht werden.

Mehrere Signale mit ähnlicher Semantik werden typischerweise zunächst in technologieunabhängige *Protocol Data Units (PDUs)* zusammengefasst, die wiederum einem Frame zugeordnet werden (siehe die Signal-PDU-Frame Baumstruktur in Abb. 3.2). Ein Frame folgt einem spezifischen Format, abhängig vom realisierenden Busprotokoll. [170, Kap. 9]

Da ein Signal bis zum Ziel-Steuergerät über *Gateways* mehrere Bussysteme unterschiedlicher Technologie oder auch konventionelle Verbindungen durchlaufen kann, wird es entsprechend ihrem Träger anders repräsentiert. Die zu übertragende Information an sich bleibt dabei immer dieselbe, sie wird nur anders interpretiert. Daher existieren für ein Signal ein oder mehrere *Signaltransmissionen*, die eine Instanz eines Signals auf den unterschiedlichen Kommunikationswegen darstellen. Analog zur Signal-PDU-Frame Struktur werden Signaltransmissionen in dazugehörige *PDU*- und *Frametransmissionen* zusammengefasst. [170, Kap. 9]

Unterstützt werden die wichtigsten Netzwerktechnologien wie CAN, *CAN Flexible Data-Rate (CAN FD)*, *Local Interconnect Network (LIN)*, FlexRay, *Media Oriented Systems Transport (MOST)* und Ethernet für serviceorientierte Architekturen. Ein AUTOSAR-konformes Design und weitere busspezifische Austauschformate wie *Data Base CAN (DBC)*, *LIN Description File (LDF)* oder *Fieldbus Exchange Format (FIBEX)* zum Import/Export werden ebenso berücksichtigt wie das Netzwerkmanagement und die Modellierung von partieller Kommunikation via Subnetze (sog. *Cluster*) zur Ressourcenschonung. [170, Kap. 9]

Nicht zuletzt wird das Routing der Signale und deren Datenelemente über das verteilte Steuergerätenetzwerk durch Automatismen wie dem *Signalrouter* [170,

Kap. 9.3.2] unterstützt. Dieser berechnet basierend auf einer konfigurierbaren Kostenfunktion die optimale Route für alle auszutauschenden Informationen vom Sender zum Empfänger. Zusätzlich benötigte Artefakte wie Gateways, Routing-Einträge, Signale, Mappings etc. werden bei Bedarf automatisiert generiert.

3.2.4 Variantenmanagement

Zur Beherrschung der Komplexität der Variantenvielfalt von Architekturalternativen und verschiedenen Ausstattungsmerkmalen wird ein Produktlinienansatz verfolgt. Mithilfe von Typ-Bibliotheken einzelner Artefakte, Artefakt-Mengen (engl. *assets*) oder gar ganzer Plattformen können mehrere Produktlinien unter Ausnutzung von drei typischen Systems Engineering Prinzipien aufgebaut werden: Dekomposition, Wiederverwendung und Instanziierung [153].

Als Resultat entsteht pro Produktlinie ein sog. 150 % Modell. Ein 150 % Modell ist eine Obermenge aller zur Verfügung stehenden Features, Funktionen, Architekturen etc., sprich Produkte innerhalb einer Baureihe. Über das eigentliche Variantenmanagement werden basierend auf dem 150 % Modell Teilmengen bzw. Derivate abgeleitet, z. B. ein Cabriolet (sog. 120 % Modelle). Aus diesem werden wiederum explizite Produkte mit spezifischen Ausstattungsmerkmalen entworfen, z. B. ein Cabriolet mit Diesel-Motor und Xenon-Scheinwerfern (sog. 100 % Modelle). Varianten können entweder auf Basis eines Feature-Modells mit Relationen und Constraints, eines Variantenmanagements nach AUTOSAR oder über Ausstattungs- und Konzeptschablonen (engl. templates) sowie dazugehörigen Alternativen modelliert werden (vgl. [175, Kap. 12] und Abb. 3.3).

3.2.5 Metriken

In Ergänzung zum EEA-Datenmodell existiert ein umfangreiches, integriertes Metrik-Framework [41], welches durch benutzerdefinierte Metrikdiagramme Berechnungen zur Analyse und Evaluation des Datenmodells und Vergleich von Architekturalternativen bzgl. nicht-funktionaler Kriterien erlaubt. Dazu zählen Kosten, Gewicht, Robustheit, Buslast oder statische Stromanalysen in bestimmten Fahrzeugzuständen [170, S. 1326 ff.].

Metriken werden via Blockdiagrammen mit Datenfluss-Semantik modelliert und können mit vordefinierten Blöcken, Java-Code als auch mit benutzerspezifischen Berechnungsblöcken (sog. *Customized Metric Blocks*) mittels zusätzlichen eclipse-Plugins Berechnungen ausführen. Sie bieten dabei vollen Zugriff auf das komplette Datenmodell und können Operationen direkt auf das Modell anwenden, bspw. zu Optimierungszwecken. Modellartefakte werden über Modellkontext-Blöcke

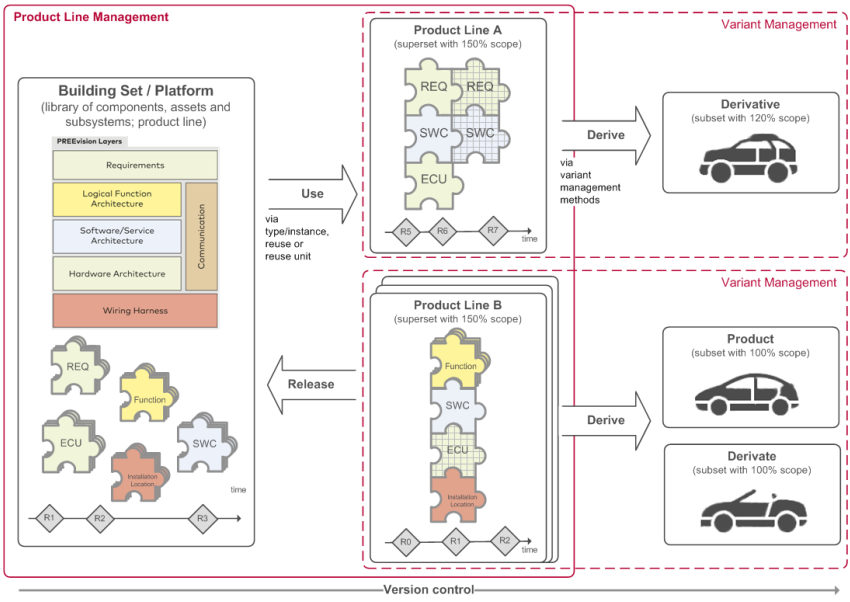


Abbildung 3.3: Produktlinienansatz von PREEvision® (Quelle: [175, S. 1751]).

oder über integrierte Modellabfragen mittels des zusätzlich integrierten Regelwerks als Eingang für Berechnungen zur Verfügung gestellt [41, Kap. 6.2]. Das Regelwerk wird mithilfe von Modell-zu-Modell-Transformationen realisiert, die in [141] entwickelt wurden und gleichzeitig als Grundlage für Modellkonsistenz-Checks dienen.

3.2.6 Metamodelle

Im Folgenden werden die für Kapitel 5 relevanten Metamodelle ausgewählter Ebenen vorgestellt. Einige der gezeigten Metamodelle basieren auf den in [41] vorgestellten, sind jedoch auf Grundlage der Version 8.5 der EEA-ADL aktualisiert, angepasst oder erweitert worden.

3.2.6.1 Logische Architektur

LA Instanz Wie in Unterabschnitt 3.2.2.2 beschrieben, werden sämtliche logischen Funktionen und ihre Ports typisiert instanziiert. Zur Modellierung einer LA-Instanz werden im Wesentlichen die in Abb. 3.4 dargestellten Artefakte verwendet. Atomare Funktionen erben alle von der abstrakten Klasse `AbstractLogicalFunction`. Zu den gängigsten zählen `SensorLogicalFunction`, `LogicalFunction` und `ActuatorLogicalFunction` zur Modellierung von datenerfassenden (Sensoren), datenverarbeitenden bzw. auf Daten reagierende Funktionen (Aktuatoren).

Zusammengesetzte Funktionen und Domänen werden durch die beiden Klassen `LogicalBuildingBlock` und `LogicalDomain` realisiert und sind von der abstrakten Klasse `AbstractLogicalBlockOwner` abgeleitet. Über die Komposition zu `AbstractLogicalBlock` können sie logische Blöcke aufnehmen, wobei ein `LogicalBuildingBlock` zusätzlich von `AbstractLogicalBlock` erbt und somit entsprechend dem Kompositum-Entwurfsmuster hierarchisiert werden kann. Jeder `AbstractLogicalBlock` verfügt über die Komposition zu `AbstractLogicalPort` über beliebig viele Ports, die über logische Assemblies vom Typ `LogicalAssemblyConnector` miteinander verbunden werden und einen eigenen Container `AbstractLogicalAssemblyOwner` besitzen.

Funktionen bzw. Blöcke, die von `AbstractMappableLogicalArtefact` erben, können auf Software- oder Hardwarekomponenten abgebildet werden, dazu zählen auch Building Blocks und Domänen, wonach ganze Funktionsverbünde mit einer ausführenden Einheit verknüpft werden können. Service- und Umgebungsfunktionen (`ServiceFunction` und `PlantFunction`) realisieren bspw. Diagnosefunktionalitäten bzw. abstrahieren die Umwelt und können daher nicht auf Software/Hardware abgebildet werden.

Typisierung Die Assoziation eines abstrakten Blocks zu `LogicalBlockType` stellt die Schnittstelle zur Typisierung dar. Das entsprechende Metamodell ist in Abb. 3.5 gezeigt. Demnach hat jede logische Funktion bzw. jeder Block eine dazugehörige, gleichnamige Typklasse mit der Endung `Type`, die alle von `LogicalBlockType` abgeleitet sind. Logische Typen werden in Paketen `LogicalTypesPackage` strukturiert. Analog besitzen auch die Ports einen `LogicalPortPrototype`, der Daten zur Verfügung stellen oder Empfänger sein kann. Durch die Spezialisierungen `LogicalPPortPrototype` und `LogicalRPortPrototype` werden die Prototypen auf Sender bzw. Empfänger beschränkt. Demnach können Sensortypen nur Sender-Port-Prototypen und Aktuatortypen nur Empfänger-Port-Prototypen besitzen. Alle anderen können über die zusätzliche Vererbung von `AbstractLogicalFunctionBlockType` beide Arten

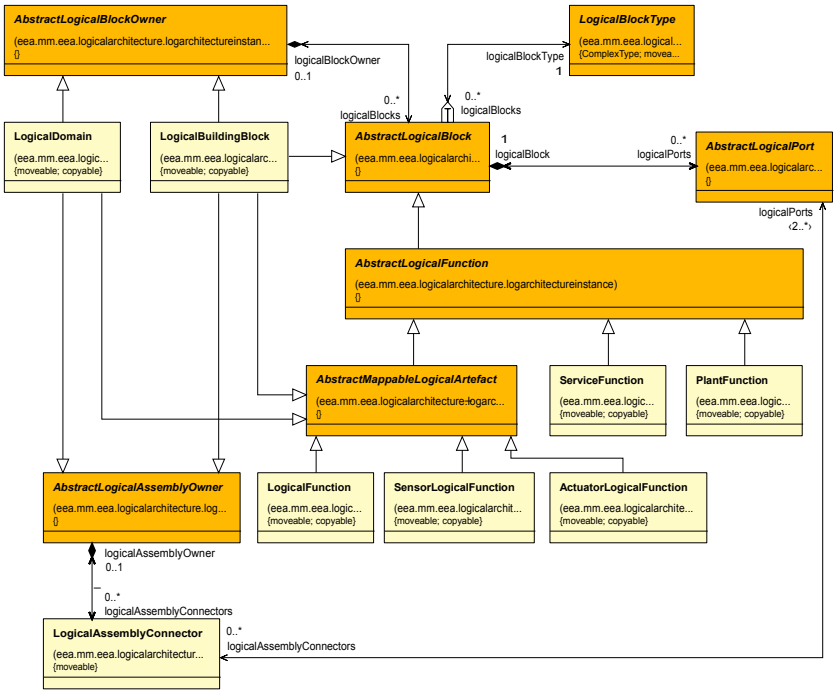


Abbildung 3.4: Metamodell einer LA-Instanz (Quelle: aktualisierte Darstellung nach [41, S. 63]).

umsetzen. Zu bemerken ist, dass durch die Beschränkung der Ports in der Typisierung somit auch die Instanzen auf Sender- oder Empfänger-Ports beschränkt sind. Für die Synchronisierung generell zwischen Typen und Instanzen sorgen Automatismen innerhalb von PREEvision® [41].

Entsprechend der Kommunikationsarten zwischen Funktionen (siehe Unterabschnitt 3.2.2.2), existieren weitere konkrete Port-Prototypen wie `LogicalSenderReceiverPortPrototype`, `LogicalClientServerPortPrototype` und `LogicalSlaveControllerPortPrototype` sowie deren Spezialisierungen zur Spezifikation von Sender und Empfänger (nicht gezeigt in Abb. 3.5).

Schnittstellen Die beschriebenen Port-Prototypen definieren zunächst lediglich die Art der Kommunikation. Welche Informationen und zusätzlichen Attribu-

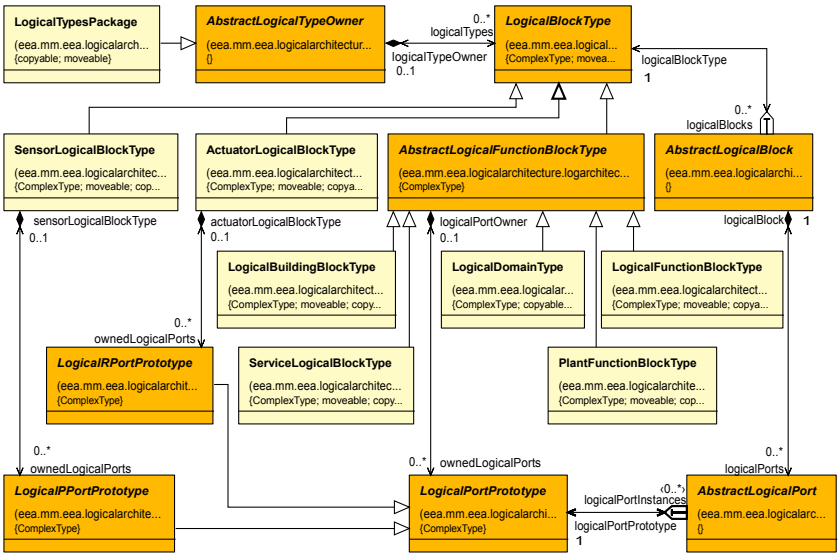


Abbildung 3.5: Metamodell der LA-Typisierung (Quelle: eigene Darstellung nach [41, S. 62]).

te tatsächlich zwischen den Funktionen ausgetauscht werden, erfolgt über die Zuweisung dazugehöriger Schnittstellen. Die Realisierung im Metamodell ist in Abb. 3.6 gezeigt.

Jeder logische Port-Prototyp kann über die Komposition seiner Superklasse zu `InterfaceAssignment` einer dem Prototyp entsprechenden, konkreten Schnittstelle zugewiesen werden, die alle von der abstrakten Klasse `Interface` erben. Unterschiedliche Informationseinheiten werden je nach Schnittstelle zwischen Funktionen übertragen. Dies wird über die Komposition der Schnittstellen zu konkreten Informationseinheiten bewerkstelligt, die von `InformationUnit` erben[16].

Die `SenderReceiverInterface` Schnittstelle erbt zusätzlich von `DataInterface` und realisiert daher eine gewöhnliche Datenübertragung zwischen Sender und Empfänger via Datenelementen (`DataElements`), die über die Spezialisierung von `DataTypedInformationUnit` einen Datentyp aufweisen können. Im Gegensatz

[16] Mit Ausnahme der Slave-Controller Schnittstelle, die eine Komposition zu `SCFunction` besitzt und MOST-Bus spezifische Informationsträger definieren [41].

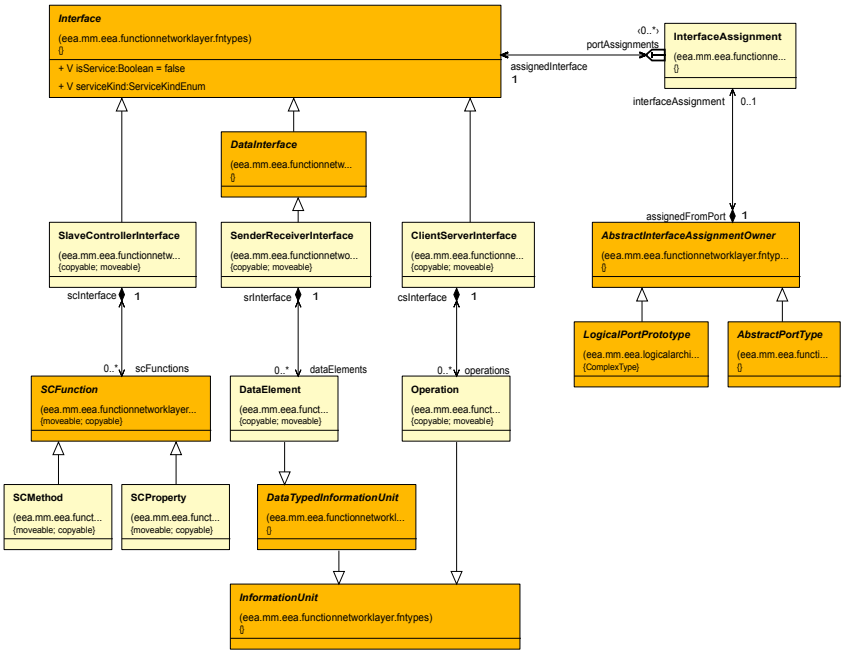


Abbildung 3.6: Metamodell der Schnittstellenzuweisung von Port-Typen (Quelle: aktualisierte Darstellung nach [41, S. 70]).

dazu erfolgt die Kommunikation von `ClientServerInterface` Schnittstellen über typlose Operationen.

Software Schnittstellen Das Konzept der Schnittstellenzuweisung zu typisierten Ports gilt analog für die System-Softwarearchitektur. Dies ist durch die Klasse `AbstractPortType` in Abb. 3.6 angedeutet. Der Unterschied besteht lediglich im Typ-Prototyp-Instanz Konzept der Softwarearchitektur (siehe Unterabschnitt 3.2.2.3), wonach analog zur LA sowohl ein Sender-/Empfänger-Typ existiert, aber auch software- bzw. AUTOSAR-spezifische Schnittstellen wie ein Trigger-Interface, die den kompatiblen *Port-Typen* der Softwareebene zugewiesen werden können.

3.2.6.2 Hardwarearchitektur

Das Metamodell der logischen Ebene der Hardwarearchitektur ist in Abb. 3.7 illustriert und zeigt sowohl die wichtigsten E/E-Hardwarekomponenten als auch deren internen Komponenten. Zentrale Klasse ist dabei die `DetailedElectricElectronic`, von der alle konkreten Artefakte wie ECUs, Sensoren, Aktoren und elektrische Komponenten wie Sicherungsboxen (`FuseRelayBox`) sowie Batterien und Generatoren (`EnergySources`) indirekt erben. Die Komposition zu `LogicalConnector` verknüpft die Komponenten über beliebig viele externe Anbinder und ihren dazugehörigen logischen Verbindungen untereinander (siehe Unterabschnitt 3.2.2.4). Bussysteme werden zusätzlich über die Typisierung mit speziellen Bustypen wie `CANType` oder `LINType` konkretisiert[17]. Über das Kompositum-Entwurfsmuster von `ComponentsComposite` und `ElectronicComposite` werden zusammengesetzte Elektrik/Elektronik-Konstrukte realisiert, die mehrere Sensoren, ECUs o. Ä. beinhalten können.

Alle Instanzen von `DetailedElectricElectronic` können interne Komponenten aufnehmen, welche im Wesentlichen aus ausführenden Einheiten (`ProcessUnits`), analogen Hardware-Modulen (`HWModules`) oder Speichereinheiten (`MemoryUnits`) bestehen. Realisiert wird dies im Metamodell über die abstrakte Oberklasse `AbstractInternalComponentArtefactOwner` und dessen Komposition zu `InternalComponentArtefact`. Dieser interne Aufbau entspricht dem Hardwarekomponenten-Diagramm in Abb. 3.2.

Da `AbstractInternalComponentArtefactOwner` und einige interne Komponenten zusätzlich von `HWDeviceOwner` erben, können diese Artefakte spezifische elektrische Bauteile wie Sicherungen, Widerstände, Kondensatoren etc. aufnehmen (`HWDevices`). Allerdings werden diese Artefakte tatsächlich der elektrischen Ebene zugeordnet und werden daher nur in Stromlaufplan-Diagrammen angezeigt (siehe Abb. 3.2).

Die Verknüpfung von externen, logischen Konnektoren zu internen Komponenten wird über *interne* logische Konnektoren und Verbindungen realisiert. Dies ist in Abb. 3.8 im oberen Teil des Metamodells dargestellt (vgl. `InternalLogicalConnector` und `InternalLogicalConnection`).

Elektrische Ebene Wie die externen logischen Verknüpfungen zwischen E/E-Komponenten, werden auch die internen zu elektrischen bzw. *internen schematischen* Verbindungen verfeinert. Dies ist in Abb. 3.8 durch die Assoziation zwischen `InternalLogicalConnection` und `InternalSchematicHWConnection` gezeigt.

[17] Das dazugehörige Metamodell zu den logischen Konnektoren und Verbindungen ist in [41, S. 78] zu finden.

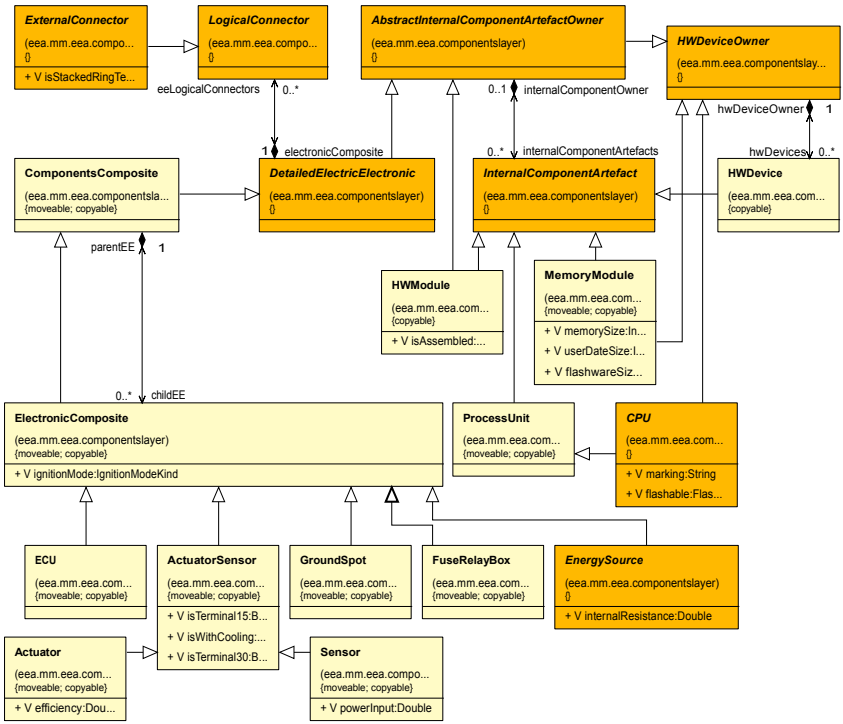


Abbildung 3.7: Metamodell der Hardwarekomponenten (Quelle: eigene Darstellung nach [41, S. 76]).

Über die abstrakte Klasse `AbstractInternalSchematicHWConnector` werden die internen elektrischen Verbindungen an unterschiedliche, konkrete Konnektoren angeschlossen. Dies können schematische Pins sein, gerichtete Konnektoren von elektrischen Bauteilen[18], bspw. ein Relais, oder ungerichtete Anschlüsse[19]. Letztere verfeinern die logischen Konnektoren von internen Komponenten, z. B. von CPUs.

Die Klasse `HasCurrentDescription` stattet Hardwarekomponenten und sämtliche schematischen Konnektoren mit Attributen zur Spezifikation eines Stromverbrauchers mit einer bestimmten Stromaufnahme aus. Dies ist nützlich, um eine statische Stromanalyse von E/E-Architekturen in unterschiedlich definier-

[18] Abstrahiert durch die Klasse `InternalSchematicConnector`.
[19] `UndirectedInternalSchematicConnector`

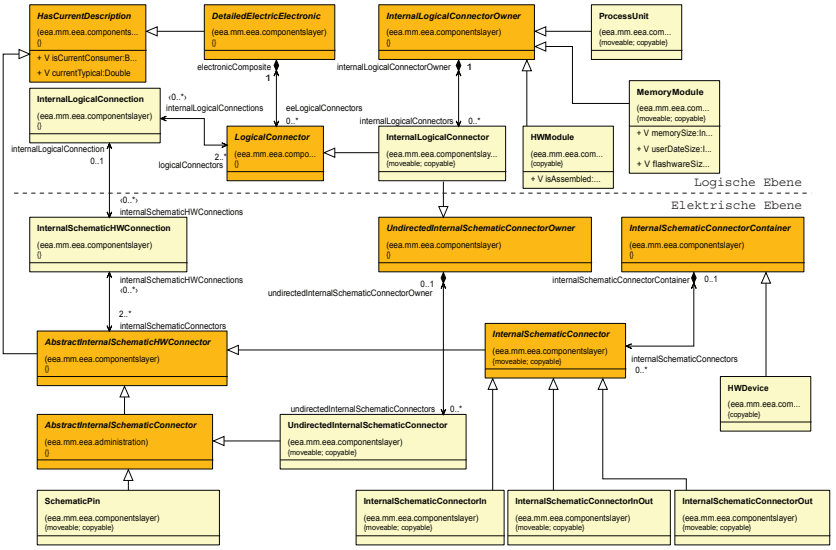


Abbildung 3.8: Metamodell zur Verknüpfung von logischen und elektrischen Hardwarekomponenten über verschiedene Verbindungssysteme (Quelle: erweiterte Darstellung nach [41, S. 78]).

ten Fahrzeugzuständen wie “Zündung an/aus” durchführen zu können [170, S. 1326 ff.].

In Abb. 3.9 ist im rechten oberen Teil des Metamodells schließlich die Verfeinerung von externen logischen Konnektoren zu schematischen Pins und Verbindungen zu sehen (siehe Unterabschnitt 3.2.2.4).

Leitungssatz Der untere Teil von Abb. 3.9 zeigt, wie in Unterabschnitt 3.2.2.4 beschrieben, die Verfeinerung der elektrischen Ebene zu Leitungssatz-Artefakten. Demnach abstrahiert eine schematische Verbindung mehrere physikalische Leitungen über die Assoziation von `SchematicConnection` und `AbstractWire`. Über Spezialisierungen kann eine physikalische Verbindung entweder eine einzelne Leitung (`SingleWire`) oder ein Kabel (`Cable`) sein. Ein Kabel kann dabei über das Kompositum-Entwurfsmuster von `Cable` und `AbstractCableChild` hierarchisch beliebig viele Adern enthalten (`Conductor`) sowie genau eine Schirmung (`Shielding`).

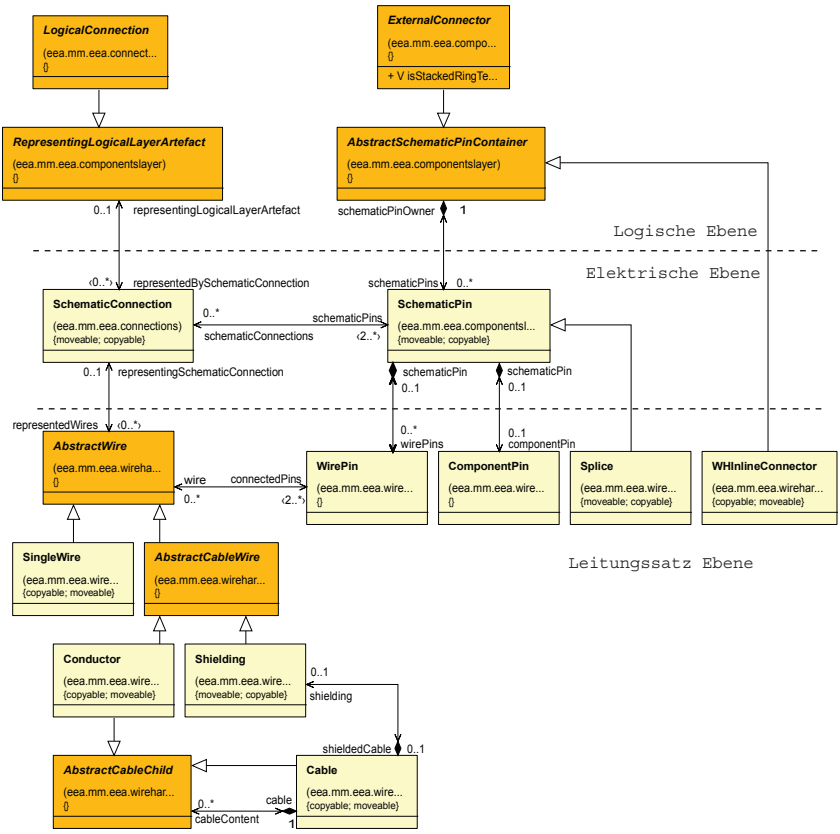


Abbildung 3.9: Metamodell der schematischen Verbindungen und des Leitungssatzes (Quelle: erweiterte Darstellung nach [41, S. 80]).

Die Leitungen verbinden dabei über leitungs- und komponentenseitige Pins (`WirePin` und `ComponentPin`) die Hardwarekomponenten, wobei ein oder mehrere Splices oder Trennstellen dazwischen liegen können. Pin-Paare werden je durch einen dazugehörigen schematischen Pin abstrahiert.

Darüber hinaus können generelle Attribute wie Kosten pro Meter oder Farbcodierungen und elektrische Eigenschaften wie Pin-Übergangswiderstand oder spezifischer Leitungswiderstand spezifiziert werden. Damit diese nicht für jede

Leitung bzw. jeden Pin definiert werden müssen, sind die wesentlichen Artefakte typisiert, die die Eigenschaften für alle Instanzen vorgeben.

Ein Ausschnitt des Metamodells der wichtigsten Typen des Leitungssatzes ist in Abb. 3.10 gezeigt. Pins, Leitungen und Kabel sind zunächst einer entsprechenden Familie untergeordnet und erben alle von `AbstractTypeFamily`. Die konkreten Familien verfügen über die Assoziation zu den speziellen Typen über eine Menge an Pin-, Leitungs- und Kabeltypen. Der spezifische Leitungswiderstand ist ein Attribut von `WireType`, während die Pins ihre Übergangswiderstände über die Klasse `HasResistanceValues` vererbt bekommen. Dabei wird zwischen `oldResistance` und `newResistance` unterschieden, um Alterungsprozesse zu berücksichtigen.

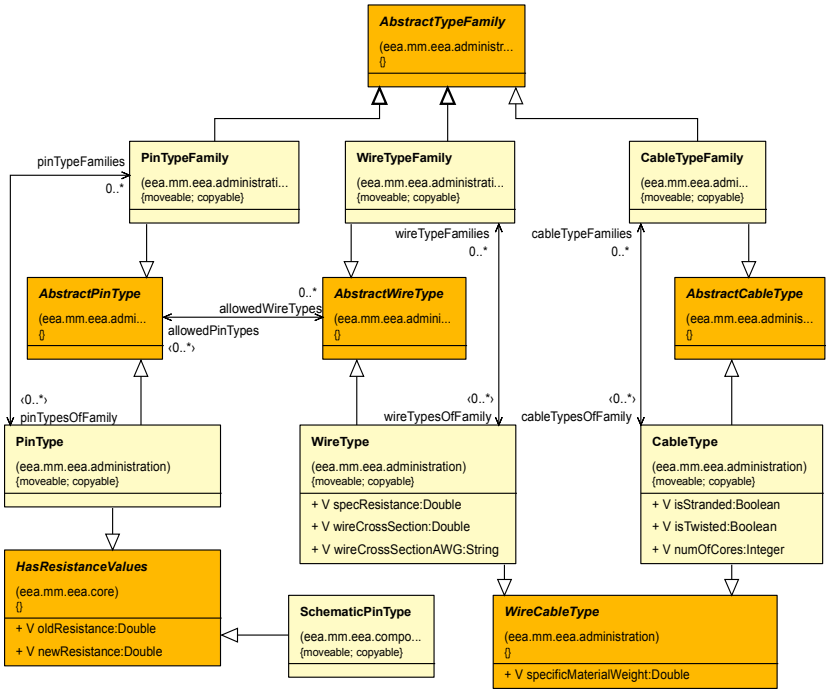


Abbildung 3.10: Metamodell-Ausschnitt der wesentlichen Leitungssatz-Typen (Quelle: eigene Darstellung nach [171]).

3.3 Simulation und Test von E/E-Architekturen

3.3.1 Überblick

Wie in Unterabschnitt 2.1.1.2 eingeführt, hat sich in der Automobilindustrie das V-Modell als Prozess zur Entwicklung und zum Test von eingebetteten elektronischen Systemen etabliert. In den einzelnen Entwicklungsschritten haben sich sowohl im absteigenden als auch im aufsteigenden Ast mehrere Test- und Simulationsmethoden zur Verifikation und Validierung durchgesetzt. Eine Übersicht über die verschiedenen Testmethoden entlang des V-Modells ist in Abb. 3.11 zu sehen.

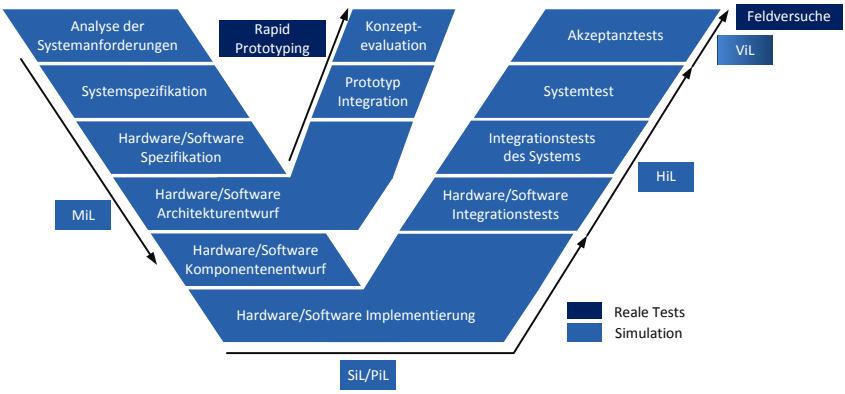


Abbildung 3.11: Das V-Modell und seine Test- bzw. Simulationsmethoden (Quelle: eigene Darstellung nach [188, S. 130] und [43]).

Die Verifikationsmethoden können zunächst in reale und virtuelle Tests bzw. Simulation unterteilt werden. Reale Tests finden generell auf dem aufsteigenden Ast der Integration statt, wenn bereits fertige Komponenten wie eine ECU, ECU-Verbünde oder Probefahrzeuge zur Verfügung stehen. Fahrzeugerprobungen finden in den letzten Entwicklungsstufen statt und sind weiterhin unverzichtbar zur Validierung und Abnahme eines Fahrzeugs (vgl. Abschnitt 1.1). Allerdings kommt diese Methode bei der immer weiter steigenden Anzahl an Szenarien, getrieben durch hochgradig vernetzte Fahrzeuge und autonomes Fahren, bzgl. Skalierbarkeit und Reproduzierbarkeit schnell an seine Grenzen [188, Kap. 8].

Als eine alternative Methode für reale Tests in frühen Entwicklungsstadien eignet sich *Rapid Prototyping*. Softwarefunktionen werden demnach in unterschiedlichem

Detailgrad, z. B. unabhängig von der Softwarearchitektur des Zielsteuergeräts, in ein Experimentiersteuergerät mit deutlich höherer Rechenkapazität als des Zielsteuergeräts mithilfe einer Bypass-Schnittstelle in das Fahrzeug integriert, um so frühzeitige Aussagen bzgl. des Funktionsverhaltens treffen zu können [154, Kap. 5].

Um kostspielige Fehlerkorrekturen bzgl. einer nicht erfüllten Spezifikation in späten Entwicklungsphasen der Integration zu vermeiden, zu denen in der Automobilindustrie bis zu „drei Jahre Entwicklungszeit vergehen und -kosten in Millionenhöhe“ [188, S. 129] entstehen können, haben sich virtuelle Test- bzw. Simulationsmethoden in den früheren Stadien etabliert. Diese werden als X-in-the-Loop Methoden bezeichnet [188, Kap. 8] [155], nach denen die eingebettete Software in den verschiedenen Entwicklungsstufen mit zunehmender Realitätsnähe gegen ein Systemmodell und dessen Umgebung getestet wird.

Bei *Model-in-the-Loop (MiL)* Simulationen werden ein oder mehrere Funktionen oder Algorithmen meist abstrakt mit modellbasierten Entwurfswerkzeugen wie MATLAB/Simulink erstellt und zur Validierung mit Modellen der Umgebung, des Fahrzeugs und Sensorik/Aktuatorik verknüpft, die separat und modular zur Verfügung stehen müssen. Die Modelle werden bspw. auf einem PC ausgeführt und sind unabhängig von der Zielhardware [188, Kap. 8] [43, 155]. Existieren in den modellbasierten Werkzeugen Code-Generatoren für die Modelle, kann der Code für Rapid-Prototyping-Zwecke verwendet werden [154, Kap. 5].

Software-in-the-Loop (SiL) Tests werden verwendet, um realisierte Softwarekomponenten - die auch generiert sein können - in einer simulierten Umgebung auszuführen, die der des Zielsystems bzgl. Performanzanforderungen recht nahe kommt, aber noch unabhängig von der zugrunde liegenden Zielhardware ist [188, Kap. 8] [154, Kap. 5]. *Processor-in-the-Loop (PiL)* ist ähnlich zu SiL mit dem Unterschied, dass die Software für den Zielprozessor kompiliert und auf diesem oder einem entsprechenden Emulator ausgeführt wird [155, 16].

Im aufsteigenden Ast werden die Softwarekomponenten sukzessive in ihre ECUs integriert, welche wiederum in das Steuergerätenetzwerk des Fahrzeugs eingebettet werden. Mit *Hardware-in-the-Loop (HiL)* Simulationen werden die Softwarefunktionen auf der finalen ECU in einer virtuellen Umgebung getestet. Die ECU enthält dabei sowohl sämtliche Softwaretreiber und das Betriebssystem des Zielprozessors als auch sämtliche zusätzliche Hardware, Peripherie und Schnittstellen nach außen [155]. In einem verteilten Steuergerätenetzwerk erfolgt zunächst der HiL-Test auf Komponentenebene, wonach jede einzelne ECU gegen seine Spezifikation getestet wird [188, Kap. 8]. Anschließend werden sie schrittweise zusammen integriert, um ein komplettes Feature in einer komplexen Wirkkette aus vernetzten Sensoren, funktionalen Steuergeräten und Aktuatoren zu verifizieren (Cluster-HiL) [154, Kap. 5] [188, Kap. 8].

Eine junge Methode in späten Entwicklungsphasen ist die *Vehicle-in-the-Loop (ViL)* Simulation, die komplementär zu den Fahrversuchen stattfindet. Hier wird das komplette Probefahrzeug in die virtuelle Umgebung eingebettet und die Sensorik des Fahrzeugs durch diese stimuliert. Die Sensorik kann dabei durch eine Schnittstelle künstlich durch das Umgebungsmodell stimuliert oder durch virtuelle Komponenten ersetzt werden. Hintergrund dieser Methode ist die zunehmende Anzahl an Fahrerassistenzfunktionen und deren Anforderungen an funktionale Sicherheit. Bedingt durch den dadurch resultierenden hohen Vernetzungs- und Automatisierungsgrad entstehen eine Vielzahl an Szenarien, die durch reale Feldversuche nicht mehr reproduzierbar und skalierbar zu realisieren sowie wirtschaftlich nicht tragbar sind [188, Kap. 8]. Analog dazu existieren erste Konzepte wie in [131], die auf Basis von Cluster-HiL und virtuellen Fahrszenarien automatisiert und kontinuierlich Fahrmanöver von Fahrerassistenzsystemen evaluieren können, ohne auf vordefinierte Testszenarien aus der Spezifikation zurückgreifen zu müssen.

Eine virtuelle Integration von AUTOSAR Softwarekomponenten (SWCs) wird in [110] und [133, Kap. 12] vorgestellt. Der Vorteil ist die frühzeitige Absicherung von SWCs und deren Schnittstelleninteraktionen unter Berücksichtigung einer abstrakten AUTOSAR Basissoftware sowie in [110] die Möglichkeit, simple Kommunikationslatenzen zu berücksichtigen, jedoch ohne deren Protokolle. Ein Ansatz zur virtuellen Integration von ECUs mittels FMUs (siehe Unterabschnitt 3.3.2.1) wird in [24] vorgestellt.

3.3.2 Simulation in frühen Entwicklungsphasen

Um das Entwicklungsrisiko zu reduzieren, werden, wie in Abschnitt 3.3 beschrieben, virtuelle Tests und Simulationen durchgeführt. MiL Simulationen und virtuelle Tests sind dabei besonders in frühen Entwicklungsstadien geeignet. Zur Verhaltensspezifikation und -simulation einer E/E-Architekturen existieren grundsätzlich zwei Möglichkeiten:

1. Das Verhalten wird auf Basis heterogener Berechnungsmodelle in ein oder mehreren Modellierungs- und Simulationswerkzeugen erstellt und miteinander kombiniert oder
2. auf Basis von modellbasierten ADLs für E/E-Architekturen spezifiziert.

Beide Ansätze werden im Folgenden detaillierter diskutiert.

3.3.2.1 Ansätze basierend auf heterogener (Co-)Simulation

Durch die heterogene Natur von E/E-Architekturen in Form von vernetzten, eingebetteten Systemen und den unterschiedlich involvierten Entwicklerdomänen werden meist in ihrer Domäne etablierte Modellierungsformalismen und Werkzeuge verwendet, die heterogenen Berechnungsmodellen folgen.

Um die Wechselwirkung zwischen den Domänen untersuchen zu können, ist jedoch eine Kombination aus den unterschiedlichen Berechnungsmodellen notwendig [31]. Eine Kombination dieser aus verschiedenen Quellwerkzeugen wird dabei Co-Simulation genannt. Neben dieser *kopplungsbasierten* Co-Simulation [142] kann weiter in *formale* Ansätze unterteilt werden. Formale heterogene Modellierungs- und Simulationswerkzeuge erlauben eine mathematisch eindeutige Beschreibung von Berechnungsmodellen sowie die beliebige Kombination aus diesen in einem einzigen Modell mittels formal beschriebener Semantiken. Zu den formalen Frameworks zählen bspw. das in Abschnitt 2.3 vorgestellte Ptolemy II oder Metro II [30] und ForSyDe [149].

Im Bereich der eingebetteten Systeme existieren neben den formalen Frameworks Modellierungsformalismen und Beschreibungssprachen, welche einige wenige Berechnungsmodelle kombinieren. Dazu zählen Formalismen wie Hybrid Automata [139], verbreitete Systembeschreibungssprachen wie VHDL-*Analog/Mixed-Signal (AMS)* [55] oder SystemC-AMS[20] [54] und etablierte Simulationswerkzeuge wie Simulink und Stateflow [104]. Letztere realisieren sog. *hybride* Systeme [31], indem sie diskrete und kontinuierliche Berechnungsmodelle kombinieren.

Bei koppelbasierten Co-Simulationen zur Untersuchung der Wechselwirkung unterschiedlicher Domänen werden nach [142] zwei Strategien verfolgt: im ersten Fall werden formale Frameworks wie Ptolemy II dazu verwendet, um als zentraler Koordinator die Ausführung zwischen heterogenen Modellen verschiedener Quellsimulatoren zu steuern. Dazu zählen Arbeiten wie [93, 7]. Auf der anderen Seite werden Simulatorkopplungen meist über eine standardisierte Schnittstelle realisiert, welche zusätzlich eine darunterliegende Laufzeitumgebung bietet. Die Simulatoren müssen eine zu diesen Schnittstellen konforme Implementierung aufweisen und ihr zugrunde liegendes Berechnungsmodell der Schnittstelle anpassen und umgekehrt. Verbreitete Standards, die eine Co-Simulationsschnittstelle spezifizieren sind die *High-Level Architecture (HLA)* [52] und die FMI Schnittstelle [116]. Zu Simulatorkopplungen und Co-Simulation zählen Arbeiten wie [28, 48, 74, 106, 142, 160]. Koppelbasierte Ansätze bieten den Vorteil, dass existierende Modelle einfach wiederverwendet werden können. Je nach Spezifikation

[20] SystemC-AMS beschränkt sich zur Lösung von kontinuierlichen Modellen im Gegensatz zu VHDL-AMS auf lineare Solver und bietet so nur eine limitierte Simulation kontinuierlicher Probleme. Zudem führt die Referenzimplementierung des linearen Solvers u. U. zu nicht lösbaren Systemen elektrischer Netzwerke [44].

und Umsetzung der Schnittstelle erlauben diese zusätzlich eine zeitlich entkoppelte und verteilte Ausführung wie etwa die HLA [142]. Die HLA Laufzeitumgebung hat allerdings den Nachteil, dass sie Co-Simulation nur über verteilte Netzwerkknoten unterstützt [24, S. 25].

Die Integration von domänenspezifischen Modellen und Simulationswerkzeugen stellt nach wie vor eine Herausforderung dar und führt selten zu einer effektiven Integration, sondern vielmehr zu anfälligen Werkzeugketten [59, 31]. Gründe dafür sind u. a. inkompatible Datenrepräsentationen, nicht dokumentierte API und weiche oder nicht spezifizierte Semantiken zwischen Berechnungsmodellen [31, 80]. Als Folge resultiert, vor allem wenn die Co-Simulationsschnittstelle nicht standardisiert ist, eine *amporhe* Heterogenität und nicht deterministische Ausführung, wenn die Semantik nicht eindeutig ist [34, 31]. Selbst die Kombination von deterministischen Berechnungsmodellen im Bereich von CPS kann zu nicht deterministischem Verhalten führen und sind Gegenstand von Forschungsprojekten wie PTIDES oder PRET. Diese verfolgen das Ziel, trotz der Kombination ein deterministisches Verhalten aufbauen zu können [82]. Grundlage dafür ist nach [82] ein deterministisches Zeitmodell, wie es mit formalen Frameworks wie PtII bereitgestellt wird. Der Vorteil des Einsatzes eines formalen Frameworks zur Modellierung und Simulation und als Co-Simulationskoordinator ist daher eine verbesserte Interoperabilität von heterogenen Berechnungsmodellen und damit den unterschiedlichen Domänen [142].

Functional Mock-up Interface FMI[21] ist eine Schnittstelle zum standardisierten Austausch und zur Co-Simulation von dynamischen Modellen unterschiedlicher Quellwerkzeuge. Initiiert wurde die Entwicklung von der Daimler AG mit dem Ziel, Verhaltens- bzw. Simulationsmodelle zwischen OEMs und Zulieferern standardisiert austauschen zu können. Ein Modell, das diese Schnittstelle implementiert wird FMU genannt. Letztere ist eine komprimierte fmu-Datei im zip-Format, die zum einen eine Schnittstellenbeschreibung des Modells in Form einer standardisierten XML-Spezifikation enthält (engl. *Model Description*). Zusätzlich beinhaltet sie eine dynamische Bibliotheksdatei oder dessen Quelltexte in C, die das kontinuierliche Verhalten des Modells mittels einer standardisierten C-Code-API implementiert. Die API orientiert sich dabei an der Semantik von FSMs.

Im Falle einer Co-Simulation dient die FMU entweder als Slave, welche die Solver zur Simulation des Modells direkt bereitstellt. Oder als Wrapper, der keinen eigenen Solver und Simulationskernel besitzt und lediglich zwischen dem Ziel-Simulationswerkzeug und dem FMI-Master vermittelt. In beiden Fällen ist die FMU im Modell eines Masters eingebettet, der die Synchronisierung der Slaves

[21] https://fmi-standard.org/, zuletzt aufgerufen am 24.09.2019.

oder Wrapper sowie deren Ausführung steuert. Die Algorithmen des Masters sind jedoch nicht Teil der FMI Spezifikation und bietet daher viele Freiheitsgrade in deren Implementierung [12, 18, 152]. Im Falle eines Modellaustauschs ein oder mehrerer FMUs werden diese, im Gegensatz zur Co-Simulation, ganzheitlich betrachtet und global durch den Solver des Master-Simulationswerkzeugs simuliert.

Der Standard definiert damit in der aktuellen Version 2.0 von 2014 eine geeignete Semantik für den Modellaustausch und Co-Simulation von zeitkontinuierlichen Modellen. Im Zuge von gesammelten Erfahrungen hat sich die Notwendigkeit ergeben, hybride Systeme mit diskreten Ereignissen zu unterstützen [19, 29]. In der zukünftigen Version des FMI Standards v2.1[22] bzw. v3.0 ist eine hybride Co-Simulation mit diskreten Events in der Feature-Liste der Alpha Version vorgesehen, aber zum Zeitpunkt des Schreibens dieser Arbeit noch nicht publiziert.

Lösungsansätze zur Integration und verteilten Co-Simulation von echtzeitfähigen Systemen, z. B. HiL-Systeme, mit Simulationsmodellen unterschiedlicher Quellwerkzeuge wurde kürzlich durch die Spezifikation des *Distributed Co-Simulation Protocol (DCP)*[23] [115] präsentiert. Dieses spezifiziert ein Kommunikationsprotokoll auf Applikationsebene nach dem Master-Slave-Prinzip, das den Austausch von Simulationsdaten und Metadaten zur Konfiguration der Co-Simulation erlaubt, unabhängig vom zugrunde liegenden physikalischen Medium wie ein CAN-Bus. Die Spezifikation wurde kompatibel zur FMI Schnittstelle entwickelt, um den Integrationsprozess von Modellen zu vereinfachen und die Wiederverwendung zu steigern. Neben dem Master-Slave Prinzip spezifizieren beide eine Semantik basierend auf Zustandsautomaten, wobei ein DCP Slave eine FMI-konforme Initialisierungsmethode implementiert. Der Prozess zur Modellintegration wird ebenfalls durch standardisierte XML-Austauschformate definiert [72].

3.3.2.2 ADL-basierte Ansätze

Eine Möglichkeit die Limitierung von MiL Simulationen auf Architekturebene zu umgehen ist, das Verhalten mithilfe von modellbasierten EEA Modellen zu spezifizieren. Wie in Abschnitt 3.1 erörtert, ist eine Verhaltensspezifikation jedoch entweder nicht oder nur limitiert möglich, indem auf externe Modelle verwiesen wird oder nur einfache Verhaltensmodelle verwendet werden. Domänenübergreifende Aspekte wie Buskommunikation oder Elektrik aus einem E/E-

[22] https://web.archive.org/web/20171224110743/https://fmi-standard.org/downloads/, zuletzt aufgerufen am 02.07.2018.

[23] https://dcp-standard.org, zuletzt aufgerufen am 24.09.2019.

Architekturmodell können somit nicht automatisiert berücksichtigt werden (vgl. Abschnitt 3.1).

Vor diesem Hintergrund sind in der Forschung Projekte entstanden, die diese Lücke schließen sollen. Ein prominentes Beispiel ist das INTO-CPS Projekt. INTO-CPS[24] ist ein von der EU gefördertes Projekt, in dem eine Werkzeugkette zur Entwicklung von CPS auf Basis der FMI-Schnittstelle im Vordergrund steht. Die Systemarchitektur bestehend aus Teilsystemen unterschiedlicher Domänen wird abstrakt mithilfe von SysML beschrieben. Mittels eines CPS-Profils für SysML kann die Schnittstellenbeschreibung in Form einer FMU-Modellbeschreibung generiert werden. Das eigentliche Verhalten muss jedoch durch Import des XML-Platzhalters im Zielwerkzeug modelliert und anschließend als fertige FMU wieder exportiert werden. Die exportierten FMUs werden in einer sog. INTO-CPS Applikation importiert, in der die FMUs aus einer Bibliothek ausgewählt werden können. Ein *Connection Diagram* Profil verbindet die FMU-Modelle und können anschließend co-simuliert werden [75, 76, 77].

Das SmartSE (*Smart Systems Engineering*) Projekt[25] [187] ist ein vergleichbarer Ansatz. Ähnlich zum INTO-CPS Projekt wird hier der Austausch von Verhaltensmodellen innerhalb der EEA-Entwicklung über FMUs adressiert und mit dem Systems Engineering Gedanken verknüpft. Es wird zudem ein FMI *Requirements Template* Annex vorgeschlagen, um den Austausch von FMUs basierend auf Anforderungen weiter zu vereinfachen.

Eine weiterführende Diskussion und Abgrenzung bzgl. ADL-basierter Ansätze wird in Abschnitt 6.7 abgehandelt.

3.3.3 Diskussion

Die virtuelle Integration von AUTOSAR SWCs in [110] und [133, Kap. 12] hat den Nachteil, dass das Verhalten für alle SWCs als AUTOSAR-konformer C-Code vorliegen muss, was in frühen Phasen der E/E-Architekturentwicklung typischerweise nicht der Fall ist, wenn dieser nicht automatisiert aus dem EEA-Modell generiert werden kann. Sämtliche ECU-Konfigurationen und Kommunikationsbeschreibungen müssen zunächst importiert werden. Zudem sind Test- und Systemmodell nur lose gekoppelt, wodurch die Stimuli in externen Testfällen spezifiziert und dann importiert werden müssen. Änderungen am zugrunde liegenden E/E-Architekturmodell haben daher Anpassungen an den importierten Beschreibungen zur Folge und werden nicht automatisiert berücksichtigt. Bei der virtuellen Integration von ECUs über FMI [24] muss ebenso generierter C-Code

[24] http://into-cps.org/, zuletzt aufgerufen am 22.06.2018.
[25] http://www.prostep.org/projekte/smart-systems-engineering/, zuletzt aufgerufen am 03.07.2018.

der Funktion vorliegen, der manuell in den virtuellen Prototyp integriert werden muss, bevor eine FMU exportiert und simuliert werden kann.

Unabhängig von ihrer Realisierung mit formalen, modellbasierten Werkzeugen oder Co-Simulation adressieren MiL Simulationen und virtuelle Tests meist die simulative Validierung individueller Funktionen oder (virtueller) Hardware/-Softwarekomponenten einer ECU [16, 24, 151, 155], [154, Kap. 5]. Die frühzeitige Simulation auf Systemebene beim OEM, sprich die Simulation der logischen Systemarchitektur einer EEA, die zunächst unabhängig von der späteren technischen Realisierung sein kann (Systemspezifikation, siehe Abb. 2.1), wird damit nicht ausreichend unterstützt.

Die MiL Simulationen und virtuellen Tests geschehen darüber hinaus weitestgehend in einem eigenständigen Prozess mit domänenspezifischen Werkzeugen, in dem die spezifischen Parameter der E/E-Architektur und Umgebungen nachmodelliert werden müssen (siehe Unterabschnitt 3.3.1 und vgl. [154, Kap. 5] [188, Kap. 8]), anstatt die Eigenschaften des zugrunde liegenden E/E-Architekturmodells gleich mitzuberücksichtigen. Zwar existieren weitere standardisierte Austauschformate wie ECU- und Systemkonfigurationen von AUTOSAR, Leitungssatz oder Kommunikationsbeziehungen via DBC, jedoch müssen diese einen Import/Export-Prozess in den jeweiligen Werkzeugen durchlaufen. Änderungen am zugrunde liegenden E/E-Architekturmodell haben daher Anpassungen an den importierten Beschreibungen zur Folge und werden nicht automatisiert berücksichtigt. Die Konsistenz zu einem bestimmten Entwicklungs- bzw. Versionsstand zwischen E/E-Architekturmodell und den importierten Beschreibungen in den externen Simulationswerkzeugen ist damit u. U. nicht sichergestellt. Die Bewertung von Architekturvarianten bzgl. der gewonnenen Simulationsdaten gestaltet sich aus demselben Grund als schwierig.

Bezüglich der ADL-basierten Ansätzen zur Verhaltenssimulation von E/E-Architekturen ist, wie in Abschnitt 3.1 erörtert, eine Verhaltensspezifikation entweder nicht (bspw. in PREEvision® bis v8.5) oder nur limitiert möglich. Auf der anderen Seite beschränkt die ausschließliche Verwendung von FMUs ausgewählter Werkzeuge, wie in den diskutierten Projekten INTO-CPS, SmartSE oder vergleichbaren Ansätzen, die frühzeitige Architekturevaluation auf abstrakter Ebene, da durch die verteilte Entwicklung der einzelnen Systemkomponenten in den Zielwerkzeugen sämtliche FMUs zur Verfügung stehen müssen, bevor eine Systemsimulation stattfinden kann. Dies ist in frühen Entwicklungsphasen typischerweise nicht gegeben, in der u. U. die Spezifikation der technischen Realisierung der Komponenten noch gar nicht verfügbar ist. Im Kontext von INTO-CPS wird ein Ansatz namens *Discrete-Event First* beschrieben, die diese Limitierung umgehen soll [121], allerdings muss auch hier die Verhaltensmodel-

lierung im externen Werkzeug erfolgen und diese als FMU exportiert werden, bevor sie in der INTO-CPS Applikation importiert und simuliert werden kann.

Die Verhaltensmodellierung, bspw. durch die Anreicherung von logischen Funktionen des Architekturmodells mittels State Charts oder ausführbaren Blöcken, kann nicht integriert und somit nicht transparent stattfinden. Dieses sog. Black-Box Verhalten zählt zu den bekanntesten Limitierungen von Co-Simulationen [19, 18]. Ein weiterer Nachteil der reinen Co-Simulation mittels FMUs ist, dass, aufgrund ihrer zeitdiskreten und zeitkontinuierlichen Semantik, ereignisdiskrete Simulationen mit DE Semantik noch nicht nativ unterstützt werden. Das heißt sie werden immer auf Basis eines diskreten Zeitschritts ausgeführt und können nicht sofort auf plötzlich auftretende Ereignisse reagieren. Aus diesem Grund ist es u. a. mit der INTO-CPS Applikation nicht möglich, FMUs mit Netzwerkkommunikation zu co-simulieren [76, Kap. 6].

Eine hybride Lösung zwischen integrierter Modellierung ausführbarer Blöcke unterschiedlicher Semantik wie Actors oder State Charts und der Möglichkeit, das abstrakte Verhalten mithilfe von FMUs im Laufe des Entwicklungsprozesses zu verfeinern und auszutauschen, ist eine geeignetere Methode. Die Struktur und die Schnittstellen des statischen Funktionsmodells und der dazugehörigen Verhaltensmodelle sollen dabei unberührt bleiben, damit die Modularität und einfache Austauschbarkeit gewahrt wird.

3.4 Bewertung von E/E-Architekturen

3.4.1 Überblick

Zur Absicherung von E/E-Architekturen in der Konzeptphase (Technologieentwicklung, siehe Unterabschnitt 2.1.1) werden typischerweise Metriken eingesetzt, um quantitative Eigenschaften bzgl. bestimmter Bewertungskriterien über eine Architektur zu erhalten und sie somit bewerten zu können. Eine *Metrik* „ist eine Berechnungsvorschrift, mit deren Hilfe die Bewertung einer E/E-Architektur bezüglich vorgegebener Bewertungskriterien in eine oder mehrere Kennzahlen durchgeführt wird.“ [41, S. 98]. Eine Übersicht über mögliche Bewertungskriterien ist in Abb. 3.12 illustriert, wobei diese nicht als absolut und vollständig angesehen werden kann, da Bewertungskriterien oft herstellerabhängig und geschütztes geistiges Eigentum sind [154] [41, Kap. 4].

Bewertungskriterien zur Untersuchung der Machbarkeit einer E/E-Architektur sind u. a. physikalische Baubarkeit, die Einhaltung von gesetzlichen Vorschriften und logisches Vernetzungskonzept. Zusätzlich sind *statische Metriken* wie Kosten, Gewicht oder Leitungsquerschnitte und -längen wichtige Aspekte bei der

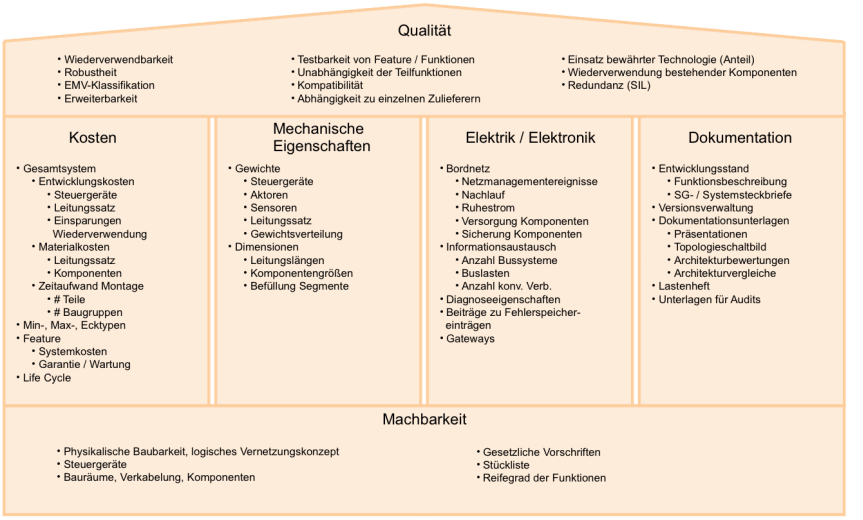


Abbildung 3.12: Mögliche Kategorisierung von Bewertungskriterien von E/E-Architekturen (Quelle: [41, S. 102]).

Bewertung von Fahrzeugvarianten, diese zählen zu den globalen Kenngrößen [153].

Darüber hinaus existieren weitere Eigenschaften, die abhängig von der zur Verfügung stehenden Funktionalität bzw. dem Equipment im Fahrzeug sind. Dazu zählen bspw. Buslast aber auch *dynamische Metriken* wie Kommunikationslatenzen bestimmter Busse oder zeitbehaftete Strombedarfe. Diese können abhängig von Betriebszuständen, Funktionsverhalten oder komponentenspezifischem Betriebsverhalten sein, bspw. ein spezifisches Busprotokoll, und können daher mit rein statischen Attributen nicht ausreichend beschrieben werden, sondern bedürfen einer simulativen Unterstützung. Diese Metriken sollen aber nicht ausschließlich getrennt voneinander betrachtet, sondern bei der Auslegung einer E/E-Architektur komplementär zu den globalen Zielen berücksichtigt werden, um innerhalb eines Fahrzeuglebenszyklus geeignete Trade-offs zwischen existierenden Architekturvarianten zu finden. [153] [41, Kap. 4] [159, Kap. 2]

Die automatisierte Berechnung dieser Metriken und der Vergleich von Architekturvarianten spielt daher eine wichtige Rolle. Dies rührt in erster Linie von mehreren Produktlinien und dem Plattformprinzip, in dem Architekturteile über Produktlinien hinweg wiederverwendet werden. Zusätzlich tragen mehrere

Ausstattungs- und Realisierungsalternativen zur massiven Steigerung der Kombinatorik bei. Eine Bewertung aller Varianten ist aber auch durch automatische, computergestützte Metrikberechnungen nicht handhabbar. Daher werden oft sog. Minimal-, Maximal- und Ecktypen abgeleitet, die eine repräsentative Menge zur Bewertung einer Gesamtarchitektur bilden oder bei Kunden besonders beliebt sind [41, Kap. 4 & Kap. 5] [153].

3.4.2 Ansätze

Zur Bewertung und zum Vergleich von Architekturvarianten bzgl. statischer und dynamischer Metriken sind bedingt durch den Wandel hin zum modellbasierten Architekturentwurf unterschiedliche Ansätze und Werkzeuge entstanden. Zunächst kann dabei zwischen Ansätzen und Werkzeugen unterschieden werden, die auf modellbasierte Architekturbeschreibungen wie SysML/MARTE, EAST-ADL oder proprietäre Sprachen zurückgreifen (vgl. Abschnitt 3.1 und [41, Kap. 4.4]) und welchen, bei denen Architekturinformationen entweder manuell hinzugefügt werden müssen oder über dedizierte Konfigurationsdateien in die Analyse-/Simulationsmodelle eingelesen werden.

Beispiele für letztere sind z. B. das kommerzielle Werkzeug Autonomie [45], der akademische Simulator ITE-Sim [181] oder analytische Ansätze, wie sie in [159] basierend auf dem kommerziellen Werkzeug Symtavision vorgestellt werden. Autonomie und ITE-Sim verwenden beide XML-Beschreibungen, um das Fahrzeugmodell bzw. die E/E-Architektur zu konfigurieren. Während sich Autonomie auf die simulative Verifikation von Regelungsfunktionen mittels automatisiert zusammengesetzten Simulink-Subsystemen konzentriert, baut ITE-Sim auf Skripte bestehend aus einfachen Kontrollstruktur-Primitiven sowie verschiedener Instruktionstypen, die in SystemC übersetzt werden, um das Verhalten zu simulieren. Die technische Steuergerätearchitektur wird mittels XML beschrieben, worin Leistungsaufnahme und Verarbeitungsrate von Instruktionen einer CPU angegeben werden können. Daraus lassen sich aus geskripteten Fahrzyklen die Stromaufnahme und Ressourcenauslastung von Steuergeräten bzw. CPUs bestimmen.

Durch ihre dateibasierte Konfiguration gestaltet sich die Handhabung von Architekturvarianten und deren Vergleich bzgl. der nicht-funktionalen Metriken allerdings als schwierig. Zudem ist eine durchgängige und ebenenübergreifende Nachverfolgbarkeit der E/E-Architektur im Vergleich zu modellbasierten Ansätzen nicht gegeben. In diesem Zusammenhang gestaltet sich die Erhaltung der Konsistenz zwischen Verhaltensmodell und technischer Architektur, insb. bei verteilter Kollaboration, ebenso als schwierig.

Mit der Etablierung des modellbasierten Entwurfs von E/E-Architekturen werden deshalb vermehrt Ansätze entwickelt, die sich auf die Wiederverwendung der modellbasierten Architekturinformationen stützen. Allerdings nutzen nach [192] nur wenige Arbeiten (9 % von 188 untersuchten Papers), die sich mit der Bewertung von nicht-funktionalen Metriken beschäftigen, überhaupt die modellbasierte Architekturbeschreibung aus oder decken nur einen Teil der E/E-Abstraktionsebenen ab. Des Weiteren ist es wichtig, eine kombinierte Analyse von sowohl statischen als auch dynamischen Metriken durchzuführen, um eine globale Betrachtungsweise der Architektur zu gewährleisten [153]. Dazu ist es notwendig, die durch Simulation oder externe Analyse gewonnenen Daten im ursprünglichen EEA-Datenmodell zur Verfügung zu stellen, um diese bspw. in weiteren Metrikanalysen weiterverarbeiten oder Anpassungen am E/E-Architekturmodell zur iterativen Verbesserung vornehmen zu können.

Beispiele für Ansätze basierend auf EAST-ADL Architekturbeschreibungen sind [192] und die Analysis-Workbench des MAENAD Projekts [97]. Die Autoren in [192] erweitern die EAST-ADL-Modelle um ein entsprechendes UML-Profil sowie um Aktivitätsdiagramme aus der SysML und MARTE-Annotationen. Zusätzlich führen sie ein UML-Profil zur Abbildung von Redundanzen ein und nutzen diese Erweiterungen um eine Architektur bzgl. statischer Metriken wie Kosten, Buslast und ECU-Auslastung unter Verwendung von evolutionären Algorithmen zu optimieren. Zudem ist eine Konvertierung des EAST-ADL-Modells in das Format des statischen Timing-Analysewerkzeugs *Qompass* möglich, in welchem auch die Optimierung stattfindet. Die Arbeit ist in die Analysis Workbench des MAENAD Projekts mit eingeflossen, welche u. a. ein Optimierungs-Framework für die EAST-ADL-Modelle beinhaltet. Die Analyse und Metrikberechnung ist bei diesen Ansätzen externen Werkzeugen vorbehalten, die über sog. Gateways lose an das EAST-ADL-Modell gekoppelt sind [41, Kap. 4.4] [97].

Das AADL Werkzeug OSATE (vgl. Unterabschnitt 3.1.2) unterstützt eine Reihe an Analyse-Plugins, welche (quasi-)statische Metriken auf Basis der modellierten, nicht-funktionalen Architektureigenschaften berechnen können. Dazu zählen eine Ende-zu-Ende-Latenzanalyse basierend auf Ausführungs- und Sampling-Zeiten von Tasks, aber auch Kommunikationsverzögerungen und Verarbeitungsverzögerungen in Abhängigkeit von Prozessorgeschwindigkeiten können berücksichtigt werden. Weitere Plugins beinhalten Analysen zu Gesamtgesicht, Ressourcenverbrauch, Buslast und Energiebudget.

Kugele et al. [73] schlagen eine domänenspezifische Sprache namens AAOL zur multi-kriteriellen Optimierung von E/E-Architekturen vor, beschränken sich aber auf statische Metriken wie Kosten, Gewicht und benötige Busbandbreite. Weitere Beispiele sind der VEIA Prozess [98], welcher jedoch keine modellbasierte Beschreibung der Metriken erlaubt, und das SPEEDS Projekt [156].

Die genannten Ansätze erlauben allerdings alle keine grafische Notation der Metrikbeschreibungen, bieten keinen automatisierten und effizienten Vergleich von Varianten (mit Ausnahme des Optimierungs-Frameworks des MAENAD Projekts) und sind nicht integriert. Eine Integration bedeutet in diesem Zusammenhang, dass Metrikberechnungen direkt auf Modellartefakte der E/E-Architektur zugreifen können und die Modellierung der Metriken in demselben Werkzeug wie die Architekturmodellierung stattfindet (realisiert durch dedizierte Metamodelle und wohldefinierte Schnittstellen). Eine fehlende Integration erschwert sowohl die Ergebnisaufbereitung als auch die Nachvollziehbarkeit der Ergebnisse, die mit Modellartefakten assoziiert sind. Zusätzlich gestaltet sich dadurch ein effizienter Variantenvergleich bzgl. der Metriken als problematisch.

Vor diesem Hintergrund wurde in [41] ein modellbasiertes und integriertes, grafisch notiertes Verfahren zur Berechnung und Visualisierung von Metriken sowie zum Architekturvergleich entwickelt. Umgesetzt ist dieses Verfahren im Werkzeug PREEvision® (siehe auch Unterabschnitt 3.2.5). Durch den integrierten Ansatz haben die Metrikbeschreibungen direkten Zugriff auf alle Architekturartefakte auf allen Abstraktionsebenen. Da Architekturvarianten über einen Produktlinienansatz und 150 % Modelle Bestandteil des eigentlichen E/E-Architekturmodells sind, können diese mithilfe der Metrikbeschreibungen effizient und automatisiert miteinander verglichen werden.

3.4.3 Diskussion

Ein gemeinsamer Nachteil aller genannten modellbasierten ADL Ansätze ist, dass diese aufgrund ihrer naturgemäß statischen Modellierung lediglich statische bzw. quasi-statische Metriken berechnen können. Das heißt Metriken können nicht oder nur indirekt auf Daten aus dynamischen Analysen und Simulationen zurückgreifen. Dies ist jedoch notwendig, um bspw. eine Kommunikationslatenz von Wirkketten in Abhängigkeit vom Funktionsverhalten und dem zugrunde liegenden Protokoll (Arbitrierung etc.) oder Strombedarfe in Abhängigkeit von Steuergeräteaktivitäten zu bestimmen. Mithilfe von Gateways, welche im MAENAD Projekt eingesetzt werden, ist dies indirekt und teilweise möglich, indem angereicherte EAST-ADL-Modelle über Gateways in ein Format des Zielwerkzeugs übersetzt und dort ausgeführt werden. Beispielsweise existieren ein AUTOSAR-Gateway zur Anbindung einer AUTOSAR-konformen Softwarearchitektur und ein Gateway zur oben bereits erwähnten frühzeitigen Timing-Analyse von EAST-ADL-Modellen im Werkzeug *Qompass*.

Diese Gateways sind auch Bestandteil der Arbeit in [192], wobei sich die Optimierung auf statische Metriken beschränkt. Die integrierte Ausführung und Ergebnisaufbereitung einer Simulation innerhalb der EAST-ADL Model-

lierungsumgebung wird nicht adressiert. Zwar ist die Rückführung der Analyse-/Simulationsergebnisse in das ursprüngliche EAST-ADL Modellierungswerkzeug über UML/MARTE-Konverter vorgesehen, die genaue Realisierung der Ergebnisrückführung zur Verwendung in weiteren modellbasierten Metrikanalysen werden jedoch nicht näher diskutiert [97]. Außerdem hat eine UML/MARTE basierte lose Kopplung die Nachteile von universellen Beschreibungssprachen (vgl. Unterabschnitt 2.1.4), sodass durch die reine Annotation von Stereotypen sehr textlastige und schwer wartbare Diagramme entstehen.

Ein Simulink-Gateway der MAENAD Analysis-Workbench zur Generierung von Simulink-Blackboxes (Platzhalter ohne internes Verhalten) stellt zwar die Konsistenz zwischen den generierten Blackboxes und den ursprünglichen EAST-ADL Funktionen sicher. Allerdings müssen die Modellierung und Simulation des eigentlichen Verhaltens, das untersucht werden soll, extern innerhalb von Simulink erfolgen. Eine Rückführung der Daten zur weiteren Analyse wird nicht adressiert. Analoges gilt für die Generierung von Stateflow-Modellen aus EAST-ADL-Modellen mit Verhaltensannotationen des Behavioral Annex (siehe Unterabschnitt 3.1.1.2), die zwar direkt das Verhalten ausführen, aber die Daten nicht rückführen können [97]. Darüber hinaus werden bei diesen Simulink-/Stateflow-Ansätzen keine abstraktionsebenenübergreifenden Architekturinformationen wie die Netzwerktopologie automatisiert mit in das Verhaltensmodell generiert.

Abschließend ist anzumerken, dass die in diesem Kapitel und in Abschnitt 3.1 diskutierten Ansätze, welche eine Generierung von Simulations- bzw. Analysemodellen erlauben, keine variantensensitive Generierung unterstützen. Das heißt Zusammenhänge zwischen EEA-Modellartefakten, die abhängig von einer Architekturvariante aktiv sind oder nicht, werden bei der Generierung nicht berücksichtigt. Dies ist allerdings eine wichtige Eigenschaft, wenn eine automatisierte Evaluation und ein Vergleich verschiedener Architekturvarianten bzgl. bestimmter Metriken untersucht werden soll. Ohne diese Berücksichtigung ist es nicht möglich ein variantenreiches E/E-Architekturmodell, d. h. eine Obermenge an möglichen Architekturvarianten, als Eingangsgröße für die Generierung eines Simulationsmodells und somit zur Analyse dynamischer Metriken einzusetzen.

Das Optimierungs-Framework des MAENAD Projekts adressiert dies teilweise. Das Framework ist prototypisch als eigenständiges Werkzeug entwickelt worden und erwartet als Eingangsgröße ein variantenreiches EAST-ADL-Modell, welches mithilfe sog. *Encodings* über ein *Variability Resolution Mechanism* Modul zu einer Menge an vollständigen Modellen aufgelöst wird, ähnlich zu den 100 % Architekturvarianten eines Produktlinienansatzes. Zurückgeliefert wird eine Menge an pareto-optimalen Architekturvarianten bzgl. der untersuchten Optimierungskriterien. Der Nachteil besteht aber darin, dass je aufgelöster Variante ein komplettes, variantenunabhängiges EAST-ADL-Modell in Form einer eaxml-Beschreibung

exportiert werden muss. Das heißt die aufgelösten Variantenmodelle sind nicht mehr Teil des ursprünglichen, variantenreichen EAST-ADL-Modells, sondern müssen einzeln konvertiert und an die externen Analysewerkzeuge übergeben werden.

Eine integrierte Variantenbetrachtung bietet hingegen den Vorteil, dass nur ein einziges E/E-Architekturmodell benötigt wird und zusätzliche Import/Export-Prozesse vermieden werden. Des Weiteren können mit einem integrierten Ansatz beliebige Modellartefakte auf allen E/E-Abstraktionsebenen auf Basis einer aktiven Variante inkludiert oder ausgeschlossen werden und Ergebnisse von Metrikberechnungen einfacher mit den Architekturartefakten verknüpft werden. Das Optimierungs-Framework besitzt darüber hinaus keine Möglichkeit - auch bedingt durch die fehlende integrierte Variantenbetrachtung - die Ergebnisse von externen Analysewerkzeugen in weiteren Metrikberechnungen zu verwenden. Modellanpassungen an der aktiven Architekturvariante innerhalb einer Analyse und daraus resultierende, notwendige Syntheseschritte wie ein Signalrouting werden ebenso wenig adressiert. Metriken sind nach [97] begrenzt auf einfache Kostensummierungen und auf eine einfache Schnittstelle zum externen Analysewerkzeug HiP-HOPS[26] zur Durchführung von Abhängigkeitsanalysen mittels Fehlerbäumen.

[26] http://www.hip-hops.eu/, zuletzt aufgerufen am 24.10.2018.

4 Ganzheitliche Methodik und Anforderungen

4.1 Motivation und Abgrenzung

Basierend auf dem untersuchten Stand der Technik und Forschung ergeben sich in den drei behandelten Themenfeldern jeweils Lücken bzw. Verbesserungspotentiale, wie es in den entsprechenden Diskussionen in Unterabschnitt 3.1.7, Unterabschnitt 3.3.3 und Unterabschnitt 3.4.3 dargelegt wurde. Diese lassen sich wie folgt zusammenfassen. Eine zusätzliche Abgrenzung gegen verwandte Arbeiten wird nach der Evaluation der in dieser Arbeit vorgestellten Ansätze und des Prototyps in Abschnitt 6.7 abgehandelt.

- **Modellbasierter Entwurf von E/E-Architekturen**
  - Keiner der diskutierten Ansätze bietet zugleich eine durchgängige Abdeckung aller relevanten EEA Ebenen sowie komplementäre, ausführbare Verhaltensspezifikationen[1].
  - Eine Modellierung ebenenübergreifender Interaktionen zwischen Verhaltensmodellen wird nicht hinreichend adressiert.
  - Die Verknüpfung von logischer, realisierungsunabhängiger Verhaltensspezifikation von Funktionen mit detaillierteren, realisierungsabhängigen Informationen aus dem Architekturmodell wird nicht adressiert. Dies gilt konsequenterweise auch für die nachfolgende Ausführung bzw. Simulation des Verhaltens.
  - Komplexere Verhaltensspezifikationen durch die Kombination verschiedener Modellierungsparadigmen/-notationen von Verhalten und mehreren Berechnungsmodellen werden nicht integriert unterstützt.
- **Simulation in frühen Entwicklungsphasen**
  - Klassische MiL Simulationen und virtuelle Integrationstests stützen sich nicht auf modellbasierte Architekturinformationen.
  - Daraus resultierende Import/Export-Prozesse verhindern eine automatisierte und effiziente Evaluation von Architekturvarianten und

[1] Eine Verhaltensmodellierung in PREEvision® hat erst in der im Juli 2018 veröffentlichten Version 9.0 Einzug gehalten, eine Simulation ist jedoch nicht möglich.

erschweren die Transparenz zwischen Architektur- und Simulationsmodellen.

- ADL-basierte Ansätze benötigen für eine durchgängige und iterativ verfeinerbare Simulation von E/E-Architekturmodellen eine Kombination aus integrierter Verhaltenssimulation und detaillierten domänenspezifischen Modellen, bspw. FMUs.

- **Bewertung und Vergleich von E/E-Architekturvarianten**
  - Die Bewertung von zugleich statischen und dynamischen Metriken wird von den diskutieren Ansätzen nicht integriert unterstützt.
  - Die Transparenz zwischen den externen Analyseergebnissen von dynamischen Metriken und dem Architekturmodell wird aufgrund der fehlenden integrierten Betrachtungsweise erschwert.
  - Weitere Metrikanalysen basierend auf den dynamischen Metrikergebnissen im EEA Modellierungswerkzeug werden nicht adressiert.
  - Iterative Modellanpassungen und daraus notwendige EEA Modell-Syntheseschritte werden nur bedingt berücksichtigt.
  - Eine variantensensitive domänen- bzw. ebenenübergreifende Synthese von Simulationsmodellen zur Analyse von dynamischen Metriken ist nicht gegeben.

Im folgenden Abschnitt wird ein Lösungskonzept zur Adressierung der o. g. Punkte präsentiert. Die Methodik integriert die drei Themenfelder modellbasierte E/E-Architekturentwicklung, die integrierte Verhaltensspezifikation sowie deren Simulation und Bewertung hinsichtlich Architekturvarianten in einer Gesamtmethodik, um eine frühzeitige und ganzheitliche Betrachtung zu gewährleisten. Zusätzlich werden die notwendigen Anforderungen dafür anhand relevanter Anwendungsfälle abgeleitet. Details zu konkreten Konzepten und zur Realisierung des Prototyps folgen in Kapitel 5.

4.2 Methodik

Der Schwerpunkt dieser Arbeit liegt in der architekturzentrierten Modellierung von ausführbarem Verhalten sowie in einer ebenenübergreifenden Simulationssynthese mehrerer Architekturaspekte und nicht-funktionalen Eigenschaften zur Simulation und dynamischen Bewertung von E/E-Architekturen. Ausgangspunkt ist daher ein einheitliches E/E-Architekturmodell, das als single-source Quelle alle relevanten Abstraktionsebenen nach Abb. 2.2 und zusätzlich die Architekturvarianten und deren Parametrisierungen enthält (vgl. Abb. 4.1 l. o.).

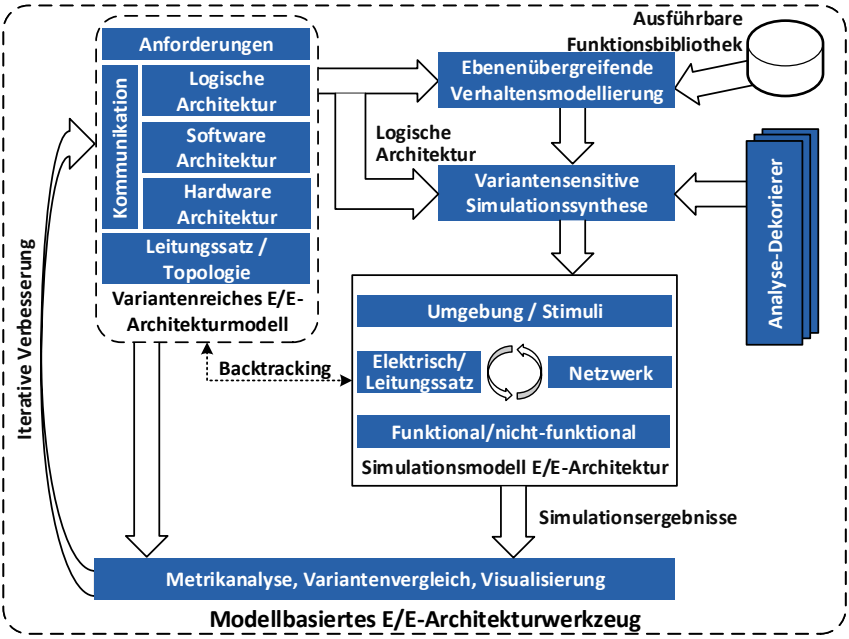


Abbildung 4.1: Ganzheitlicher Ansatz zur integrierten Simulationssynthese und dynamischen Bewertung modellbasierter E/E-Architekturen.

Dieses statische Architekturmodell wird mit ausführbaren Verhaltensmodellen angereichert, die aus einer Bibliothek stammen, integriert im E/E-Architekturwerkzeug neu entwickelt und ebenenübergreifend miteinander interagieren können. Eine nachgelagerte Simulationssynthese generiert basierend auf der realisierungsunabhängigen logischen Architektur ein einheitliches Simulationsmodell, das abhängig von der Konfiguration über sog. Dekorierer sowohl mehrere Domänen des Architekturmodells als auch zusätzliche, für die Analyse dynamischer Metriken instrumentierende, Simulationsartefakte enthalten kann.

Der Ansatz stellt eine Kombination aus Top-Down und Bottom-Up Verfahren dar, indem die zunächst realisierungsunabhängige logische Architektur als Eingangsgröße für eine ebenenübergreifende Simulationssynthese dient. Die Wahl der logischen Architektur als Eingangsgröße hat den Vorteil, dass diese typischerweise Ausgangspunkt neuer Innovationen und Funktionen darstellt. Sie bleibt innerhalb einer Fahrzeuggeneration und darüber hinaus über Jahre stabil, während die technische Realisierung über die Zeit unterschiedliche Ausprägungen

annehmen kann [153]. Die logische Architektur stellt somit das Fundament für weitere Entwicklungsphasen dar.

Durch ihre Verknüpfung mit mehreren Abstraktionsebenen der Architektur und des komplementär modellierten Verhaltens, kann bereits in frühen Phasen eine automatisch synthetisierte und domänenübergreifende Simulation bereitgestellt werden, die die einzelnen Aspekte auf den unteren Abstraktionsebenen sowie nicht-funktionale Eigenschaften in Kombination mit dem logischen Verhalten berücksichtigt. Bei Änderungen auf den unteren, detaillierteren Ebenen werden diese bei erneuter Ausführung der Synthese automatisch in der neuen Simulationsiteration berücksichtigt. Die Synthese geschieht darüber hinaus ausgehend von einem 150 % Modell (vgl. Unterabschnitt 3.2.4) variantensensitiv, damit die Synthese und die nachfolgenden Simulations- und Analyseschritte für mehrere Architekturvarianten durchgeführt werden und somit ein Variantenvergleich stattfinden kann. Die Simulationsergebnisse werden schließlich in der EEA-Entwicklungsumgebung verwendet, um Metrikanalysen sowie deren Aufbereitung bzw. Visualisierungen durchzuführen. Um die Transparenz bzw. die Konsistenz zwischen Architektur- und Simulationsmodellartefakten zu gewährleisten, wird ein Backtracking angewandt, welches sich auf die eindeutige *Universally Unique Identifier (UUID)* der ursprünglichen Architekturartefakte stützt.

Insbesondere werden durch diesen integrierten Charakter Werkzeugwechsel und fehleranfällige Import/Export-Prozesse zwischen E/E-Architekturwerkzeug und externen Simulations- und Analysewerkzeugen gezielt vermieden. Basierend auf der Evaluation der Ergebnisse können iterative Verbesserungen bzw. Anpassungen an Architektur- und/oder Verhaltensmodellen vorgenommen werden, welche Grundlage für automatisierte Evaluationsvorgänge sind. Zusätzlich erlaubt die integrierte Betrachtungsweise die iterative und automatisierte Variantenbewertung bzgl. statischer und insbesondere dynamischer Metriken gleichzeitig, was zu einer globalen Betrachtung der E/E-Architektur beiträgt. Der Ansatz der Simulationssynthese in Kombination mit der modellbasierten, dynamischen Architekturevaluation mittels Metriken stellt eine erhebliche Produktivitätssteigerung und Fehlervermeidung bei der iterativen, verteilten Entwicklung von Architektur und ausführbarem Verhalten dar und berücksichtigt zudem die Kollaboration an ein und demselben Modell.

In den folgenden Abschnitten werden die zur Erfüllung der beschriebenen Methodik notwendigen Anforderungen abgeleitet. Darüber hinaus werden wichtige Parameter und Metriken beschrieben, die während der Simulationssynthese Berücksichtigung finden sollen. Diese werden anhand relevanter Anwendungsfälle und Fragestellungen (vgl. Abschnitt 1.1), die in der Konzeptphase zur Evaluation von E/E-Architekturvarianten herangezogen werden können, abgeleitet.

4.3 Anforderungen

4.3.1 Modellbasierte Verhaltensspezifikation im Architekturmodell

Ausgehend von den Verbesserungspotentialen in Abschnitt 4.1 und basierend auf den diskutierten modellbasierten Architektursprachen in Abschnitt 3.1 und Abschnitt 3.2 soll eine Verhaltensspezifikation im Architekturmodell möglich sein. Die architekturzentrierte Verhaltensmodellierung hat den Vorteil, dass eine architekturweite Systemevaluierung möglich ist. Eine konsistente, formale Beschreibung zwischen Architektur- und Verhaltensmodell bezogen auf bestimmte Versionsstände ist somit ebenfalls gegeben. Zudem ermöglicht es die Verbindung und Nachvollziehbarkeit zu ebenenübergreifenden Artefakten wie Anforderungen oder ausführende Hardwarekomponenten. Dies ist eine notwendige Anforderung, um realisierungsabhängige Aspekte der Architektur wie Netzwerkkommunikation in der nachfolgenden Simulation des Verhaltens berücksichtigen zu können.

Das Verhalten soll insbesondere auf der realisierungsunabhängigen Ebene der E/E-Architektur, der logischen Architektur, möglich sein, da diese in der Regel über Jahre stabil bleibt, während sich die E/E-Architektur auf Basis der Funktionsarchitektur inkrementell ändert [153]. Um ggf. bereits bestehende logische Funktionsarchitekturen durch das Anreichern bzw. das Verfeinern von Verhalten nicht ändern zu müssen, soll das Verhalten durch Assoziationen mit den statischen Funktionsblöcken verbunden werden können, bspw. über Mapping-Mechanismen. Analoges gilt vor allem auch für gegebene Schnittstellenspezifikationen der Funktionen und Informationseinheiten, die zwischen diesen ausgetauscht werden.

Zur Erhöhung der Wiederverwendbarkeit von Funktionen und Verhaltensmodellen sollen diese nach dem Typ-Instanz Prinzip spezifiziert werden können. Darüber hinaus soll zur Steigerung der Produktivität eine Funktionsbibliothek zur Verfügung stehen, aus der ausführbare Verhaltensblöcke instanziiert werden können. Je nach Entwicklungsstand und Anwendungsfall ist sowohl Verhalten unterschiedlichen Detailgrades notwendig und es sind unterschiedliche Modellierungsparadigmen und Berechnungsmodelle mehr oder weniger für bestimmte Anwendungen geeignet. Datenflusslastige Applikationen und Berechnungen werden so eher mit Blockschaltbildern bzw. Actor-orientiert dargestellt, während zustandsorientiertes Verhalten mit FSMs oder State Charts abgebildet werden. Daher soll zum einen eine iterative Verfeinerung und Austauschbarkeit von Verhalten gewährleistet werden. Zum anderen sollen zustandsorientiertes Verhalten mittels State Charts und Actor-orientierte Modelle miteinander kombiniert wer-

den, um komplexeres Verhalten mit gängigen Modellierungsparadigmen und Berechnungsmodellen realisieren zu können.

Neben der horizontalen Interaktion von Verhaltensmodellen, die über die Schnittstellen der logischen Funktionen abgeleitet werden, sollen darüber hinaus vertikale, ebenenübergreifende Abhängigkeiten berücksichtigt werden. So lässt sich z. B. ein vom Betriebsmodus abhängiges Funktionsverhalten untersuchen, in welchem Funktionen einen Hardwarestatus anfordern können und ihr funktionales Verhalten nur ausführen, wenn die ECU betriebsbereit ist. Somit wird eine Evaluation des Verhaltens abhängig von aktiven Betriebsmodi ermöglicht. Die getrennte und unabhängige Modellierung der Funktionalität von Applikationen und der ausführenden Plattform sowie deren Einfluss auf die Applikation durch Mappings ist darüber hinaus eine leistungsfähige Methode zur Entwurfsraumexploration (engl. *Design Space Exploration (DSE)*) der Gesamtarchitektur [150]. Demnach können Entwurfsentscheidungen, wie die Wahl der Topologie der Hardwarearchitektur (z. B. eine zentrale oder verteilte Steuergeräte-Architektur), Single- oder Multicore Mikroprozessoren, Scheduling-Verfahren und die Verteilung von Funktionen auf die ausführende Hardware, signifikanten Einfluss auf das Verhalten einer Applikation haben [3]. Durch die Entwurfsraumexploration in frühen Entwicklungsphasen können diese Entwurfsentscheidungen rechtzeitig evaluiert und verglichen werden.

Ein weiteres Beispiel ist die Berücksichtigung von detaillierten elektrischen Komponenten oder physikalische Eigenschaften wie Leitungslängen. Dazu sollen erweiterte Modellierungsmöglichkeiten bereitgestellt werden, die das abstrakte Verhalten mit den realisierungsabhängigen Artefakten verknüpfen, damit diese vertikalen Abhängigkeiten bei der Synthese der Simulation berücksichtigt werden können. Die vertikale Verknüpfung soll weitestgehend durch bereits etablierte Modellierungskonzepte wie Mappings, Port-Schnittstellen etc. hergestellt werden. Zum einen, um die Entkopplung der Ebenen aufrechtzuerhalten und zum anderen, um den Mehraufwand der Modellierung zu reduzieren.

4.3.2 Integrierte Simulation

Die Simulation des spezifizierten Verhaltens soll integriert stattfinden. Der Begriff der Integration orientiert sich dabei an [41, Kap. 5]. Zum einen bedeutet Integration, dass die Simulation in derselben Entwicklungsumgebung wie das Design der E/E-Architektur stattfinden soll. Grundlage dafür ist die bereits diskutierte integrierte Verhaltensspezifikation, jedoch soll auch die Ausführung der Simulation funktional in die Laufzeitumgebung des Entwicklungswerkzeugs integriert sein. Dies ermöglicht die Visualisierung von Simulationsergebnissen wie Signalverläufe und verhindert Import/Export-Prozesse mit externen Werkzeugen, die

für eine automatisierte Evaluation von Varianten nachteilig sind. Zum anderen bedeutet integriert, dass Parameter aus E/E-Architekturartefakten automatisiert mit in die Simulation einfließen können, bspw. die Baudrate eines Bussystems, ohne dass diese Informationen über Austauschformate wie DBC einem externen Simulationswerkzeug zur Verfügung gestellt werden müssen. Integriert heißt auch, dass die Simulationsergebnisse mit Artefakten des E/E-Architekturmodells in Beziehung gesetzt werden können, sodass sie für die Analyse in weiteren Metriken (siehe Abschnitt 5.4) weiterverwendet und in Oberflächenkomponenten wie Tabellen dargestellt werden können. Dazu sollen die synthetisierten Simulationsartefakte basierend auf der UUID der ursprünglichen E/E-Architekturartefakte miteinander in Beziehung gesetzt werden.

4.3.3 Domänen- bzw. ebenenübergreifende Simulation

Der Schwerpunkt der integrierten Simulation liegt in der Synthese einer domänen- bzw. ebenenübergreifenden Simulation, die das spezifizierte Verhalten auf der realisierungsunabhängigen Ebene mit realisierungsabhängigen Informationen aus unteren Ebenen verknüpft. Dies beinhaltet die vertikale Interaktion von Verhalten und die Anreicherung um realisierungsabhängige Informationen. Dazu zählen die zugrunde liegende Steuergeräte-Topologie und die damit zusammenhängende Netzwerkkommunikation zwischen den Funktionen, Ausführungsaspekte von Funktionen aber auch Verfeinerungen aus der elektrischen und physikalischen Ebene (siehe Unterabschnitt 4.3.1).

Zur Synthese des Simulationsmodells ist daher eine Analyse des zugrunde liegenden Metamodells als komplexe Datenstruktur der E/E-Architektur und der komplementären Verhaltensspezifikation notwendig, zu denen naturgemäß Abhängigkeiten bestehen. Um eine Übertragbarkeit und einfachere Portierbarkeit auf andere Metamodelle zu ermöglichen, soll die Softwarearchitektur zur Analyse und Synthese modular und erweiterbar gestaltet sein sowie abstrakte Schnittstellen bereitstellen, die den Ablauf der Synthese kapseln.

Eine weitere wichtige Anforderung ist, dass der Zielsimulator zur integrierten Ausführung mehrere Berechnungsmodelle (siehe Unterabschnitt 2.2.2) und eine Kombination derer unterstützen soll, da heterogene Verhaltensspezifikationen unterstützt werden sollen. Aber auch aufgrund der realisierungsabhängigen Informationen der E/E-Architektur sind unterschiedliche Domänen notwendig. Eine Netzwerksimulation basiert i. d. R. auf Discrete-Event Berechnungsmodellen (siehe Unterabschnitt 2.2.2.1), während die Simulation elektrischer Komponenten und analoger Anteile auf kontinuierliche Berechnungsmodelle zurückgreifen (siehe Unterabschnitt 2.2.2.3 und Unterabschnitt 2.2.2.4).

4.3.3.1 Simulation der logischen Funktionsarchitektur

Wie in Unterabschnitt 4.3.1 bereits eingeführt, soll das Verhalten gleicher Funktionen unterschiedliche Detailgrade aufweisen können, um iterative Verfeinerungen zu erlauben. Dies kann ein abstrakter, ereignisbasierter Informationsaustausch ohne eine bestimmte Funktionalität sein, bis hin zu detailliertem Funktionsverhalten mit komplexen Berechnungen. Der Detailgrad ist sowohl abhängig vom Entwicklungsstand aber auch von der Art, was analysiert werden soll. So ist bspw. für eine reine Latenzanalyse einer funktionalen Wirkkette das detaillierte Funktionsverhalten eher weniger von Belang als die Dauer der Berechnung und der Kommunikation zwischen den Funktionen. Dies soll über die Möglichkeiten der integrierten Verhaltensmodellierung über State Charts und ausführbare Actors adressiert werden. Zur Unterstützung des Endnutzers bei der Modellierung von abstraktem Verhalten, sollen nicht-funktionale Eigenschaften wie Sendeverhalten und Timing von Funktionen werkzeuggestützt in ein entsprechendes Verhaltensmodell überführt werden können.

Nicht mehr weiter verfeinerbares Verhalten mittels der integrierten Verhaltensmodellierung soll an domänenspezifische Modelle delegiert werden können (siehe Unterabschnitt 4.3.5).

4.3.3.2 Simulation der Netzwerkkommunikation

Die Simulation der Netzwerkkommunikation ist eine Anforderung, die zwei wichtige Metriken von E/E-Architekturen und vernetzten Funktionen adressiert. Zum einen die Buslast [41, Kap. 4] und zum anderen die Kommunikationslatenz als Bestandteil einer Ende-zu-Ende Latenz von Funktionsnetzwerken [159]. Die Kommunikation kann dabei intern mehrere Bussysteme durchlaufen.

Da die Buslast-Metrik für Aussagen über die Einhaltung von Latenzen nicht geeignet ist [159], soll daher zusätzlich die Simulation der Netzwerkkommunikation berücksichtigt werden. Neben der reinen Latenz ist aber auch der Einfluss der zusätzlichen Kommunikationslatenz auf das eigentliche Funktionsverhalten ein wichtiger Aspekt, der damit frühzeitig untersucht werden kann. Die innerhalb der E/E-Architektur stattfindende Kommunikation soll dabei über die gegebene Architekturmodellierung automatisiert synthetisiert werden. Der Empfang externer Kommunikation soll über die Verhaltensmodellierung von Stimuli erfolgen.

Die Simulation der Netzwerkkommunikation bzw. deren Latenz soll dabei nicht als Garantie für die spätere Umsetzung im Fahrzeug dienen. Sondern um in frühen Entwicklungsphasen abhängig von der Funktionsverteilung Abschätzungen über das funktionale Verhalten unter Einfluss der Kommunikation und über die

Buslast- bzw. Latenzverteilungen der zugrunde liegenden Vernetzungstopologie treffen zu können.

4.3.3.3 Simulation von elektrischen Komponenten und Strombedarfe

Wie schon durch die Bezeichnung gegeben, betreffen weitere relevante Eigenschaften zur Analyse bzw. zur Bewertung die Elektrik und Elektronik. Hinsichtlich der Elektrik spielt das Bordnetz eine wichtige Rolle, das vor allem auf die Leistungsversorgung und auf das Massekonzept abzielt. Das ist insbesondere bei der in Zukunft weiter verstärkten Entwicklung von Elektrofahrzeugen von Bedeutung. Dazu zählt auch die Dimensionierung des Absicherungskonzeptes von Komponenten des Fahrzeugs. [41, Kap. 4][96, Kap. 3.5]

Zur Bewertung dieser Kriterien sind die Stromaufnahmen von Komponenten und somit Ströme auf den versorgenden Leitungen und Spannungen an Versorgungs-Pins notwendig, wodurch die Versorgung und der Strombedarf bei unterschiedlichen Betriebszuständen des Fahrzeugs bzw. von Sensoren, Steuergeräten und Aktuatoren berechnet werden können.

Statische Analyse Die Modellierung solcher *statischer* Strombedarfe ist mit existierenden Beschreibungssprachen wie der EAST-ADL (vgl. sog. *GenericConstraints* in [96]) und der EEA-ADL von PREEvision® möglich (vgl. Unterabschnitt 3.2.6.2). Die Analyse dieser Strombedarfe kann nach [96] entweder unabhängig oder abhängig von einem bestimmten Betriebszustand (engl. *mode*) sein. Diese Art der Bestimmung der Leistungsversorgung und Strombedarfe wird bspw. von PREEvision® bereits unterstützt, wobei Leitungswiderstände und Masseverbindungen vernachlässigt werden [170, Kap. 10.3]. Eine ähnliche Art der Analyse wird mit der im Bereich der Avionik eingesetzten Beschreibungssprache AADL und dem dazugehörigen Werkzeug OSATE prototypisch umgesetzt (vgl. Unterabschnitt 3.1.2).

Quasi-statische Analyse Neben den statischen Strombedarfe kann, angelehnt an drei Kategorien in [96, S. 103], zwischen *quasi-statischer* und *dynamischer Analyse* unterschieden werden. Die quasi-statische Analyse soll dabei u. a. den zeitabhängigen Stromverbrauch von Komponenten adressieren, indem die Zeitdauer eines bestimmten Betriebszustandes von Steuergeräten, Aktuatoren etc. abhängig von ihrem Funktionsverhalten zusätzlich mitberücksichtigt werden. Ein weiterer Vorteil dieser Art ist, dass Ruhe- und Nachlaufströme von Komponenten betrachtet werden können, die die Grundlage für die Berechnung von maximalen Standzeiten eines Fahrzeugs sind, bevor Abschaltszenarien ausgeführt werden

müssen [41, Kap. 4]. Die Bewertung hinsichtlich der Stromaufnahme abhängig von zustandsorientiertem Funktionsverhalten und der Zeit ist mit den statischen Modellierungs- und Analysemitteln nicht mehr realisierbar. Dazu bedarf es weiterer Simulationsmöglichkeiten des E/E-Architekturmodells basierend auf den modellierten Strombedarfen und des Verhaltens. Dies soll mithilfe der ebenenübergreifenden Verhaltensmodellierung und Simulationssynthese unter Nutzung von State Charts adressiert und mit bestehenden Attributen zur Spezifikation von Strombedarfen verknüpft werden. Die Wahl von State Charts wird hierbei aufgrund ihrer naturgemäßen Zustandsorientierung dem Actor-orientierten Verhalten bevorzugt, jedoch ist auch eine Kombination aus beiden möglich.

Dynamische Analyse Neben den Stromwerten in den einzelnen Betriebszuständen über ein Zeitintervall berücksichtigt die dynamische Analyse auch Wertänderungen innerhalb dieses Zeitintervalls. Dadurch sollen instantane Stromänderungen bzw. auch Spannungsänderungen in Form von Signalverläufen berücksichtigt werden können, um bspw. Spitzenwerte zu bestimmen oder Mittelungen von Strömen und Spannungen über die Zeit simulieren zu können. Die Signalverläufe innerhalb eines Zustands sind typischerweise abhängig vom Funktionsverhalten, die Modellierung dessen zur Berücksichtigung in der Simulation des dynamischen Stromverhaltens wird jedoch weder von der EAST-ADL unterstützt [96, S. 103] noch von PREEvision® und den weiteren Beschreibungssprachen aus Abschnitt 3.1. Daher sollen hier Modellierungsmechanismen erweitert werden, um kontinuierliche Signale auf der realisierungsunabhängigen Ebene abbilden zu können. Diese sollen außerdem die Verknüpfung zu detaillierteren Abstraktionsebenen herstellen, um die nachfolgend diskutierten und wichtigen Anwendungsfälle zu adressieren.

Der Leitungssatz und das Massekonzept sind kritische Auslegungspunkte, die in der dynamischen Analyse mitberücksichtigt werden sollen. So kann es bedingt durch die unterschiedlichen Stromaufnahmen der verteilten Komponenten in der E/E-Architektur zu signifikanten Spannungsabfällen auf den Leitungen kommen, die sich auf die Spannungsstabilität an den versorgenden Pins der Komponenten auswirkt. Eine Spannungsversorgung unter einem bestimmten Grenzwert kann zu fehlerhaften Verhalten von Funktionen oder gar ECU-Resets führen, was insbesondere bei sicherheitsrelevanten Funktionen kritisch ist [42].

Aber nicht nur der Leitungssatz kann das Funktionsverhalten beeinflussen. Ausführende Hardwarekomponenten wie Mikrocontroller in Steuergeräten und intelligenten Sensoren/Aktuatoren enthalten typischerweise Hardware-Module, welche signalaufbereitende Aufgaben für die Mikrocontroller und Aktuatoren erfüllen [35]. Aus diesem Grund sollen des Weiteren grundlegende elektrische und elektronische Komponenten wie Schalter, Widerstände, Kapazitäten, Induktivitä-

ten, Dioden etc. mit in die Simulation einfließen. Einerseits um die Auswirkungen auf das abstrakt modellierte Verhalten auf der realisierungsunabhängigen Ebene frühzeitig untersuchen zu können. Andererseits, um den Einfluss auf versorgende Pins ausführender Hardwarekomponenten wie Mikrocontroller oder Aktuatoren analysieren zu können.

4.3.4 Metrikbasierte Variantenbewertung

Bei der Bewertung von E/E-Architekturen müssen zahlreiche Varianten berücksichtigt werden, die auch die Bewertung selbst beeinflussen. Wie in Abschnitt 3.4 diskutiert, sollen dabei Berechnungen bzgl. bestimmter Metriken vorgenommen werden. Neben den dort diskutierten statischen Metriken sollen vor allem jene dynamischen Metriken bei der Bewertung von Varianten berücksichtigt werden, welche nur durch zusätzliche Simulationsmöglichkeiten analysierbar sind. Die Ergebnisse dieser dynamischen Metriken durch die Simulation sollen zudem einen integrierten Charakter aufweisen, damit diese mit weiteren Metrikberechnungen im EEA-Entwurfswerkzeug kombiniert und mit ggf. assoziierten EEA-Artefakten in Beziehung gesetzt werden können. Eine integrierte Betrachtungsweise von Varianten ist außerdem wichtig für deren automatische, effiziente Bewertung und Vergleich mithilfe der Metrikberechnungen (vgl. Unterabschnitt 3.4.3).

Hinsichtlich der Bewertung und des Vergleichs von Varianten bzgl. der dynamischen Metriken besteht eine Abhängigkeit zum umgebenden Werkzeug PREEvision®, welches für die prototypische Umsetzung eingesetzt wird. Architekturvarianten und Realisierungsalternativen werden dort wie in Unterabschnitt 3.2.4 beschrieben nach einem Produktlinienansatz und mithilfe des integrierten Variantenmanagements gebildet. Eine bestimmte Architekturvariante wird schließlich als *aktiv* ausgewählt. In diesem Modus werden automatisiert sog. Regel-Propagationen angewandt, die eine Art Filter über alle Abstraktionsebenen darstellen. Artefakte werden somit entweder als *aktiver* und *inkludierter* Bestandteil der aktuellen Variante gekennzeichnet oder nicht[2]. Somit lassen sich diese Artefakte über einfache binäre Abfragen identifizieren.

Dieser Eigenschaft soll sich die Synthese des Simulationsmodells bedienen. Einerseits, um nur die relevanten Artefakte der aktiven Variante für die Synthese zu berücksichtigen. Andererseits, um nur ein einziges variantenreiches E/E-Architekturmodell als Eingabe zu benötigen, das für die Bewertung aller zur Verfügung stehenden Varianten konstant vorgehalten werden soll.

[2] Inkludiert deshalb, weil bestimmte Artefakte zwar aktiv sein, aber trotzdem von der aktuellen Variante ausgeschlossen werden können, z. B. bei Realisierungsalternativen.

Schließlich soll die metrikbasierte Bewertung auch eine iterative Verfeinerung bzw. Anpassung erlauben, sowohl des E/E-Architekturmodells als auch der komplementären Verhaltensmodelle. Durch die möglichen iterativen Modellanpassungen sind ggf. weitere Syntheseschritte des E/E-Architekturmodells notwendig, welche jedoch durch den integrierten Charakter der Metriken durch Automatismen des umgebenden Werkzeugs PREEvision® abgefangen werden können.

4.3.5 Integration von domänenspezifischen Modellen

Bedingt durch die integrierte Verhaltensmodellierung über State Charts und ausführbare Actors, welche weitestgehend durch die Bibliothek des Zielsimulators vorgegeben ist, können diese Möglichkeiten bei komplexerem Verhalten an ihre Grenzen stoßen bzw. einen nicht mehr weiter verfeinerbaren Detailgrad aufweisen. Beispielsweise lassen sich detailliertere Modelle der Umgebung und der Fahrzeugdynamik mit den Möglichkeiten der integrierten Verhaltensmodellierung bei komplexeren Szenarien nur noch schwer abdecken. Andererseits führt die verteilte Entwicklung dazu, dass Modelle von Zulieferern oder verschiedenen Abteilungen innerhalb des OEMs in die logische Gesamtfunktionsarchitektur integriert werden müssen, ohne dabei die bestehenden Modelle und Schnittstellen darauf anpassen zu müssen. Dies ist auch einer der Haupttreiber der immer weiter verbreiteten FMI Schnittstelle (siehe Unterabschnitt 3.3.2.1).

Daher sollen für die Verfeinerung des Verhaltens Möglichkeiten zur Einbindung bzw. zur Co-Simulation von domänenspezifischen Modellen über etablierte Schnittstellen wie FMI möglich sein. Die domänenspezifischen Modelle sollen dabei mittels der integrierten Verhaltensmodellierung über ausführbare Actors einfach austauschbar sein. Konsequenterweise ist dies auch eine Anforderung an den Simulator, der diese domänenspezifischen Modelle integrieren und auch ausführen können soll.

4.3.6 Zusammenfassung der Anforderungen

Die analysierten Anforderungen können wie folgt zusammengefasst werden:

Integrierte Verhaltensspezifikation und Simulation

REQ01 Das Verhalten soll integriert und komplementär zum Architekturmodell spezifiziert werden können.

REQ02 Die Modellierung soll modular zum Architekturmodell erweiterbar sein, ohne die Schnittstellen der logischen Funktionen ändern zu müssen.

REQ03 Die Spezifikation des Verhaltens soll auf der realisierungsunabhängigen Ebene der E/E-Architektur beginnen können.

REQ04 Nicht-funktionale Eigenschaften der realisierungsunabhängigen Ebene wie Timing sollen automatisiert in ein ausführbares, abstraktes Verhalten überführt werden können.

REQ05 Die Modellierung des Verhaltens soll iterativ verfeinerbar und austauschbar sowie aus einer Funktionsbibliothek instanziierbar sein.

REQ06 Es sollen zustandsorientierte und Actor-orientierte Modellierungsparadigmen sowie deren Kombination mit verschiedenen Berechnungsmodellen unterstützt werden.

REQ07 Neben der horizontalen Interaktion von logischen Funktionen soll eine vertikale, ebenenübergreifende Interaktion zur Berücksichtigung von detaillierten elektrischen und physikalischen Komponenten sowie Zuständen von Steuergeräten ermöglicht werden.

REQ08 Die Ausführung bzw. Simulation des Verhaltens soll funktional integriert im EEA Entwicklungswerkzeug stattfinden, um Import/Export-Prozesse zu minimieren.

REQ09 Die synthetisierten Artefakte des Simulationsmodells und die Ergebnisse sollen basierend auf der UUID der ursprünglichen Artefakte des Architekturmodells miteinander in Beziehung gesetzt werden.

Domänen- und ebenenübergreifende Simulation

REQ10 Das realisierungsunabhängige Verhalten soll durch Analyse der EEA Abstraktionsebenen automatisch mit realisierungsabhängigen Informationen als domänenübergreifende Simulation angereichert werden.

REQ11 Die Softwarearchitektur zur Simulationssynthese soll modular und erweiterbar sein, um die Ergänzung von weiteren Architekturaspekten in der Simulation sowie die Übertragbarkeit und Portabilität auf andere Metamodelle zu vereinfachen.

REQ12 Es soll die Simulation von Netzwerkkommunikation zwischen Steuergeräten und weiteren, nicht-funktionalen Eigenschaften wie Ausführungsaspekte von Funktionen möglich sein sowie die vertikale Interaktion von realisierungsunabhängigem Verhalten mit Steuergeräte-Verhalten.

REQ13 Die Analyse von quasi-statischen Strombedarfe soll möglich sein.

REQ14 Die Simulation von elektrischen Komponenten innerhalb von Steuergeräten sowie die Berücksichtigung von Leitungswiderständen und die Analyse von dynamischen Strombedarfe soll möglich sein.

REQ15 Es soll ein einfach zu integrierender Simulator verwendet werden, der deterministische, heterogene Berechnungsmodelle sowie Actor-orientierte und FSM-basierte Modellierungsparadigmen unterstützt.

Metrikbasierte Variantenbewertung

REQ16 Dynamische Metriken basierend auf den Simulationsergebnissen, die transparent in weiteren Metrikberechnungen verwendet werden können, sollen analysiert werden können.

REQ17 Die Bewertung mehrerer Architekturvarianten bzgl. dynamischer Metriken soll mittels einer variantensensitiven Simulationssynthese realisiert werden.

REQ18 Mehrere vorgegebene Architekturvarianten sollen automatisch bzgl. statischer und dynamischer Metriken bewertet und verglichen werden können.

REQ19 Eine tabellarische Ausgabe oder ein Export der Bewertungsergebnisse für nachgelagerte Analysen soll möglich sein.

REQ20 Es soll eine iterative Verfeinerung bzw. Anpassung möglich sein, sowohl des E/E-Architekturmodells als auch der komplementären Verhaltensmodelle.

Integration von domänenspezifischen Modellen

REQ21 Das Verhalten soll mittels domänenspezifischer Modelle weiter verfeinert und bspw. über FMI co-simuliert werden können.

REQ22 Die domänenspezifischen Modelle sollen einfach über Actors ausgetauscht werden können.

REQ23 Der einzusetzende Simulator soll Schnittstellen zur Co-Simulation unterstützen.

5 Integrierte Simulation und dynamische Bewertung von E/E-Architekturen

In diesem Kapitel wird zunächst ein Überblick über das Gesamtkonzept zur Realisierung der integrierten Methodik aus Kapitel 4 gegeben. Dies umfasst auch die Wahl des zugrunde liegenden E/E-Architekturmodells und des Zielsimulators zur prototypischen Realisierung. Konkrete Konzepte zur integriert ausführbaren und ebenenübergreifenden Verhaltensmodellierung werden in Abschnitt 5.2 detailliert diskutiert. Details zur Analyse des zugrunde liegenden E/E-Architekturmodells und zu den Abbildungs-/Synthesevorschriften des Simulationsmodells folgen in Abschnitt 5.3. Anschließend folgt basierend auf der integrierten Simulationssynthese ein generisches Konzept für eine metrikbasierte Bewertung von Architekturvarianten bzgl. statischer und dynamischer Metriken gleichzeitig (Abschnitt 5.4). Dies beinhaltet eine konkrete Umsetzung anhand einer integrierten Simulations- und Analysemetrik zur Bewertung von Architekturvarianten bzgl. Kommunikationslatenzen von funktionalen Wirkketten und den damit verbundenen Buslasten.

Die in diesem Kapitel präsentierten Methoden und Konzepte sowie die prototypischen Realisierungen basieren auf den eigenen Publikationen [BRB17, BB18, BKB19a, BNB19, BKB19b, BKB19c].

5.1 Konzept

Das Gesamtkonzept zur Realisierung der integrierten Methodik ist in Abb. 5.1 dargestellt. Details zu den einzelnen Komponenten werden in den darauffolgenden Abschnitten erläutert. Dazu zählen die Begründung der Wahl des Zielsimulators Ptolemy II (Unterabschnitt 5.1.2) sowie die generische Softwarearchitektur zur Interpretation und ebenenübergreifenden Aggregation des E/E-Architekturmodells (Unterabschnitt 5.1.3) für die Synthese des Simulationsmodells (Unterabschnitt 5.1.4).

5.1.1 Überblick

Wie in Abschnitt 4.2 bereits eingeführt, ist die Ausgangslage für eine architekturzentrierte Modellierung und Simulation von E/E-Architekturen ein einheitliches und umfangreiches Datenmodell, das alle relevanten Abstraktionsebenen einer E/E-Architektur beschreibt. Zusammen mit einer integrierten Verhaltensmodellierung lässt sich daraus eine ausführbare Architektur synthetisieren.

Zur Modellierung der E/E-Architektur können bspw. die bereits diskutieren EAST-ADL oder PREEvision® zum Einsatz kommen. Auf der einen Seite stellt PREEvision® das umfangreichste Metamodell bereit, das alle relevanten Ebenen von den Anforderungen bis zur physikalischen Topologie in einem durchgängigen Metamodell abdeckt. Auf der anderen Seite bot PREEvision® bis zur Version 8.5 keine Möglichkeit Verhalten zu spezifizieren, während die EAST-ADL einfache Zustandsautomaten als Verhalten annotieren kann, aber nicht alle Abstraktionsebenen abdeckt. PREEvision® unterstützt ab Version 9.0 die Verhaltensmodellierung mittels State Charts [175, Kap. 8.4] und egalisiert damit die Nachteile gegenüber der EAST-ADL und den in Abschnitt 3.1 untersuchten Ansätzen.

Nichtsdestotrotz wurde bereits vor der Einführung der State Charts PREEvision® als Werkzeug zur prototypischen Umsetzung gewählt, da es das umfangreichste Metamodell bietet und Voraussetzung für eine ebenen- und domänenübergreifende Simulationssynthese ist. Allerdings unterstützen die behandelten, integrierten Verhaltensmodellierungen nur zustandsbasierte Notationen, welche nicht direkt ausführbar sind und komplexeres Verhalten nicht ausreichend adressieren (vgl. Kapitel 4). Um den in Unterabschnitt 4.3.6 spezifizierten Anforderungen zur integrierten Verhaltensmodellierung und Simulation gerecht zu werden, wird in dieser Arbeit eine neue Ebene eingeführt, die die integrierte Spezifikation von *ausführbarem* Verhalten erlaubt. Diese Ebene wird als *Behavioral Logical Architecture (BLA)* bezeichnet und ist in Abb. 5.1 unterhalb der realisierungsunabhängigen logischen Architektur (LA) angesiedelt.

Die BLA-Ebene erlaubt eine Metamodell unabhängige Modellierung von Actororientiertem Verhalten, indem die statischen, logischen Funktionen der LA-Ebene auf der BLA-Ebene durch detailliertes Verhalten aus einer Funktionsbibliothek verfeinert werden. Die Bibliothek besteht dabei aus Actors, welche nativ durch den Zielsimulator zur Verfügung gestellt werden oder eine in einen Actor gekapselte, werkzeugunabhängige FMU repräsentieren können (vgl. *PtII & FMU Actor Lib* in Abb. 5.1). Beide werden als Funktionsblocktypen importiert und werden auf der BLA-Ebene instanziiert. Zusätzlich können die Actors mit State Chart Verhalten verknüpft werden und mittels zusätzlicher Mechanismen mit State Charts auf der Hardware-Ebene interagieren (siehe Abschnitt 5.2). In Kombination mit den Mappings zur LA und detaillierteren Abstraktionsebenen wird die Verbindung

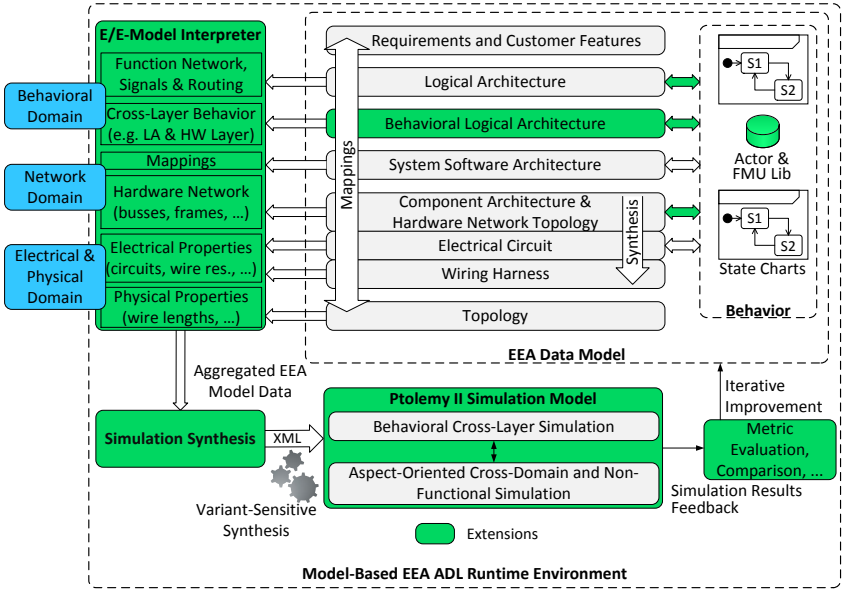


Abbildung 5.1: Konzept zur Synthese einer integrierten ebenen- und domänenübergreifenden Simulation modellbasierter E/E-Architekturen (Quelle: erweiterte Darstellung nach [BRB17, BKB19a]).

der Verhaltensblöcke zu realisierungsabhängigen Architekturinformationen hergestellt und ist eine Voraussetzung für die ebenen- und domänenübergreifende Simulation. Gleichzeitig stellen die Mappings die durchgängige Nachvollziehbarkeit der Verhaltensblöcke über alle Ebenen sicher.

Der *E/E-Model Interpreter* ist dafür zuständig, die notwendigen Informationen aus den relevanten Abstraktionsebenen zu extrahieren und diese mit den Verhaltensspezifikationen zu verknüpfen, um daraus ein ebenen- und domänenübergreifendes Simulationsmodell zu synthetisieren. Dazu zählen neben der ebenenübergreifenden Verhaltenssimulation, z. B. auf der LA- und Hardware-Ebene, die Netzwerkkommunikation zwischen den Funktionen in Abhängigkeit ihrer Abbildung auf ausführende ECUs (Netzwerkdomäne). Aber auch elektrische und physikalische Eigenschaften, die bspw. die quasi-statische und dynamische Analyse der Strombedarfe erlauben (elektrische und physikalische Domäne).

Die *Simulation Synthesis* sorgt zum einen für die Abbildung des zugrunde liegenden Metamodells der E/E-Architektur auf das Metamodell des Zielsimulators.

Zum anderen werden die aggregierten Informationen des Interpreters für die eigentliche Synthese des Simulationsmodells verwendet und dient somit als Back-End.

Das synthetisierte Simulationsmodell wird funktional integriert im E/E-Architekturwerkzeug ausgeführt und besteht prinzipiell aus zwei Teilen:

1. Der ebenenübergreifenden Verhaltenssimulation und
2. der aspektorientierten Simulation [3] der nicht-funktionalen Architekturanteile wie die Netzwerkkommunikation in Verbindung mit dem spezifizierten Verhalten.

Die aspektorientierte Simulation wird dabei automatisch synthetisiert und verfeinert das spezifizierte Verhalten auf Basis der interpretierten Architekturinformationen, ohne dass diese durch Austauschformate wie DBC einem externen Simulationswerkzeug zur Verfügung gestellt werden müssen. Import/Export-Prozesse sind wie in Unterabschnitt 4.3.2 diskutiert anfällig bzgl. Modellinkonsistenzen und Werkzeugbrüchen sowie nachteilig für eine automatisierte Variantenbewertung.

Der Begriff der Aspektorientierung stammt ursprünglich aus dem Software Engineering Bereich mit dem Ziel, die Komplexität von funktionalem Code durch nachträgliche Integration von Logging-Aspekten zu reduzieren und Fehler im funktionalen Code durch die Anpassung zu vermeiden. Dies wird durch die automatische Verwebung des funktionalen Codes mit dem Logging-Code während der Kompilierung erreicht, wodurch eine modulare und getrennte Entwicklung erreicht wird [3].

Bei der aspektorientierten Simulation wird dieses Konzept auf *ausführbare* Modelle bzw. im Speziellen auf Actor-orientierte Modelle übertragen und basiert auf sog. *Quantity Manager*, welche in [30] eingeführt wurden. Es wird dabei eine Trennung zwischen dem funktionalen Verhaltensmodell und nicht-funktionalen, aber dennoch wichtigen Eigenschaften, die Einfluss auf das zu untersuchende Verhalten haben können, vorgenommen. Das heißt die Struktur und die Schnittstellen des zu untersuchenden Verhaltensmodells müssen durch die dynamische Verwebung mit diesen nicht-funktionalen Teilmodellen zur Laufzeit nicht verändert werden. Die Modelle können dadurch modular in der jeweiligen Domäne von Experten entwickelt und mit geringem Aufwand in das funktionale Verhalten eingekoppelt werden. Analoges kann hier auf die Entwicklung von E/E-Architekturen übertragen werden, in der Experten auf ihrer jeweiligen Abstraktionsebene die Modelle entwerfen, welche letztendlich durch die in dieser Arbeit erarbeiteten, ebenenübergreifenden Mechanismen und der Simulationssynthese miteinander verwoben werden.

Die Teilmodelle der nicht-funktionalen Aspekte fungieren dabei als eine Art Mediator, welche an das ursprüngliche Kernmodell annotiert werden und eine Verfeinerung des Systemverhaltens darstellen, z. B. die Netzwerkkommunikation zwischen zwei logischen Funktionen. Aspekte können an Actors oder an deren Ports annotiert werden. Die aspektorientierte Modellierung hat nicht nur bei der Modellkonstruktion bzw. bei der Synthese den Vorteil, dass abhängig von der Konfiguration des Interpreters die Aspekte modular hinzu synthetisiert werden. Sondern auch die Aspekte an sich können je nach verfügbarem Detailgrad beliebig in ihrem Verhalten verfeinert werden.

Durch die integrierte Ausführung der Simulation im EEA-Entwicklungswerkzeug wird schließlich die Visualisierung der Simulationsergebnisse sowie die Weiterverarbeitung in Analysemetriken realisiert. Die Weiterverarbeitung der Simulationsdaten ist eine Voraussetzung für die integrierte Variantenbewertung sowie für mögliche iterative Modellverbesserungen.

5.1.2 Zielsimulator

Der Zielsimulator für die prototypische Umsetzung soll zum einen die in Unterabschnitt 4.3.6 spezifizierten Anforderungen erfüllen als auch die beschriebenen Konzepte in Unterabschnitt 5.1.1 hinsichtlich der Simulation unterstützen. In dieser Arbeit wurde das in Abschnitt 2.3 ausführlich vorgestellte Modellierungs- und Simulationswerkzeug Ptolemy II gewählt. Die Wahl wird anhand der betroffenen Anforderungen aus Unterabschnitt 4.3.6 nachfolgend begründet.

- **REQ05**: Die Modellierung des Verhaltens soll iterativ verfeinerbar und austauschbar sowie aus einer Funktionsbibliothek instanziierbar sein.
  - Eine hierarchische Modellierung und Kapselung von Verhalten werden durch zusammengesetzte Actors unterstützt.
  - Eine quelloffene und einfach erweiterbare Actor-Bibliothek steht zur Wiederverwendung zur Verfügung.
- **REQ06**: Es sollen zustandsorientierte und Actor-orientierte Modellierungsparadigmen sowie deren Kombination mit verschiedenen Berechnungsmodellen unterstützt werden.
  - PtII unterstützt sowohl die Actor-orientierte und zustandsorientierte Modellierung via Modal Models als auch deren Kombination mit verschiedenen Berechnungsmodellen (siehe Abschnitt 2.3).
- **REQ08**: Die Ausführung bzw. Simulation des Verhaltens soll funktional integriert im EEA Entwicklungswerkzeug stattfinden.

  - PtII ist ein quelloffenes Werkzeug mit gut dokumentierten Schnittstellen und Quellcode, das vollständig in Java implementiert ist. Da PREEvision® ebenfalls vollständig in Java und auf eclipse Basis implementiert ist, lässt sich PtII vergleichsweise einfach in die PREEvision® Umgebung als zusätzliches Plugin einbinden. Auf Datenstrukturen und Klassen lässt sich somit direkt zugreifen, ohne ggf. verlustbehaftete Konvertierungen oder Wrapper-Klassen für andere Sprachen zur Verfügung stellen zu müssen.
  - Zudem stehen sämtliche Features von eclipse beiden Werkzeugen zur Verfügung, wie bspw. der o. g. Plugin Mechanismus, die *Standard Widget Toolkit (SWT)*[1]-basierte Integration der graphischen Oberfläche oder das serviceorientierte *Open Service Gateway initiative (OSGi)* [130] Framework. Letzteres bietet eine dynamische Integration und Verwaltung von Plugins sowie deren Kommunikation über eine *Service Platform* [193].
  - Da bspw. auch für die EAST-ADL eclipse-Implementierungen in Java existieren (*EAST-ADL Tool Platform (EATOP)*) [1, S. 27 ff.], kann PtII auch dort zum Einsatz kommen und ist somit nicht unmittelbar auf PREEvision® beschränkt.

- **REQ12**: Es soll die Simulation von Netzwerkkommunikation zwischen Steuergeräten und weiteren, nicht-funktionalen Eigenschaften möglich sein sowie die vertikale Interaktion von realisierungsunabhängigem Verhalten mit Steuergeräte-Verhalten.
  - Zur Simulation von orthogonalen und nicht-funktionalen Eigenschaften unterstützt PtII die aspektorientierte Simulation und dazugehörige Actors, z. B. einen CAN-Bus. Sogenannte *Execution Aspects* adressieren die Simulation von Ausführungsaspekten von Funktionen [3, 136].
  - Die vertikale Interaktion zwischen realisierungsunabhängigem Verhalten mit Verhalten von Hardwarekomponenten lässt sich sowohl aspekt- als auch Actor-orientiert lösen.

- **REQ13**: Die Analyse von quasi-statischen Strombedarfe soll möglich sein.
  - Die quasi-statische Analyse ist aufgrund der Simulation von zeitbehafteten Modal Models in Kombination mit der vertikalen Interaktion mit Steuergeräten möglich.

- **REQ14**: Die Simulation von elektrischen Komponenten innerhalb von Steuergeräten sowie die Berücksichtigung von Leitungswiderständen und die Analyse von dynamischen Strombedarfe soll möglich sein

[1] https://www.eclipse.org/swt/, zuletzt aufgerufen am 21.12.2018.

  - Die Simulation von kontinuierlichen Signalen wird sowohl durch die CT- und die QSS-Domäne adressiert. Der Fokus von PtII auf die hierarchische und heterogene Kombination von MoCs unterstützt dabei die deterministische Simulation mit anderen Domänen.
  - Darüber hinaus kann der Einfluss des elektrischen Verhaltens auf das funktionale Verhalten wiederum durch einen modular erweiterbaren Aspekt realisiert werden.

- **REQ22** & **REQ23**: Die domänenspezifischen Modelle sollen einfach über Actors ausgetauscht werden können. Der einzusetzende Simulator soll die Schnittstellen zur Co-Simulation unterstützen.
  - Es werden sowohl FMI als auch HLA durch dedizierte Actors unterstützt [87, 78]. Eine weitere HLA Schnittstelle wurde darüber hinaus in einer Vorarbeit entwickelt [RBB+14].
  - Zusätzlich bietet der Einsatz von PtII als heterogenes, formales Framework den Vorteil eines zentralen Co-Simulationskoordinators zur deterministischen Ausführung von heterogenen Modellen auf Basis seines deterministischen Zeitmodells (vgl. [RBB+14] sowie Unterabschnitt 2.3.2.3 und 3.3.2.1).

5.1.3 E/E-Modell Interpreter

Der *E/E-Model Interpreter* in Abb. 5.1 ist dafür zuständig, die notwendigen Informationen aus den relevanten Abstraktionsebenen zu extrahieren und diese mit den Verhaltensspezifikationen zu verknüpfen. Der Interpreter fungiert somit als eine Art Front-End, der das zugrunde liegende Architekturmodell und Verhalten analysiert. Das Klassendiagramm in Abb. 5.2 zeigt die Struktur des Interpreters mit den wichtigsten Klassen und deren Beziehungen.

Prinzipiell besteht der Interpreter aus zwei Teilen, der `IModelDirector` und der `IModelInterpreter` Schnittstelle. Die `IModelDirector` Schnittstelle dient dabei zur Entkopplung der Interpretation des Architekturmodells und der Konstruktion des Zielsimulationsmodells und stellt sämtliche zur Synthese notwendigen Schnittstellen dem Client zur Verfügung. Zur Abstrahierung von den zugrunde liegenden Metamodellen des Architektur- und Simulationsmodells werden zwei Klassenparameter `SRC_ARTIFACT_T` und `DST_ARTIFACT_T` eingeführt, die in den implementierenden Klassen durch die abstrakten Basisklassen der zugrunde liegenden Metamodelle aufgelöst werden. Auf die spezifischen Methoden zur Konstruktion des Simulationsmodells wird in Unterabschnitt 5.1.4 genauer eingegangen. Wichtig auf `AbstractModelDirector` Seite ist an dieser Stelle die statische Datenstruktur `eeaToTargetArtifact`, die die Abbildung der Architektur-

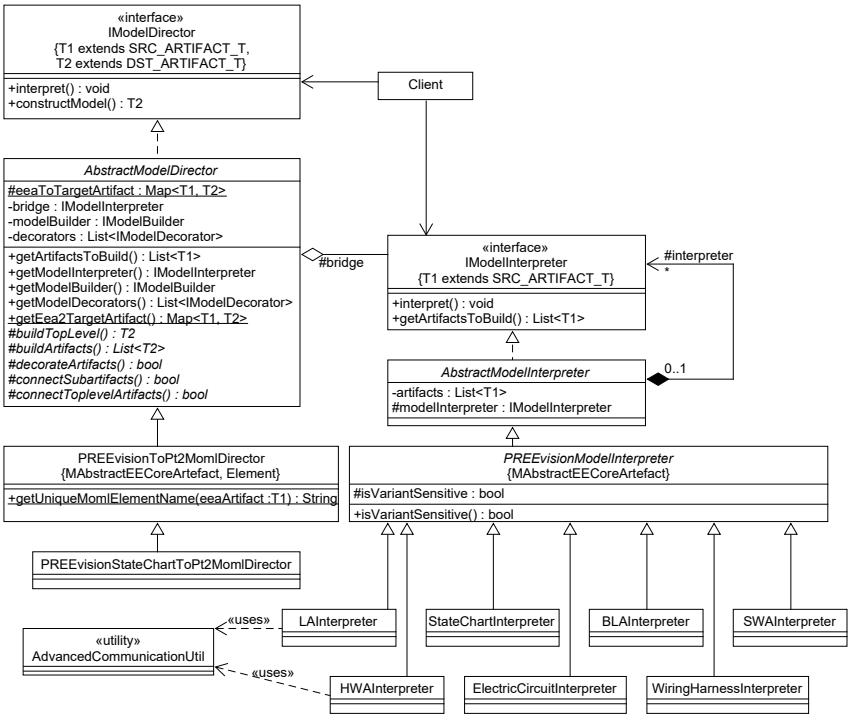


Abbildung 5.2: Klassendiagramm zur Interpretation des zugrunde liegenden E/E-Architekturmodells und Schnittstelle zur Entkopplung der Transformation in das Ziel-Simulationsmodell.

auf Simulationsmodellartefakte repräsentiert. In der konkreten für die Transformation und Konstruktion zuständige Klasse `PREEvisionToPt2MomlDirector` wird diese verwendet, um eine auf der UUID des ursprünglichen Architekturartefakts basierende eindeutige Bezeichnung des transformierten PtII Artefakts zu erzeugen.

Über das Bridge-Entwurfsmuster wird die Entkopplung zum Interpreter realisiert, in welchem die abstrakte Klasse `AbstractModelDirector` die Implementierung der `interpret()` Methode an einen austauschbaren `IModelInterpreter` delegiert wird, welcher die spezifische Implementierung der Interpretation realisiert. Analoges gilt für die `getArtifactsToBuild()` Methode, welche die vom Client übergebene Artefakte der logischen Architektur zurückliefert. Für jede

Abstraktionsebene der E/E-Architektur existiert eine separate, konkrete Interpreter Klasse, die im Falle von PREEvision® von einer abstrakten Oberklasse erben und u. a. die abstrakte Basisklasse des Klassenparameters auflöst. Zusätzlich werden an dieser Stelle die analysierten Architekturartefakte bei aktiviertem Flag `isVariantSensitive` nach der aktiven Variante gefiltert, damit eine variantensensitive Simulation realisiert werden kann.

Die Interpreter können mithilfe des Dekorierer-Entwurfsmuster kombiniert und geschachtelt werden, über welche die Interpretation des E/E-Architekturmodells gesteuert wird. So kann bspw. die Verfeinerung des Verhaltens durch elektrische Simulation mit einem fehlenden `ElectricCircuitInterpreter` einfach ausgelassen werden. Als Wurzel-Interpreter wird der `LAInterpreter` herangezogen, der die realisierungsunabhängige logische Architektur als Ausgangspunkt interpretiert und alle weiteren Aspekte über die jeweilig verschachtelt instanziierten Interpreter berücksichtigt.

5.1.4 Simulationssynthese

Die Simulationssynthese in Abb. 5.1 dient als Back-End zur Transformation und Konstruktion des Ziel-Simulationsmodells auf Basis der aggregierten Informationen aus dem Interpreter. Wie schon beim Interpreter dient der `IModelDirector` zur Entkopplung der beiden Prozesse Interpretation und Konstruktion und stellt dem Client die nötigen Schnittstellen zur Verfügung. Die Konstruktion besteht wiederum aus zwei Teilen, die zum einen das spezifizierte Actor-orientierte Verhalten und State Charts auf den entsprechenden Ebenen realisiert. Zum anderen werden die domänenspezifischen Verhaltensverfeinerungen (bspw. die elektrische Domäne) und andere nicht-funktionalen Eigenschaften aspektorientiert generiert.

5.1.4.1 Klassendiagramm

Die zweiteilige Synthese spiegelt sich in der Softwarearchitektur wider, ein Ausschnitt des Klassendiagramms ist in Abb. 5.3 gezeigt.

Die Klasse `AbstractModelDirector` definiert dazu zunächst abstrakte Methoden, die in der konkreten Klasse, die für die Transformation des Quellmodells in das Zielmodell zuständig ist, implementiert werden müssen. Die abstrakten Methoden erfüllen folgende Aufgaben, wobei die Parameter der Übersichtlichkeit halber nicht in Abb. 5.3 gezeigt sind:

- `buildTopLevel()`: Erzeugt den Container auf oberster Hierarchieebene, der das vollständige Simulationsmodell enthält.

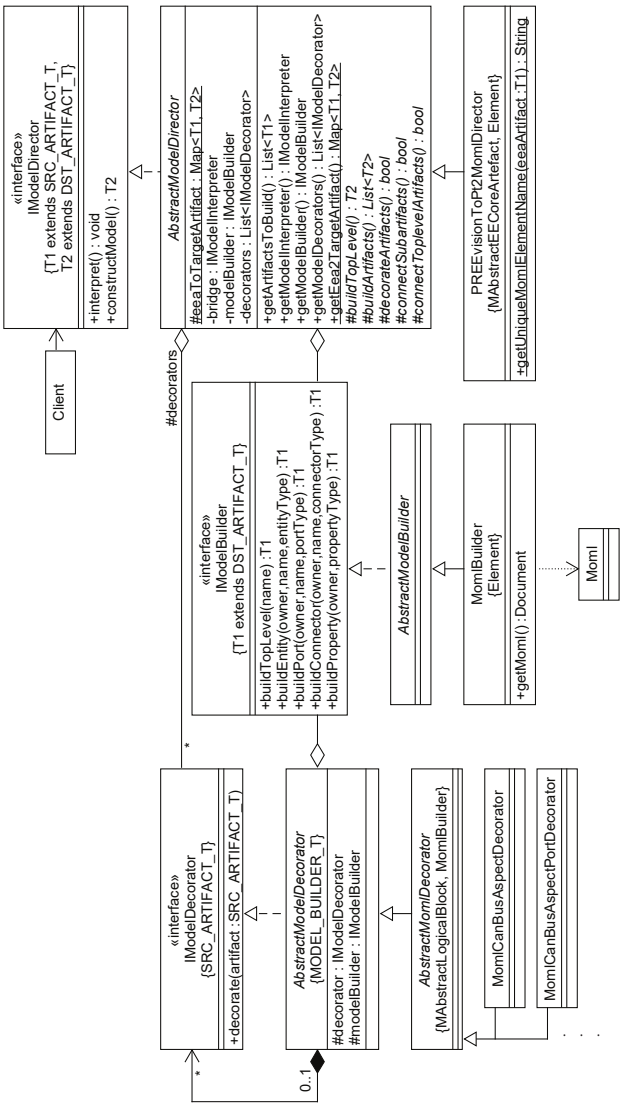


Abbildung 5.3: Klassendiagramm zur Konstruktion des Ziel-Simulationsmodells. Der Ausschnitt an konkreten Klassen realisiert dabei die Transformation und Konstruktion aus dem EEA-Modell (PREEvision®) in das Simulationsmodell (PtII).

- `buildArtifacts()`: Erzeugt das spezifizierte, ebenenübergreifende Verhalten in Form von hierarchischen Verhaltensblöcken auf der BLA-Ebene, die das modellierte Actor-Netzwerk bzw. State Chart Artefakte des Zielsimulators enthalten (vgl. Abschnitt 5.2).
- `connectSubartifacts()`: Verknüpft die in den hierarchischen Blöcken enthaltenen Actors bzw. State Chart Artefakte.
- `connectToplevelArtifacts()`: Verknüpft die generierten, hierarchischen Blöcke auf oberster Ebene.
- `decorateArtifacts()`: Dekoriert das spezifizierte Verhalten mit den domänenübergreifenden bzw. nicht-funktionalen Architekturaspekten.

Die Implementierung der eigentlichen Konstruktion Simulationsmodells aus der Verhaltensspezifikation ist dabei über das Builder-Entwurfsmuster zum `AbstractModelDirector` hin über die Schnittstelle `IModelBuilder` entkoppelt. Essentielle Methoden zur Konstruktion eines Actor-orientierten Modells sind dabei in der Schnittstelle definiert, bspw. zur Konstruktion eines Blocks eines bestimmten Typs in einem entsprechenden Container bzw. *Owner* mit der Methode `buildEntity()`. Der Name enthält dabei den eindeutigen, auf der UUID des ursprünglichen Architekturartefakts basierenden, eindeutigen Namen. Dieser wird vom konkreten `IModelDirector` übergeben, im vorliegenden Falle über die statisch zugreifbare Methode `getUniqueMomlElementName()`. Der Container und die Typen sind über die Parameter `owner` bzw. endend mit `Type` in den Methoden signalisiert, die je eine bestimmte Metaklasse im Zielmodell besitzen, abgeleitet von der abstrakten Basisklasse `DST_ARTIFACT_T`.

Alternativ ist bei der Konstruktion eines serialisierten Modells die Angabe des Typs auch über Strings möglich. Im Falle der konkreten Klasse `MomlBuilder` zur Erzeugung des PtII Modells, wird letzteres über die Methode `getMoml()` als XML-Dokument zurückgeliefert. Methoden, die spezielle Funktionalitäten oder Abhängigkeiten zum Zielsimulator enthalten, sind in der entsprechenden konkreten Builder-Klasse zu ergänzen.

Ähnlich zur Generierung der Verhaltensspezifikation ist die genaue Implementierung zur Konstruktion der domänenübergreifenden bzw. der nicht-funktionalen Architekturaspekte über ein oder mehrere `IModelDecorator` entkoppelt. Diese Dekorierer sind hauptsächlich zur Abbildung der aspektorientierten Anteile des Simulationsmodells vorgesehen. Jedoch sind sie auch dazu geeignet, instrumentierende Simulationsartefakte automatisch zu generieren. Die Dekorierer greifen über die zur Verfügung gestellten Artefakte der realisierungsunabhängigen logischen Architektur vom generischen Typ `SRC_ARTIFACT_T` auf die aggregierten, realisierungsabhängigen Informationen der konkreten Interpreter-Klassen zurück (nicht dargestellt in Abb. 5.3).

Ein abstrakter Dekorierer `AbstractModelDecorator` kann dabei ähnlich zu den Interpretern über das Dekorierer-Entwurfsmuster mehrere Modell-Dekorierer verschachtelt kombinieren. Dies wird z. B. bei der Generierung der Netzwerkkommunikation für das PtII-Zielmodell ausgenutzt, um einerseits den Aspekt selbst zu generieren und andererseits die betroffenen Ports mit busspezifischen Eigenschaften wie Framegröße zu dekorieren. Des Weiteren besitzt ein abstrakter Dekorierer einen Klassenparameter, der den Typ des dazugehörigen Modell-Builders angibt und in den konkreten Klassen aufgelöst wird, da nur das Dekorieren desselben Ziel-Simulationsmodells erlaubt ist. Die konkrete Implementierung der Konstruktion der domänenübergreifenden und nicht-funktionalen Architekturanteile ist dabei wiederum über das Builder-Entwurfsmuster entkoppelt.

5.1.4.2 Syntheseablauf

Das Flussdiagramm zur Simulationssynthese ist in Abb. 5.4 dargestellt. Ausgangspunkt ist das zu interpretierende E/E-Architekturmodell auf Basis der logischen Architektur, das wie beschrieben über die konkreten Interpreter-Klassen analysiert wird. Im Anschluss wird in einem rekursiven Prozess jeder Verhaltensblock der BLA-Ebene abgearbeitet. Abhängig davon, ob es sich dabei um eine Actor-orientierte oder State Chart Verhaltensbeschreibung handelt, werden entweder die darin befindlichen Actors rekursiv generiert und verknüpft. Andernfalls werden basierend auf den State Chart Beschreibungen die in das Zielmodell transformierten State Chart Artefakte generiert und verknüpft. Im Falle von hierarchischen Verhaltensblöcken wird zusätzlich überprüft, ob diese eine spezielle Domäne bzw. einen Director (vgl. Abschnitt 2.3) spezifiziert haben, welcher generiert werden muss.

Nachdem alle Verhaltensblöcke auf oberster Ebene generiert wurden, folgt die Dekorierung jener mit den domänenübergreifenden und nicht-funktionalen Architekturaspekten sowie mit instrumentierenden Artefakten. Wichtig anzumerken ist an dieser Stelle, dass die Dekorierung i. d. R. nur für die Verhaltensblöcke auf oberster BLA-Ebene erfolgen muss, da die entsprechenden Aspekte wie Netzwerkkommunikation auf Basis der LA Artefakte abgeleitet werden und aufgrund den eingeführten Mappings für die dazugehörigen Verhaltensblöcke valide sind (vgl. Unterabschnitt 5.2.1). Die Dekorierung von instrumentierenden Actors kann u. U. auch innerhalb von hierarchischen Verhaltensblöcken stattfinden, um bspw. Messwerte innerhalb des modellierten Actor-Netzwerks abzugreifen, die nicht direkt mit den Ports nach außen verbunden sind (vgl. Unterabschnitt 5.4.3.2).

Schließlich werden die generierten Gegenstücke der Verhaltensblöcke im Simulationsmodell auf oberster Ebene miteinander verbunden. Abhängig vom zugrunde liegenden Berechnungsmodell der Simulation kann es bedingt durch die au-

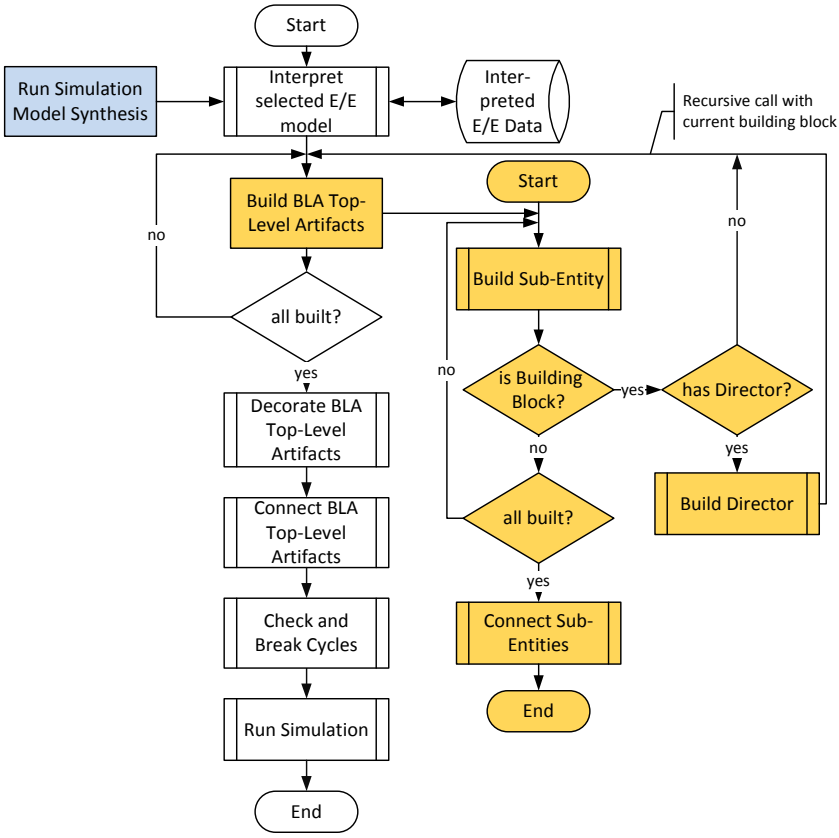


Abbildung 5.4: Flussdiagramm der Simulationssynthese (Quelle: erweiterte Darstellung nach [BRB17]).

tomatische Verknüpfung notwendig sein, evtl. auftretende Zyklen im Modell aufzubrechen, um ein valides und ausführbares Modell zu erhalten. Dies ist bspw. bei der DE Domäne von PtII der Fall, bei der eine direkte Rückkopplung zwischen Actors zu einer Zero-Delay Schleife führt (siehe Unterabschnitt 2.3.3.1). An dieser Stelle wird das generierte Modell nochmals analysiert und die Schleifen durch automatisches Einfügen von verzögernden Actors in die Rückkopplung aufgebrochen. Im Falle von PtII und einem DE Director werden hier `Microstep` Actors verwendet, welche eine Verzögerung bzgl. n im Zeitmodell der Superdense Time (τ, n, m) bewirken, d. h. die Modellzeit schreitet nicht voran.

5.2 Ausführbare Verhaltensmodellierung

5.2.1 Behavioral Logical Architecture

Die Behavioral Logical Architecture dient dazu, die statischen Funktionsblöcke der LA mit detailliertem, ausführbarem Verhalten zu verfeinern und somit ein zunächst von der zugrunde liegenden E/E-Architektur realisierungsunabhängiges Verhalten zu spezifizieren.

Zur Spezifikation des Verhaltens werden ausführbare Actors aus einer dazugehörigen Funktionsbibliothek verwendet, die die Actors des Zielsimulators als Funktionsblocktypen hinterlegt. Diese Typen können durch eine entsprechende Import-Funktion, die die Metamodell-Transformation zwischen der EEA ADL und Zielsimulator vornimmt, automatisiert als zusätzliche Bibliothek abgelegt werden (siehe Unterabschnitt 5.3.2.3). Die Bibliothek kann sowohl reguläre Actors, aber auch werkzeugunabhängige FMU-Actors enthalten. Letztere stellen die in der FMU-Modellbeschreibung (vgl. Unterabschnitt 3.3.2.1) spezifizierten Schnittstellen als typisierte Ein- und Ausgangs-Ports bereit.

Grundidee der Verhaltensspezifikation ist, dass jede *atomare* logische Funktion der statischen LA-Ebene durch einen zusammengesetzten, *hierarchischen* Block auf der BLA-Ebene verfeinert wird (Abb. 5.5). Innerhalb dieses BLA Building Blocks wird das tatsächliche Verhalten durch Instanziierung von Actors aus der Bibliothek und deren Vernetzung modelliert. Durch die Entkopplung der beiden Ebenen ist es notwendig, diese über zusätzliche Mechanismen miteinander in Beziehung zu setzen. Darauf basierend wird in der späteren Synthese identifiziert, welches Verhalten die logischen Funktionen hinterlegt haben und wie das Funktionsnetzwerk strukturiert ist. Dazu werden zwei zusätzliche Typen von Mappings zwischen LA und BLA eingeführt:

1. Block *Instanz* Mappings

2. *Port-Prototyp* Mappings

Die zusätzlichen Mappings sind in Abb. 5.5 illustriert. Im Grunde repräsentiert jede atomare logische Funktion die Grenze zur BLA, die über ein *Block-Mapping* mit dem im hierarchischen logischen Block der BLA hinterlegten Verhalten verknüpft wird. Daher wird das Mapping auch auf den hierarchischen Block auf oberster Ebene durchgeführt. Wichtig ist hier das Mapping zwischen einer *Instanz* einer atomaren LA Funktion und eines hierarchischen BLA Blocks. Dies ermöglicht den Austausch von Realisierungsalternativen des hinterlegten Verhaltens über die einfache Änderung des Mappings auf einen anderen Building Block vom selben Typ. Mappings von hierarchischen Blöcken auf LA-Ebene werden nicht erlaubt, sondern dienen zur strukturellen Hierarchisierung und Dekomposition der gesamten logischen Funktionsarchitektur. Analog dazu kann das Verhalten innerhalb eines BLA Building Blocks auf oberster Hierarchieebene durch weitere hierarchische Blöcke abstrahiert werden, die allerdings nicht über Mappings referenziert werden dürfen.

Im Gegensatz dazu werden die Mappings der Ports der Funktionen auf *Prototypen*-Basis durchgeführt. Funktionen und Ports werden in den diskutierten Architekturbeschreibungssprachen in Anlehnung an AUTOSAR typisiert instanziiert und Prototypen genannt. Port-Prototypen werden dabei ausschließlich in einer 1-zu-1 Beziehung aufeinander abgebildet. Dies bringt mehrere Vorteile mit sich:

- Die Konsistenz der Schnittstellen zwischen einer atomaren logischen Funktion und des hinterlegten Verhaltens im Building Block wird sichergestellt.
- Die Port-Prototyp-Mappings müssen nur genau einmal durchgeführt werden, solange sich die Prototypen nicht ändern.
- Die Verbindung zwischen den BLA Building Block Instanzen auf oberster Ebene können während der Synthese auf Basis der Port-Prototypen-Mappings und der Vernetzung der logischen Funktionen auf LA-Ebenen automatisch miteinander verknüpft werden. Verknüpfungen auf BLA-Ebene müssen so nicht dupliziert werden.

Die Kombination aus den beiden Mapping-Typen trägt stark zur Erhöhung der Wiederverwendung bei, da bspw. mehrere logische Funktionen vom selben Typ in mehreren LA Subsystemen instanziiert werden können, die unterschiedliche Realisierungsalternativen des hinterlegten Verhaltens aufweisen können. Die dazugehörigen BLA Building Blocks müssen jedoch nicht mehrmals instanziiert und auch nicht redundant zur LA verknüpft werden, weil dies automatisiert während der Synthese geschieht.

Das beschriebene Block-Mapping geht von einer 1-zu-1 Beziehung aus. Alternativ dazu ist es auch möglich, dass mehrere atomare logische Funktionen auf genau

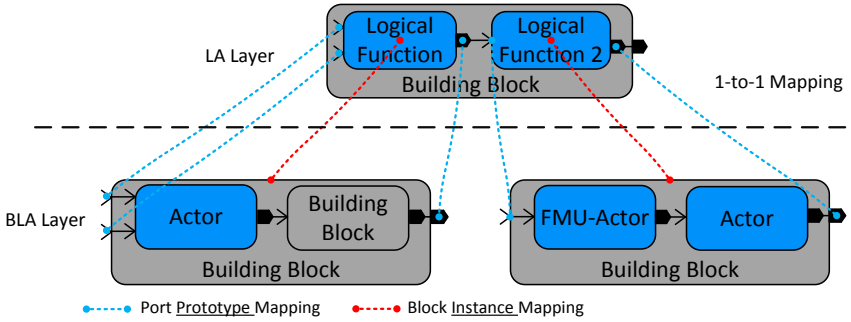


Abbildung 5.5: Prinzip der Verhaltensmodellierung auf BLA-Ebene und die Verknüpfung zur statischen LA-Ebene über 1-zu-1 Mappings (Quelle: modifizierte Darstellung nach [BRB17]).

einen BLA Building Block abgebildet werden, der dann die Summe der Port-Prototypen enthalten muss. Diese Methode ist in Abb. 5.6 dargestellt und wird als n-zu-1 Mapping bezeichnet.

Allerdings kann dieses Vorgehen von Nachteil sein und die Wiederverwendung einschränken, da für eine valide Verhaltenssynthese immer alle n logischen Funktionen als Verbund vorhanden sein müssen, was u. U. nicht immer der Fall ist. Zum Beispiel, wenn eine dieser Funktionen in einer aktiven Architekturvariante nicht berücksichtigt wird. Die bevorzugte Methode ist daher, die logischen Funktionen und das dazugehörige Verhalten atomar zu halten und eine 1-zu-1 Abbildung durchzuführen, um diese Nachteile zu vermeiden.

Durch den Schichtenansatz in Kombination mit den Mappings zur Verknüpfung der beiden Ebenen, kann die Verhaltensspezifikation als unabhängig vom zugrunde liegenden Metamodell des E/E-Architektur angesehen werden. Voraussetzung ist lediglich, dass das EEA-Metamodell eine Hierarchisierung von Komponenten auf der logischen Ebene erlaubt, um atomare logische Funktionen innerhalb eines hierarchischen Blocks auf der BLA-Ebene verfeinern zu können. Hierarchisierung zur Beherrschung von Komplexität (*teile und herrsche*) ist eine verbreitete Methode und wird von etablierten Architekturbeschreibungssprachen wie den diskutierten EAST-ADL, AADL und auch der EEA-ADL in PREEvision® unterstützt. Ein zusätzlicher Vorteil ist, dass bestehende logische Architekturen durch ggf. nachträgliche Verhaltensspezifikationen nicht geändert werden müssen, sondern durch die Mappings entkoppelt sind. Dies trägt stark zur Wiederverwendung und inkrementellen Entwicklung von logischen Architekturen bei, die typischerweise über Jahre stabil bleiben [153].

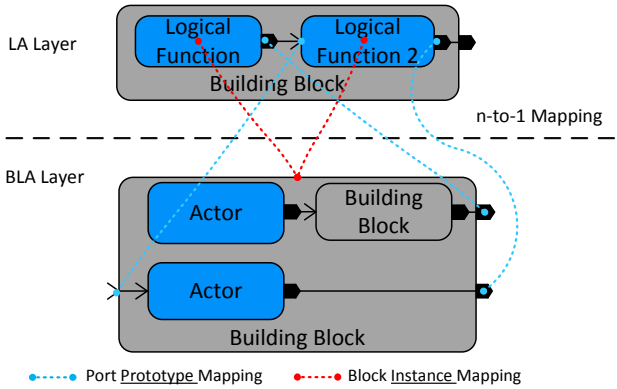


Abbildung 5.6: Prinzip der Verknüpfung zur statischen LA-Ebene über n-zu-1 Block-Mappings (Quelle: modifizierte Darstellung nach [BRB17]).

Analog dazu ermöglicht der Schichtenansatz die Realisierung von unterschiedlichen Zielsimulatoren, die eine Actor-orientierte Modellierung anbieten, wie bspw. MATLAB/Simulink. Notwendig dazu ist der Import der entsprechenden Bibliothek und ein passendes Back-End zur Synthese.

5.2.2 State Charts

Um die Modellierung von komplexen reaktiven Systemen zu adressieren, soll die Actor-orientierte Beschreibung um zustandsorientiertes Verhalten mittels State Charts erweitert bzw. kombiniert werden (vgl. Unterabschnitt 4.3.1). Dabei wird in dieser Arbeit die in PREEvision® v9.0 eingeführte Teilmenge von UML-konformen State Charts ausgenutzt. Hauptgrund für die Wahl sind die standardisierte UML Notation und die Modellierung der Schnittstellen auf Basis von AUTOSAR (siehe Unterabschnitt 5.2.2.1), das die Übertragbarkeit auf andere Architekturbeschreibungssprachen fördern soll, insb. in Kombination mit der im letzten Abschnitt vorgestellten Behavioral Logical Architecture und den später diskutierten ebenenübergreifenden Mechanismen.

5.2.2.1 Prinzip der State Chart Modellierung in PREEvision®

State Charts können an Architekturartefakte auf unterschiedlichen Abstraktionsebenen annotiert werden, insb. der logischen Architektur. Aber auch Kom-

ponenten der Hardware-Ebene wie ECUs oder Prozessoren können mit State Charts verfeinert werden. Dabei wird ein State Chart einem Architekturartefakt als Kindartefakt hinzugefügt. Abhängig von der Abstraktionsebene gestalten sich die Schnittstellen zwischen Architekturartefakt und annotiertem State Chart anders und weisen unterschiedliche Daten-Provider und -Konsumenten auf.

Wie in Unterabschnitt 3.2.6.1 beschrieben wird die Kommunikation zwischen logischen Funktionen mittels typisierten Ports modelliert, welche eine dedizierte Schnittstelle zugewiesen bekommen. Die Schnittstelle spezifiziert dabei die tatsächliche Information, die ausgetauscht wird, z. B. Datenelemente bei Sender-Empfänger-Schnittstellen oder Operationsaufrufe bei Client-Server-Schnittstellen. Diese Art der Modellierung folgt dem AUTOSAR Standard und gilt ebenso für die Softwarearchitektur. Die spezifizierten Datenelemente der individuellen Schnittstellen bzw. Ports bilden sodann die Schnittstelle für das hinterlegte State Chart, indem die Datenelemente in den Guard- und Action-Ausdrücken der Zustandstransitionen referenziert werden. Die Modellierung ist schematisch in Abb. 5.7 dargestellt.

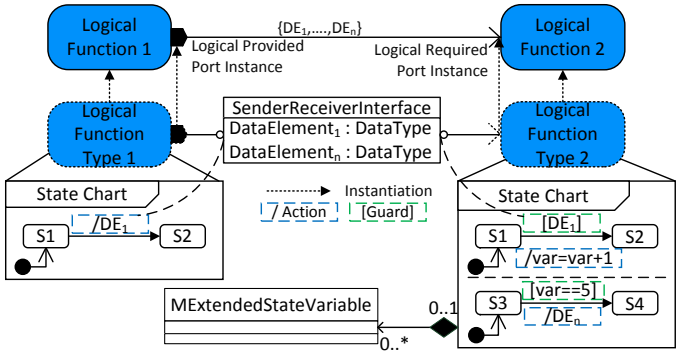


Abbildung 5.7: Prinzip zur Verhaltensmodellierung in PREEvision® durch Verfeinerung von logischen Funktionstypen mit State Charts (Quelle: [BKB19a]).

Eine Verhaltensmodellierung mit State Charts in der Softwarearchitektur erfolgt aufgrund der AUTOSAR-Orientierung analog. Eine Ausnahme bildet die Hardware-Ebene, auf der State Charts weitestgehend auf Instanzen der Komponenten hinterlegt werden und die Daten-Provider und -Konsumenten der Schnittstellen abweichend von Datenelementen sind, z. B. Signal Transmissionen.

5.2.2.2 Erweiterte Zustandsautomaten

Ein Nachteil der State Chart Modellierung in PREEvision® ist die fehlende Möglichkeit, erweiterte Zustandsautomaten (engl. *Extended Finite State Machine (EFSM)*) [4] durch interne numerische Variablen zu realisieren, welche zur signifikanten Reduktion der Komplexität von State Charts beitragen und der Zustands- und Transitions-Explosion entgegenwirken können [136, S. 199 f.]. Deshalb wird wie in Abb. 5.7 dargestellt der Einsatz von erweiterten Zustandsautomaten vorgeschlagen. Dazu wird eine Komposition des State Charts mit der vorgeschlagenen Metaklasse `MExtendedStateVariable` angewandt. Ein typisches Beispiel für den Einsatz einer erweiterten Zustandsvariable ist eine Zählvariable, die in einer Action inkrementiert wird und bei Erreichen eines bestimmten Zählerwertes eine Übergangsbedingung zum Zielzustand erfüllt (vgl. *var* Variable in Abb. 5.7).

5.2.3 Verknüpfung von Actor-orientiertem Verhalten mit State Charts

Die Verknüpfung von Actor-orientiertem Verhalten und State Charts wird über die neu eingeführte BLA-Ebene realisiert. Da die horizontale Kommunikation zwischen den State Charts über die Schnittstellen der logischen Funktionstypen, d. h. deren Ports stattfindet, kann dies auf die BLA-Ebene übertragen werden. Zur Unterscheidung der Art von Verhaltensbeschreibung der logischen Funktionen während der Simulationssynthese, wird ein entsprechender Building Block vom Typ *ModalModel* gewählt (in Anlehnung an PtII), welcher auf die State Chart Beschreibung des Funktionstyps verweist.

Damit eine konsistente Verknüpfung und Kommunikation zwischen logischen Funktionen mit Actor-orientiertem Verhalten und welchen mit State Chart Beschreibungen gewährleistet wird, sind zusätzliche Mappings notwendig. Diese sind als Sub-Mappings der bereits eingeführten Port-Prototyp-Mappings zu verstehen und referenzieren die durch die Port-Schnittstelle spezifizierten Datenelemente oder Operationen. Jeder Port eines regulären oder Modal Model Building Blocks auf oberster BLA-Ebene repräsentiert somit ein Datenelement oder eine Operation und werden auf Basis der strukturellen Verknüpfung auf LA-Ebene miteinander verbunden. Das Konzept zur Verknüpfung der beiden Modellierungsparadigmen auf der BLA-Ebene ist in Abb. 5.8 gezeigt. Die Port-Prototyp-Mappings und Datenelement Sub-Mappings sowie das initiale Block-Mapping werden dabei automatisiert generiert (siehe Unterabschnitt 5.3.4).

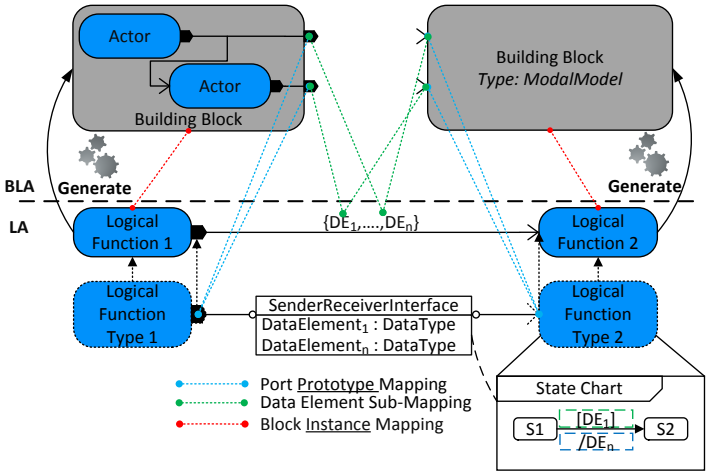


Abbildung 5.8: Konzept zur Verknüpfung der Actor-orientierten Verhaltensmodellierung mit State Charts auf der BLA-Ebene (Quelle: [BKB19a]).

5.2.4 Verknüpfungen mit detaillierteren Abstraktionsebenen

Die eingeführten LA/BLA-Mappings sind Voraussetzung für die Verbindung des spezifizierten Verhaltens mit realisierungsabhängigen Aspekten der unteren Abstraktionsebenen. Typischerweise werden ausgehend von den logischen Funktionen Mappings zu ausführenden Hardwarekomponenten wie Prozessoren, Verarbeitungseinheiten oder auch analoge Hardwaremodule hergestellt, die angeben, ob die Funktion in Hardware oder Software realisiert wird (vgl. Unterabschnitt 3.2.3.1). Alternativ kann eine logische Funktion auch weiter in Software-Funktionen aufgeteilt werden, welche dann auf Hardwarekomponenten abgebildet werden. Dies wird hier jedoch für die weitere Beschreibung nicht betrachtet. Generell kann eine logische Funktion auf mehrere ausführende Hardwarekomponenten abgebildet werden (*n*-zu-1 Mapping), jedoch sollte eine Funktion i. d. R. genau einer zugewiesen werden. Andernfalls ist zum einen nicht eindeutig ersichtlich, welcher Teil der Funktion auf welcher Hardwarekomponente ausgeführt wird. Zum anderen entstehen nicht eindeutige Signalpfade zwischen den Hardwarekomponenten [132, S. 133 ff.] [175, Kap. 9.3].

Ein wichtiges Merkmal ist, dass durch die bidirektionalen LA/BLA-Mappings sämtliche Abbildungen der logischen Funktionen auf Artefakte detaillierterer Ebenen und die sich daraus abgeleiteten Informationen auch für die BLA Buil-

ding Blocks gelten. Sowohl für Actor-orientiertes Verhalten als auch für Building Blocks, die auf State Chart Beschreibungen verweisen. Somit kann das spezifizierte Verhalten mit realisierungsabhängigen Informationen wie der Netzwerkkommunikation zwischen logischen Funktionen verfeinert werden. Dieses Vorgehen ist in Abb. 5.9 exemplarisch dargestellt.

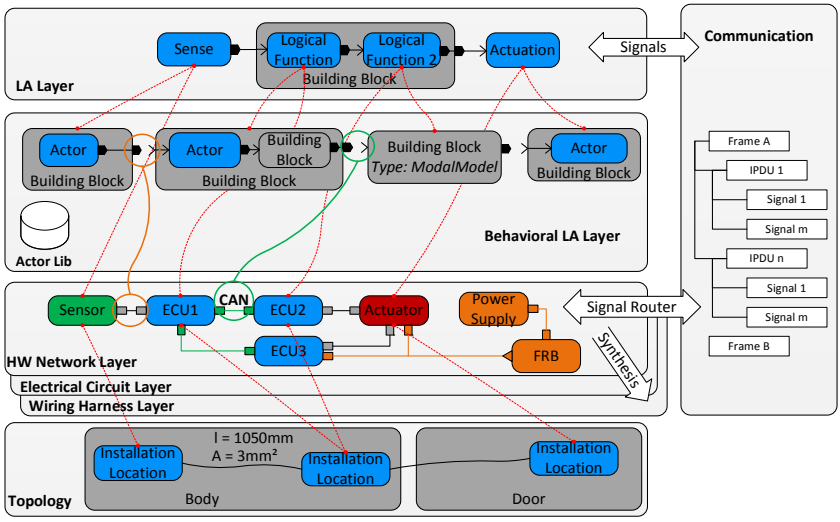


Abbildung 5.9: Mappings der logischen Funktionen auf detailliertere Abstraktionsebenen stellen die Verbindung des Verhaltens mit realisierungsabhängigen Informationen her (Quelle: [BRB17]).

Basierend auf den Mappings der logischen Funktionen ergibt sich in diesem Beispiel durch ein entsprechendes Routing der spezifizierten Datenelemente und die damit zusammenhängenden Signale eine CAN-Kommunikation zwischen den Funktionen auf der LA-Ebene. Diese Kommunikation gilt somit auch für die abgebildeten Verhaltensblöcke auf der BLA-Ebene und verfeinern deren Informationsaustausch mit einem entsprechenden CAN-Aspekt während der Simulationssynthese. Analog wird für konventionelle Verbindungen, z. B. zwischen einem Sensor und einer ECU, der Leitungssatz und dessen Verlegewege durch die Topologie berücksichtigt. Dadurch können elektrische und physikalische Eigenschaften wie spezifischer Widerstand und Leitungslänge für die analoge Verbindung bestimmt werden, die wiederum als Verfeinerung in die Simulation einfließen können. Auf die Modellierungserweiterungen zur Berücksichtigung von elektrischen Informationen und des Leitungssatzes wird in Unterabschnitt 5.2.5

eingegangen. Die Extraktion und Aggregation der realisierungsabhängigen Informationen für die Simulationssynthese ist Aufgabe des E/E-Model Interpreters (vgl. Unterabschnitt 5.1.3). Die spezifische Interpretation wird in Abschnitt 5.3 genauer beschrieben.

5.2.5 Ebenenübergreifende Verhaltensmodellierung

In den folgenden Abschnitten werden Konzepte vorgestellt, die eine ebenenübergreifende Verhaltensmodellierung erlauben und somit das realisierungsunabhängige Verhalten der LA-Ebene mit realisierungsabhängigen Architekturinformationen verknüpft. Die realisierungsabhängigen Informationen können dabei unterschiedliche Ausprägungen besitzen und auch auf unterschiedlichen Ebenen stattfinden. Ausschlaggebend ist die Frage, wie die diversen Verhaltensmodellierungen und realisierungsabhängigen Informationen auf den unteren Ebenen mit dem Verhalten der LA bzw. der BLA gekoppelt werden können. Dies wird in den folgenden Abschnitten adressiert.

5.2.5.1 Ebenenübergreifende State Charts

Wie in Unterabschnitt 5.2.2.1 eingeführt, erlaubt PREEvision® die State Chart Modellierung an Architekturartefakten auf mehreren Abstraktionsebenen. Als Basis zur ebenenübergreifenden Verhaltensmodellierung dient dazu die mögliche State Chart Beschreibung von Hardwarekomponenten. Die Verknüpfung mit dem Verhalten auf logischer Ebene ist jedoch nicht möglich und bedarf zusätzlicher Mechanismen.

Zur Schaffung dieser Möglichkeit orientiert sich das nachfolgend vorgestellte Konzept am AUTOSAR Standard zur Modellierung der Kommunikation zwischen Funktionen mittels typisierten Ports und dazugehörigen Schnittstellen (vgl. Unterabschnitt 3.2.6.1 und 5.2.2.1). Angewandt wird dies nachfolgend auf die logische Architektur. Durch die AUTOSAR-Orientierung lässt sich das Konzept jedoch auch auf State Chart Beschreibungen auf der Softwarearchitektur übertragen.

Basisservice Interfaces Zur Herstellung der Kommunikation bzw. Interaktion zwischen Verhalten von logischen Funktionen und den durch Mappings abgebildeten Hardwarekomponenten, werden am AUTOSAR Standard orientierte *Basisservice Interfaces* ausgenutzt. Service Interfaces sind im AUTOSAR Standard spezifiziert und dienen zur Kommunikation von SWCs mit Diensten der Service-Ebene der Basissoftware (vgl. Abb. 2.3). Auf diese Weise werden einem State

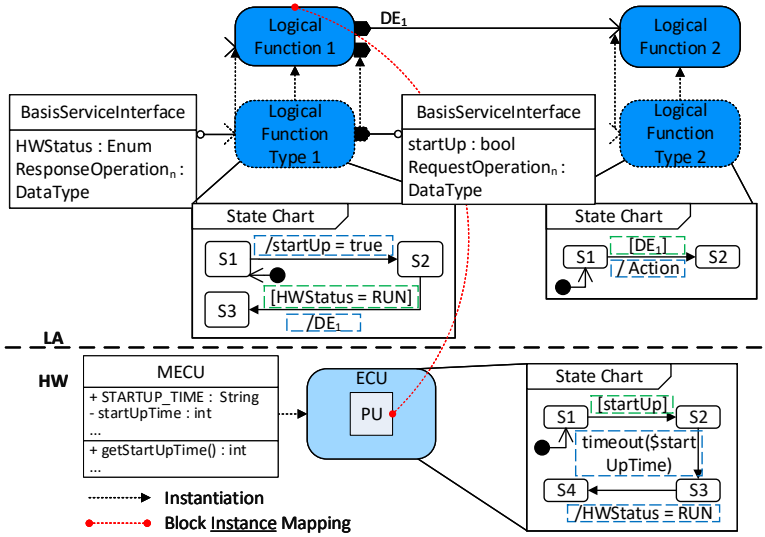


Abbildung 5.10: Ebenenübergreifende Verhaltensmodellierung auf LA- und Hardware-Ebene mit State Charts. Die Interaktion zwischen den Ebenen wird mit an AUTOSAR orientierten Basisservice Interfaces gelöst (Quelle: [BKB19a]).

Chart einer Hardwarekomponente zusätzliche Datenelemente oder Operationen zur Interaktion mit dem State Chart der dazugehörigen logischen Funktion zur Verfügung gestellt und umgekehrt. Das Konzept zur Interaktion eines State Charts einer Hardwarekomponente lässt sich sowohl mit einem State Chart als auch einer Actor-orientierten Verhaltensbeschreibung einer logischen Funktion anwenden. Grund dafür ist die Zusammenführung der beiden Beschreibungsarten in einem Building Block auf BLA-Ebene, welche eine eindeutige Beziehung zwischen Ports und Datenelementen besitzt (siehe Abb. 5.8).

Im Falle einer State Chart Beschreibung dienen die Datenelemente oder Operationen eines Basisservice Interfaces an einem Eingangsport zur Verwendung in Guard- bzw. in Action-Ausdrücken im dazugehörigen State Chart der Hardwarekomponente. Umgekehrt dienen die Datenelemente oder Operationen des Interfaces eines Ausgangsports zur Verwendung in Action- bzw. in Guard-Ausdrücken im State Chart der Hardwarekomponente. Ein ebenenübergreifendes Beispielmodell ist in Abb. 5.10 gezeigt.

Auf diese Weise lässt sich z. B. ein vom Betriebsmodus abhängiges Funktionsverhalten untersuchen, in welchem Funktionen einen Hardwarestatus anfordern können und ihr funktionales Verhalten nur ausführen, wenn die ECU betriebsbereit ist. Durch dieses Modellierungskonzept der Betriebsmodi-Interaktion ist es möglich, beliebige Zustände zu erweitern. Beispielsweise Fehlerzustände und damit verbunden abweichendes Funktionsverhalten wie Fail-Operational oder Fail-Safe Verhalten.

ECU Attribute & spontane FSMs Basierend auf den eingeführten, erweiterten Zustandsautomaten mittels Zustandsvariablen wird die Verwendung von spezifischen Attributen der Hardwarekomponenten ermöglicht. So lassen sich Attribute wie die Aufstartzeit einer ECU referenzieren und in Guard-Ausdrücken verwenden. Um solche zeitlichen Eigenschaften in der Simulation der State Charts ausschöpfen zu können, sind zusätzlich spontane Zustandsautomaten [136, Kap. 8.5] notwendig, die nicht nur auf Events an ihren Eingängen reagieren, sondern selbst zeitlich abhängige Events erzeugen können. Aus diesem Grund wird die aus PtII verfügbare Funktion `timeout` zur Verwendung in Guard-Ausdrücken eingeführt, welche ein Event nach Ablauf der als Argument angegebenen Modellzeit triggert (siehe das ECU State Chart in Abb. 5.10). In der EAST-ADL werden spontane Zustandsautomaten über sog. Temporal Constraints realisiert.

Strombeschreibungen Zur Adressierung der Analyse von quasi-statischen Strombedarfen werden in Kombination mit den Hardware State Charts zusätzliche Modellierungsmechanismen vorgestellt. Je nach Granularität der State Chart Beschreibung auf Hardware-Ebene weist auch die quasi-statische Strombeschreibung einen entsprechenden Detailgrad auf. So kann bspw. wie in Abb. 5.10 gezeigt das State Chart auf der ausführenden ECU hinterlegt sein, aber auch auf der internen Verarbeitungseinheit. Zur Modellierung der Stromaufnahme werden an dieser Stelle bestehende Modellierungsartefakte wiederverwendet, bspw. die *GenericConstraints* im Falle der EAST-ADL, die ein *current* Attributtyp mit beliebigen Artefakten assoziieren können [33, Kap. 22], oder das *isCurrentConsumer* Attribut der Metaklasse `HasCurrentDescription` in PREEvision® (siehe Abb. 3.8).

Die Verknüpfung zum Verhalten ist allerdings noch nicht gegeben. Aus diesem Grund wird ein Mapping an Transitionen von Hardwarezuständen vorgeschlagen, welches die Stromaufnahme der Komponente im aktuellen Zustand beschreibt. Die Anwendung im Falle von PREEvision® ist in Abb. 5.11 schematisch dargestellt und verknüpft eine Transition mit einem `CurrentDescriptorType`.

Das Mapping auf den Typ der Strombeschreibung ist wichtig, da bei Wiederverwendung des State Charts auf weiteren Hardwarekomponenten die Instanz der Strombeschreibung einen anderen Wert aufweisen kann. Ein Strombeschrei-

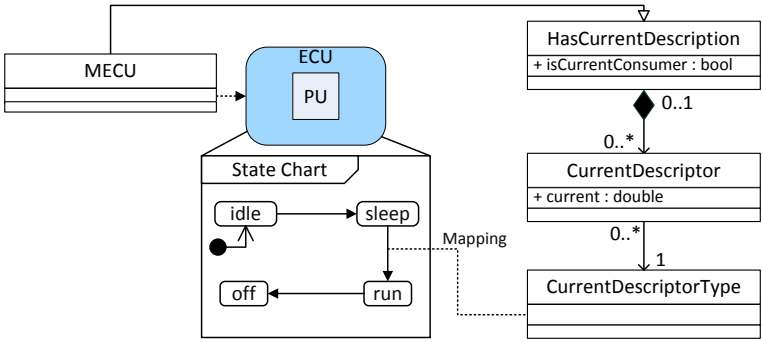


Abbildung 5.11: Das Mapping einer Strombeschreibung an eine Transition von Hardwarezuständen adressiert die quasi-statische Stromaufnahme.

bungstyp kann z. B. ein Standby-Zustand sein, der auf unterschiedlichen Sensoren oder Aktuatoren typischerweise andere Ströme annimmt. Anzumerken ist, dass das aktuelle Metamodell von PREEvision® eine Strombeschreibung auf internen Verarbeitungseinheiten etc. nicht erlaubt und in diesem Zusammenhang erweitert werden müsste, um eine auf internen Komponenten definierte Strombeschreibung zu berücksichtigen. Das Mapping einer Strombeschreibung auf Zustandstransitionen lässt sich u. a. auch auf die EAST-ADL übertragen, bei der ein *GenericConstraint* mit dem o. g. Attributtyp mit beliebigen Artefakten assoziiert werden kann. Zusammen mit der zeitlichen Simulation von spontanen State Charts lässt sich somit eine von den Betriebsmodi abhängige Stromaufnahme analysieren.

5.2.5.2 Modellierungserweiterungen zur elektrischen Simulation

Die bisher beschriebene Verhaltensmodellierung beschränkt sich auf die logische Vernetzung von Hardwarekomponenten wie Steuergeräten, Sensoren/Aktuatoren etc. Die elektrische Ebene, der Leitungssatz und dessen Verlegewege durch die Topologie wurden bisher nicht berücksichtigt. Um jedoch dynamische Analysen von Strombedarfen und den Einfluss von elektrischen Komponenten sowie des Leitungssatzes auf das realisierungsunabhängige Verhalten untersuchen zu können (vgl. Unterabschnitt 4.3.3.3), sind zusätzliche Modellierungsmechanismen notwendig. Diese Mechanismen werden herangezogen, um die genannten Aspekte während der Simulationssynthese automatisiert zu integrieren. Das Konzept für die ebenenübergreifende Verhaltensmodellierung zur Berücksichtigung von

elektrischen Komponenten und des Leitungssatzes ist schematisch in Abb. 5.12 dargestellt.

Die dort exemplarisch gezeigte elektrische Ebene (engl. *Electric Circuit (EC)* Layer) beinhaltet einen einfachen Spannungsteiler vor dem Eingangspin und einen Einweg-Gleichrichter am Ausgangspin eines Mikrocontrollers, auf welchen eine logische Funktion bzw. deren Verhalten abgebildet ist. Nun sind nicht unbedingt alle logischen Funktionen und ihr Verhalten dafür vorgesehen durch analoge Komponenten verfeinert zu werden, da bspw. das spezifizierte Verhalten lediglich Information repräsentiert und kein kontinuierliches Signal. Zur Unterscheidung und Identifizierung während der Simulationssynthese, welche Ports durch elektrische Komponenten verfeinert werden können und sollen, werden zwei grundlegende E/E-Architektur-Modellierungserweiterungen eingeführt:

1. Spezielle Datentypen für Datenelemente an Port-Schnittstellen von logischen Funktionen.
2. Eine neue Menge an ebenenübergreifenden Mappings.

Analoge Ports Grundlage ist wiederum das an AUTOSAR orientierte Konzept bestehend aus typisierten Ports, welche über eine dedizierte Schnittstelle Datenelemente als Information austauschen. Definiert wird Folgendes:

Analoger Port *Ein logischer Port, der genau ein Datenelement mit einem Applikations-Datentyp* ELECTRICAL VOLTAGE *besitzt, wird als analoger Port spezifiziert. Ein analoger Port kann einen beliebigen Signalverlauf annehmen.*

Die Spezifikation von analogen Ports ist beispielhaft auf der LA-Ebene in Abb. 5.12 zu sehen. Ein analoger Port kann dabei mithilfe eines Actor-Netzwerks beliebige Signalverläufe annehmen, d. h. beliebige Amplituden und Frequenzen, digitale Spannungspegel oder PWM-Signale.

Analoge Port-Mappings Wie in Unterabschnitt 5.2.4 beschrieben wird eine logische Funktion typischerweise auf genau eine ausführende Hardwarekomponente abgebildet, deren Verhalten in einem BLA Building Block gekapselt wird. Durch die Einführung der analogen Ports können nun individuelle Ausgangsports als Actor-orientierte Spannungsquelle fungieren, die an die Hardwarekomponente angeschlossene elektrischen Schaltungen stimulieren.

Um jedoch während der Simulationssynthese zu identifizieren, welche elektrischen Schaltungen und ggf. externe Verbindungen in Form von konventionellen Leitungen zwischen einem analogen Ausgangsport und einem oder mehreren analogen Eingangsports berücksichtigt werden müssen, sind zusätzliche Informationen im Modell notwendig. Im Speziellen müssen die den analogen Ports

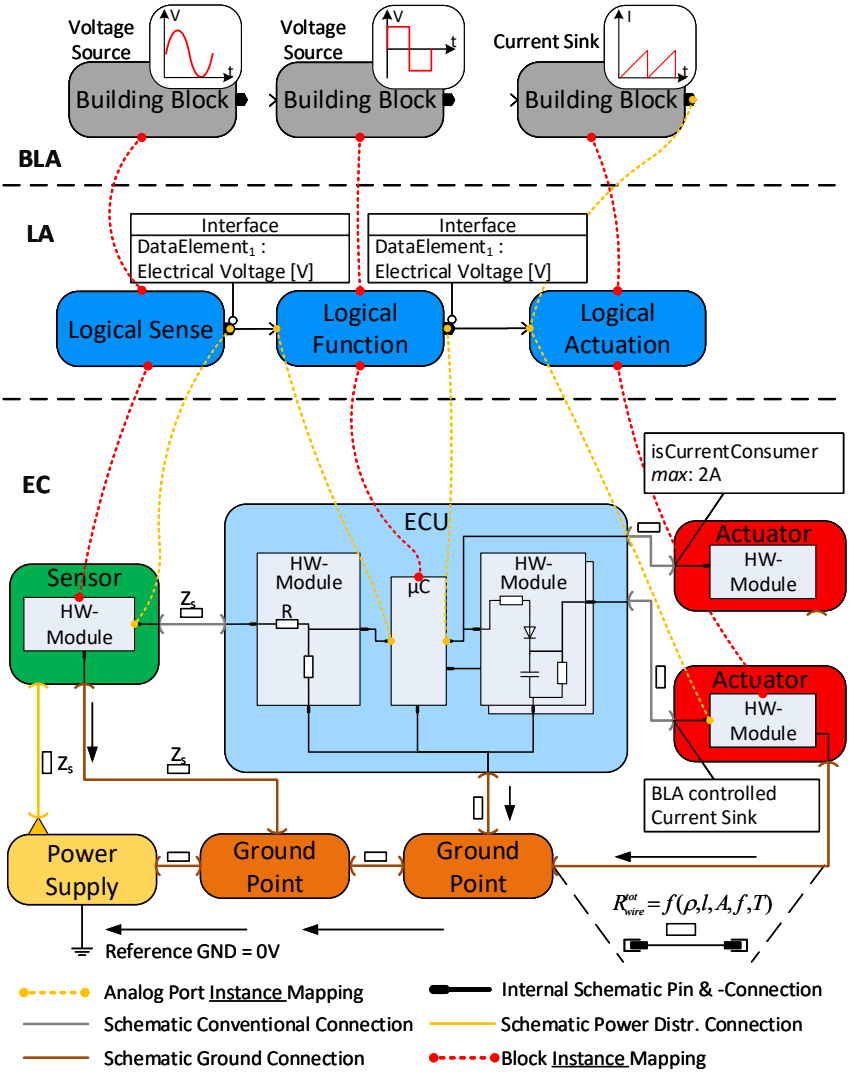


Abbildung 5.12: Modellierungserweiterungen zur elektrischen und leitungssatz-sensitiven Verfeinerung von Verhalten (Quelle: [BB18]).

zugehörigen internen Hardware-Pins bekannt sein, die den Start- und Endpunkt auf elektrischer Ebene definieren. Zwischen diesen werden die elektrischen Schaltungen und ggf. Leitungen berücksichtigt. Hauptgrund für diese Notwendigkeit ist, dass die ausführenden Hardwarekomponenten typischerweise mehr als einen Pin aufweisen. Ein weiterer Grund ist, falls eine logische Funktion auf mehr als eine ausführende Hardwarekomponente innerhalb eines Steuergeräts abgebildet werden soll.

Um die Start- und Ende-Pins zu definieren, werden zusätzliche Mappings in Form von *analogen Port-Mappings* eingeführt, die genau einen analogen Port auf genau einen internen Hardware-Pin[2] abbilden. Im Gegensatz zu den Port-Prototyp-Mappings zwischen LA und BLA werden diese Mappings auf Instanz-Ebene angewandt, da IO-Pins von Mikrocontrollern und Hardwaremodulen i. d. R. nicht typisiert instanziiert werden. Darüber hinaus muss eine eindeutige Beziehung zwischen dem stimulierenden/konsumierenden analogen Port und dessen Hardware-Pin bestehen, da eine logische Funktion mehrere analoge Ports enthalten kann. Letztere können jedoch je nach Verhaltensmodellierung verschiedene Signalverläufe annehmen.

Des Weiteren gelten folgende Modellierungsbedingungen:

- Ein analoger Port muss auf einen Hardware-Pin abgebildet werden, dessen interne Hardwarekomponente jener der abgebildeten logischen Funktion entspricht.
- Start- und Ziel-Hardware-Pin können sich sowohl innerhalb derselben ECU als auch auf unterschiedlichen ECUs befinden.

In Abb. 5.12 sind die Mappings exemplarisch über Sensor, ECU und Aktuator hinweg gezeigt.

Stromverbraucher Ergänzend zu den analogen Ausgangsports können analoge Eingangsports als Stromverbraucher fungieren, z. B. dann, wenn im empfangenden Hardwaremodul keine elektrische Schaltung als Last modelliert ist. Zur Adressierung von statischen, quasi-statischen und dynamischen Strombedarfen werden zunächst zwei Typen von Stromverbrauchern betrachtet, die in Kombination mit den elektrischen Schaltungen während der Verhaltenssimulation berücksichtigt werden: *statische* und *dynamische* Verbraucher. Im Falle eines dynamischen Verbrauchers wird das Stromprofil wiederum durch den dazugehörigen BLA Building Block als Actor-Netzwerk modelliert (siehe Abb. 5.12). Zur Modellierung der statischen Verbraucher werden wie in Unterabschnitt 5.2.5.1 beschrieben bestehende Modellierungsartefakte zur Strombeschreibung wiederverwen-

[2] `UndirectedInternalSchematicConnector` im PREEvision® Metamodell, vgl. Abb. 3.8.

det. Im Falle von PREEvision® das `isCurrentConsumer` Attribut der Metaklasse `HasCurrentDescription`.

Statische Verbraucher Unabhängig vom zugrunde liegenden Metamodell der E/E-Architektur werden die genannten Attribute zur Spezifikation eines Stromverbrauchers an einem Hardware-Pin verwendet. Das Attribut wird dabei entweder direkt am internen Hardware-Pin oder, wie in Abb. 5.12 am oberen Aktuator dargestellt, an einem direkt verbundenen schematischen Pin spezifiziert. In Kombination mit dem analogen Port-Mapping wird so ein logischer Eingangsport als Stromverbraucher spezifiziert. Gleichzeitig signalisiert ein gesetztes Stromverbraucherattribut den Endpunkt der Verfeinerung zwischen einem analogen Ausgangsport und einem oder mehreren analogen Eingangsports. Auf diese Art ist es möglich, dass auch Hardwarekomponenten berücksichtigt werden können, die nicht vollständig als Last modelliert sind oder keine entsprechende Funktion entlang des logischen Pfades abgebildet haben. Dies ist in Abb. 5.12 beim oberen Aktuator der Fall.

Dynamische Verbraucher Schließlich wird zur Unterscheidung zwischen einem statischen und einem durch einen BLA Building Block definierten dynamischen Stromverbraucher ein zusätzliches analoges Port-Mapping durchgeführt. Ein bereits spezifizierter analoger Eingangsport einer logischen Funktion wird zusätzlich auf einen separaten Ausgangsport des dazugehörigen BLA Building Blocks abgebildet. Grund dafür ist schlicht die zugrunde liegende Actor-orientierte Realisierung des Stromprofils, das über einen Ausgangsport zur Verfügung gestellt werden muss. In Abb. 5.12 ist der dynamische Stromverbraucher exemplarisch am logischen Aktuator gezeigt.

Quasi-statische Verbraucher Wenn eine logische Funktion eine State Chart Beschreibung statt einer Actor-orientierten Realisierung hinterlegt hat, ist es auf Grundlage der eingeführten ebenenübergreifenden Verhaltensmodellierung mittels Basisservice Interfaces möglich, quasi-statische Stromverbraucher während der Simulationssynthese mit in die elektrische Simulation einzubinden. Dafür sind folgende Modellierungsbedingungen zu erfüllen:

- Logische Funktion und abgebildete, ausführende Hardwarekomponente oder deren umgebende ECU besitzen eine State Chart Beschreibung, die über Basisservice Interfaces miteinander kommunizieren können.
- Das State Chart der Hardware besitzt mindestens ein Mapping zu einer Strombeschreibung.

- Der zum analogen Port dazugehörige Hardware-Pin ist Bestandteil der abgebildeten Hardware.
- Der Hardware-Pin ist als Stromverbraucher markiert.

Der während der Simulationssynthese automatisch generierte Stromausgangsport für das State Charts der Hardwarekomponente definiert sodann die Stromaufnahme. In diesem Fall wird der zu diesem Stromausgangsport gehörige Eingangsport des Dummy-Blocks mit der aspektorientierten elektrischen Simulation dekoriert (vgl. Unterabschnitt 5.3.8).

5.2.5.3 Leitungssatzsensitive Simulation

Die beschriebenen Modellierungserweiterungen beschränkten sich auf die Verfeinerung des spezifizierten Verhaltens auf BLA-Ebene oder mit State Charts durch elektrische Schaltungen auf der EC-Ebene. Verfeinerungen von konventionellen Leitungen sowie Masseverbindungen wurden vernachlässigt. Auf EC-Ebene stellen diese elektrischen Verbindungen Abstraktionen dar, die tatsächlich auf der Leitungssatz-Ebene (engl. *Wiring Harness (WH)* Layer) durch eine Menge an physikalischen Leitungen, Trennstellen und Leitungssatz-Konnektoren verfeinert werden. Hardware-Pins an Hardwarekomponenten wie Sensoren, Aktuatoren oder ECUs sind schematische Darstellungen und stellen auf Leitungssatz-Ebene tatsächlich ein Paar aus Header-Pin einer Komponente und Leitungs-Pin dar (vgl. Abb. 2.2 und Abb. 3.9 sowie die verfeinerte Masseleitung in Abb. 5.12). In PREEvision® kann der Leitungssatz durch die *Electric Circuit Synthesis* automatisiert erstellt werden (vgl. Unterabschnitt 3.2.2.4).

Die Stabilitätsanalyse der Spannungsversorgung von Steuergeräten unter dem Einfluss der Spannungsabfälle über Leitungen ist wie in Unterabschnitt 4.3.3.3 beschrieben ein wichtiges Kriterium bei der E/E-Architekturauslegung.

Ohmsche Leitungswiderstände Die eingeführten analogen Port-Mappings bilden die Voraussetzung zur Berücksichtigung der Leitungswiderstände in der elektrischen Simulation, sowohl von konventionellen Signalleitungen aber auch von versorgenden Leitungen unter Einfluss der modellierten elektrischen Schaltungen und Stromverbrauchern. Im E/E-Architekturmodell besitzen Leitungen einen bestimmten Leitungstyp[3] (engl. *wire type*), der durch bestimmte elektrische und physikalische Eigenschaften charakterisiert ist und welche für alle instanziierten Leitungen gelten. Dazu zählen der spezifische Leitungswiderstand ρ und der Leitungsquerschnitt A. Leitungs-Pins werden ebenfalls typisiert instanziiert

[3] vgl. die Metaklasse `WireType` in Abb. 3.10.

und besitzen elektrische Eigenschaften wie Übergangswiderstände[4]. Über die Mappings von Hardwarekomponenten und Leitungssatz-Artefakten zur Topologie, lässt sich nach dem Routing der Leitung durch typischerweise mehrere Topologie-Segmente die Gesamtlänge der Leitung l_{tot} bestimmen. Diese ergibt sich aus der Summe der individuellen Topologie-Segmentlängen. Durch die gegebenen Leitungseigenschaften lässt sich der ohmsche Gesamtwiderstand einer einzelnen Leitung zusammen mit den Übergangswiderständen der Pins mit

$$R_{wire}^{tot} = \rho \frac{l_{tot}}{A} + R_{PinType\_1} + R_{PinType\_2} \tag{5.1}$$

berechnen, wobei die Pins unterschiedlichen Typs und somit die Übergangswiderstände unterschiedlich groß sein können. Die Verfeinerung einer schematischen, elektrischen Verbindung in einen Pfad auf Leitungssatz-Ebene ist in Abb. 5.13 auf Basis der elektrischen Ebene in Abb. 5.12 nochmals detaillierter dargestellt. Dort ist auch die Einbindung eines Stromverbrauchers in Form einer zusätzlichen ECU2 gezeigt, die nicht Teil des logischen Pfades ist. Die gestrichelten Linien verweisen auf die Knotenpunkte eines Widerstands in einer elektrischen Netzliste.

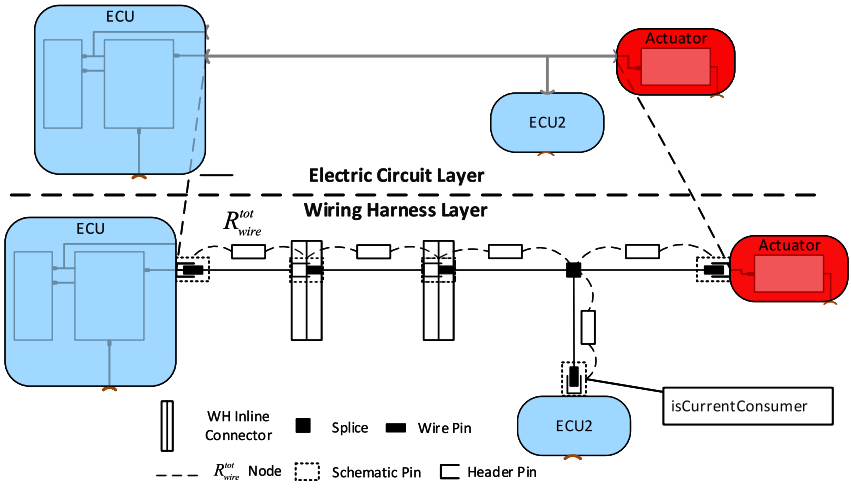


Abbildung 5.13: Verfeinerung einer schematischen, elektrischen Verbindung in einen Pfad an Leitungssatz-Artefakten und dazugehörige ohmsche Leitungswiderstände.

[4] vgl. die Metaklasse `HasResistanceValues` in Abb. 3.10.

Masseverbindungen Bei den bisherigen Betrachtungen der elektrischen Schaltungen und Stromverbraucher als Verhaltensverfeinerung wurde eine Referenzmasse von $0\,V$ angenommen. Jedoch kann es auch bei Masseleitungen zu Spannungsabfällen kommen, da diese typischerweise an einen bestimmten Massepunkt (engl. *Ground Point*) im Chassis des Fahrzeugs angeschlossen sind und nicht direkt an der Masse der versorgenden Komponente, z. B. der Batterie. Dies führt zu zusätzlichen Spannungsabfällen über die Masseleitungen zwischen den Massepunkten, bis die Referenzmasse erreicht ist und kann ungewollte Spannungsoffsets hervorrufen [42].

Daher werden auf ähnliche Weise auch die Masseleitungen mit in die elektrische Simulation eingebunden. Eine notwendige Modellierungsbedingung dafür ist, dass die Hardwarekomponenten über entsprechende Massepunkte mit ihrer Hauptenergieversorgung wie der Batterie verbunden sind. Dies ist in Abb. 5.12 exemplarisch für den Sensor gezeigt.

Auf Basis des modellierten Massennetzes wird zu dessen Einbindung folgende Vorgehensweise verfolgt:

- Für jeden Masse-Pin einer involvierten Hardwarekomponente wird der kürzeste Pfad der Masseleitungen über die Massepunkte bis zur Energiequelle zurückverfolgt.
- Der Masse-Pin der Energiequelle wird dabei als die Referenzmasse von $0\,V$ angenommen.

Die optimale Verbindung der Masse-Pins der Hardwarekomponenten zu den verfügbaren Massepunkten im Fahrzeug wird dabei als separates Problem betrachtet und ist explizit nicht Teil der Zurückverfolgung. Diese Optimierung kann mit existierenden Lösungen, bspw. mit der *Ground Spot Optimizer* Funktion von PREEvision® [170, Kap. 10.3], im Vorfeld gelöst werden. Letztere liefert in Verbindung mit dem Leitungssatz-Router (siehe Unterabschnitt 3.2.2.4) als Ergebnis die kürzesten Leitungslängen der Masseverbindungen durch die Topologie zurück.

Leitungsimpedanz Abschließend ist zu bemerken, dass der Leitungswiderstand tatsächlich eine komplexe, frequenzabhängige Leitungsimpedanz Z_S aufweist. Allerdings sind die induktiven und kapazitiven Leitungsbeläge stark von der Topologie abhängig, sowohl hinsichtlich des kompletten Leitungssatzes als auch von der Anordnung von mehreren Leitern in Kabeln, sodass sich vor allem die Koppel-Induktivitäten und -Kapazitäten nur schwer auf Basis der zur Verfügung stehenden Topologie-Informationen aus dem Modell bestimmen lassen. Jedoch lassen sich insb. für Versorgungsleitungen Annahmen bzgl. der Frequenzabhängigkeit und des RLC-Modells treffen, ohne große Genauigkeitsverluste einbüßen

zu müssen [42]. Ausgehend davon wäre als Alternative und Erweiterung denkbar, eine komplexe Impedanz als Attribut einer Leitung oder als Last zu modellieren.

5.3 E/E-Architekturzentrierte Analyse und Simulationssynthese

In diesem Abschnitt werden konkrete Konzepte der integrierten E/E-Architekturanalyse und Simulationssynthese vorgestellt. Die prototypische Umsetzung der Architektur- und Verhaltensmodellierung findet dabei vollständig in PREEvision® statt, während über die Simulationssynthese ein ausführbares Ptolemy II Modell generiert und in PREEvision® integriert ausgeführt wird. Gleichzeitig können die Simulationsergebnisse auf verschiedenste Arten visualisiert sowie zurückgespeist werden.

Zunächst wird erläutert, wie und wo die ebenenübergreifende Verhaltensmodellierung und die Simulationssynthese in den Modellierungsprozess existierender E/E-Architekturmodelle und speziell von PREEvision® einzuordnen sind (vgl. Unterabschnitt 5.3.1), bevor auf die realisierte Implementierung und Integration in PREEvision® eingegangen wird (Unterabschnitt 5.3.2). Anschließend folgen die grundlegenden Abbildungsvorschriften zur Überführung der Architekturartefakte des E/E-Architektur-Metamodells auf das Simulations-Metamodell (Unterabschnitt 5.3.3).

Die Generierung der Building Blocks der logischen Verhaltensarchitektur auf oberster Hierarchieebene (BLA-Schicht, siehe Abb. 5.1) und deren Mappings werden in Unterabschnitt 5.3.4 beschrieben. Auf Basis der Abbildungsvorschriften werden in den darauffolgenden Abschnitten Details zur Interpretation des E/E-Architekturmodells und zur ebenen- bzw. domänenübergreifenden Synthese des PtII Simulationsmodells erläutert (vgl. Unterabschnitt 5.3.5 bis 5.3.8).

Abschließend folgt eine kurze Diskussion der aktuellen Limitierungen in Unterabschnitt 5.3.9.

5.3.1 Modifizierter Modellierungsprozess

Das Flussdiagramm des modifizierten Modellierungsprozesses mit den für diese Arbeit wichtigsten Modellierungs- und Syntheseschritten sowie den neu hinzugefügten Schritten ist in Abb. 5.14 illustriert.

Der Ablauf folgt prinzipiell einem Top-Down-Ansatz beginnend bei der Spezifikation der Anforderungen und Kundenwünschen sowie deren Mapping unter-

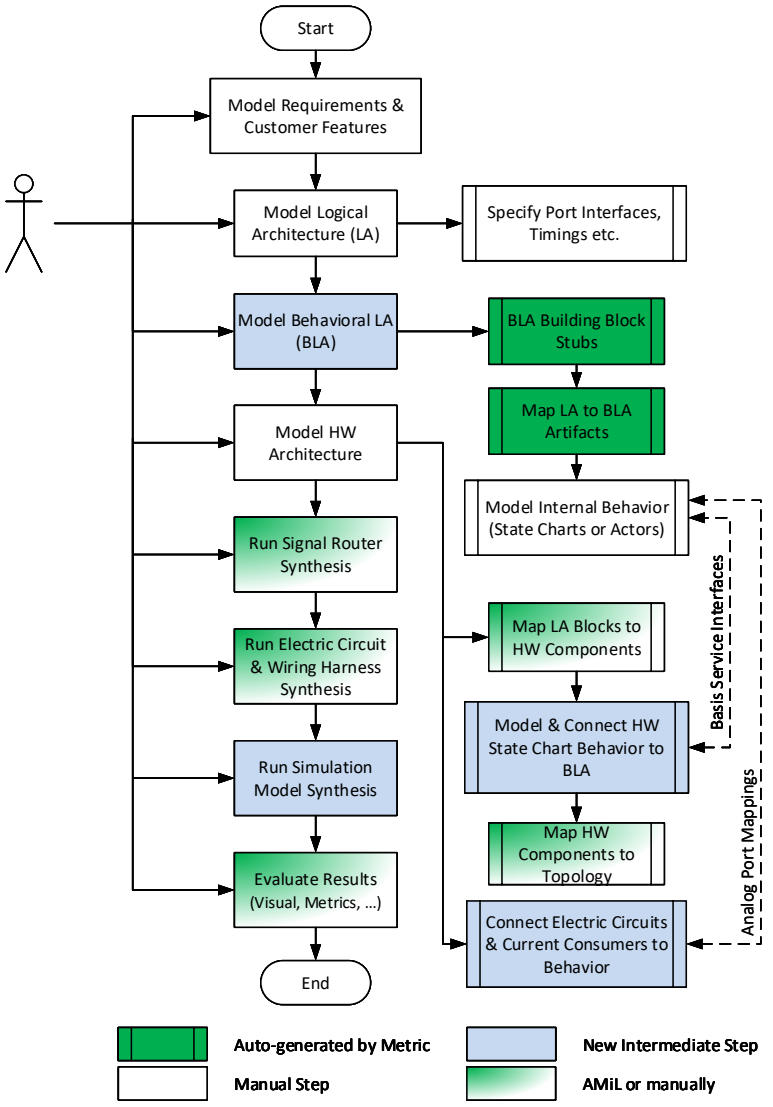


Abbildung 5.14: Modifizierter Modellierungsprozess mit den wichtigsten Modellierungs- und Syntheseschritten sowie den in dieser Arbeit entstandenen Erweiterungen.

einander und zu beliebigen Artefakten auf unteren Schichten. Der Schritt ist nur der Vollständigkeit halber gezeigt, da der relevante Startpunkt für diese Arbeit die logische Architektur im nächsten Schritt ist. Ein Top-Down Ansatz beginnend bei den Anforderungen und der realisierungsunabhängigen Systemarchitektur folgt zum einen dem in der Industrie etablierten V-Modell Prozess (vgl. Unterabschnitt 2.1.1.2). Zum anderen bleibt die logische Systemarchitektur in der Praxis über Jahre stabil und dient daher gleichzeitig als Grundlage für alle weiteren Entwicklungsschritte über Fahrzeuggenerationen hinweg [153]. Dadurch ist es jedoch aufgrund verteilten Entwicklung und des schichtenorientierten Ansatzes auch möglich, die technische Umsetzung der E/E-Architektur parallel und inkrementell in einem Bottom-Up-Vorgehen zu entwickeln und schließlich mit der logischen Architektur in Verbindung zu setzen.

Der erste Schritt nach den Anforderungen und Kundenwünschen ist somit die Modellierung der logischen Fahrzeugarchitektur (LA), die aus mehreren Teilfunktionen und Funktionsdomänen zusammengesetzt ist. Die Spezifikation der Ports und deren Schnittstellen ist dabei essenziell für eine konsistente Kommunikation und Informationsaustausch zwischen Funktionen. Generell ist eine konsistente LA eine notwendige Voraussetzung für alle weiteren Modellierungs- und Syntheseschritte, sowohl für die E/E-Architektur selbst als auch für die Simulationssynthese.

Anschließend folgt als neuer Zwischenschritt die in dieser Arbeit hinzugefügte ebenenübergreifende Verhaltensmodellierung bestehend aus Actor-orientiertem Verhalten und State Charts, die je nach Modellierungsart entweder auf der eingeführten BLA-Ebene erfolgt oder mit State Chart Editoren. Zusammengeführt werden jedoch beide auf der BLA-Ebene mittels dedizierten Building Blocks (vgl. Unterabschnitt 5.2.3). Zur Reduzierung des Modellierungsaufwands für den Nutzer und der Fehleranfälligkeit werden die benötigten Building Block Hüllen und insb. die Mappings zwischen den LA und BLA Artefakten automatisiert generiert (siehe Unterabschnitt 5.3.4).

Basierend auf der logischen Architektur und des dazugehörigen Verhaltens können die Funktionen auf die Steuergeräte der Vernetzungsarchitektur abgebildet werden. Dies kann entweder manuell durch den Nutzer erfolgen oder, mithilfe des später vorgestellten AMiL Ansatzes (siehe Abschnitt 5.4), automatisiert in einer Metrik. Zusätzlich erfolgt als optionaler Schritt, je nach gewünschtem Detailgrad, die Verknüpfung von State Charts von Steuergeräten mittels der eingeführten ebenenübergreifenden Modellierung mit logischem Verhalten. Abhängig davon, ob elektrisches und leitungssatzsensitives Verhalten gewünscht ist, sind die Hardwarekomponenten auf die Topologie abzubilden sowie die elektrische Ebene nach Unterabschnitt 5.2.5.2 und 5.2.5.3 mit dem logischen Verhalten zu verknüpfen.

Die folgenden Syntheseschritte des Signal Routings sowie der elektrischen und Leitungssatz-Ebene durch PREEvision® sind je nach zu berücksichtigenden Domänen bzw. Eigenschaften der Hardware-Ebenen notwendig, um ein konsistentes Simulationsmodell zu erhalten, das im Anschluss ausgeführt wird. Diese Syntheseschritte als auch die Evaluation der Simulationsergebnisse können wiederum entweder manuell, z. B. die visuelle Verifizierung von Signalen einer Funktion, oder automatisiert mithilfe des AMiL Ansatzes durchgeführt werden, z. B. die automatisierte Evaluation von statischen und dynamischen Metriken.

5.3.2 Implementierung und Integration in PREEvision®

5.3.2.1 Simulationssynthese und -ausführung

Die Implementierung und Integration der Simulationssynthese sowie die Ausführung der Simulation wird vollständig mithilfe des Metrik-Frameworks in PREEvision® realisiert. Die Realisierung der Synthese und Simulation innerhalb des Metrik-Frameworks hat den Vorteil, dass voller Zugriff auf das Datenmodell der E/E-Architektur besteht und sowohl die Synthese als auch die Ausführung in benutzerdefinierten Metrikblöcken (engl. Customized Metric Blocks, vgl. Unterabschnitt 3.2.5) gekapselt werden können. Dies führt zu einem weiteren Vorteil, dass die Synthese und Simulation zusammen mit weiteren Metrikberechnungsblöcken integriert und grafisch in entsprechenden Metrikdiagrammen modelliert werden können. Ein Anwendungsfall dafür ist bspw. das in Abschnitt 5.4 vorgestellte AMiL Konzept.

Das Metrik-Diagramm mit den elementaren, benutzerdefinierten Metrikblöcken zur Simulationssynthese und -ausführung ist in Abb. 5.15 dargestellt. Diese sind jeweils als Java Plugins realisiert und erweitern die benutzerdefinierten Metrikblöcke über sog. *Extension Points*.

Die Simulationssynthese beinhaltet dabei tatsächlich die in Unterabschnitt 5.1.3 und 5.1.4 vorgestellten E/E-Model-Interpreter- und Simulationssynthese-Komponenten, welche im *Pt2SimulationModelSynthesizer* Metrikblock gekapselt sind. Dementsprechend erwartet der Metrikblock die zu simulierende logische Architektur oder Subsysteme davon, die über einen sog. Modellkontextblock [41, Kap. 6.2.2] am dazugehörigen Eingang bereitgestellt werden. Als Ergebnis wird das synthetisierte und serialisierte MoML-Modell an den für die Ausführung der Simulation zuständigen Block *Pt2Executor* geschickt. Zusätzlich zu den logischen Funktionen werden zwei optionale Eingangsports bereitgestellt, *VariantSensitive* und *CurrentDescriptorType*. Ersterer erwartet einen booleschen Parameter, der die variantensensitive Synthese (de-) aktiviert. Der zweite Eingang gibt an, welcher Wert eines statisch modellierten Stromverbrauchers bei einer elektrischen und

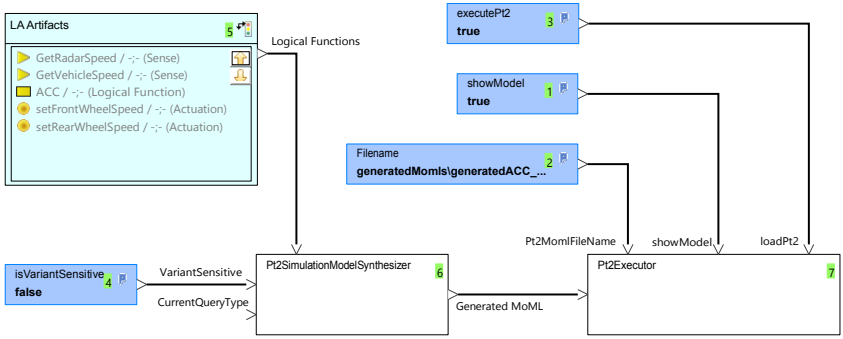


Abbildung 5.15: Metrikdiagramm mit den elementaren Metrikblöcken zur Synthese und Ausführung eines ebenenübergreifenden PtII Simulationsmodells.

leitungssatzsensitiven Synthese ausgelesen werden soll. Dies kann entweder der minimale, typische oder maximale Wert sein, oder ein `CurrentDescriptorType` Artefakt, das die statische Stromaufnahme in einem bestimmten Betriebsmodus von Verbrauchern angibt. Ist dieser Eingangsport nicht angeschlossen, wird die elektrische und leitungssatzsensitive Synthese deaktiviert.

5.3.2.2 Integration der Behavioral Logical Architecture

Die von PREEvision® für Entwickler bereitgestellte Plattform in Form einer modifizierten eclipse-Entwicklungsumgebung bietet keine Möglichkeit, eigenständige Metamodell-Erweiterungen vorzunehmen, die für die neu eingeführte BLA-Ebene und deren Verknüpfung zu anderen Ebenen via Mappings notwendig wären. Für die Umsetzung und Integration in PREEvision® wird daher auf den Produktlinienansatz nach Abb. 3.3 zurückgegriffen, wonach jede Produktlinie aus einem 150 % Modell besteht, das alle Abstraktionsebenen umfasst. Die konkrete Umsetzung ist in Abb. 5.16 schematisch dargestellt.

Demnach findet die Modellierung der eigentlichen E/E-Architektur inkl. der logischen Architektur innerhalb einer Produktlinie statt, die mithilfe von ein oder mehreren Bibliotheken bestehend aus Komponenten, Funktionstypen, Subsystemen etc. durch Anwendung der Prinzipien Wiederverwendung und Instanziierung aufgebaut wird. Analog dazu wird nun die BLA-Ebene mittels einer separaten, sog. ausführbaren Produktlinie integriert, deren LA-Ebene als Stellvertreter zur Actor-orientierten Verhaltensmodellierung dient. Das Verhalten innerhalb der

Building Blocks wird mithilfe einer zur Produktlinie komplementären, ausführbaren Bibliothek bestehend aus Actors modelliert, welche in logischen Funktionstypen gekapselt sind. Bereits bestehende, modellierte Building Blocks können über die Wiederverwendungsfunktion (engl. *reuse*) von PREEvision® mehrfach verwendet werden. Die Port-Prototyp- und Block-Mappings zwischen den beiden Ebenen werden über sog. Modellkontexte (engl. *Model Context*) [175, Kap. 18.3.4.1] hergestellt. Ein Modellkontext assoziiert ein Artefakt mit einer Menge an beliebig weiteren Artefakten, z. B. eine logische Funktion mit einem Building Block und umgekehrt.

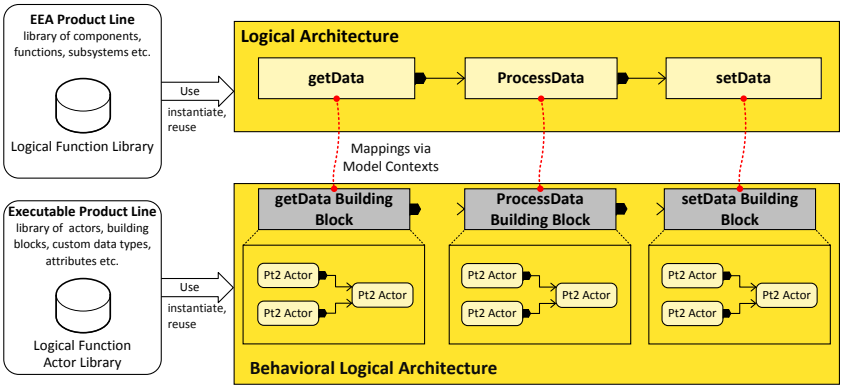


Abbildung 5.16: Integration der BLA-Ebene über eine dedizierte, ausführbare Produktlinie (in Anlehnung an Abb. 3.3), deren LA-Ebene als Stellvertreter zur Actor-orientierten Verhaltensmodellierung dient. Die Actors sind in logischen Funktionstypen gekapselt.

5.3.2.3 Import der Ptolemy II Actor-Bibliothek

Zur Steigerung der Produktivität der Verhaltensmodellierung ist, wie in Abb. 5.16 dargestellt, eine Bibliothek an Actors herangezogen worden. Die Actors werden dabei in instanziierbaren logischen Funktionstypen gekapselt. Genau genommen werden die Actor-Klassen in Funktionstypen gekapselt (siehe auch Unterabschnitt 5.3.3), da PtII MoML-Modelle objektorientiert aufgebaut sind. Dabei kann ein Actor sowohl eine Instanz einer referenzierten Java-Klasse sein als auch ein benutzerdefiniertes MoML-Modell, das als Klasse (MoML-Element `class`) definiert wurde [83, 85].

Die zur Verfügung stehenden Actors werden dabei über eine Metrik automatisiert in PREEvision® importiert, welche in Abb. 5.17 realisiert ist. Der benutzerdefinierte Metrikblock *PtII Actor Import* erwartet dabei zwei Eingänge. Die zu importierende Bibliothek *MoML Lib* ist eine XML-Datei, die die Klassenbeschreibung der Actors enthält. Diese wird entweder explizit angegeben oder, wie in Abb. 5.17 exemplarisch dargestellt, relativ zum Klassenpfad des PtII Plugins. Wird keine Datei angegeben wird, sofern noch nicht vorhanden, eine Basisbibliothek mit den grundlegenden Actors und Directors importiert. Der zweite Eingang gibt das Zielpaket an, in das die Actors als logische Funktionstypen der BLA-Ebene abgelegt werden. In Abb. A.1 im Anhang ist ein Ausschnitt der importierten PtII Actor-Bibliothek zu finden.

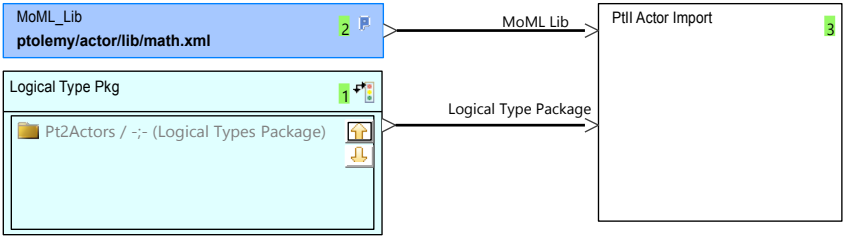


Abbildung 5.17: Metrik zum Import von PtII Actor-Bibliotheken sowie benutzerdefinierten MoML-Klassen.

5.3.3 Abbildungsvorschriften des ebenenübergreifenden Verhaltens

In diesem Abschnitt werden die Abbildungsvorschriften zur Überführung der ebenenübergreifenden Verhaltensmodelle der BLA-Ebene und der State Charts auf Artefakte im PtII Simulationsmodell zusammengefasst.

5.3.3.1 Abbildung der Behavioral Logical Architecture

Da die Behavioral Logical Architecture als LA einer ausführbaren Produktlinie in PREEvision® integriert wurde, basieren die Abbildungsvorschriften des spezifizierten Verhaltens auf LA Artefakten. Wie in Unterabschnitt 3.2.2.2 beschrieben, folgt die LA einem Typ-Instanz-Prinzip, vergleichbar mit dem Klassen-Instanz-Prinzip aus der Objektorientierung, welches auch bei PtII-Modellen zum Tragen kommt. Auf dieser Basis sind in Tabelle 5.1 die Abbildungsvorschriften zusammengefasst.

Tabelle 5.1: Abbildungsvorschriften zwischen Artefakten der Behavioral LA und PtII Artefakten (Quelle: erweiterte Fassung nach [BRB17]).

BLA Artefakt	Beschreibung	Ptolemy II Artefakt
`LogicalFunction`	Instanzen von atomaren, logischen Funktionstypen; repräsentieren eine Instanz eines konkreten Actors aus der Bibliothek, dieser kann atomar oder zusammengesetzt sein.	Instanz einer abgeleiteten `TypedAtomicActor` oder `TypedCompositeActor` Klasse
`LogicalFunction-BlockType`	Spezifiziert den Typ einer atomaren logischen Funktion; kapselt die konkrete Actor-Klasse, die instanziiert werden soll.	Konkrete Actor-Klasse bzw. ein als MoML-`class` spezifizierter, zusammengesetzter Actor.
`LogicalBuildingBlock`	Hierarchisiert atomare, logische Funktionen oder wiederum logische Building Blocks; repräsentiert gewöhnliche, zusammengesetzte Actors oder davon abgeleitete.	Instanz einer (konkreten) `TypedCompositeActor` Klasse
`LogicalBuildingBlockType`	Spezifiziert den Typ eines Building Blocks.	(Konkrete) `TypedCompositeActor` Klasse.
`LogicalProvidedPort`	Ausgangsport	`TypedIOPort` mit Flag `isOutput`
`LogicalRequiredPort`	Eingangsport	`TypedIOPort` mit Flag `isInput`
`LogicalBuildingBlock` Custom Attribute	Spezifiziert das Berechnungsmodell bzw. den Director eines ausführbaren, undurchlässigen Composite Actors.	Konkreter `Director`
`LogicalAssemblyConnector`	Verbindet logische Ports.	`TypedIORelation`
`GenericAttribute`	Parameter eines atomaren oder zusammengesetzten Actors.	`Parameter`

Bezüglich der Abbildung der Building Blocks ist anzumerken, dass diese standardmäßig in einen `TypedCompositeActor` transformiert werden, wenn der Typ des Building Blocks nicht explizit einer speziellen in PtII vorhandenen Klasse entspricht. Ein Beispiel hierfür ist ein Building Block vom Typ `ptolemy.domains.modal.modal.ModalModel` als Stellvertreter für ein hinterlegtes State Chart eines logischen Funktionstyps. Dieser Stellvertreter-Block wird während der Synthese in ein Modal Model übersetzt, welches intern das transformierte State Chart enthält. Eine weitere Anwendung dafür ist, die Instanz einer zusammengesetzten Actor-Klasse mit spezifischen Parametern oder Actors zu erweitern.

Neben der automatischen Synthese eines Directors in bestimmte zusammengesetzte Actors wie den Kommunikations- und Ausführungsaspekten (vgl. Unterabschnitt 5.3.6.3 und 5.3.7.2), kann es aufgrund der verwendeten Actors bzw. der zu modellierenden Funktion erforderlich sein, einen Director explizit zu modellieren.

Zur Modellierung des Berechnungsmodells bzw. des Directors eines zusammengesetzten Actors in einem BLA Building Block werden sog. *Custom Attributes* [175, Kap. 5.14] in PREEvision® ausgenutzt. Diese stellen benutzerdefinierte Attribute dar, welche optional einen Datentyp aufweisen und mit beliebigen Metamodell-Artefakten assoziiert werden können. Zudem ist es möglich, dass die Attribute nur innerhalb einer bestimmten Produktlinie den assoziierten Metamodell-Artefakten zur Verfügung stehen, um die Flexibilität der Produktlinien zu gewährleisten. Zur Abbildung der Director und ihrer wichtigsten Eigenschaften wurden daher benutzerdefinierte Attribute und Datentypen modelliert, welche nur die der BLA-Ebene verfügbaren Building Blocks der ausführbaren Produktlinie anreichern und nicht die in der regulären LA. Ein Ausschnitt der modellierten benutzerdefinierten *Domain* Attribute für Building Blocks ist in Abb. 5.18 zu sehen, in der die Spezifikation eines DE Directors für einen *IDMController* vom Typ *AccController* vorgenommen wird.

Generische Attribute (`GenericAttributes`) können sowohl auf dem Funktionstyp als auch auf der Instanz spezifiziert werden. Die auf dem Typ spezifizierten Attribute entsprechen dabei Parametern, die in der Actor-Klasse definiert sind und werden bspw. während dem Import von Bibliotheken auf dem Funktionstyp angelegt. Es können aber auch zusätzliche, für die Instanz geltende Parameter auf der jeweiligen Funktion angelegt werden. In der Regel werden Parameter auf Building-Block-Instanzen für zusammengesetzte Actors angelegt, welche dann von allen beinhaltenden Funktionen referenziert werden können.

Beim Instanziieren einer Funktion übernimmt PREEvision® allerdings die auf dem Typ definierten `GenericAttributes` nicht für die Instanz. Zur Vermeidung der manuellen Nachbearbeitung der Instanzen wurde daher eine Metrik entwickelt, die die `GenericAttributes` eines Funktionstyps für alle vorhandenen Instanzen automatisiert generiert, falls diese noch nicht angelegt wurden. Zusätzlich zur Metrik wurde ein sog. Metrikausführer [41, Kap. 6.2.14] konfiguriert, der es erlaubt, die Metrik in einen externen Kontext einzubetten, bspw. im Kontextmenü von bestimmten Artefakten. Ein Aufruf des Metrikausführers über das Kontextmenü eines Funktionstyps ist in Abb. 5.19 dargestellt.

5.3.3.2 Abbildung von State Charts

State Charts werden durch sog. Modal Models in PtII abgebildet und simuliert, die im Grunde hierarchische EFSMs darstellen und durch einen FSM Director

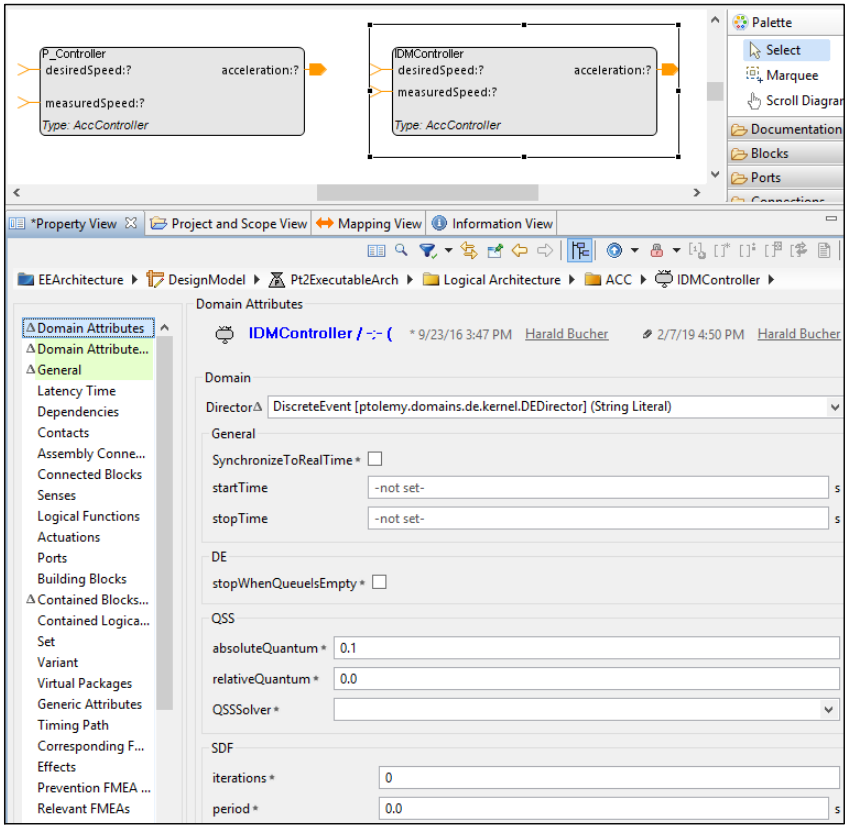


Abbildung 5.18: Ausschnitt der benutzerdefinierten *Domain* Attribute eines Building Blocks zur Spezifikation eines Directors und seinen Eigenschaften.

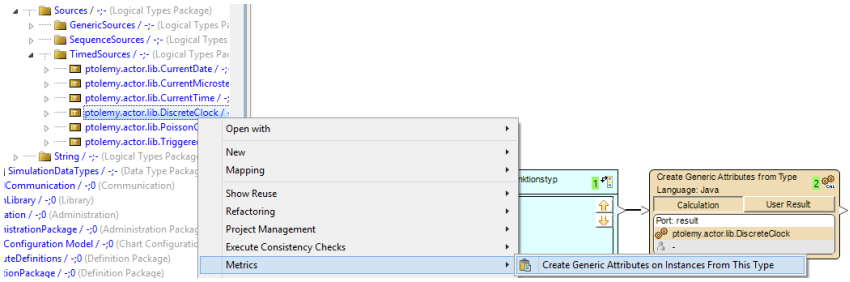


Abbildung 5.19: Aufruf eines Metrikausführers auf einem logischen Funktionstyp zur automatisierten Generierung der auf dem Typ definierten `GenericAttributes` in den vorhandenen Instanzen.

gesteuert werden. Jeder Zustand kann eine weitere Zustandsmaschine oder aber ein Actor-orientiertes Submodell, das einem anderen Director folgt, enthalten (vgl. Unterabschnitt 2.3.3.3). Da Modal Models spezialisierte, zusammengesetzte Actors sind, kommunizieren diese ebenso wie andere Actors über Ports miteinander.

In Tabelle 5.2 sind die grundlegenden Überführungsvorschriften zwischen State Chart Artefakten der UML Teilmenge und Modal Models in PtII zusammengefasst.

Tabelle 5.2: Abbildungsvorschriften zwischen der UML State Chart Teilmenge in PREEvision® und PtII Modal Models (Quelle: [BKB19c]).

UML State Chart Artefakt	PtII Modal Model Artefakt
einfacher state, choice & junction pseudo-state	state
initial pseudo-state	state mit Flag `isInitialState`
final state	state mit Flag `isFinalState`
composite state	*state machine refinement* state
orthogonal state	*default refinement* state
deep history state	history transition
(externe) state transition	ordinary transition
guard condition	guard expression
IO/variable action	output/set action expression

Ein orthogonaler Zustand, der häufig auch als AND-Zustand bezeichnet wird und aus mehreren nebenläufigen Regionen besteht, wird in einen hierarchischen Zustand mit einem sog. *default refinement* transformiert. Dieser verfeinerte Zustand

enthält ein Actor-orientiertes Submodell, welches für jede nebenläufige Region des State Charts ein eigenständiges Modal Model bereitstellt. Die Modal Models dieser Regionen werden entweder von einem DE oder *Synchronous-Reactive (SR)* Director [136, Teil II, Kap. 5] gesteuert, welcher die Semantik der Nebenläufigkeit während der Ausführung spezifiziert[5] [89]. Datenabhängigkeiten zwischen den Regionen werden während der Synthese analysiert und die betroffenen Modal Models kommunizieren diese Daten über entsprechende Ports. Die Datenabhängigkeiten können dabei während der Synthese basierend auf den referenzierten Datenelementen der State Charts als explizite Metamodell-Artefakte analysiert werden, im Gegensatz zum simplen Namensabgleich in der UML Semantik [89]. Die expliziten Referenzen erlauben es, die bereitstellenden und konsumierenden Regionen der Datenelemente eindeutig zu identifizieren und erhöht damit die Modularität des State Chart Designs.

Weitere UML State Chart Artefakte, wie z. B. lokale oder interne Transitionen, werden aktuell nicht unterstützt. Wichtig anzumerken ist schließlich, dass die Ausführungssemantik der State Charts während der Simulation durch die Semantik der Modal Models in PtII vorgeschrieben ist.

5.3.4 Generierung der Behavioral Logical Architecture

5.3.4.1 Generierung von Building Block Hüllen

Zur Reduzierung des Modellierungsaufwands der BLA-Ebene für den Entwickler wurde eine Metrik entworfen, die nach Abb. 5.8 sowohl die Building Block-Hüllen generiert als auch insb. die Port-Prototyp- und Block-Mappings zwischen der LA und der Verhaltensebene herstellt. Die Metrik unterscheidet demnach auch zwischen Actor-orientiertem und State Chart Verhalten. Für letzteres wird ein Building Block vom Typ `ptolemy.domains.modal.modal.ModalModel` je hinterlegtem State Chart eines logischen Funktionstyps generiert. Die Block-Mappings sind dabei initial und können bei Bedarf durch den Entwickler durch einen Building Block mit alternativer Realisierung ausgetauscht werden, während die Port-Prototypen-Mappings einmalig generiert werden. Das entsprechende Metrikdiagramm ist in Abb. 5.20 abgebildet. Die Metrik erwartet als obligatorische Eingänge die zu generierenden logischen Funktionen der LA und ein logisches

[5] In der vorliegenden Arbeit wurde der DE Director als nebenläufige Ausführungssemantik von Regionen gewählt, um potenzielle Probleme in Feedback-Schleifen zwischen Modal Models bei Verwendung des SR Directors zu vermeiden. Dieser folgt einer *fixed-point* Semantik, die ggf. eine iterative Ausführung der Feedback-Schleifen bis zur Konvergenz benötigt. Ein Modal Model muss dazu u. U. Annahmen bzgl. der Ausgänge treffen, ohne die dazugehörigen Eingangswerte zu kennen, was bei komplexen Modellen zu extrem schwer nachzuvollziehendem Verhalten führen kann [136, Teil II, Kap. 8.4.2].

Funktionspaket innerhalb der BLA-Produktlinie, in welchem die generierten Building Blocks abgelegt werden.

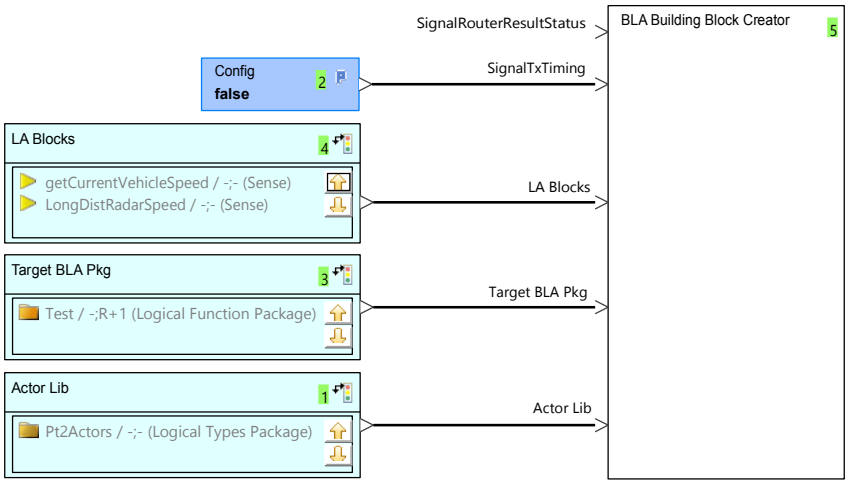


Abbildung 5.20: Metrik zur Generierung der Behavioral Logical Architecture und dazugehöriger Mappings basierend auf der logischen Architektur nach Abb. 5.8.

Komplementär dazu wurde ein Metrikausführer konfiguriert, somit lassen sich die BLA Artefakte und Mappings komfortabel über das Kontextmenü einer logischen Funktion erzeugen. Abb. 5.21 zeigt einen Screenshot für den Aufruf des Metrikausführers auf einer logischen Funktion. Der Metrikkontextblock *LA Blocks* wird in diesem Fall mit der logischen Funktion *GetCurrentVehicleSpeed*, auf welcher der Metrikausführer aufgerufen wird, überschrieben.

5.3.4.2 Berücksichtigung von Timing Constraints

Über den optionalen booleschen Parameter *SignalTxTiming* wird die Generierung von abstraktem, ereignisbasiertem Verhalten aktiviert, welches auf die modellierten Timing Constraints der logischen Architektur zurückgreift. Detailliertes, funktionales Verhalten per State Charts oder Actors wird dabei nicht adressiert.

Ausgangspunkt dafür sind die Beziehungen zwischen logischen Funktionen und deren spätere Kommunikation mittels auszutauschenden Signalen zwischen den jeweiligen Steuergeräten. Wie in Unterabschnitt 3.2.6.1 beschrieben geschieht

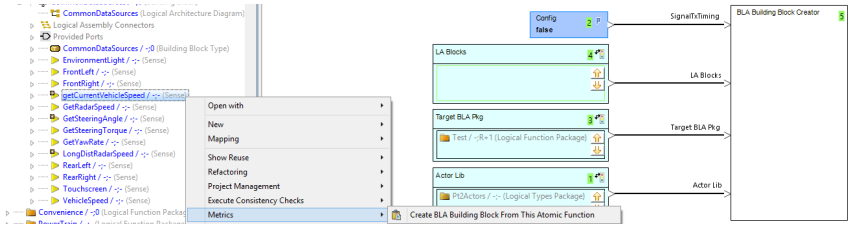


Abbildung 5.21: Metrikausführer zur Generierung eines BLA Building Blocks und dazugehöriger Mappings über das Kontextmenü einer logischen Funktion.

die Modellierung der Kommunikation zwischen logischen Funktionen analog zur Softwarearchitektur AUTOSAR-konform. Typisierte Ports tauschen über dedizierte Schnittstellen die Informationen in Form von Datenelementen oder Operationen aus. Abb. 5.22 veranschaulicht das Prinzip der Kommunikationsmodellierung auf LA-Ebene und die Realisierung auf Hardware-Ebene mithilfe von Signalen und Signaltransmissionen (vgl. Unterabschnitt 3.2.3.2). Die integrierte Signalrouter-Funktionalität erzeugt dabei die notwendigen Mappings der Datenelemente auf die auszutauschenden Signale, welche schließlich in Form von Signaltransmissionen zwischen den ECUs ausgetauscht werden.

Logische Port-Prototypen erlauben zusätzlich die Spezifikation von Timing Constraints, welche u. a. den Sendemodus und die Zykluszeit beinhalten und werden während dem Signalrouting basierend auf einer konfigurierbaren Strategie an alle dazugehörigen Signaltransmission als Signal-Timing weiter propagiert (vgl. Abb. 5.22 und [175, S. 1221 ff.]).[6] Während der Sendemodus und die Zykluszeit obligatorische Eigenschaften sind, sind weitere Angaben wie Offset oder Toleranz (absolut oder relativ) modellierbar. In Abb. 5.22 ist die Timing-Propagation basierend auf den Timing Constraints an Senderports gezeigt sowie die zusätzliche Angabe eines Offsets. Diese Art von Propagation resultiert in Signal-Timings mit der am Sender modellierten Zykluszeit von 100 *ms*. Im Vergleich dazu würde eine Propagation auf Basis der Timing Constraints von Empfängerports in diesem Beispiel zu einem Signal-Timing von 50 *ms* führen.

Ausgehend von einem erfolgreichen Signalrouting wird mithilfe dieser Informationen ein zyklisches Sendeverhalten in Form eines `DiscreteClock` Actors mit dem resultierenden Signal-Timing in den Building Block generiert. Bei aktiviertem Parameter *SignalTxTiming* ist daher zusätzlich die Angabe der Actor-Bibliothek notwendig. Damit der benutzerdefinierte Metrikblock in automatisierten Me-

[6] Analoges ist auch für die Software-Ebene anwendbar.

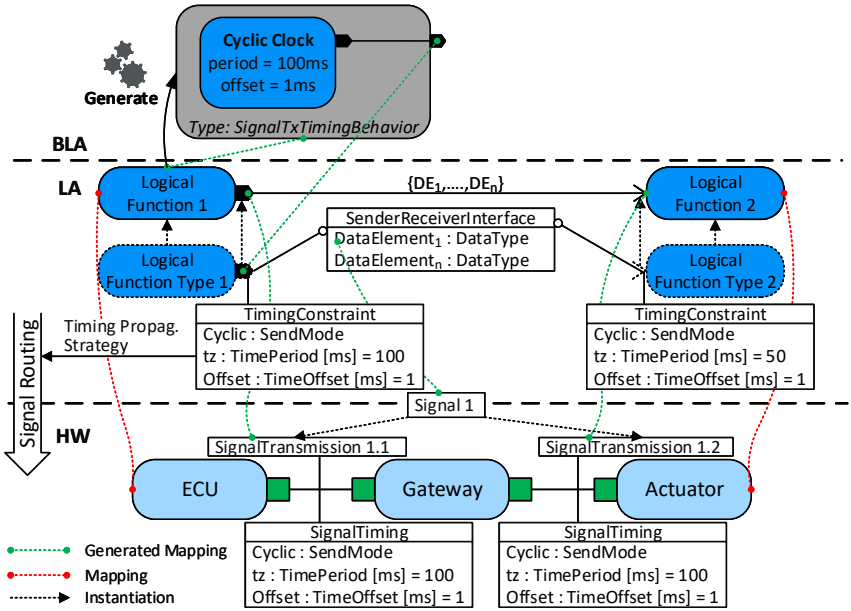


Abbildung 5.22: Prinzip der Generierung von zyklischem Sendeverhalten auf Basis von spezifizierten Timing Constraints an logischen Port-Prototypen (Quelle: [BNB19]).

trikberechnungen eingebettet werden kann, welche ein Signalrouting mittels des vordefinierten Signalrouter-Metrikblocks beinhalten, kann über den zusätzlichen Eingang *SignalRouterStatus* der Status abgefangen werden. Bei fehlerhaftem Routing wird die Generierung des Signal-Timing-Verhaltens abgebrochen.

Anzumerken ist schließlich, dass ein generierter Block mit internem Signal-Timing unter bestimmten Bedingungen regeneriert bzw. aktualisiert werden muss. Beispielsweise wenn die Timing Constraints der Port-Prototypen sich ändern, eine andere Propagationsstrategie im Signalrouter gewählt wird, die Mappings der Funktionen auf die ECUs sich ändern oder eine neue Architekturvariante aktiviert wird.

5.3.5 State Chart-Synthese und -Listener

5.3.5.1 Synthese

Die Überführung der State Charts in ein ausführbares PtII Modell während der Simulationssynthese erfolgt gemäß den Abbildungsvorschriften in Unterabschnitt 5.3.3. Vor der Überführung sammelt die Klasse `StateChartInterpreter` aus Abb. 5.2 des E/E-Modell Interpreters alle State Charts der logischen Funktionen und ggf. die dazugehörigen State Charts der abgebildeten Hardwarekomponenten. Zusätzlich liefert sie die dazugehörigen internen Zustände, Transitionen etc. mittels der UUID des jeweiligen State Charts über die öffentlichen Schnittstellen zurück. Die eigentliche Synthese und Überführung in das ausführbare PtII Modell ist in der Klasse `PREEvisionStateChartToPt2MomlDirector` realisiert, welche von der grundlegenden Synthese in der Klasse `PREEvisionToPt2MomlDirector` abgeleitet ist. Details zur prototypischen Implementierung der abgeleiteten Klasse, die als Basis für die Synthese der ebenenübergreifenden Verhaltensmodellierung in Unterabschnitt 5.2.5 und [BKB19a] dienen, sind in [Kam18, Kap. 5.2] zu finden.

Die Schnittstelle zwischen den State Charts und den dazugehörigen Architekturartefakten sowie die Abhängigkeiten innerhalb der State Charts in Form von Datenprovidern und -konsumenten bildet eine dedizierte, abstrakte Metamodell-Klasse. Dazu zählt die Klasse `MDataInstance`, welche Assoziationen zu weiteren Metamodell-Klassen herstellt, um abhängig vom konkreten Architekturartefakt auf einer Abstraktionsebene unterschiedliche Arten an Daten wie die Datenelemente der logischen Port-Schnittstellen in den State Charts verwenden zu können (vgl. Unterabschnitt 5.2.2.1). Ein vereinfachtes Metamodell-Klassendiagramm ist in Abb. 5.23 dargestellt.

Die linke Seite stellt dabei die Schnittstelle eines State Charts dar, welches bspw. Datenelemente eines Empfängerports konsumieren bzw. einlesen kann, während die rechte Seite die Schnittstelle eines State Charts repräsentiert, welches bspw. Datenelemente an einem dazugehörigen Ausgangsport setzt. Im Grunde existieren drei Typen von Assoziationen zur Klasse `MDataInstance`:

1. Abstrakte Datenprovider und -konsumenten: Dazu zählen die Datenelemente auf der LA- und Software-Ebene oder Signaltransmissionen auf der Hardware-Ebene, welche in einem State Chart referenziert werden können.
2. Abstrakte Provider- und konsumierende Ports: Dazu zählen die logischen Port-Prototypen aber auch logische, schematische und elektrische Hardware-Konnektoren (vgl. Unterabschnitt 3.2.6.2).

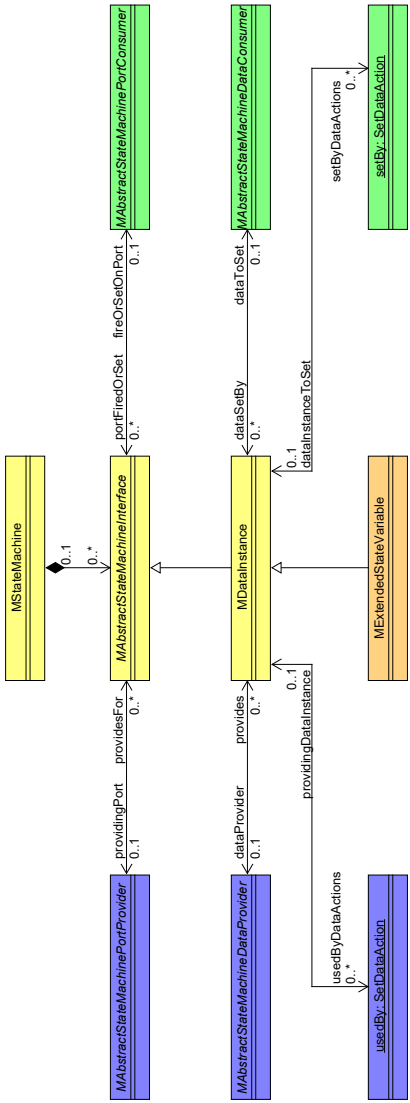


Abbildung 5.23: Vereinfachtes Metamodell der PREEvision® State Charts und deren Schnittstellen zu Architekturartefakten sowie die vorgeschlagene Ergänzung um erweiterte Zustandsvariablen (Quelle: eigene Darstellung nach [176]). Blau/grün: konsumierende/setzende Schnittstelle; orange: EFSM Erweiterung.

3. Data Actions, in denen die referenzierten Daten gesetzt oder gelesen werden.

Die Klasse `MExtendedStateVariable` repräsentiert dabei die vorgeschlagene Ergänzung um EFSMs durch Zustandsvariablen. Die Vererbung von `MDataInstance` hat den Vorteil, dass die Zustandsvariable als Schnittstelle integriert und die bestehenden Assoziationen benutzt werden können. Durch die Assoziation zu den abstrakten Datenprovider und -konsumenten können so bestehende Architekturartefakte, die nicht an Ports gebunden sind, bspw. UML Attribute in einem Klassendiagramm der Software-Ebene, als Zustandsvariable verwendet werden.

Die Assoziationen werden zusätzlich während der Synthese u. a. dazu verwendet, um Plausibilitätschecks durchzuführen. Beispielsweise werden die Basisservice Interfaces dahingehend überprüft, ob ein referenziertes Datenelement in einem Hardware-State Chart tatsächlich einem Basisservice Interface der abgebildeten logischen Funktion angehört, bevor die dazugehörigen Ports miteinander verbunden werden.

Im synthetisierten PtII-Modell werden die Funktions- und Hardware-State Charts je in dedizierten Modal Models gekapselt und kommunizieren via Ports miteinander, welche den Basisservice Interfaces entsprechen. Ein zusätzlicher Ausgangsport wird für die Simulation der quasi-statischen Stromaufnahme einer ECU generiert (vgl. Unterabschnitt 5.2.5.1).

5.3.5.2 Listener

In [BKB19a] wurde ein Listener-Entwurfsmuster, basierend auf [BNB19], umgesetzt, das dem in Unterabschnitt 5.4.2.2 präsentierten Konzept zur Rückführung der Simulationsdaten folgt, um diese in der PREEvision® Umgebung weiterverarbeiten zu können. Dabei wurde, wie in Abb. 5.34 skizziert, ein Listener für die `ModalController` Klassen von PtII umgesetzt, da die Ausführungsinformationen nicht über gewöhnliche Actors abgegriffen werden können. Analog greift ein Beobachter aufseiten von PREEvision® die zurückgeführten Daten als zusätzlicher Service ab und speichert sie in einer CSV-Datei (vgl. Abb. 5.35). Die zurückgeführten Daten beinhalten u. a. den Namen und die UUID jedes State Charts, den aktuellen und den im vorherigen Simulationsschritt aktiven Zustand, den Zeitstempel (Modellzeit) sowie die Output- und Variablen-Actions.

5.3.6 Analyse und Synthese der Netzwerktopologie

5.3.6.1 Vorbedingungen

Zur Berücksichtigung der realisierungsabhängigen vernetzten Steuergeräte-Topologie und den verwendeten Bussystemen während der Simulationssynthese, ist eine notwendige Voraussetzung eine konsistent modellierte logische Architektur bzgl. der Schnittstellen zwischen den Funktionen. Die dazugehörigen Datenelemente werden dann wie in Unterabschnitt 5.3.4.2 beschrieben durch die integrierte Signalrouter-Funktionalität Signalen zugeordnet. Basierend auf den Mappings der Funktionen auf ihre ausführenden Hardwarekomponenten werden die kürzesten Kommunikationspfade mit den geringsten Kosten bestimmt, auf denen die ECUs schließlich Signaltransmissionen austauschen. Ein korrektes Signalrouting stellt somit die hinreichende Bedingung zur Analyse der Kommunikationsbeziehungen zwischen den Funktionen in der vernetzten Steuergeräte-Topologie dar.

5.3.6.2 Analyse

Ausgehend vom bereits durchgeführten Signalrouting wird während der Simulationssynthese die Analyse der Kommunikationsbeziehungen zwischen den Funktionen durchgeführt. Dazu werden die bereits berechneten kürzesten Kommunikationspfade anhand der Signaltransmissionen zurückverfolgt. Abb. 5.24 veranschaulicht die dabei involvierten Artefakte, die bei der Zurückverfolgung berücksichtigt werden.

Ausgangspunkte sind vom Signalrouter zusätzlich angelegte, spezielle Mappings, die einen logischen Senderport mit jenen Signaltransmissionen verknüpft, welche von einer bestimmten Anbindung der Quell-ECU ausgehen. Analoges gilt für einen logischen Empfängerport und der dazugehörigen Signaltransmissionen, welche von einer bestimmten Anbindung der Ziel-ECU empfangen werden. Dies können sowohl konventionelle Verbindungen als auch Bussysteme sein. Abhängig von der zugrunde liegenden Netzwerktopologie kann eine Quell-Signaltransmission jedoch ein oder mehrere Gateways durchlaufen, die unterschiedliche Bussysteme und -segmente miteinander koppeln, bis das Signal die Ziel-ECU erreicht. Dazu werden die Signaltransmissionen in den Gateways durch sog. *Gateway Routing Entries (GRE)* zur empfangenden ECU geroutet. Ein Routing-Eintrag übersetzt dabei eine eintreffende Signaltransmission in eine ausgehende Signaltransmission, die jedoch i. d. R. einem anderen Bussystem oder auch einer konventionellen Verbindung angehört.

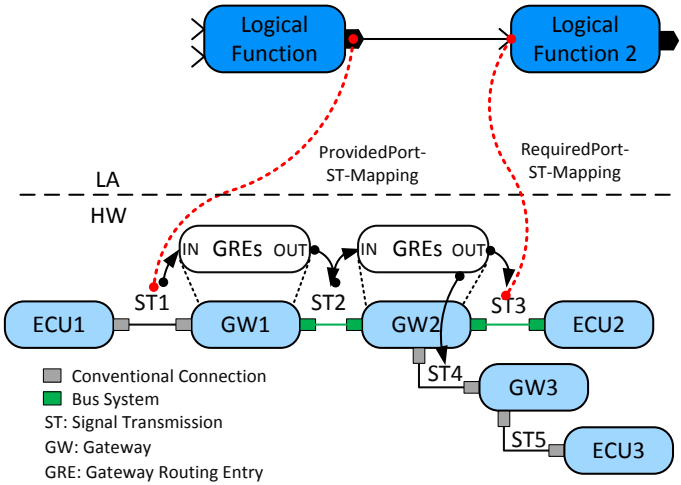


Abbildung 5.24: Analyse und Extraktion der realisierungsabhängigen Netzwerkkommunikation zwischen logischen Funktionen (Quelle: [BRB17]).

Zur Einbindung der Netzwerkkommunikation zwischen logischen Funktionen in der Simulationssynthese werden also alle Signaltransmission und die damit zusammenhängenden Bussysteme auf dem Kommunikationspfad zwischen Quell- und Ziel-ECU betrachtet. Dazu wird ein Breitensuche-Algorithmus eingesetzt, welcher auf einen Graphen bestehend aus einer Menge an GRE-Knoten und der dazugehörigen Menge an Signaltransmission-Kanten angewandt wird. Begonnen wird bei denjenigen GRE, welche als eingehende Signaltransmission die dem Senderport-Mapping zugehörige enthalten. Der Algorithmus wird solange ausgeführt, bis diejenigen Ziel-GRE gefunden sind, welche als ausgehende Signaltransmission die dem logischen Empfängerport zugehörigen enthalten. Zusätzlich muss die gefundene Ziel-Signaltransmission demselben Signal wie dem der Quell-Signaltransmission angehören. Der Breitensuche-Algorithmus findet dabei immer den kürzesten Weg zwischen Quell- und Zielknoten, da die Signaltransmissionen (Kanten) alle das gleiche Gewicht aufweisen [10, Teil III, S. 144]. Diese Vereinfachung ist gültig, da das gewichtete Routing der Signale bzgl. des kürzesten Pfades basierend auf einer Kostenfunktion und die damit einhergehende Generierung der Signaltransmissionen zuvor vom Signalrouter durchgeführt wurde.

Der Pseudo-Code zur Extraktion der Netzwerkkommunikation zwischen logischen Funktionen ist in [BRB17] zu finden. Der Algorithmus liefert sowohl die involvierten Signaltransmissionen als auch das dazugehörige Bussystem zurück, auf welchem eine Signaltransmission übertragen wird.

5.3.6.3 Synthese

Die nicht-funktionalen und realisierungsabhängigen Anteile des Verhaltens auf LA-Ebene werden wie in Abschnitt 5.1 beschrieben als aspektorientierte Simulation realisiert. Dazu werden die in PtII bereitgestellten sog. *Communication Aspects* (dt. Kommunikationsaspekte) und *Execution Aspects* (dt. Ausführungsaspekte) verwendet, die wie gewöhnliche Actors atomare oder hierarchisch zusammengesetzte Blöcke sein können. Die Kommunikationsaspekte werden dabei auf Empfängerports und die Ausführungsaspekte auf einen Actor selbst dekoriert.

Hinsichtlich der Netzwerktopologie wird zwischen zwei Funktionen pro extrahiertem Bussystem ein Kommunikationsaspekt Actor synthetisiert, in welchem die Netzwerkkommunikation gekapselt wird. Somit kann die Netzwerkkommunikation mit beliebigem Detailgrad, abhängig von den zur Verfügung stehenden Modellen, verfeinert werden.

Zur Simulation einer CAN-Kommunikation stellt PtII bspw. einen atomaren CAN-Bus Actor bereit, welcher im Rahmen dieser Arbeit u. a. um die Handhabung variabler Frame-Größen erweitert wurde. Alternativ zum zusammengesetzten Kommunikationsaspekt kann auch ein atomarer Aspekt synthetisiert werden, wenn keine zusätzliche Logik in Form eines Actor-Netzwerks notwendig ist. Empfängerports werden dabei mit den entsprechenden CAN-Bus-Aspekten jener CAN-Bussysteme dekoriert, welche während der Analyse extrahiert wurden. Die busspezifischen Eigenschaften wie Baudrate werden dabei mithilfe des zur Signaltransmission zugehörigen Bussystems bzw. des dazugehörigen Bus-Clusters[7] extrahiert und als Parameter des CAN-Bus-Actors synthetisiert. Frame-spezifische Eigenschaften wie CAN-Frame-ID und Größe des CAN-Frames werden mithilfe der zur Signaltransmission zugehörigen CAN-Frametransmission und des damit verknüpften CAN-Frames extrahiert, welche als Parameter des aspektdekorierten Empfängerports synthetisiert werden.

Die Synthese ist exemplarisch in Abb. 5.25 dargestellt und wird im Wesentlichen in den in Abb. 5.3 gezeigten Klassen `MomlCanBusAspectDecorator` und `MomlCanBusAspectPortDecorator` realisiert, die auf die analysierten und extrahierten Kommunikationsbeziehungen des `LAInterpreters` zurückgreifen. Als Director wird ein DE Director synthetisiert, um eine zeitbehaftete und ereignisba-

[7] Metaklasse `MBusSystem` bzw. `MCluster`.

sierte Simulation der Aspekte, welche in der Regel Verzögerungen enthalten, abzubilden [136]. Dabei wird ein Datenelement, das einem Port eines BLA Building Blocks entspricht, über den mit einem CAN-Bus-Aspekt dekorierten Empfängerport zum CAN-Kommunikationsaspekt umgeleitet und dort dem CAN-Protokoll folgend abhängig von der Baudrate und Frame-Größe verzögert. Zusätzlich zur CAN-Kommunikation ist in diesem Beispiel eine weitere Umleitung des Datenelements bzw. der dazugehörigen Signaltransmission in einen Gateway-Aspekt dargestellt, bis das Datenelement schließlich beim ursprünglichen Empfängerport des zusammengesetzten Actors *Function 2 Composite* empfangen wird.

Der verschachtelte Aspekt des Gateways enthält dabei eine entsprechende State Chart Verhaltensbeschreibung in Form eines Modal Models (vgl. Unterabschnitt 5.3.5.1). Der Aspekt kann aber auch eine einfache Routing-Verzögerung darstellen, wenn keine State Chart Beschreibung hinterlegt ist. Neben der ebenenübergreifenden Verhaltensmodellierung mithilfe der Basisservice Interfaces ist es somit aufgrund der Analyse der Netzwerktopologie auch möglich, Hardwarekomponenten zu berücksichtigen, die keine dazugehörige logische Funktion entlang des Kommunikationspfads besitzen.

5.3.7 Ausführungsaspekte von Funktionen

5.3.7.1 Abstrakte Ausführungsaspekte

Neben der ebenenübergreifenden Verhaltensmodellierung von State Charts, welche bereits eine detailliertere Beschreibung der E/E-Architektur darstellt und Ausführungsaspekte von Funktionen in Abhängigkeit von Steuergeräte-Betriebsmodi berücksichtigt, können auch abstraktere Ausprägungen von Ausführungsaspekten von Funktionen infrage kommen. Beispielsweise in früheren Entwicklungsstadien, wenn noch keine Hardwarearchitektur vorliegt oder eine gegebene Hardwarearchitektur noch nicht in ihrem Verhalten beschrieben worden ist. Verhaltensbeschreibungen von Funktionen in frühen Entwicklungsstadien enthalten zudem typischerweise keine internen Modelle über Ausführungszeiten, da diese ohne Implementierungsdetails und Informationen über die ausführende Hardwareplattform schwer abzuschätzen sind – selbst mit komplexen Modellen der Architektur in späteren Entwicklungsphasen [3].

Im Vergleich zu den Timing Constraints an logischen Port-Prototypen, die das Sendeverhalten einer Funktion an einzelnen Ports spezifizieren, wird ein Latenzattribut ausgenutzt, das die Latenz- bzw. die Ausführungszeit der gesamten Funktion spezifiziert. Somit lässt sich eine Ausführungslatenz von Funktionen in der Simulation berücksichtigen, ohne dass Hardwaredetails von ausführenden Komponenten verfügbar sein und ohne dass diese Verzögerungen explizit im

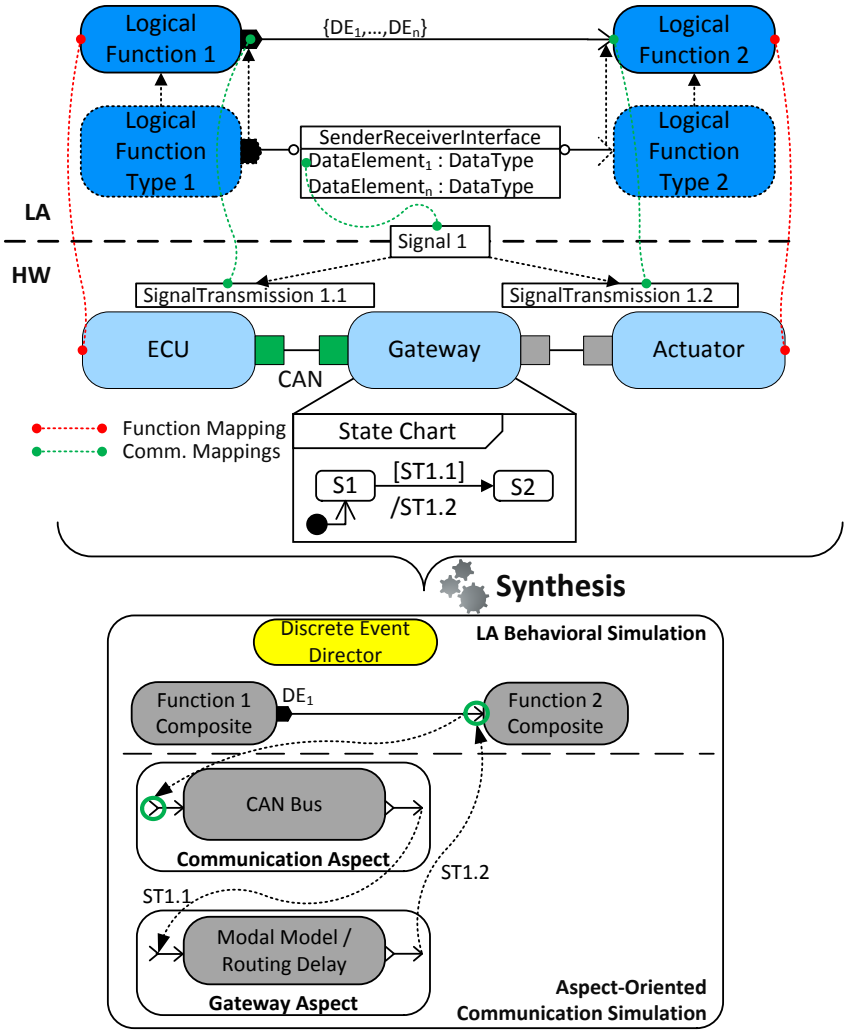


Abbildung 5.25: Schematische Darstellung der Synthese von Netzwerkkommunikation und weiteren Hardwarekomponenten zwischen zwei logischen Funktionen in Form von kaskadierten PtII Kommunikationsaspekten.

internen Verhalten der Funktion modelliert werden müssen. Die Latenz wird somit aus der Architekturspezifikation abgeleitet und als Anforderung angesehen. PREEvision® bietet zur Modellierung von Latenzen eine Eigenschaft *Latency Time* (Metaklasse `LatencyTime`), die die Spezifikation einer minimalen, typischen und maximalen Latenz erlaubt. Dies ist in Abb. 5.26 als Eigenschaft einer logischen Funktion exemplarisch dargestellt, welche eine maximale Ausführungszeit von 10 *ms* definiert.

5.3.7.2 Synthese und Funktionsweise von Ausführungsaspekten

Die Ausführungsaspekte werden während der Simulation in PtII durch die gleichnamigen Ausführungsaspekt-Actors abgebildet. Durch die Verwendung von zusammengesetzten Actors kann analog zu den Kommunikationsaspekten das zu verfeinernde Ausführungsverhalten über ein PtII-Submodell mit beliebigem Detailgrad[8] modelliert werden. Im Gegensatz zu den Kommunikationsaspekten werden diese auf den Actors selbst dekoriert.

Vor jedem Aufruf der `fire()` Operation eines Actors zu einem bestimmten Zeitpunkt durch den Director, kontaktiert letzterer zunächst den Ausführungsaspekt als eine Art Vermittler. Ein Actor wird dann so lange pausiert, bis der Vermittler diesen zur Ausführung freigibt und dem ausführenden Director signalisiert, dass dieser Actor fortgesetzt und gefeuert werden kann. Falls der Vermittler dem Director mitteilt, dass der Actor pausiert werden muss und nicht unmittelbar nach seiner Anfrage die notwendigen Ressourcen zur Ausführung erhält, fährt der Director mit dem nächsten Actor in der Warteschlange fort. Zwischenzeitlich eingetroffene Events am pausierten Actor werden dabei in ihrer zeitlichen Reihenfolge abgearbeitet, sobald der Vermittler dem zuständigen Director signalisiert, dass der Actor fortgesetzt werden kann [136, Kap. 10.4] [3].

Zusammengesetzte Ausführungsaspekte verwenden für Anfragen und zur Fortsetzung eines Actors sog. `ExecutionRequestPorts` und `ExecutionResponsePorts`, wobei diese als Token den Quell-Actor sowie dessen Ausführungszeit bereitstellen bzw. erwarten. Letztere wird im dazugehörigen Aspekt des dekorierten Actors als Parameter angegeben. Implementiert wird die Analyse und Synthese der Ausführungsaspekte in der Klasse `MomlAbstractExecutionAspectDecorator`. Die abstrakte Klasse analysiert mithilfe der Interpreter-Klassen, welche Art von Ausführungsaspekten auf die Funktionen angewandt werden, z. B. eine reine Latenz oder ein Multicore-Ausführungsaspekt (siehe Unterabschnitt 5.3.7.3). Die konkret zu synthetisierenden

[8] Beschränkt durch die Modellierungsmöglichkeiten von PtII. Detaillierteres, domänenspezifisches Verhalten kann bspw. durch Co-Simulation mit FMUs erreicht werden.

Ausführungsaspekte werden in spezialisierten Klassen implementiert[9] und über das Dekorierer-Entwurfsmuster dem abstrakten Dekorierer bei der Erzeugung bzw. Konfiguration als Verfeinerung übergeben. Die Synthese eines Latenz-Ausführungsaspekts, der eine Funktion um die als Eigenschaft angegebene Latenzzeit $t_{L,max}$ verzögert, ist in Abb. 5.26 links mit der *Logical Function 1* exemplarisch dargestellt. Als Director werden analog zu den Kommunikationsaspekten DE Director verwendet, um eine zeitbehaftete und ereignisbasierte Simulation zu ermöglichen. Diese kommen typischerweise zur Simulation digitaler Hardware/Software-Systeme zum Einsatz (vgl. Unterabschnitt 2.2.2.2).

Funktionen, die durch einen regulären BLA Building Block oder durch ein dem State Chart zugehörigen Modal Model Building Block realisiert sind, können gleichermaßen mit Ausführungsaspekten dekoriert werden. Generelle Voraussetzung ist lediglich, dass die resultierenden zusammengesetzten Actors *opaque* sein, d. h. einen Director besitzen müssen, da sie sonst keine ausführbare Einheit als Ganzes bilden und ein Ausführungsaspekt keinen Sinn ergeben würde (vgl. Unterabschnitt 2.3.2). Bei Modal Models ist dieser Umstand inhärent gegeben, da diese immer von einem *FSMDirector* gesteuert werden. Bei Funktionen, die über reguläre BLA Building Blocks realisiert sind und Ausführungsaspekte aufweisen sollen, muss ein entsprechender Director explizit modelliert werden.

5.3.7.3 Ausführungsaspekte gemeinsam genutzter Ressourcen

Eine weitaus interessantere Eigenschaft von Ausführungsaspekten ist die Möglichkeit, den Zugriff auf gemeinsam genutzte Ressourcen zu modellieren und ihre Auswirkungen auf das Verhalten der Applikation bzw. dessen Timing in der Simulation zu untersuchen.

Im Folgenden wird die erweiterte Verwendung von Ausführungsaspekten betrachtet, die die Zuweisung von Funktionen auf ihre ausführende Mikroarchitektur im E/E-Architekturmodell bereits auf der LA-Ebene berücksichtigt. Dies erlaubt die Analyse der spezifizierten Ausführungszeit von Funktionen durch ihre Ausführung auf gemeinsam genutzten Ressourcen zur Laufzeit sowie deren Einfluss auf das Verhalten.

Um dies zu realisieren wurde in PREEvision® die Möglichkeit geschaffen, Funktionen nicht nur auf Verarbeitungseinheiten oder Mikroprozessoren abzubilden, sondern auch einem internen Rechenkern eines Mikroprozessors zuzuweisen. Dazu werden Mappings via Modellkontexten zu den Kernen hergestellt, welche Sub-Mappings zu den nativ vorhandenen LA/Hardware-Mappings repräsentieren. Einerseits, um die Abwärtskompatibilität zu bereits bestehenden LA/Hardware-

[9] `MoMLExecutionLatencyOnlyAspectDecorator` und `MoMLMulticoreExecutionAspectDecorator`.

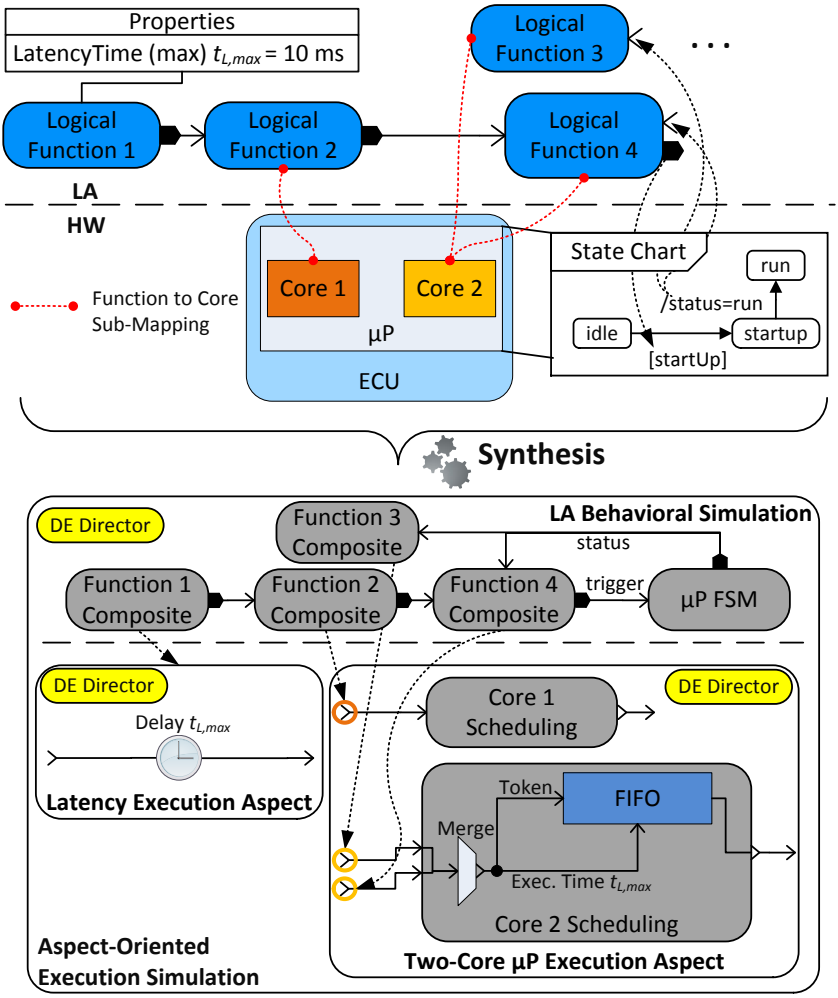


Abbildung 5.26: Exemplarische Darstellung der Synthese von Ausführungsaspekten unter Berücksichtigung gemeinsam genutzter Ressourcen von logischen Funktionen, abhängig von ihrer Abbildung auf ausführende Hardwarekomponenten, in Form von zusammengesetzten Ausführungsaspekten (Quelle: [BKB19c]),

Mappings und ihre Notwendigkeit für den Signal-Router zu gewährleisten. Andererseits stellen die Sub-Mappings eine Verfeinerung dar, die während der Synthese als zusätzliche Information zur Fallunterscheidung herangezogen werden, ob und welche Art von Ausführungsaspekten erzeugt werden müssen.

Die Abbildung von Funktionen auf Kerne ist in Abb. 5.26 durch die Zuweisung der logischen Funktionen zwei bis vier schematisch dargestellt. In der exemplarischen Funktionszuweisung wird während der Synthese ein zusammengesetzter Ausführungsaspekt-Actor erstellt, welcher intern zwei reguläre, zusammengesetzte Actors enthält, von denen je einer für die Abarbeitung von Funktionen auf einem Rechenkern zuständig ist. Abhängig von der Anzahl an Funktionen auf einem Kern werden ein oder mehrere `ExecutionRequestPorts` als Eingänge für die Funktionen im Ausführungsaspekt erzeugt. Die dargestellte Implementierung von Core 2 folgt dabei der in [3]: Die Funktionen werden im entsprechenden Kern zusammengeführt und über einen *First In First Out (FIFO)*-Puffer abgearbeitet. Die auf einer Funktion spezifizierte Latenz $t_{L,max}$ wird dabei als Ausführungszeit-Parameter des dazugehörigen Actors synthetisiert und entspricht der Bearbeitungszeit im FIFO.

Der Detailgrad, bis zu welchem die ausführende Hardwareplattform berücksichtigt werden kann, hängt naturgemäß von den Möglichkeiten der Modellierung im E/E-Architektur-Entwurfswerkzeug und von der möglichen Abbildung in ein ausführbares Modell im Zielsimulator ab. Nichtsdestotrotz sind ausführlichere Ausführungsmodelle erweiterbar, wie z. B. ein prioritätsbasiertes Scheduling der Funktionen. Durch die Berücksichtigung der Mappings von logischen Funktionen auf Softwarekomponenten der AUTOSAR-konformen Softwarearchitektur ist es möglich, die den Softwarekomponenten dort zugewiesenen Scheduling-Verfahren und Eigenschaften wie Prioritäten zu extrahieren und einen entsprechenden Aspekt-Actor zu synthetisieren. Ein prioritätsbasiertes Scheduling ist durch die Verwendung des `FixedPriorityScheduler` Ausführungsaspekts der PtII Actor-Bibliothek direkt möglich, wie bspw. in [11] demonstriert wurde.

Durch den modularen, aspektorientierten Ansatz lassen sich sowohl die abstrakten als auch die detaillierten Ausführungsaspekte leicht mit dem ebenenübergreifenden Konzept mittels Basisservice Interfaces kombinieren, wie es in Abb. 5.26 schematisch für das State Chart des Mikroprozessors dargestellt ist. Analoges gilt auch für die dazugehörigen Funktionen, die entweder Actor-orientiert oder als State Chart realisiert und mit Ausführungsaspekten dekoriert werden können.

5.3.7.4 Ausführungsaspekte von Modal Models

In der zum Zeitpunkt des Verfassens dieser Arbeit vorliegenden Version von PtII hat sich gezeigt, dass Ausführungsaspekte für Modal Models nicht akkurat

ausgelegt sind. Grund dafür ist die Ausführungssemantik von FSMs bzw. Modal Models in PtII (vgl. Unterabschnitt 2.3.3.3), im Speziellen von sog. schwach transienten Zuständen (engl. *weakly transient states*) [136, Kap. 6, S. 210]: bei jedem Feuern eines Modal Models verursacht eine auftretende Zustandstransition eine Anfrage im ausführenden Director, es zur selben Modellzeit erneut zu feuern, unabhängig davon, ob weitere Events an den Eingängen auftreten werden.

Im Falle eines zeitbehafteten ausführenden Directors bedeutet dies, dass eine Zustandstransition zu einem bestimmten Zeitpunkt (τ, n) eine erneute Ausführung im nächsten Mikroschritt mit dem Zeitstempel $(\tau, n+1)$ bewirkt (vgl. das Zeitmodell von PtII in Unterabschnitt 2.3.2.3). Dies stellt sicher, dass eine unmittelbar aktive Transition, z. B. eine ohne Übergangsbedingung, im Zielzustand zum Zeitpunkt $(\tau, n+1)$ sofort ausgeführt wird, ohne dass die Modellzeit fortschreitet. Ist der Zielzustand ein hierarchischer Zustand, werden seine Verfeinerungen ebenfalls zum Zeitpunkt $(\tau, n+1)$ gefeuert. Dies stellt sicher, dass z. B. die Transition vom initialen Pseudozustand eines State Charts ausgeführt wird, welche nach [127, S. 310 f.] keine Übergangsbedingung aufweisen dürfen, und sich der hierarchische Zustand somit im definierten Initialzustand befindet, bevor die Modellzeit voranschreitet. Zusammenfassend bewirkt das erneute Feuern nach Transitionen in weiteren Mikroschritten, dass sich die hierarchische FSM als Ganzes zum Zeitpunkt τ in einer stabilen Zustandskonfiguration befindet.

Diese intentionale und durchaus korrekte Semantik hat jedoch auch zur Folge, dass jeder mit einer Zustandstransition verbundene Wiederaufruf im nächsten Mikroschritt – zur selben Modellzeit – eine Kontaktaufnahme zum Ausführungsaspekt bewirkt, welcher pro Transition eine zusätzliche, künstliche Ausführungszeit verursachen würde. Diese zusätzlichen Latenzen, die von sog. puren Events ohne verknüpfte Tokens ausgelöst werden, sind nicht realistisch und auch nicht korrekt, da sie der ursprünglichen Semantik von Modal Models widersprechen[10].

Aus diesem Grund wurde in [BKB19c] eine Unterstützung für Ausführungsaspekte von Modal Models im ausführenden DE Director entwickelt. Darin werden Ausführungsanfragen von Modal Models, welche von den zuvor genannten puren Events ausgelöst werden, nicht erneut zum Ausführungsaspekt weitergeleitet. Stattdessen wird das Modal Model nach der ersten Kontaktaufnahme im Ausführungsaspekt, ausgelöst durch ein Eingangsport-Event oder ein Timeout-Event, unmittelbar im nächsten Mikroschritt gefeuert, sodass die zusätzlichen Latenzen vermieden werden. Somit wird eine stabile Zustandskonfiguration erreicht, ohne dass zusätzliche Modellzeit verstreicht.

[10] Die schwach transienten Zustände verursachen zusätzlich verstreichende Modellzeit, bevor eine stabile Zustandskonfiguration der FSM erreicht ist.

5.3.8 Elektrische und leitungssatzsensitive Synthese

In diesem Abschnitt wird beschrieben, wie die ebenenübergreifenden Konzepte der Verhaltensmodellierung zur elektrischen und leitungssatzsensitiven Simulation aus Unterabschnitt 5.2.5.2 und 5.2.5.3 umgesetzt werden. Das Konzeptbild zur Realisierung der Analyse und der aspektorientierten Synthese ist in Abb. 5.27 dargestellt.

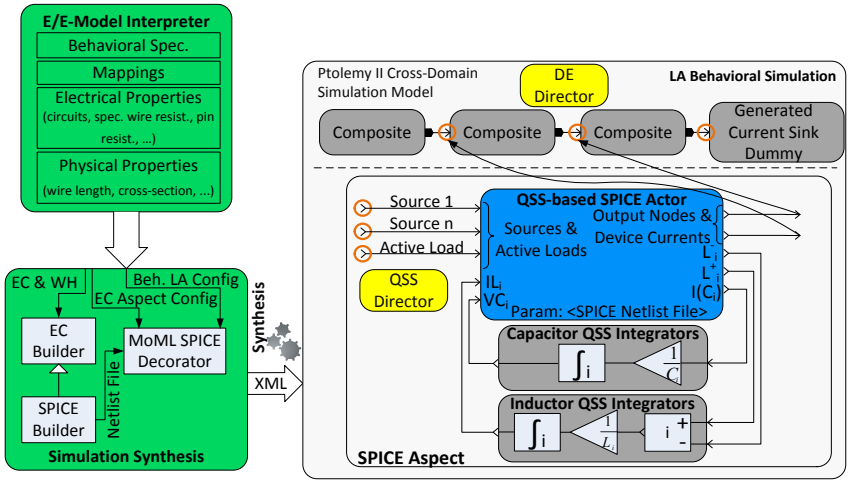


Abbildung 5.27: Schematische Darstellung der Synthese von elektrischer und leitungssatzsensitiver Simulation mithilfe eines zusammengesetzten Aspekt-Actors (Quelle: modifizierte Darstellung nach [BB18]).

Die elektrische Simulation wird dabei wiederum aspektorientiert als nichtfunktionale Verfeinerung des logischen Verhaltens auf LA-Ebene realisiert und trennt somit auch klar die verschiedenen, involvierten Domänen in der Simulation wie diskretes Verhalten auf der LA/BLA und kontinuierliches auf der elektrischen Ebene. Da die Verfeinerung der elektrischen Simulation auf Basis der eingeführten analogen Ports stattfindet, werden die Signale an den logischen Empfängerports an den Aspekt umgeleitet, in welchem die elektrische Simulation ausgeführt wird, bevor das durch die Schaltungen veränderte Signal im vorgesehenen Empfänger weiterverarbeitet wird. Dieses Prinzip ist vergleichbar mit den bereits vorgestellten Kommunikationsaspekten von PtII zur Realisierung der Netzwerkkommunikation und lässt sich auf diesen Anwendungsfall übertragen.

Daher wird für die elektrische Simulation ein zusammengesetzter Kommunikationsaspekt-Actor ausgenutzt, um die Signale der analogen Ports umzuleiten. Im zusammengesetzten Aspekt-Actor können nun wiederum adäquate Modelle zur elektrischen Simulation angewandt werden.

Hierzu wird ein neuartiger Ansatz vorgestellt, welcher ein SPICE-basiertes Simulationsmodell zur aspektorientierten elektrischen Simulation in einem *SPICE Aspect* Actor kapselt und mit QSS-Methoden verknüpft, was eine DE-basierte Simulation des logischen Verhaltens mit der elektrischen Ebene erlaubt (vgl. Abb. 5.27). Ein Hauptgrund für die Wahl zur Generierung einer SPICE-basierten Netzliste ist die Erhöhung der Wiederverwendbarkeit und Modularität der synthetisierten Netzliste, da diese mit jedem SPICE kompatiblen Simulator ohne signifikante Anpassungen ausgetauscht werden kann.

Im Folgenden wird detaillierter auf die Analyse des ebenenübergreifenden Verhaltens und die Synthese des SPICE-Aspekts eingegangen.

5.3.8.1 Modellanalyse

Für die Analyse des ebenenübergreifenden Verhaltens in Verbindung mit der detaillierten elektrischen Ebene, des Leitungssatzes und der Topologie ist der E/E-Model Interpreter zuständig. Im Wesentlichen sind dies die Klassen `ElectricCircuitInterpreter` und `WiringHarnessInterpreter` aus Abb. 5.2, die drei Hauptaufgaben erfüllen:

1. Parsen der modellierten Schaltungen auf elektrischer Ebene, die mit analogen Ports verknüpft sind, und ggf. des zwischen den analogen Hardware-Pins liegenden Leitungssatzes von konventionellen Verbindungen (vgl. Abb. 5.12) sowie die Generierung einer dazugehörigen Netzliste.
2. Anlegen von Datenstrukturen als Konfiguration für die Synthese des SPICE-Aspekts.
3. Anlegen von Datenstrukturen als Konfiguration für die Dekorierung der analogen Empfängerports mit dem SPICE-Aspekt.

Generierung der Netzliste Jeder analoge Senderport wird während der Interpretation in eine Spannungsquelle in der zu simulierenden Netzliste übersetzt, welche entsprechend dem modellierten Verhalten im BLA Building Block während der Simulation stimuliert werden. Der hauptsächlich für die Extraktion der angeschlossenen, elektrischen Schaltungen und des Leitungssatzes zuständige `ElectricCircuitInterpreter` erstellt während der Interpretation die restliche Netzliste. Diese wird schließlich für die Synthese des SPICE-Aspekts benötigt

(siehe Abb. 5.27). Zur Erzeugung der Netzliste wird das Builder-Entwurfsmuster verwendet, um eine Entkopplung zwischen Interpretation der elektrischen Hardwarearchitektur und der Generierung der zu simulierenden Netzliste zu erhalten.

Während in der vorliegenden Umsetzung eine SPICE-Netzliste erzeugt wird, sind auch andere Modelle zur Simulation von elektrischen Schaltungen denkbar. Um diese Möglichkeit offen und erweiterbar zu halten, werden über die Schnittstelle `IElectricCircuitBuilder` in Abb. 5.28 abstrakte Methoden und Klassenparameter zur Erzeugung der einzelnen Komponenten sowie der resultierenden Netzliste bereitgestellt. In einer abgeleiteten Klasse werden schließlich die Klassenparameter in die jeweiligen Klassen des Ziel-Metamodells aufgelöst und die Methoden implementiert. Zur zusätzlichen Entkopplung zwischen Interpreter und der tatsächlichen Implementierung zur Erzeugung der Netzliste bzw. des Zielmodells wird das Bridge-Entwurfsmuster angewandt (vgl. Abb. 5.28 unten).

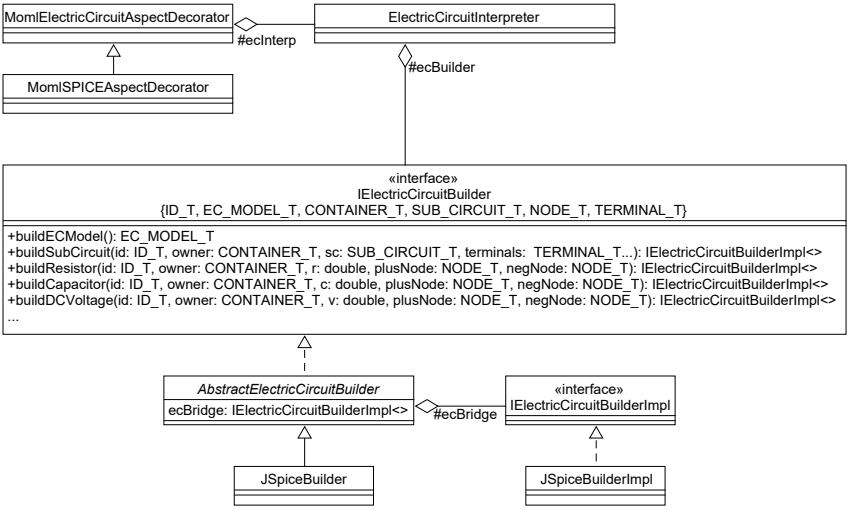


Abbildung 5.28: Klassendiagrammausschnitt zur Erzeugung von elektrischen Netzlisten mittels generischer Schnittstellen und zur Synthese des dazugehörigen Aspekts für die Verfeinerung des Verhaltens mit der elektrischen Simulation.

Die Generierung der SPICE-Netzliste ist durch die Klassen `JSpiceBuilder` bzw. `JSpiceBuilderImpl` realisiert, während die Synthese des SPICE-Aspekts in der abgeleiteten Klasse `MomlSPICEAspectDecorator` vorgenommen wird. Über die Assoziation ihrer Oberklasse zum Interpreter werden alle nötigen Datenstruktu-

ren für die Synthese, die während der Interpretation im vorherigen Schritt aufgebaut wurden, sowie die generierte Netzliste bezogen. Jeder abgeleitete Dekorierer ist dabei nur zu bestimmten Zielmodellen elektrischer Schaltungen und Netzlisten kompatibel bzw. dafür implementiert, was durch Kompatibilitätschecks abgefangen wird.

Interpretation der Netzliste **Vorbedingung:** Die Ausführung der Electric Circuit Synthese und des Leitungssatz-Routers (vgl. Unterabschnitt 3.2.2.4 und 3.2.2.4) vor der Interpretation des E/E-Architekturmodells sind eine notwendige Voraussetzung zur Einbindung des Leitungssatzes und Massenetzes. Darauf aufbauend werden die damit zusammenhängenden ohmschen Leitungswiderstände in die Netzliste integriert.

Wie in Unterabschnitt 5.2.5.2 beschrieben werden ausgehend von einem Hardware-Pin, der genau auf einen analogen Senderport abgebildet ist, die daran angeschlossenen, modellierten Schaltungen extrahiert und in die Netzliste generiert. Zusätzlich werden ggf. Leitungswiderstände und modellierte, elektrische Schaltungen auf der Ziel-ECU, falls sich der analoge Empfängerport nicht auf derselben ECU befindet, interpretiert und der Netzliste hinzugefügt. Ebenso werden die Verbindungen der Hardwarekomponenten zu Masse-Pins und damit zu ggf. vorhandenen Masseleitungen sowie die modellierten Stromverbraucher extrahiert und in die Netzliste aufgenommen. Dabei dienen die internen schematischen Verbindungen[11] von modellierten elektrischen Schaltungen als Knotenpunkte in der SPICE-Netzliste. Im Gegensatz dazu werden als Knotenpunkte für Leitungswiderstände die schematischen Pins bzw. Splices[12] verwendet (vgl. Abb. 5.13).

Mithilfe der durch die analogen Port Mappings eindeutig festgelegten Start- und Ende-Pins, dient die Spannung der simulierten SPICE-Netzliste am Knotenpunkt des Ende-Pins als Eingangsgröße für den analogen Empfängerport des dazugehörigen Actors. Diese Information wird u. a. im `ElectricCircuitInterpreter` zur späteren Synthese des SPICE-Aspektes gespeichert, um die korrekte Knotenspannung zurück an den Empfängerport zu leiten (vgl. Abb. 5.27).

Der Aufbau zur Interpretation der relevanten E/E-Architekturebenen EC, WH und der Topologie ist in den Klassendiagrammen in Abb. 5.29 und Abb. 5.30 zu sehen.

Ausgangspunkt ist dabei der `ElectricCircuitInterpreter`, der nach der Interpretation eine Reihe an getter-Methoden zur Verfügung stellt, die für die Konfiguration der Synthese des SPICE-Aspekts verwendet werden. Insbesondere stellt die-

[11] Metaklasse `InternalSchematicHWConnection`, vgl. Abb. 3.8.
[12] Metaklassen `SchematicPin` bzw. `Splice`, vgl. Abb. 3.9.

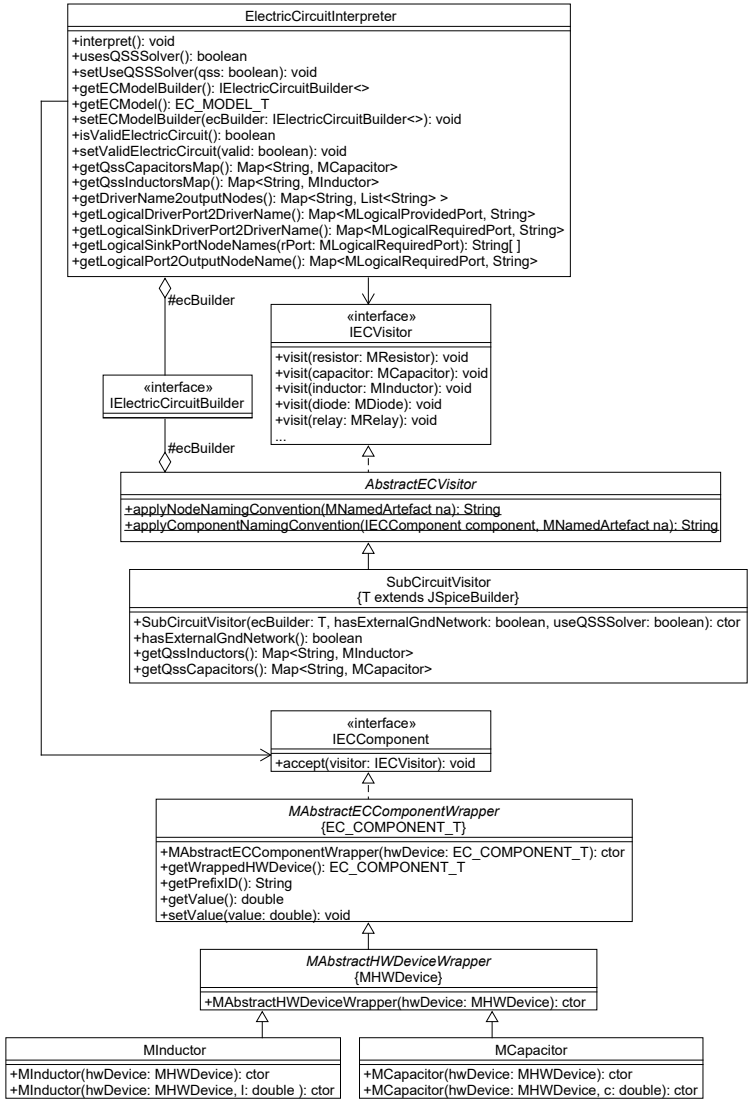


Abbildung 5.29: Klassendiagrammausschnitt zur Interpretation von elektrischen Schaltungen zwischen zwei analogen Ports mithilfe des Besucher-Entwurfsmusters.

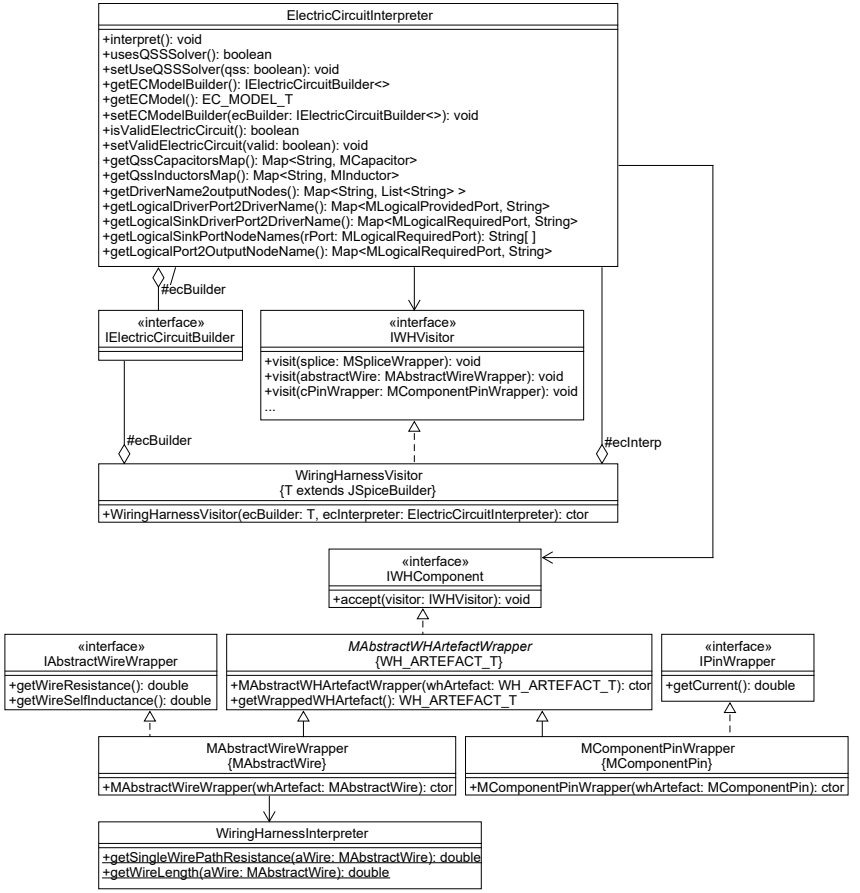


Abbildung 5.30: Klassendiagrammausschnitt zur Interpretation des Leitungssatzes zwischen zwei analogen Ports sowie des Massenetzes mithilfe des Besucher-Entwurfsmusters.

ser ein Flag `isValidCircuit` bereit, welches signalisiert, ob die erzeugte SPICE-Netzliste valide ist. Dazu wird eine Methode der Klasse `JSpiceBuilderImpl` ausgenutzt, die eine Menge von Plausibilitätsannahmen durchführt, bspw. ob eine Masse vorhanden ist oder ob alle Knotenpunkte an mindestens zwei Bauelementen angeschlossen sind. Durch die zuvor beschriebene Assoziation zum `IElectricCircuitBuilder` wird die Generierung der einzelnen Komponenten der Netzliste vom Prozess der Interpretation entkoppelt. Die konkrete Netzliste, in diesem Fall die SPICE-Netzliste, wird durch die Methode `getECModel()` zurückgeliefert.

Der `ElectricCircuitInterpreter` steuert den Prozess zur Extraktion der elektrischen Schaltungen zwischen den analogen Sender- und Empfängerports sowie die Generierung der Netzliste. Um eine weitere Entkopplung zwischen der Analyse der Hardwarearchitektur und der Behandlung der involvierten Artefakte zu gewährleisten, wird das Besucher-Entwurfsmuster eingesetzt. Die Behandlung von elektrischen Komponenten auf der EC-Ebene ist in Abb. 5.29 gezeigt, während die Behandlung des Leitungssatzes durch dedizierte Besucher in Abb. 5.30 dargestellt ist.

Die Schnittstelle `IECVisitor` deklariert dazu für jedes Element in einer elektrischen Schaltung eine entsprechende `visit()` Methode, die in der konkreten Klasse `SubCircuitVisitor` implementiert werden. Durch die Assoziation ihrer Basisklasse `AbstractECVisitor` zur Klasse `IElectricCircuitBuilder` haben die `visit()` Methoden Zugriff auf die Schnittstelle zur Erstellung der entsprechenden Netzlistenkomponente. Die zweite, notwendige Schnittstelle, die den Besucher eines Artefakts der elektrischen Ebene definiert, ist die `IECComponent` Schnittstelle. Diese müsste jedoch in der jeweiligen zugrunde liegenden Metamodell-Klasse implementiert werden, was in der verwendeten PREEvision® Entwicklungsplattform nicht möglich ist.

Um dies zu umgehen, und zudem eine generische Methode zur Behandlung beliebiger Metamodell-Artefakte darstellt, wird eine abstrakte Wrapper-Klasse `MAbstractECComponentWrapper` umgesetzt. Diese implementiert die Komponenten-Schnittstelle für den Besucher und hält eine Referenz auf die Basisklasse des zugrunde liegenden Metamodell-Artefakts vor. Diese Basisklasse wird zunächst durch den generischen Klassenparameter `EC_COMPONENT_T` abstrahiert und in der abgeleiteten Klasse `MAbstractHWDeviceWrapper` auf die abstrakte Basisklasse für elektrische Bauelemente in PREEvision® aufgelöst[13]. Die exemplarisch gezeigten, spezialisierten Klassen `MInductor` und `MCapacitor` sind schließlich die durch den Besucher behandelten Artefakte.

Ein analoges Vorgehen wird auf die Interpretation von Artefakten des Leitungssatzes in Abb. 5.30 übertragen. Die Analyse des Leitungssatzes und des

[13] Metaklasse `HWDevice`, vgl. Abb. 3.7

Massenetzes der involvierten Hardwarekomponenten geht zunächst auch vom `ElectricCircuitInterpreter` aus und folgt dabei der Beschreibung in Unterabschnitt 5.2.5.3. Sobald dieser eine an einen elektrischen Pin angeschlossene schematische Verbindung vorfindet, z. B. eine konventionelle Verbindung zwischen zwei analogen Ports oder eine Masseverbindung, werden die repräsentierenden Leitungen sowie damit verbundene Artefakte wie Splices und Pins durch den `WiringHarnessVisitor` behandelt.

Die Leitungswiderstände einzelner Leitungen werden in der Wrapper-Klasse `MAbstractWireWrapper`[14] bestimmt, welche auf die Klasse `WiringHarness-Interpreter` zurückgreift. In der Methode `getSingleWirePathResistance()` wird der Leitungswiderstand plus Übergangswiderstände der Leitungspins berechnet, sofern diese spezifiziert sind (vgl. Gl. (5.1)). Zusätzlich werden, sofern spezifiziert, die Übergangswiderstände der angeschlossenen Komponenten-Pins addiert. Darüber hinaus bestimmt die Methode die Gesamtlänge der gegebenen Leitung mithilfe der durch den Router festgelegten Topologie-Segmente, den über den Leitungstyp assoziierten spezifischen Widerstand, den Leitungsquerschnitt sowie die Übergangswiderstände der spezifizierten Pin-Typen[15].

Über die Schnittstelle `IPinWrapper` werden schließlich die über das Flag `isCurrentConsumer` spezifizierten, statischen Strombeschreibungen an den Pins extrahiert und als Stromverbraucher der Netzliste hinzugefügt. Wie in Abb. 5.13 dargestellt signalisieren die gesetzten Flags auch Endpunkte in der Interpretation des Leitungssatzes und erlauben daher die Einbindung dritter ECUs.

5.3.8.2 Synthese des SPICE-Aspekts und Simulationsprinzip

Die Synthese ist schematisch in Abb. 5.27 illustriert, während die Umsetzung weitestgehend durch den Dekorierer `MomlSPICEAspectDecorator` in Abb. 5.28 implementiert ist. Neben der eigentlichen Synthese des SPICE-Aspekts, werden die zu den analogen Senderports dazugehörigen Empfängerports mit dem SPICE Aspekt dekoriert, um die Signale zu den entsprechenden Eingangsports im Aspekt umzuleiten, in welchem sie schließlich als Stimuli für den SPICE Actor dienen. Darüber hinaus werden während der Synthese automatisch zusammengesetzte Dummy-Actors für jeden analogen Port erstellt und verknüpft, welcher als quasi-statischer oder dynamischer Stromverbraucher konfiguriert ist (siehe Unterabschnitt 5.2.5.2). Die Stromprofile der dazugehörigen Ausgangsports werden während der Simulation zum SPICE-Aspekt umgeleitet, in dem sie als Stimuli für

[14] Die Wrapper-Klasse repräsentiert die abstrakte Basisklasse `AbstractWire` einer Leitung im PREEvision® Metamodell, vgl. Abb. 3.9. Dies können je nach Modellierung der E/E-Architektur sowohl einzelne Leitungen sein als auch Leiter in einem Kabel.

[15] Metaklassen `WireType` und `PinType`, vgl. Abb. 3.10.

die dynamischen Stromverbraucher in der Netzliste dienen (vgl. die *Active Load* Eingänge im SPICE-Aspekt in Abb. 5.27).

Wie in Abb. 5.27 zu erkennen werden zur elektrischen Simulation der SPICE-Netzliste die in Unterabschnitt 2.2.2.4 eingeführten QSS Methoden zur numerischen Integration verwendet, welche in PtII durch die QSS Domäne implementiert sind (vgl. Unterabschnitt 2.3.3.2). Dies ermöglicht die DE-basierte Simulation der kontinuierlichen, elektrischen Modelle mit dem restlichen Verhalten unter gleichzeitiger Verwendung der beschriebenen Kommunikations- und Ausführungsaspekte. Die Hauptkomponenten im SPICE-Aspekt sind der in dieser Arbeit entwickelte QSS-basierte SPICE Actor, QSS Integratoren und der QSS Director, wobei die letzten beiden den Zeitfortschritt im Aspekt steuern.

Simulationsprinzip Der prinzipielle Ablauf der aspektorientierten, elektrischen Simulation ist die Ausführung einer Transientenanalyse der generierten SPICE-Netzliste, die durch die modellierten analogen Ports und Stromverbraucher durch Events stimuliert werden. Bei jedem empfangenen Event eines analogen Ports oder eines QSS Integrators wird genau eine Iteration der Transientenanalyse des SPICE Actors ausgeführt. Innerhalb dieser Iteration wird der Gleichstrom-Arbeitspunkt der linearisierten Schaltung (engl. *Direct Current Operating Point (DCOP)*) der Netzliste berechnet, bis der SPICE-Kernel konvergiert, d. h., wenn die Abweichung von zwei aufeinanderfolgenden DCOPs kleiner als ein Toleranzparameter ist.

Aufgrund der DE-basierten Simulation der elektrischen Schaltungen, ist der Modellierer der Signale von analogen Ports dafür verantwortlich, ein ausreichend hoch abgetastetes Signal als Stimuli für die SPICE-Co-Simulation zur Verfügung zu stellen, um akkurate Ergebnisse zu erzielen. Eine elegantere Lösung ist jedoch aufgrund des QSS-Ansatzes möglich, indem die Modellierung von stückweise stetigen Signalen mithilfe der in der QSS Domäne eingeführten `Smooth Tokens`. Damit wird ein Signal an einem Eingangsport des SPICE Actors zwischen zwei auftretenden Events mit Zeitstempeln t_1, t_2 mithilfe der spezifizierten Ableitungen in einem Smooth Token extrapoliert, wenn der SPICE Actor durch ein Event eines anderen Eingangssignals im Intervall (t_1, t_2) gefeuert wird[16].

QSS-basierter SPICE Actor Zur Simulation der SPICE-Netzliste wird JSpice [118] als Kernel des neu realisierten Actors eingesetzt – ein in Java implementierter, SPICE-basierter Schaltungssimulator, welcher sich auf das Rapid Prototyping

[16] Wenn der stückweise stetige Signalverlauf durch eine Sequenz von Smooth Tokens mit seinen Ableitungen exakt beschrieben werden kann, so ist auch die Extrapolation des Wertes zu einem Zeitpunkt zwischen zwei auftretenden Smooth Tokens exakt [87].

von Schaltungen fokussiert. Dazu verwendet dieser simple, linearisierte Äquivalenzschaltungen von nicht-linearen Bauteilen wie Dioden oder *Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs)*. Die Implementierung des Kernels in Java hat den Vorteil, dass dieser einfach in einen neuen Actor integriert werden kann und die Schnittstellen des Kernels direkt angesprochen bzw. einfach erweitert/angepasst werden können. Die Co-Simulation mit einem externen Werkzeug entfällt, während gleichzeitig die generierte Netzliste einfach in einen beliebig anderen SPICE-kompatiblen Simulator importiert werden kann. Der Actor erwartet dabei als obligatorischen Parameter die generierte Datei der Netzliste (vgl. Abb. 5.27).

JSpice verwendet zum Aufbau der Systemgleichungen der Netzliste die *Modified Nodal Analysis (MNA)* [50] Methode, welche in nahezu allen gängigen Schaltungssimulationen die Basis ist. Der Beitrag jedes Schaltungselements zur Systemgleichung wird dabei als Stempel (engl. *stamp*) bezeichnet, wobei JSpice im Wesentlichen die in [178] grundlegend vorgestellten Stempel verwendet. Die Systemgleichung kann dabei ausgedrückt werden als [196]:

$$\mathbf{A}\mathbf{x}(t) + \mathbf{C}\frac{d\mathbf{x}(t)}{dt} = \mathbf{B}\mathbf{u}(t) \tag{5.2}$$

wobei **A**, **B** und **C** Koeffizienten-Matrizen darstellen, welche die Stempel jedes Schaltungselements beinhalten. $\mathbf{x}(t)$ ist der Vektor an zu lösenden Unbekannten und $\mathbf{u}(t)$ der Vektor an bekannten Quellen, welcher alle unabhängigen Spannungs- und Stromquellen enthält. Während **A** die Admittanz-Matrix darstellt und zusätzlich Koeffizienten aller frequenzunabhängigen Schaltungselemente enthält, z. B. von gesteuerten Spannungsquellen, repräsentiert **C** die Koeffizienten der frequenzabhängigen Energiespeicherelemente in Form von Kapazitäten und Induktivitäten. **B** enthält schließlich die bekannten Werte der unabhängigen Spannungs- und Stromquellen. Die Unbekannten der MNA in $\mathbf{x}(t)$ beinhalten sowohl die Potentiale an Knoten als auch Ströme durch Bauteile wie Spulen oder Spannungsquellen.

Der SPICE Actor erwartet als Eingänge die zu stimulierenden Quellen sowie die aktiven, zu stimulierenden Lasten. Im vorliegenden Fall der Synthese sind dies Stromverbraucher, jedoch sind auch andere Formen von Lasten wie Verbraucher mit konstanter Leistungsaufnahme denkbar. Dabei muss der Name jedes Ports identisch mit der ID jenes Bauteils in der SPICE-Netzliste sein, das stimuliert werden soll. Die Ausgänge des Actors repräsentieren die berechneten Knotenpotentiale sowie die Ströme durch die Bauteile. Die Namen der Ausgänge müssen dabei den IDs der Knoten bzw. für die Ströme der ID des Bauteils in der Netzliste in der Form $I(< device\_netlist\_ID >)$ entsprechen (siehe Abb. 5.27).

An den Ausgängen produziert der Actor abhängig vom parametrisierten QSS Solver Smooth Tokens, welche entweder keine Ableitung (QSS1) oder, zusätzlich zum berechneten Wert einer Knotenspannung bzw. eines Stromes, die erste Ableitung (QSS2) und die zweite Ableitung (QSS3) enthalten. In den individuellen Ausführungsoperationen einer Iteration in PtII erfüllt der Actor folgende Aufgaben:

- `preinitialize()`:
  - Überprüft, ob der QSS-basierte SPICE Actor sich innerhalb eines zusammengesetzten Actors mit einem QSS Director befindet.
  - Setzt den Datentyp der Ports auf `double`[17].
- `initialize()`:
  - Setzt interne Actor-Zustände wie den zuletzt berechneten DCOP zurück, liest die SPICE-Netzliste ein und überprüft die Namen der Ports auf Konformität mit obiger Namenskonvention.
  - Instanziiert eine angepasste, abgeleitete Klasse der Schnittstelle für eine Transientenanalyse von JSpice, um genau eine Iteration in jener durchzuführen. Das heißt die Berechnung des DCOP zum aktuellen Zeitschritt, da der Zeitfortschritt vom QSS Director gesteuert wird.
- `fire()`:
  - Konsumiert alle verfügbaren Eingangs-Tokens und aktualisiert die Stimuliwerte aller zu den Ports zugehörigen Quellen in der Netzliste.
  - Berechnet den aktuellen DCOP mit den aktualisierten Eingangswerten mithilfe des JSpice Kernels.
  - Berechnet die Ableitungen (bis zur zweiten) der für die QSS Integratoren benötigten Smooth Tokens mithilfe der in der letzten Iteration berechneten DCOP- und Smooth Token-Werte und sendet die Tokens aller spezifizierten Knoten und Ströme an die dazugehörigen Ausgänge.
- `postfire()`:
  - Aktualisiert die internen Zustände des Actors wie etwa die zuletzt gesandten Smooth Tokens und den DCOP.

QSS Integratoren In einer Transientenanalyse werden reaktive Elemente wie Kapazitäten und Induktivitäten typischerweise durch eine Äquivalenzschaltung aus Spannungs- bzw. Stromquelle mit Reihen- bzw. Parallelwiderstand um den

[17] Ein Smooth Token ist von einem Double Token abgeleitet und wird daher unterstützt.

aktuellen Arbeitspunkt linearisiert. Die Äquivalenzschaltung und die Werte der Bauelemente hängen vom eingesetzten Solver zur klassischen, zeit-diskreten Integration ihrer Spannungs-/Strombeziehung ab. Aus diesem Grund hängt der Stempel der linearisierten Äquivalenzschaltung vom aktuellen Zeitschritt h des eingesetzten, zeit-diskreten Solvers ab [178].

Die Abhängigkeit der Stempel vom aktuellen Zeitschritt gilt für die gesamte MNA Systemgleichung und würde bedeuten, dass diese zur Laufzeit an diejenige DE-basierte Quelle angepasst werden muss, die den aktuell kleinsten Zeitschritt aufweist. Asynchron auftretende Events von Quellen machen diesen Umstand jedoch schnell komplex.

Um diese Abhängigkeit in der MNA während der Simulation zu vermeiden, werden die frequenzabhängigen Anteile der Kapazitäten $I_C(t) = C\frac{dV_C(t)}{dt}$ und Induktivitäten $V_L(t) = L\frac{dI_L(t)}{dt}$ aus Gl. (5.2) extrahiert und unabhängig voneinander durch QSS Integratoren gelöst. Statt der Zeit quantisieren diese den Zustand selbst und produzieren asynchron zueinander ein Event am Ausgang, sobald der quantisierte vom kontinuierlichen Zustand um ein Quantum abweicht[18]. Vergleicht man den Spannungs-/Stromzusammenhang der Kapazitäten und Induktivitäten mit der einer ODE nach Gl. (2.1), so ist zu erkennen, dass für eine Kapazität

$$\dot{x} = \dot{V_C} = \frac{1}{C} I_C(V,t) = f(x,t) \tag{5.3}$$

gilt. Analoges gilt für Induktivitäten. Da diese reguläre ODEs darstellen, können sie nach Gl. (2.6) (ohne Abhängigkeit von $\mathbf{u}(t)$) mit den QSS Methoden effizient gelöst werden. Über die Integration

$$V_C(t+\Delta t) = V_C(t) + \frac{1}{C}\int_t^{t+\Delta t} I_C(\tau)d\tau \tag{5.4}$$

und

$$I_L(t+\Delta t) = I_L(t) + \frac{1}{L}\int_t^{t+\Delta t} V_L(\tau)d\tau \tag{5.5}$$

erhält man somit mit den QSS Integratoren die Spannung über der Kapazität bzw. den Strom durch die Induktivität. Verbunden mit dem QSS-basierten SPICE Actor in Abb. 5.27 berechnet letzterer den Strom durch die Kapazitäten bzw. die

[18] Die QSS Integratoren in PtII können über einen zusätzlichen Parameter `propagateInputDerivatives` den am Eingang eintreffenden Smooth Token als Ableitung des integrierten Signals im gleichen Zeitschritt weiter propagieren. Dies ist möglich, da das stückweise stetige Signal am Eingang die Ableitung des Ausgangs darstellt und daher sofort weiter propagiert werden kann, bevor ein Quantisierungs-Event auftritt. Somit erhalten die empfangenden Actors des integrierten Signals eher aktualisierte Signalinformationen, was daher die Genauigkeit der Simulation erhöhen und gleichzeitig in einigen Fällen die Laufzeit weiter reduzieren kann [87].

Knotenpotentiale an den Induktivitäten als Eingang für die QSS Integratoren, welche wiederum die Spannung über den Kapazitäten bzw. den Strom durch die Induktivitäten als Eingang für den SPICE Actor liefern.

Da die QSS*m* als auch die LIQSS*m* Methoden explizite Integrationsverfahren darstellen (siehe Unterabschnitt 2.2.2.4 und [111]), können die Kapazitäten und Induktivitäten während der Synthese je durch eine simple Äquivalenzschaltung aus Spannungs- bzw. Stromquelle in der Netzliste ersetzt werden. Überträgt man den SPICE-Aspekt in Abb. 5.27 auf das generische Blockschaltbild in Abb. 2.9, entspricht der SPICE Actor im Wesentlichen den statischen Funktionen zur Berechnung von $f(\mathbf{q}(t), \mathbf{u}(t), t)$ aus Gl. (2.6), wobei die Ausgänge der QSS Integratoren $\mathbf{q}(t)$ und die Eingangssignale der analogen Ports $\mathbf{u}$(t) repräsentieren.

Der vorgestellte Ansatz bietet dabei folgende Vorteile:

- Es ist lediglich ein frequenzunabhängiges, lineares Gleichungssystem durch den SPICE Actor zu lösen, sobald ein Event eines QSS Integrators oder eines analogen Ports eintrifft.
- Ein zusätzlicher Director zur Lösung von algebraischen Schleifen in elektrischen Schaltungen wie in [87] vorgestellt ist nicht notwendig, da diese durch den SPICE Actor implizit aufgebrochen werden. Dieser produziert Smooth Tokens an seinen Ausgängen, sobald der SPICE Kernel zum aktuellen DCOP konvergiert.

5.3.9 Limitierungen

In den folgenden Unterabschnitten werden Limitierungen bzgl. der Implementierung von einzelnen Konzepten der Modellierung und Synthese erläutert.

5.3.9.1 Generierung der Behavioral Logical Architecture

Die Synthese von Port-Prototypen-Mappings unterstützt nur ein Datenelement. Die automatische Generierung von Verhalten über Timing Constraints unterstützt aktuell ausschließlich zyklische Sender. Weitere Sendemodi wie spontane oder ereignisgesteuerte Sender sind jedoch auf Grundlage der modellierten Architekturinformationen erweiterbar.

5.3.9.2 Analyse und Synthese der Netzwerktopologie

Die aktuelle Implementierung unterstützt CAN-Bus-Aspekte und abstrakte Gateway-Aspekte in Form von einfachen Routing-Verzögerungen, welche als

Latenzattribute auf einem Gateway modelliert werden. Die Einbindung von State Charts kann aber aufgrund der modularen Konzeption der Synthese über Dekorierer analog erfolgen. Auch die Berücksichtigung weiterer Bussysteme ist erweiterbar, sobald entsprechende Simulationsmodelle implementiert sind.

5.3.9.3 State Chart-Synthese

Die Synthese von State Charts wurde auf Grundlage einer vorläufigen Alpha-Version von PREEvision® v9.0 umgesetzt, da zum Zeitpunkt der Entwicklung noch kein finales Release zur Verfügung stand. In dieser Alpha-Version erfolgte die Spezifikation von Guard- und Action-Ausdrücken durch Strings, die in der finalen Version jedoch durch explizite Metamodell-Artefakte repräsentiert werden. Zur korrekten Synthese von State Charts bzw. deren Guards und Actions in der offiziellen v9.0, muss daher eine Portierung auf das aktuelle Metamodell erfolgen. Die grundlegende Abbildungsvorschrift auf Modal Models bleibt jedoch bestehen. Die Abbildung von Zustandsvariablen zur Realisierung von EFSMs wurde aufgrund der fehlenden Möglichkeit zur Metamodellerweiterung durch generische Attribute auf dem zum State Chart gehörigen BLA Building Block umgesetzt, die während der Synthese als Parameter des Modal Model Actors angelegt werden.

5.3.9.4 Ausführungsaspekte von Funktionen

Bei der Synthese eines abstrakten Ausführungsaspektes in Form einer einfachen Latenz, wird standardmäßig die maximal spezifizierte Latenz als worst-case angenommen.

Die Berücksichtigung von Mappings logischer Funktionen auf Kerne ist aktuell begrenzt auf die Synthese bzw. Simulation des FIFO-Schedulings. Durch die Abbildung des Schedulings mithilfe von Ausführungsaspekten ist die Erweiterung auf weitere Verfahren über dedizierte Dekorierer einfach, ohne den restlichen Syntheseprozess zu beeinflussen. Aufgrund des modularen Schichtenaufbaus sowohl des E/E-Architekturmodells als auch der in dieser Arbeit entwickelten Softwarearchitektur der Interpreter, können Mappings von logischen Funktionen zur Software-Ebene in der Klasse `SWAInterpreter` berücksichtigt werden. In Kombination mit einem Dekorierer können so spezifische Scheduling-Verfahren wie prioritätsbasierte Verfahren mit in die Simulation synthetisiert werden. Ein entsprechender `FixedPriorityScheduler` Ausführungsaspekt wird bereits durch die PtII-Bibliothek bereitgestellt.

5.3.9.5 Elektrische und leitungssatzsensitive Simulation

Die Analyse und Generierung der elektrischen Netzliste unterstützt aktuell konventionelle Verbindungen, jedoch nicht leistungsversorgende Verbindungen zwischen Hardwarekomponenten. Zur Berücksichtigung von Leitungswiderständen in der Simulation müssen daher ECUs etc. und Energiequellen wie Batterien und Generatoren über konventionelle Verbindungen angebunden werden. Zur Erweiterung von leistungsversorgenden Verbindungen muss allerdings lediglich die Interpretation der logischen Ebene der Hardwarearchitektur während der Synthese angepasst bzw. erweitert werden, da die Repräsentation auf der elektrischen Ebene und im Leitungssatz dieselben Metaklassen verwenden (vgl. Abb. 3.9).

Die Synthese bzw. Simulation von elektrischen Verbrauchern sind auf Stromverbraucher begrenzt. Entweder durch die native Erweiterung im PREEvision® Metamodell oder durch die Definition von *Custom Attributes* auf Hardwarekomponenten wäre auch eine erweiterte Synthese von anderen Verbraucherarten möglich, z. B. konstante Leistungsverbraucher.

5.4 Dynamische metrikbasierte Variantenbewertung

Auf Grundlage der bisher vorgestellten Konzepte zur integrierten Verhaltensspezifikation und Simulation ist es nun möglich, dynamische Metriken von E/E-Architekturen zu analysieren. Damit diese auch mit weiteren Metrikberechnungen basierend auf den statischen Attributen des Architekturmodells kombiniert werden können, ist ein Konzept notwendig, um die simulierten Ergebnisse integriert und transparent im E/E-Architekturentwurfswerkzeug verfügbar zu machen. In Kombination mit der variantensensitiven Simulationssynthese wird somit eine Variantenbewertung und ein Variantenvergleich ermöglicht. Das Konzept zur Adressierung der dazugehörigen Anforderungen aus Unterabschnitt 4.3.6 wird im Folgenden genauer beschrieben und wird als AMiL bezeichnet.

5.4.1 Architecture-Model-in-the-Loop Konzept

Ausgangspunkt und Voraussetzung für eine automatisierte Variantenbewertung und einen Variantenvergleich ist ein variantenreiches E/E-Architekturmodell, das über weitere Mechanismen individuelle Architekturvarianten und Realisierungsalternativen bereitstellen kann. Der Prozess zur Variantenbildung in PREEvision® folgt bspw. einem Produktlinienansatz, wobei einzelne Architekturartefakte einfach aber effektiv über ein Flag als aktiver und inkludierter Bestand-

teil einer aktuellen Variante markiert werden können (siehe Unterabschnitt 3.2.4 und 4.3.4). Bei der EAST-ADL kommt ein sog. *Variability Resolution Mechanism* Modul zum Einsatz, das aus einem variantenreichen Modell mehrere separate, invariante EAST-ADL-Modelle erzeugt (vgl. Unterabschnitt 3.4.3).

Bei dem nachfolgend vorgestellten AMiL Konzept, welches in Abb. 5.31 dargestellt ist, wird ein integrierter Ansatz verfolgt und erwartet als Eingangsgröße eine Menge an individuellen Architekturvarianten als Untermenge des gesamten 150 % E/E-Architekturmodells. Der integrierte Charakter bietet u. a. die Vorteile, dass nur ein einziges E/E-Architekturmodell vorgehalten werden muss, die Ergebnisse der Metrikberechnungen transparent mit den ursprünglichen Architekturartefakten verknüpft und iterative Modellanpassungen vorgenommen werden können (vgl. Unterabschnitt 3.4.3).

Die AMiL Bewertung mittels Metrikberechnungen in Abb. 5.31 besteht prinzipiell aus zwei Schleifen:

- Die *äußere* Schleife iteriert über die zur Verfügung gestellten Architekturvarianten, die untersucht und bewertet werden sollen. Die Ergebnisse der inneren Schleife werden in einem nachgelagerten Nachbearbeitungs- und Vergleichsblock weiter aufbereitet oder können zur externen Aufbereitung exportiert werden.
- Die *innere* Schleife führt die eigentliche Analyse und Metrikberechnungen einer aktiven Variante durch, sowohl für (quasi-)statische Metriken als auch für dynamische mithilfe der Simulationsergebnisse. Als Resultat liefert die innere Schleife eine Tabelle bestehend aus den Teilergebnissen der konfigurierten Metriken für jede Architekturvariante. Tabellen sind eine effiziente und effektive Methode große Datensätze verständlich darzustellen und zu vergleichen. Dies wurde bereits in früheren Arbeiten wie [102] erfolgreich gezeigt. Darüber hinaus sind Tabellen u. a. zentrale Elemente in PREEvision® und insb. des Metrik-Frameworks.

5.4.1.1 Inner Loop Handler

Der *Inner Loop Handler* repräsentiert die zentrale Steuereinheit der inneren Schleife und ist verantwortlich für die Koordination der Metrikberechnungen und der Simulationsausführung. So werden bspw. Parameter und Konfigurationen für die beinhalteten Metrikberechnungsblöcke gesetzt, das Abbruchkriterium der aktuellen Variante überprüft und ggf. weitere Metrikberechnungen bzw. Simulationsiterationen getriggert. Schließlich werden die Ergebnistabellen der Metrikberechnungen an die äußere Schleife weitergeleitet. Die Realisierung des Inner Loop Handlers ist dabei abhängig von den involvierten Metriken und des

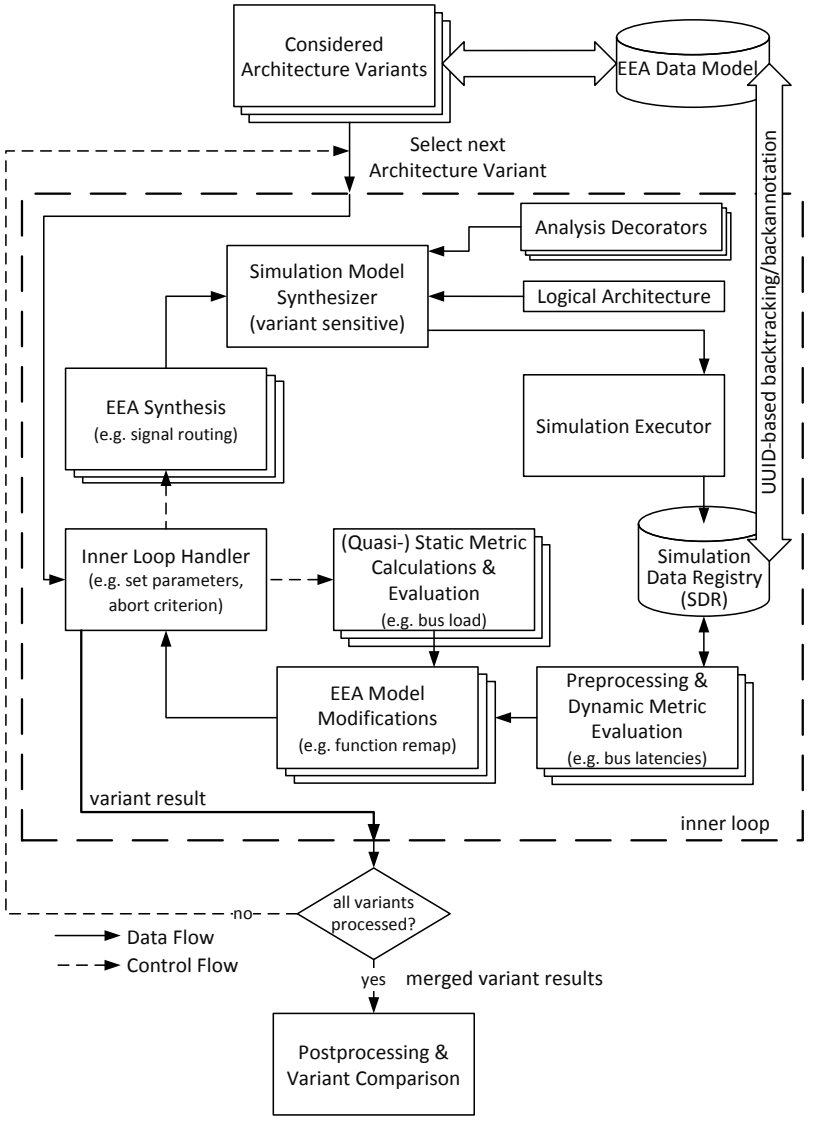


Abbildung 5.31: Ansatz zur integrierten und automatisierten Bewertung von E/E-Architekturvarianten bzgl. statischer und dynamischer Metriken gleichzeitig (Quelle: [BNB19]).

damit zusammenhängenden Abbruchkriteriums und ist deshalb individuell an die zu betrachtende Metrikanalyse anzupassen. Zum Beispiel die Verbindungen zu den eigentlichen Metrikberechnungsblöcken und deren Konfiguration mit weiteren Architekturartefakten und Parametern.

- **Eingangsgrößen**
  - Aktuelle Architekturvariante.
- **Ausgangsgrößen**
  - Von Metrikanalyse abhängige Konfigurationen und Architekturartefakte der aktiven Variante als Eingangsgrößen für angeschlossene Metrikblöcke.
  - Ergebnistabelle der Metrikanalysen der aktiven Variante.

5.4.1.2 EEA-Syntheseblöcke

Der erste Schritt nach einer Iteration der inneren Schleife oder nachdem eine neue Variante von der äußeren Schleife aktiv wird, sind dedizierte Syntheseoperationen, um ein aktualisiertes E/E-Architekturmodell und damit ein konsistentes Simulationsmodell für die dynamischen Metrikberechnungen zu erhalten. Diese Operationen können bspw. ein erneutes Signal-Routing nach Anpassung der Funktionsverteilung auf das Steuergeräte-Netzwerk innerhalb einer aktiven Variante beinhalten, um die Kommunikationspfade zwischen den Funktionen zu aktualisieren. Deshalb werden diese Blöcke vom Inner Loop Handler in der Ausführungsreihenfolge vor den Metrikberechnungsblöcken und vor der Simulationssynthese getriggert.

Ähnlich zum Inner Loop Handler sind die EEA-Syntheseblöcke auf das vorliegende Analyseszenario zuzuschneiden. Wenn bspw. keine Netzwerkkommunikation in der Simulation berücksichtigt oder keine Buslast berechnet wird, so kann das Signal-Routing als Syntheseblock entfallen. Andernfalls ist dies zum Erhalt von konsistenten Kommunikationsbeziehungen zwingend notwendig. Auch die Ausführungsreihenfolge bei mehreren EEA-Syntheseoperationen kann wichtig sein und muss bei der Modellierung der inneren Schleife beachtet werden, etwa wenn abhängig von einem Signal-Routing Verhaltensblöcke auf BLA-Ebene erzeugt werden sollen. Zusätzlich können abhängig von der Syntheseoperation weitere Architekturartefakte als Eingangsgröße notwendig sein, auf die die Synthese angewandt wird, z. B. die logische Architektur für das Signal-Routing.

- **Eingangsgrößen**
  - Konfiguration der EEA-Syntheseblöcke.

- **Ausgangsgrößen**
  - Ergebnisstatus der Syntheseoperation und ggf. -ergebnis als komplexe Datenstruktur.

5.4.1.3 Simulation Model Synthesizer

Der *Simulation Model Synthesizer* ist für die Interpretation des E/E-Architekturmodells sowie für die Synthese des Simulationsmodells zuständig und erfüllt somit die Aufgaben der in Unterabschnitt 5.1.3 und 5.1.4 vorgestellten Konzepte des E/E-Model Interpreters bzw. der Simulationssynthese. Die notwendige Eingangsgröße für die Simulationssynthese ist die realisierungsunabhängige logische Architektur bzw. die logischen Funktionen des zu simulierenden Teilsystems. Dabei kann ein variantenreiches LA Modell, z. B. in Form eines 150 % Modells, als Eingang verwendet werden, welches über die variantensensitive Synthese nur die relevanten logischen Funktionen in der aktiven Variante berücksichtigt. Dies gilt auch für die über die Mappings assoziierten Artefakte auf allen Abstraktionsebenen, die während der Synthese von den Interpretern analysiert werden.

Zusätzlich zur LA sind optional ein oder mehrere *Analysis Decorator* (dt. Analyse-Dekorierer) vorgesehen, die das Simulationsmodell während der Synthese abhängig vom Typ des Analyse-Dekorierers mit zusätzlichen Actors instrumentiert. Im einfachsten Fall kann ein Analyse-Dekorierer ein simpler Monitor sein, der vom Benutzer manuell auf BLA-Ebene modelliert wird und der entsprechend abgegriffene Wert in der E/E-Entwicklungsumgebung weiterverarbeitet werden kann. Auf der anderen Seite können Analyse-Dekorierer auch komplexere Formen annehmen, die abhängig von den analysierten Informationen des E/E-Architekturmodells während der Synthese automatisiert generiert werden sollen, damit das eigentliche Verhaltensmodell davon unberührt und die Modularität erhalten bleibt. Dafür sind die in Unterabschnitt 5.1.4.1 eingeführten `AbstractModelDecorators` verantwortlich, um bspw. die Netzwerkkommunikation zwischen Funktionen zu simulieren oder um die o. g. instrumentierenden Actors zu generieren, welche bspw. den Zeitstempel eines eintreffenden Datums messen.

Zur Rückführung und Weiterverarbeitung der Simulationsdaten im E/E-Entwicklungswerkzeug werden erweiterte *Sink*-Actors (dt. Senken) eingeführt, die die Schnittstelle zur Kommunikation zwischen Actor-orientierten Simulationswerkzeugen und der E/E-Entwicklungsumgebung bilden. Darauf wird in Unterabschnitt 5.4.2 genauer eingegangen.

- **Eingangsgrößen**
  - Logische Architektur, welche analysiert und simuliert werden soll.

  - Analyse-Dekorierer.
- **Ausgangsgrößen**
  - Simulationsmodell – serialisiert oder als komplexe Datenstruktur.

5.4.1.4 Simulation Executor

Der *Simulation Executor* ist hauptsächlich für die Ausführung des synthetisierten Simulationsmodells verantwortlich. Des Weiteren fungiert dieser Block aber auch als Schnittstelle, die die Simulationsdaten, welche über die o. g. Sink-Actors aufgezeichnet wurden, empfängt und in einer sog. *Simulation Data Registry (SDR)* abspeichert. Je nach Typ der eingesetzten Sink-Actors eines Analyse-Dekorierers enthält die SDR spezifische Datenstrukturen, die die aufgezeichneten Ergebnisse kapseln (vgl. Unterabschnitt 5.4.2.2). Die SDR mit ihren Datenstrukturen wird am Ausgang bereitgestellt, damit empfangende Metrikberechnungsblöcke die für sie relevanten Datenstrukturen extrahieren können.

Wie in Unterabschnitt 5.1.3 eingeführt erhalten sämtliche Artefakte des Simulationsmodells die UUID des ursprünglichen Architekturartefakts als Suffix. Diese UUIDs werden bei den Einträgen der jeweiligen Sink-Actors in der SDR mit berücksichtigt, um die Artefakte beider Modelle und insb. die damit zusammenhängenden Simulationsergebnisse eindeutig mit den Architekturartefakten in Beziehung zu setzen (siehe die bidirektionale Verbindung zwischen dem Architekturmodell und der SDR in Abb. 5.31).

- **Eingangsgrößen**
  - Simulationsmodell.
- **Ausgangsgrößen**
  - Datenstrukturen der aufgezeichneten Simulationsdaten.

5.4.1.5 Metrikberechnungsblöcke

Die *Metrikberechnungsblöcke* sind für die Berechnung der eigentlichen Metrik zuständig und orientieren sich an der Definition in [41, S. 159]. In dieser Arbeit werden die Berechnungen weiter unterteilt in (quasi-)statische und dynamische Metriken. Hinsichtlich der dynamischen Metriken ist pro Analyse-Dekorierer ein dedizierter Block vorgesehen, welcher die für ihn dazugehörige Datenstruktur aus der SDR filtert, vorverarbeitet sowie die tatsächliche Metrik berechnet und evaluiert. Darüber hinaus stellt ein solcher Block über die in der SDR hinterlegten und extrahierten UUIDs die Verbindung der Simulationsdaten mit den Archi-

tekturartefakten her. Die (quasi-)statischen Metriken werden direkt auf Basis der bereitgestellten Architekturartefakte und deren statischen Attribute berechnet. Die in Datenstrukturen gekapselten Resultate werden schließlich an optionale Modifikationsblöcke oder direkt an den Inner Loop Handler durchgeschleift.

- **Eingangsgrößen**
  - SDR Einträge gekapselt in dedizierten Datenstrukturen (nur dynamische Metrikberechnungsblöcke).
  - Konfigurationsartefakte vom Inner Loop Handler.
- **Ausgangsgrößen**
  - Datenstrukturen der Metrikergebnisse.

5.4.1.6 EEA-Modifikationsblöcke

Die Metrikergebnisse können in diesen Blöcken weiterverarbeitet und analysiert werden, um Modifikationen sowohl am Architekturmodell als auch am Verhaltensmodell vornehmen zu können. Abhängig vom Typ der empfangenen Metrik-Datenstruktur können bspw. Algorithmen implementiert werden, um auf Basis der Ergebnisse Optimierungen am Modell vornehmen zu können, z. B. ein Allokationsalgorithmus zur Funktionsverteilung[19].

Die Ergebnisse werden schließlich an den Inner Loop Handler weitergereicht, welcher sie in Tabellen aufbereitet und, sobald das Abbruchkriterium erfüllt ist, zur Zusammenfassung an die äußere Schleife weiterleitet.

5.4.2 Konzept zur Integration der Simulation

In diesem Abschnitt wird ein generisches Konzept zur Kopplung von Actor-orientierten Simulations- bzw. Analysewerkzeugen mit modellbasierten E/E-Architekturwerkzeugen vorgestellt. Zunächst werden erweiterte Sink-Actors eingeführt, die die Schnittstelle zur Übertragung der simulierten Daten darstellen. Anschließend wird auf die Rückführung der Daten und die Verarbeitung im E/E-Architekturwerkzeug eingegangen.

[19] Die Optimierung wird in dieser Arbeit als optionaler Aspekt zur Erweiterung betrachtet und wird nicht weiter vertieft.

5.4.2.1 Sink-Actor Konzept

Sink-Actors werden generell als Datensenken eingesetzt und besitzen i. d. R. nur Eingangsports. Typische Beispiele sind Monitor-Actors zur Beobachtung und Anzeige eines aktuellen numerischen Wertes eines Tokens oder Plotter-Actors zur Visualisierung eines Signalverlaufs. Durch den integrierten Ansatz kann die Visualisierung von Signalverläufen via Plotter-Actors verhältnismäßig einfach in die umgebende Anwendung eingeblendet werden (vgl. Unterabschnitt 6.1.1). Die tatsächliche Kommunikation zwischen Simulator und E/E-Entwurfswerkzeug durch Rückführung von bestimmten Tokens, bspw. zur Weiterverarbeitung in Metrikberechnungen, wird damit jedoch noch nicht adressiert.

Aus diesem Grund werden erweiterte Sink-Actors eingeführt, die eine OSGi [130] Schnittstelle zum Datenaustausch mit dem E/E-Entwurfswerkzeug bereitstellen und werden als *OSGi-Event Sink-Actor* bezeichnet. Durch die Platzierung der Schnittstelle in einem Actor ist es möglich, an beliebigen Stellen im Verhaltensmodell Werte abzugreifen und zurückzuführen, um diese schließlich in der SDR abzuspeichern. Diese Senken können dann wie klassische Sink-Actors bspw. als Monitore im Verhaltensmodell auf BLA-Ebene modelliert, oder aber automatisiert durch die Synthese an bestimmten Stellen im Simulationsmodell als instrumentierende Elemente platziert werden. Abb. 5.32 zeigt den schematischen Aufbau eines OSGi-Event Sink-Actors zum Abgriff eines bestimmten Tokens zwischen zwei Actors.

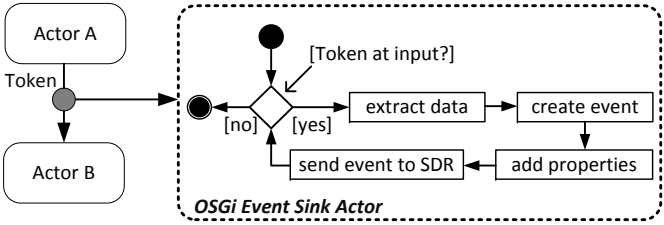


Abbildung 5.32: Schematischer Aufbau eines OSGi-Event Sink-Actors zum Datenabgriff als Grundlage zur Kommunikation mit dem E/E-Entwurfswerkzeug (Quelle: [BNB19]).

Zusätzlich zum eigentlichen Wert des empfangenen Tokens sind jedoch noch weitere Parameter notwendig, die zurückgeführt werden müssen. Zu den wichtigsten zählen hier die UUIDs der ursprünglichen Architektur- bzw. Verhaltensartefakte, die in Verbindung mit dem Sink-Actor und damit mit dem abgegriffenen Simulationswert stehen. Dazu gehört bspw. die UUID des umgebenden BLA

Building Blocks und des Ports, an dem der Sink-Actor angeschlossen ist. Des Weiteren wird der Typ eines jeden Sink-Actors mitübertragen, um aufseiten des E/E-Architekturwerkzeugs festzustellen, wie das Event weiterverarbeitet und in einer entsprechenden Datenstruktur gekapselt werden muss, die für den Austausch zwischen den Metrikblöcken notwendig ist. Dies ist vor allem dann von Belang, wenn es sich um komplexere Analyse-Dekorierer handelt, die zusätzlich spezifische, von der Art der Analyse abhängige Parameter enthalten. Dazu zählen z. B. ein Zeitstempel und eine Liste benachbarter Ports, die zur Rekonstruktion und Analyse einer Ende-zu-Ende-Latenz einer funktionalen Wirkkette notwendig sind. Die verschiedenen Anwendungsfälle und Typen der OSGi-Event Sink-Actors zur manuellen und automatisierten Platzierung sind in Abb. 5.33 schematisch dargestellt.

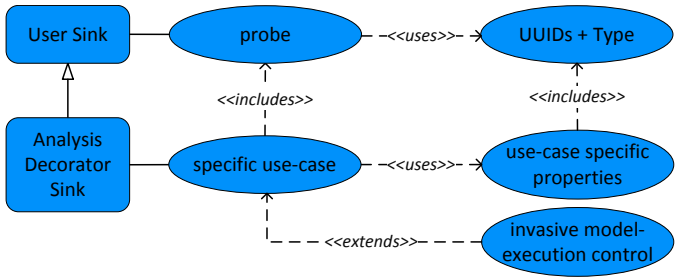


Abbildung 5.33: Übersicht über die verschiedenen OSGi-Event Sink-Actor Typen und deren Parametern (Quelle: erweiterte Darstellung nach [BNB19]).

Eine dritte Methode zur Rückführung von Simulationsdaten ist in Abb. 5.33 als Erweiterung eines anwendungsspezifischen Analyse-Dekorierers dargestellt. In diesem Fall ist ein Zugriff im Quellcode des Simulators bzw. Analysewerkzeugs notwendig. Ein Beispiel dafür ist das Erfassen von internen Modellzuständen wie der aktuelle Zustand eines State Charts, die über Actors nicht oder nur umständlich abgreifbar sind.

5.4.2.2 Rückführung und Verarbeitung der Simulationsdaten

Wie im letzten Unterabschnitt 5.4.2.1 bereits angedeutet wird für die Kommunikation zwischen Simulator und E/E-Architekturwerkzeug OSGi als generische Schnittstelle verwendet. Einerseits, um hinsichtlich der prototypischen Umsetzung mit PREEvision® und PtII den integrierten Charakter aufrecht zu erhalten.

Durch die Java-basierte Implementierung beider Werkzeuge und die Einbindung von PtII als zusätzliches eclipse-Plugin können sämtliche eclipse-Features ausgenutzt werden können, darunter auch eine Implementierung der OSGi Service-Plattform [193]. Andererseits, weil die OSGi Service-Plattform auf dem Whiteboard-Entwurfsmuster [129] basiert, das den Vorteil bietet, dass sich die involvierten Plugins gegenseitig nicht kennen müssen, sondern durch eine sog. Service Registry voneinander entkoppelt werden [193]. Ein Überblick über den Aufbau der OSGi Service-Plattform ist in [Neu18] zu finden, detailliertere Ausführungen in [193]. Der schematische Aufbau zur Kopplung über OSGi ist in Abb. 5.34 gezeigt.

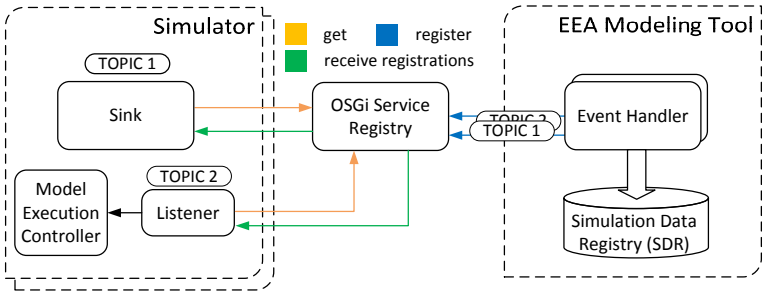


Abbildung 5.34: Prinzip der OSGi-basierten Kopplung von Simulator und E/E-Architekturwerkzeug. Event-Listener (rechts) registrieren sich als Service mit einem bestimmten Topic und kommunizieren via der OSGi Service-Plattform mit den Quellen (links), die Events mit demselben Topic senden (Quelle: modifizierte Darstellung nach [BNB19]).

Das E/E-Architekturwerkzeug fungiert nach dem Prinzip der OSGi Service-Plattform dabei als *Event-Listener*, die sich jeweils als Service bei der Registry mit einem oder mehreren eindeutigen *Topic* registrieren, an welchen der Service interessiert ist. Umgekehrt fungiert der Simulator, der die Simulationsdaten erzeugt und über Events bereitstellt, als Konsument des Services bzw. eines Topics und erhält über die globale Service Registry als Vermittler alle Instanzen des registrierten Services, an welche die Events entsprechend dem registrierten Topic schließlich gesandt werden. Der Datenaustausch geschieht dabei über *OSGi-Events*, welche die auszutauschenden Informationen über ein oder mehrere *Properties* kapseln. Properties sind key-value-Paare bestehend aus einem String-Bezeichner und einem dazugehörigen beliebigen Objekt.

Ein OSGi-Event Sink-Actor in Abb. 5.32 erzeugt ein solches OSGi-Event, füllt dieses mit dem abgegriffenen Token sowie den zusätzlich angegebenen Parametern mithilfe der Properties und sendet es mit einem bestimmten Topic an den Event-Listener. Die Topics dienen demnach dazu, Events von unterschiedlichen Quellen zu unterscheiden und zu filtern, die bspw. von Sink-Actors oder von internen Modellzuständen während der Ausführung erzeugt werden (vgl. den model-execution control Anwendungsfall in Abb. 5.33). Im letzteren Fall wird, wie in Abb. 5.34 skizziert ein Listener bereitgestellt, der auf Seiten des Simulators an der entsprechenden Komponente registriert wird und die OSGi-Events mit den gewünschten Informationen des ausführenden Modells zurücksendet.

Der Ablauf der Verarbeitung von OSGi Events auf Seiten des E/E-Architekturwerkzeugs ist in Abb. 5.35 als Sequenzdiagramm dargestellt. Konkret findet die Verarbeitung im Block Simulator Executor aus Abb. 5.31 statt, der die aus den OSGi-Events erzeugten Datenstrukturen an die Metrikberechnungsblöcke weiterreicht.

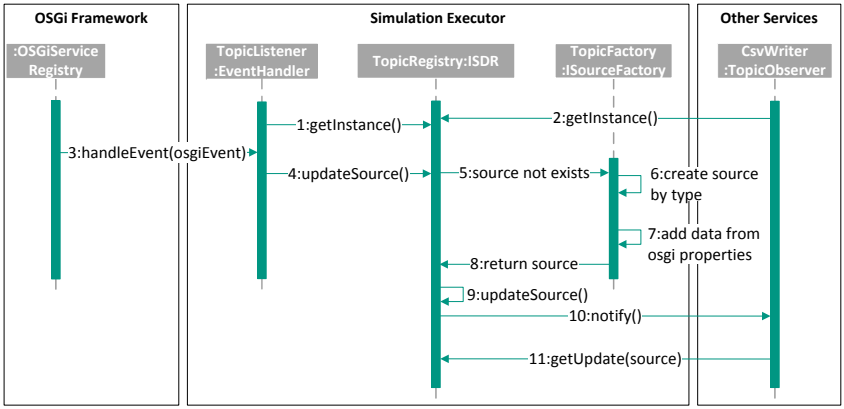


Abbildung 5.35: Sequenzdiagramm für den Ablauf der OSGi-Eventverarbeitung aufseiten des E/E-Architekturwerkzeugs im Simulation Executor Block (Quelle: erweiterte Darstellung nach [Kam18]).

Die Kommunikation zwischen Simulator und dem empfangenden Simulation Executor Block aufseiten des E/E-Architekturwerkzeugs wird vollständig durch die OSGi Service-Plattform, mithilfe des von OSGi bereitgestellten Services `EventAdmin`, abgewickelt. Sobald ein Event eintrifft, ruft das OSGi-Framework den über das Topic adressierten `TopicListener` auf, der die `EventHandler` Schnittstelle von OSGi implementiert und das Event verarbeitet (Schritt 3). Die Topics werden dazu verwendet, die verschiedenen Event-Quellen wie die Sink-

Actor-Typen zu unterscheiden, entsprechende Datenstrukturen über eine Factory anzulegen und diese in einer globalen, Topic-spezifischen SDR abzuspeichern bzw. zu aktualisieren, falls die Quelle bereits einen Eintrag besitzt (Schritte 4-9).

Die Unterscheidung der Quellen ist ein zweistufiger Prozess: zuerst wird nach dem Topic gefiltert, das jeweils eine globale SDR besitzt. Anschließend erzeugt eine konkrete Factory abhängig vom im OSGi-Event hinterlegten Quellen-Typ eine entsprechend konkrete Datenstruktur aus den restlichen im OSGi-Event enthaltenen Properties. Die konkrete Datenstruktur wird schließlich in der dazugehörigen Topic-SDR abgelegt. Diese speziellen Datenstrukturen sollen dabei eine gemeinsame, abstrakte Basisklasse besitzen, damit alle davon abgeleiteten Datenstrukturen in der Topic-SDR abgelegt werden können. Dieses Vorgehen ist bspw. zur Unterscheidung der verschiedenen Typen an Sink-Actors notwendig, die unterschiedlich komplexe Datenstrukturen aufweisen können, aber dennoch vom selben Topic stammen.

Bei den Topic-SDRs handelt es sich um globale Registries, welche das Singleton-Entwurfsmuster realisieren, wie an den Schritten 1 und 2 in Abb. 5.35 zu erkennen ist. Somit ist es möglich, angelehnt an der OSGi Service Registry, die aus den Simulationsergebnissen erzeugten Datenstrukturen anderen Services, außerhalb des Simulation Executor Blocks, im E/E-Architekturwerkzeug zur Verfügung zu stellen und über Änderungen zu informieren. Dazu werden das Observer- bzw. das Listener-Entwurfsmuster angewandt. Als Beispiel ist in Abb. 5.35 ein `CsvWriter` gezeigt, der die Ergebnisse in eine CSV-Datei exportiert.

5.4.3 Prototypische Realisierung

In diesem Abschnitt wird die prototypische Umsetzung des AMiL-Ansatzes zur Adressierung der in Unterabschnitt 4.3.4 beschriebenen Anforderungen vorgestellt. Als Anwendungsfall für die Variantenbewertung wird dabei die in Unterabschnitt 4.3.3.2 erläuterte Netzwerkkommunikation zwischen Funktionen gewählt, da mit dieser die Kombination aus quasi-statischer Buslast-Metrik und die durch Simulation gewonnene dynamische Metrik der Kommunikationslatenz abgedeckt werden kann. Gleichzeitig werden mit diesem Anwendungsfall weitere Freiheitsgrade auf mehreren Abstraktionsebenen in der Variantenbewertung berücksichtigt. Dazu zählen die involvierten Funktionsumfänge bzw. Wirkketten, die Abbildung von Funktionen auf unterschiedliche Steuergeräte sowie die Variation der Steuergeräte- und Bustopologie selbst.

Im Folgenden wird erläutert, wie die maximale Kommunikationslatenz einer funktionalen Wirkkette mithilfe der Simulation der involvierten Bussysteme und

den zurückgeführten Simulationsdaten in einer dynamischen Metrik berechnet werden kann.

5.4.3.1 Latenzanalyse funktionaler Wirkketten

Zur einfacheren Beschreibung wird die Analyse der maximalen Kommunikationslatenz zunächst anhand einer exemplarischen Wirkkette eines Spurhalteassistenten (engl. *Lane Departure Warning (LDW)*) in der logischen Architektur von PREEvision® abgeleitet, welche in Abb. 5.36 abgebildet ist. Die annotierten Infokästen repräsentieren dabei die variantenabhängigen Mappings auf eine Hardwarekomponente.

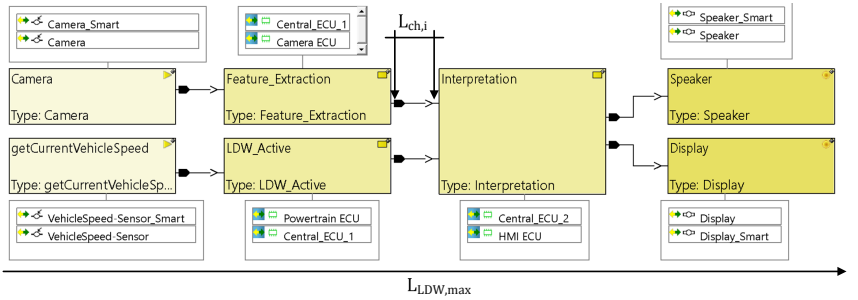


Abbildung 5.36: Beispielwirkkette eines Spurhalteassistenten in der logischen Architektur (Quelle: [BNB19]).

Eine Wirkkette besteht typischerweise aus mehreren Pfaden von einem Sensor bis zu einem Aktuator, welche durch die logischen Verbindungen zwischen den Funktionen gegeben sind. Eine logische Verbindung zwischen eines Sender- und Empfängerports wird im Folgenden als *Kanal* (engl. *channel*) bezeichnet. Ein Kanal ist definiert durch eine Sender- und Empfängerfunktion und eine dazugehörige Latenz L_{ch_i}.

Durch Kommunikationsabhängigkeiten während der Simulation mit anderen Wirkketten, die sich denselben Bus teilen, ist die Latenz eines Kanals L_{ch_i} jedoch nicht unbedingt konstant, bspw. aufgrund von Kollisionen. Die maximale Kommunikationslatenz einer Wirkkette, im Beispiel die des Spurhalteassistenten L^{max}_{LDW}, ist daher definiert als der längste Pfad der maximalen Kanallatenzen $L^{max}_{ch_i}$:

$$L^{max}_{LDW} = max\left(\sum_{\forall i \in CH(p)} L^{max}_{ch_i}\right), \forall p \in P \tag{5.6}$$

$$L_{ch_i}^{max} = max\left(L_{ch_i}^{e}\right), \forall e \in EV, \tag{5.7}$$

wobei $CH(p)$ die Menge aller Kanäle auf einem Pfad p ist, P die Menge aller gültigen Pfade der Wirkkette und EV die Menge aller gesandten Events eines Senderports.

5.4.3.2 Dekoration von Latenzanalyse-Sink-Actors

Um die maximale Kommunikationslatenz zu bestimmen, müssen die maximalen Latenzen der Kanäle aller Pfade bekannt sein, die während der Simulation innerhalb der Wirkketten auftreten. Dazu ist es notwendig, das eigentliche Verhaltensmodell mit zusätzlichen Mechanismen zu instrumentieren, um diese Messungen durchführen zu können. Damit jedoch die Performanz der Simulation durch die zusätzlichen Berechnungen der maximalen Kanallatenzen und der Gesamtlatenz am Ende der Simulation nicht zusätzlich belastet wird, wird eine minimale Instrumentierung des Verhaltensmodells verfolgt und die Berechnungen der Latenzen an das E/E-Architekturwerkzeug ausgelagert. Gleichzeitig werden so die Ergebnisse der Berechnungen im E/E-Architekturmodell verfügbar gemacht und können mit den entsprechenden Architekturartefakten assoziiert werden.

Zur Erfüllung dieser Anforderungen wird auf das in Abb. 4.1 vorgestellte Konzept der Analyse-Dekorierer zurückgegriffen, die das Verhaltensmodell während der Synthese automatisch mit den instrumentierenden Elementen für einen bestimmten Analysefall anreichern. Im vorliegenden Fall wurde dieser durch einen weiteren Dekorierer in der Klasse `MoMLCommunicationLatencyAnalysisDecorator` realisiert. In Abb. 5.37 ist das generische Konzept gezeigt, wie das Verhaltensmodell instrumentiert wird, um die Kanallatenzen von Wirkketten und somit die maximale Gesamtlatenz der Kommunikation einer Wirkkette bestimmen zu können.

Zur Rückführung der benötigten Informationen ist auf Basis des Sink-Actor Konzepts aus Abb. 5.33 ein *Latenzanalyse-Sink-Actor* realisiert worden. Diese werden während der Synthese automatisch an jeden Eingangs- und Ausgangsport eines zusammengesetzten Actors, welcher zu einer Wirkkette gehört, verbunden und senden die benötigten Informationen über das in Unterabschnitt 5.4.2 vorgestellte OSGi-Konzept an den Simulation Executor (vgl. Abb. 5.35) E/E-Architekturwerkzeug zurück. Ein OSGi-Event enthält dabei u. a. den Zeitstempel eines Tokens, der während der Simulation gesandt oder empfangen wurde.

Da die Netzwerkkommunikation aspektorientiert simuliert wird, trifft ein gesandter Token mit einem verzögerten Zeitstempel beim Empfänger ein, wodurch

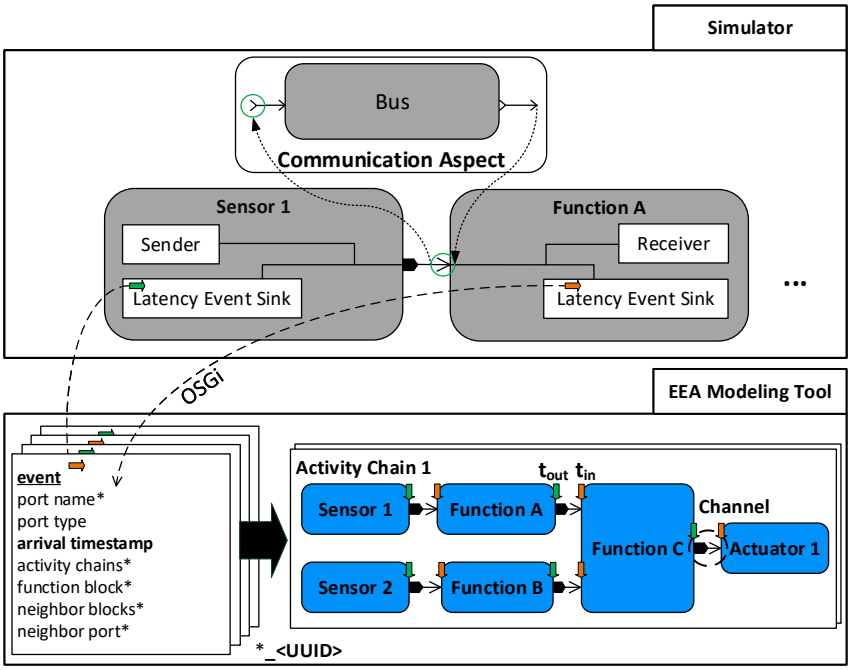


Abbildung 5.37: Konzept des Latenzanalyse-Sink-Actors zur Rekonstruktion der Wirkketten im E/E-Architekturwerkzeug und zur Bestimmung deren maximaler Kommunikationslatenz. (Quelle: erweiterte Darstellung nach [BNB19]).

die Kanallatenz durch die Differenz der beiden Zeitstempel mit

$$L^{e}_{ch_i} = t^{e}_{in}(receiverPort) - t^{e}_{out}(senderPort) \tag{5.8}$$

berechnet werden kann.

Die gezeigten weiteren Eigenschaften eines OSGi-Events eines Latenzanalyse-Sink-Actors sind nötig, um die Wirkketten aufseiten des E/E-Architekturwerkzeugs zu rekonstruieren und mittels den berechneten Kanallatenzen somit die maximale Kommunikationslatenz der Wirkkette durch den längsten Pfad zu bestimmen. Die Eigenschaften werden dabei während der Synthese automatisch aus den zugrunde liegenden Wirkketten des LA Modells ausgelesen und als Parameter den Latenzanalyse-Sink-Actors hinzugefügt, um diese zusammen mit dem

Zeitstempel eines Tokens zur Rekonstruktion der Wirkketten zurückzusenden. Dazu zählen der verbundene Port, der Port-Typ (Sender/Empfänger), der dazugehörige logische Funktionsblock und die Wirkketten, in welchen dieser vorhanden ist, sowie die benachbarten Ports/Blöcke. Wichtig ist hierbei der Suffix der UUID der ursprünglichen Architekturartefakte, um eine eindeutige Rekonstruktion der Wirkketten und ihrer Kanallatenzen zu erlangen.

Die empfangenen OSGi-Events werden schließlich basierend auf dem übertragenen OSGi Topic und des Sink-Typs gefiltert, in einer dazugehörigen Datenstruktur gekapselt und in der SDR gespeichert, welche zur Verarbeitung in weiteren Metrikberechnungsblöcken genutzt werden kann. Im Falle der beschriebenen Latenzanalyse wurde ein dynamischer Metrikberechnungsblock zur Latenzanalyse umgesetzt, welcher im nächsten Unterabschnitt näher erläutert wird. Weitere Details zur Implementierung sind in [Neu18] zu finden.

5.4.3.3 Metrikberechnungsblock zur Latenzanalyse

Dieser Metrikberechnungsblock ist für die im letzten Abschnitt erläuterte Rekonstruktion der Wirkketten, die Berechnung der Kanallatenzen sowie für die Bestimmung der maximalen Gesamtlatenz einer Wirkkette verantwortlich. Grundlage dafür ist die als Eingang zur Verfügung gestellte SDR mit den dazugehörigen Datenstrukturen der Latenzanalyse-Sink-Actors. Basierend auf den SDR-Einträgen wird die in Abb. 5.38 gezeigte Datenstruktur aufgebaut, um zunächst nach Gl. (5.7) die maximalen Kanallatenzen zu berechnen.

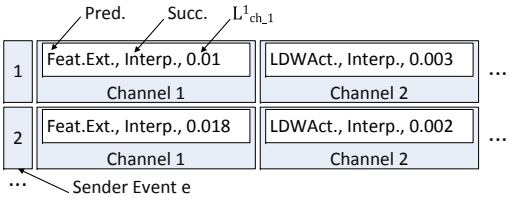


Abbildung 5.38: Datenstruktur einer Wirkkette bestehend aus Kanälen (Quelle: [BNB19]).

Die Berechnung des längsten Pfades und damit der maximalen Kommunikationslatenz einer Wirkkette nach Gl. (5.6) wird schließlich in der vorliegenden Realisierung mithilfe des Dijkstra-Algorithmus gelöst, wobei die Funktionen die Knoten und die Kanäle die gewichteten Kanten repräsentieren.

5.4.3.4 Integrierte Simulations- und Analysemetrik

Die prototypische Realisierung des hier vorgestellten AMiL Konzepts in Form einer integrierten Simulations- und Analysemetrik, wurde anknüpfend an die in Abschnitt 5.3 vorgestellte Simulationssynthese im Metrik-Framework von PREEvision® umgesetzt. Ein vereinfachtes Metrikdiagramm auf Basis der bereits vorgestellten benutzerdefinierten Metrikblöcke zur Simulationssynthese und -ausführung (vgl. Abb. 5.15) sowie des im letzten Abschnitt beschriebenen Latenzanalyse-Metrikblocks ist in Abb. 5.39 dargestellt. Einige Parameter- und Nachverarbeitungs-Metrikblöcke sind der Übersichtlichkeit halber nicht abgebildet.

Die Metrik ist dabei konform zum generischen Konzept in Abb. 5.31. Der blaue Rahmen spiegelt dabei die Grenze zur inneren Schleife des AMiL wider, wobei die benutzerdefinierten Metrikblöcke (weiß) sowie die *Inner Loop Handler*-, *BusLoad-Handler*- und die äußeren Blöcke die wesentlichen Komponenten aus Abb. 5.31 repräsentieren.

Der innere *Change Variant*-Block ist für den Wechsel der aktuellen Architekturvariante zuständig, die durch den äußeren Metrikkontextblock *Architecture Variants* nacheinander durchgeschaltet werden, sobald die innere Schleife berechnet wurde. Die innere Schleife des AMiL Konzepts wird durch einen Schleifenblock [41, Kap. 6.2.11] realisiert.

Die *Synthesizer*- und *Executor*-Metrikblöcke entsprechen jenen in Abb. 5.15, wobei zur variantensensitiven Synthese das entsprechende Flag aktiviert und das Flag `showModel`, zur iterativen Ausführung der Simulation ohne GUI, deaktiviert sein müssen. Als obligatorischer Eingang werden die logischen Funktionen der zu untersuchenden Wirkketten bereitgestellt.

Der *Latency Analysis*-Metrikblock realisiert die in Unterabschnitt 5.4.3.3 beschriebene Analyse der Kommunikationslatenzen von Wirkketten und erwartet die SDR-Einträge der Latenzanalyse-Sink-Actors. Die Latenzanalyse entspricht demnach einem *dynamischen* Metrikberechnungsblock des AMiL Konzepts.

Der *HW Remapper*-Block ist ebenfalls ein in dieser Arbeit realisierter benutzerdefinierter Metrikblock, welcher die Funktionszuweisung der Wirkketten auf ihre ausführenden Hardwarekomponenten der aktuellen Architekturvariante iterativ ändert. Als Eingang erwartet dieser Block von der Latenzanalyse eine Liste der analysierten Wirkketten und ihren maximalen Latenzen und tauscht die Zwischenergebnisse der aktuellen Funktionszuweisung mit dem *Inner Loop Handler*-Block aus, welcher die Teilergebnisse in Tabellen zusammenfasst. Generell wird der Kontroll- und Konfigurationsfluss von involvierten Metrikberechnungsblö-

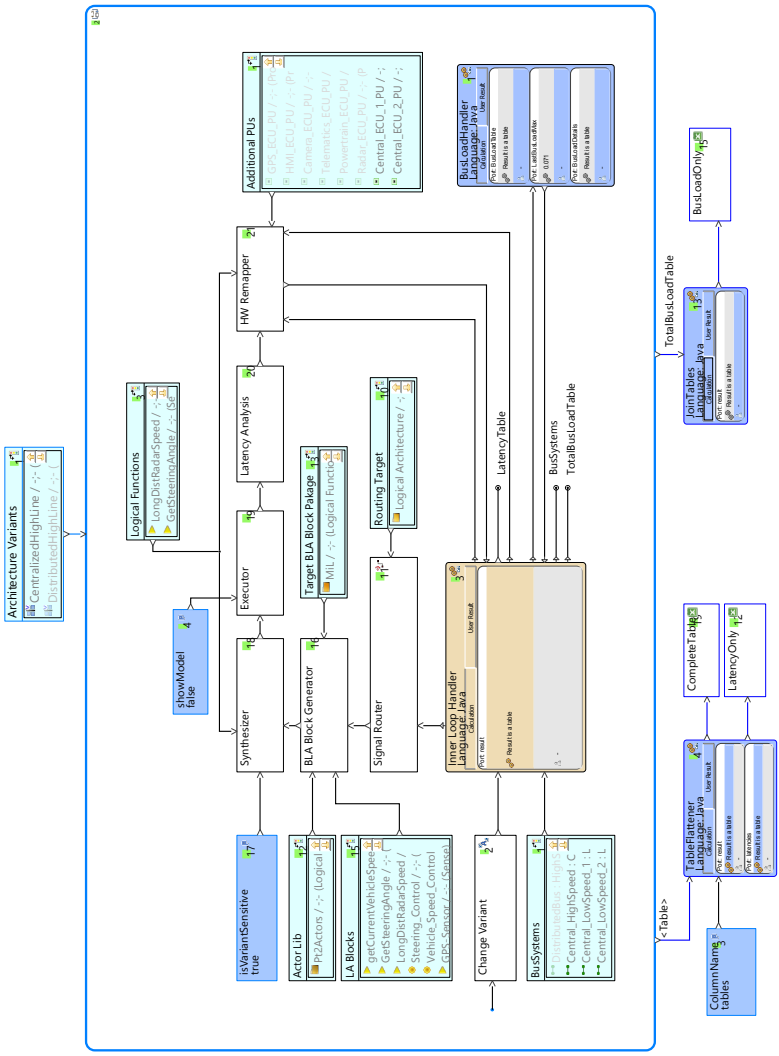


Abbildung 5.39: Prototypische Realisierung des AMiL Konzepts in Form einer integrierten Simulations- und Analysemetrik zur Untersuchung von Kommunikationslatenzen von Wirkketten und die daraus resultierenden Buslasten verschiedener Architekturvarianten (Quelle: modifizierte Darstellung nach [BNB19]).

cken der inneren AMiL-Schleife vom Inner Loop Handler mit internen Schleifen [41, Kap. 6.2.12] des Metrik-Frameworks gesteuert[20].

Demnach entspricht der HW Remapper-Block einem *EEA Modifikationsblock* des AMiL Konzepts. Der zusätzliche Metrikkontextblock *Additional PUs* definiert dabei – neben den PUs, die durch das aktuelle Mapping von Funktionen implizit gegeben sind – die für die Mapping-Änderung zur Verfügung stehenden PUs.7 Die transparent gelisteten PUs signalisieren, dass diese nicht Teil der aktiven Variante sind und finden daher keine Berücksichtigung.

Nach der Änderung der Funktionszuweisung zu ihren ausführenden Hardwarekomponenten und vor der Ausführung der nächsten Iteration der Simulationssynthese wird durch den vordefinierten Signal Router-Block ein Routing durchgeführt, um die ggf. geänderten Kommunikationspfade zwischen den Funktionen zu aktualisieren. Der Inner Loop Handler-Block konfiguriert den Signalrouter dabei mit der zur aktuellen Variante dazugehörigen Routing-Konfiguration und triggert diesen in einer neuen internen Schleife zur Simulation und Latenzanalyse, bis das Abbruchkriterium erreicht ist, bspw. nach einer festgelegten Anzahl an Iterationen. Der Signal-Router-Block zählt zu den *EEA Syntheseblöcken* des AMiL Konzepts. Ein weiterer EEA Syntheseblock ist *BLA Block Generator*, welcher die zu den logischen Funktionen abgebildeten Verhaltensmodelle in den BLA Building Blocks aktualisiert, bspw. um Änderungen der Signal-Timings nach dem Routing im Verhaltensmodell zu berücksichtigen (vgl. Unterabschnitt 5.3.4.2).

Zur Berechnung der Buslast als quasi-statische Metrik wurde die von PREEvision® gelieferte Buslast-Metrik wiederverwendet, adaptiert und in die innere AMiL-Schleife integriert. Im Wesentlichen ist dafür der Block *BusLoadHandler* zuständig, welcher vom Inner Loop Handler in der korrekten Ausführungsreihenfolge getriggert wird, d. h. nach jener inneren Schleife, die das Signal-Routing ausführt. Tatsächlich ist auch der BusLoadHandler mit weiteren Blöcken verbunden, insb. zur busspezifischen Bestimmung der Lasten. Diese sind aufgrund der Übersichtlichkeit nicht im Diagramm visualisiert, werden jedoch dennoch ausgeführt, da diese im Metrikmodell der Buslast verbunden sind[21].

Damit auch Buslasten von CAN FD-Bussystemen berechnet werden können, wurden einige CAN FD-spezifische Erweiterungen in verschiedenen Metrikblöcken der Buslast-Metrik implementiert, bspw. um das erweiterte CAN FD-Frameformat zu berücksichtigen und die Bitrate der Datenphase eines CAN FD-Clusters

[20] Interne Schleifen dienen dazu, Teilergebnisse zwischen Metrikblöcken mittels dedizierten, internen Schleifenports auszutauschen. Jeder Block kann über mehrere interne Schleifen verfügen, wobei seine eigene Ausführung so lange unterbrochen wird, bis die Teilergebnisse der internen Schleifen zurückgeliefert wurden. [41, Kap. 6.2.12]

[21] PREEvision® erlaubt es, Artefakte aus einem Diagramm visuell zu entfernen, ohne dass das Artefakt im Modell gelöscht wird. Das vollständige Diagramm der grundlegenden Buslast-Metrik ist in [41, S. 219] zu finden.

auszulesen. Dabei wurden die Annahmen getroffen, dass keine optionalen Stuff-Bits auftreten und die erhöhte Bitrate der Datenphase für den gesamten Frame gilt (Hintergrund hierzu siehe Unterabschnitt 6.5.3).

Die Ergebnisse jeder Architekturvariante in Form von Tabellen werden schließlich durch die internen Ausgangsports des Schleifenblocks an die äußere Schleife weitergegeben. Die Tabelle *LatencyTable* enthält dabei die maximale Kommunikationslatenz je Wirkkette zu jedem Iterationsschritt inkl. der dazugehörigen Mappings der Funktionen auf ihre Hardwarekomponenten. Die Tabelle *BusSystems* enthält die involvierten Bussysteme einer Architekturvariante. Des Weiteren enthält die Tabelle *TotalBusLoadTable* die Buslasten aller involvierten Bussysteme einer Architekturvariante. Das Ergebnis jeder Variante wird am Schleifen-Ausgangsport *TotalBusLoadTable* durch den Berechnungsblock *JoinTables* in einer Tabelle zusammengeführt.

Alle Tabellen als Teilergebnisse einer Architekturvariante werden am speziellen Ausgangsport `<Table>` [41, Kap. 6.2.11] zu einer Gesamttabelle zusammengefasst, wobei die erste Spalte die Architekturvariante enthält und alle weiteren Spalten die o. g. Teilergebnisse als verschachtelte Tabellen. Diese resultierende, verschachtelte Tabelle wird im nachgelagerten Berechnungsblock *TableFlattener* aufgebrochen und anschließend in eine Excel-Datei exportiert. Die Excel-Datei enthält dabei drei Arbeitsblätter zur weiteren Nach- und Aufbereitung der Ergebnisse. Diese umfassen zum einen nur die Buslasten und nur die Latenzen sowie alle Ergebnisse aller analysierten Architekturvarianten.

5.4.4 Limitierungen

Der HW Remapper-Metrikblock der umgesetzten AMiL-Metrik führt zufällige Mappings von Funktionen aus.

Die Analyse von Kommunikationslatenzen ist durch fehlende Netzwerk- bzw. Bus-Actors in PtII begrenzt auf CAN-Bus Simulationen (vgl. Unterabschnitt 5.3.9.2). Sobald weitere Busse zur Simulation in der Bibliothek zur Verfügung stehen, kann die Synthese um diese durch weitere Dekorierer modular erweitert und in der AMiL-Metrik automatisch berücksichtigt werden. Die wiederverwendete Buslast-Metrik unterstützt bereits weitere Bussysteme wie etwa LIN und FlexRay.

Als Sendemodi werden von der Synthese aktuell zyklische Sender unterstützt (vgl. Unterabschnitt 5.3.9.1). Durch die Kapselung der Generierung der BLA Blöcke in einem eigenen Metrikblock muss die AMiL-Metrik nach der Erweiterung von weiteren Sendemodi nicht angepasst werden, sondern nur die Implementierung innerhalb des Blocks *BLA Block Generator*.

6 Evaluation der Ansätze

In diesem Kapitel werden die in Kapitel 5 vorgestellten Methoden und Konzepte und die darin entwickelten Prototypen anhand von einigen Anwendungsfällen demonstriert und evaluiert. Die nachfolgend präsentierten Ergebnisse basieren auf den eigenen Publikationen [BRB17, BB18, BKB19a, BNB19, BKB19b, BKB19c].

6.1 Integrierte und domänenübergreifende Simulation einer ACC Applikation

In diesem Abschnitt wird eine ACC Applikation in der logischen Architektur und das ausführbare Verhalten mittels Bibliothekskomponenten in Form der importierten PtII Actor-Bibliothek modelliert und simuliert. Zusätzlich wird eine dazugehörige Hardware-Vernetzungsarchitektur modelliert, worauf die logischen Funktionen abgebildet werden. Dabei werden folgende Ziele verfolgt:

- Modellierung von ausführbarem, Actor-orientierten ACC Verhalten.
- Untersuchung von verschiedenen, austauschbaren Realisierungen durch die eingeführten LA/BLA Block-Mappings.
- Einfluss der vernetzungsabhängigen Buskommunikation zwischen Steuergeräten auf das realisierungsunabhängige Verhalten.
- Einfluss von abstrakten Ausführungsaspekten (Funktionslatenzen) auf das realisierungsunabhängige Verhalten.

6.1.1 Modellierung der ACC Applikation und der Vernetzungsarchitektur

Die Modellierung der ACC Applikation in Form einer Wirkkette, den dazugehörigen BLA Building Blocks, die das ausführbare Verhalten kapseln, der Vernetzungsarchitektur sowie die Mappings sind in Abb. 6.1 abgebildet.

Die ACC Wirkkette besteht aus den zwei Sensorfunktionen `GetRadarSpeed` und `GetVehicleSpeed`, die die Geschwindigkeit des vorausfahrenden und des eigenen

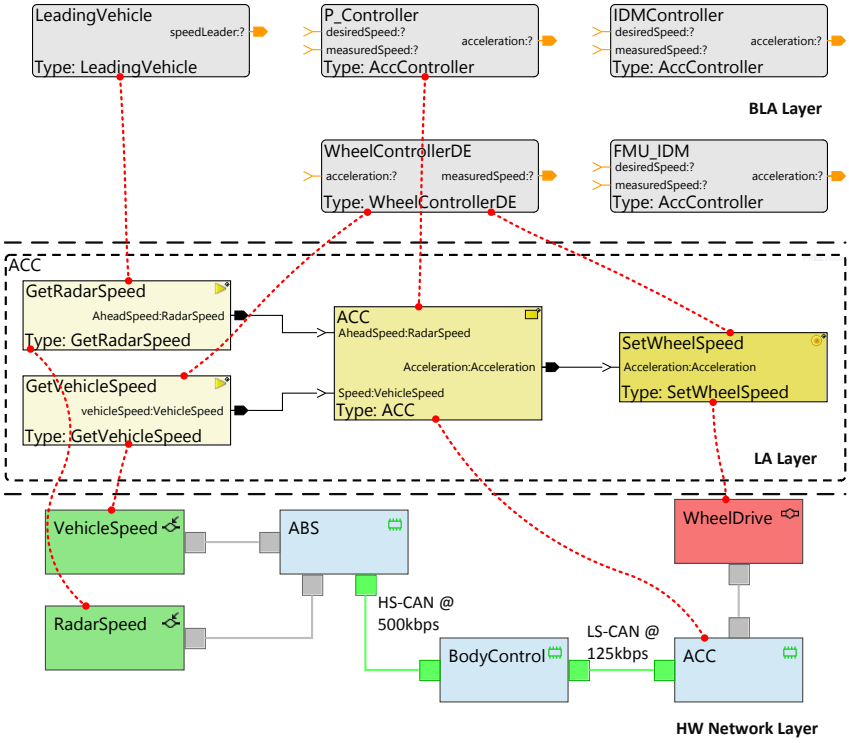


Abbildung 6.1: Modellierung einer ACC Applikation auf logischer Architektur Ebene, Verhaltensebene und der Hardware-Vernetzungsebene sowie die dazugehörigen Mappings (Quelle: erweiterte Darstellung nach [BRB17]).

Fahrzeugs für den ACC Algorithmus liefern. Letzterer berechnet darauf basierend die Beschleunigung, welche von der Aktuatorfunktion SetWheelSpeed in die Sollgeschwindigkeit umgewandelt wird.

Die dazugehörigen BLA Building Blocks sowie ein initiales Block-Mapping und die Port-Prototyp-Mappings wurden durch die Metrik in Abb. 5.8 erstellt. Das ausführbare Verhalten besteht intern aus einem Actor-Netzwerk, wobei in Abb. 6.1 drei Realisierungsalternativen vom Typ AccController zu sehen sind, von denen in diesem Abschnitt die Alternativen P_Controller und IDMController untersucht werden. Der P_Controller Block bildet einen einfachen P-Regler mit einer Schleifenverstärkung von $K_p = 10$, welche in Abb. 6.2 über den Const Actor parametrisiert ist. Zusätzlich wird die berechnete Beschleunigung durch die zwei Parameter $a_{max} = 3,0\,m/s^2$ und $a_{min} = -9,0\,m/s^2$ mithilfe des Limiter Actors begrenzt, um die physikalischen Limitierungen des Fahrzeugs zu berücksichtigen[1].

Der P-Regler ist in der kontinuierlichen Domäne bzw. innerhalb eines Continuous Directors modelliert, stellt jedoch durch den PeriodicSampler Actor eine diskrete Schnittstelle nach außen bereit (vgl. Unterabschnitt 2.3.2). Zur Visualisierung der simulierten Beschleunigung, Geschwindigkeiten und der Distanz werden TimedPlotter Actors modelliert.

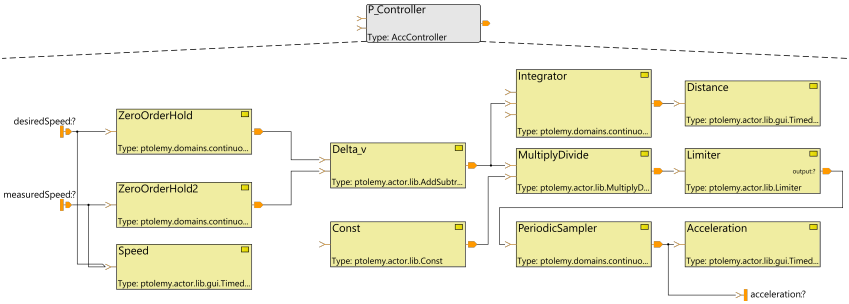


Abbildung 6.2: Actor-orientierte Realisierung der ACC Applikation in Form eines P-Reglers zur Berechnung der ACC Beschleunigung.

Zur konzeptionellen Demonstration des n-zu-1 Mappings von mehreren logischen Funktionen auf einen Verhaltensblock werden die beiden Funktionen GetVehicleSpeed und SetWheelSpeed auf den Building Block WheelControllerDE (siehe Abb. 6.3) abgebildet. Dieser fungiert dabei gleichzeitig

[1] $a_{min} = -9,0\,m/s^2$ entspricht einer Bremsverzögerung bei blockierenden Reifen auf trockener Straße [61, 62], die maximale Beschleunigung ist abhängig von der Motorisierung.

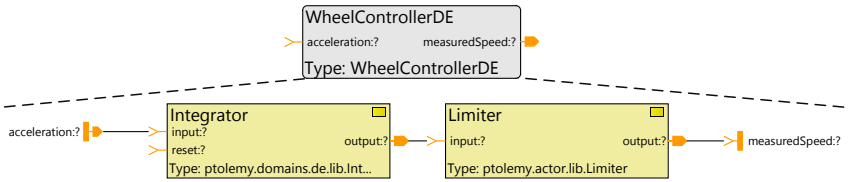


Abbildung 6.3: Actor-orientierte Realisierung der `GetVehicleSpeed` und `SetWheelSpeed` Funktionen. Der Controller wird von einem DE Director gesteuert.

als Dynamikmodell zur Umsetzung der Beschleunigung in eine Geschwindigkeit und zur Bereitstellung letzterer als Sensorwert. An dieser Stelle sei jedoch angemerkt, dass aus Gründen der Modularität und Wiederverwendbarkeit von Funktionsblöcken und damit verbunden von Verhaltensblöcken stets eine 1-zu-1 Abbildung zu bevorzugen ist. Dies ist insbesondere bei Anwendung von Varianten oder bei iterativer Generierung der Verhaltensblöcke innerhalb des AMiL Konzepts notwendig. Spätestens sobald Ausführungsaspekte von Funktionen auf geteilten Ausführungsressourcen eine Rolle spielen, ist ein n-zu-1 Mapping nicht mehr anwendbar, da nicht mehr eindeutig entschieden werden kann, welcher Ressource der Verhaltensblock zugewiesen werden soll.

Die Hardwarearchitektur in Abb. 6.1 besteht aus Sensor- und Aktuatorkomponenten, die je auf die entsprechenden logischen Funktionen abgebildet sind sowie aus einem Steuergerät, auf welchem die `ACC` Funktion ausgeführt wird. Die Sensoren und der Aktuator sind dabei über konventionelle Verbindungen mit den Steuergeräten verbunden, während die Steuergeräte untereinander über dedizierte High-Speed und Low-Speed CAN-Busse kommunizieren. Wie leicht aus der Abbildung zu erkennen ist, müssen beim vorliegenden Mapping die Sensorwerte über die beiden CAN-Busse zum ACC-Steuergerät übertragen werden, wobei die beiden Steuergeräte `ABS` und `BodyControl` als Gateway fungieren. Die durch den Signal-Router für beide Busse erzeugten CAN-Frametransmissionen der modellierten `RadarSpeed` und `VehicleSpeed` CAN-Frames erhalten über dazugehörige CAN ID-Listen eine CAN Frame-ID.

In Tabelle 6.1 sind einige wichtige Parameter von Artefakten der logischen Architektur, der Hardwarearchitektur und der Netzwerkkommunikation zusammengefasst, die für die nachfolgenden Simulationen angewandt werden.

Tabelle 6.1: Modellierte Parameter der logischen und Hardware-Architektur der ACC Applikation.

Artefakt	Parameter	Wert
Logische Architektur		
`GetVehiceSpeed`	Latency Time (max.)	50 *ms*
`SetWheelSpeed`		50 *ms*
Hardware-Architektur		
HS-CAN Bus	Baud-Rate	500 *kbps*
LS-CAN Bus		125 *kbps*
Netzwerkkommunikation		
CAN-Frame `RadarSpeed`	Frame ID	$0x123$
CAN-Frame `VehicleSpeed`		$0x125$
—„—	Frame Größe	108 *Bit*[a]
—„—		

[a] 64 *Bit Daten* + *Header Bits* + *Trailer Bits* (ohne Stuff-Bits).

6.1.2 Synthese

In den folgenden Unterabschnitten werden zwei Realisierungsalternativen der `ACC`-Funktion durch einfaches Ändern des Mappings auf einen Building Block vom Typ `AccController` untersucht. Zusätzlich wird der Einfluss sowohl der CAN-Kommunikation als auch von abstrakten Ausführungsaspekten in Form von simplen Verzögerungen untersucht.

In Abb. 6.4 ist das synthetisierte und visualisierte PtII-Simulationsmodell für die Realisierung des `IDMControllers` sowie die Einbindung der Netzwerkkommunikation und von Ausführungsaspekten der beiden Funktionen `GetRadarSpeed` und `SetWheelSpeed` (vgl. Tabelle 6.1) zu sehen. Durch das n-zu-1 Mapping der beiden Funktionen wird die Latenz des realisierenden Blocks `WheelControllerDE` zu 100 *ms* addiert[2].

Wie erwartet werden die Datenelemente der Radar- und Geschwindigkeits-Sensorfunktionen gemäß dem Signal-Routing über beide LS-CAN- und HS-CAN-Busse übertragen. Dementsprechend werden zwei dedizierte CAN-Kommunikationsaspekte mit den dazugehörigen Baudraten synthetisiert. Die Sensorwerte werden an den Empfängerports des `IDMControllers` demnach jeweils an die beiden Kommunikationsaspekte umgeleitet, wobei die Ports mit der modellierten CAN Frame-ID und der Frame-Größe dekoriert werden. Der Ausführungsaspekt

[2] Im vorliegenden Beispiel ist jede Funktion einer dedizierten Sensor-/Aktuator-Hardwarekomponente zugewiesen. Durch die Kausalität bei einem äquivalenten 1-zu-1 Mapping der Beschleunigungsumsetzung in die Geschwindigkeit und deren anschließende Detektion in einem Sensor, können die Latenzen addiert und als konservativer worst-case aufgefasst werden.

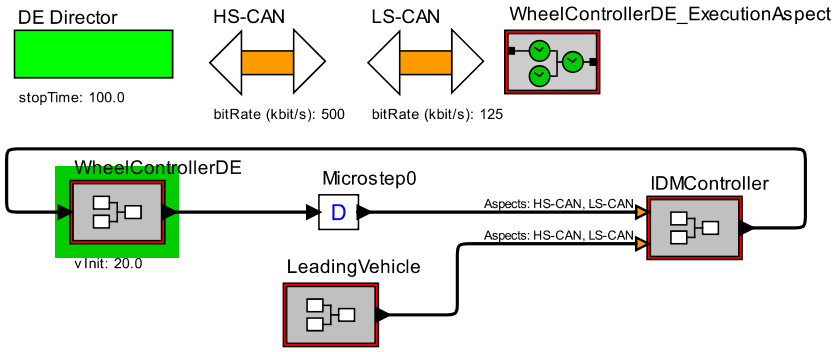


Abbildung 6.4: Generiertes Simulationsmodell der ACC-Applikation, visualisiert in der grafischen Benutzeroberfläche Vergil von PtII (Quelle: modifizierte Darstellung nach [BRB17]).

wird dagegen auf dem `WheelControllerDE` Actor dekoriert und ist durch den grünen Rahmen visualisiert. Zusätzlich ist die als Parameter modellierte Anfangsgeschwindigkeit des eigenen Fahrzeugs *vInit* zu sehen sowie ein `Microstep0` Actor. Letzterer wird während der Synthese automatisiert eingefügt, um die präsente Zero-Delay-Schleife im Rückkopplungspfad aufzubrechen (vgl. *Check and Break Cycles* in Abb. 5.4).

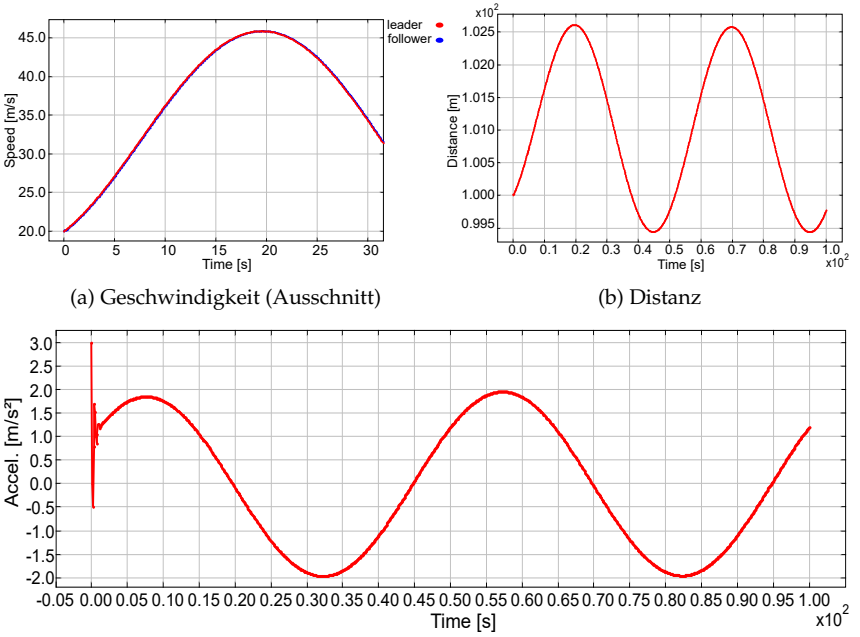
Zusätzlich zu den Beschleunigungsgrenzen sind als Parameter eine initiale Geschwindigkeit beider Fahrzeuge von 20 m/s und eine globale Sampling-Frequenz von 10 Hz für die nachfolgenden Simulationen modelliert.

6.1.3 Simulationsergebnisse und Diskussion

6.1.3.1 Simulation der P-Regler Realisierung mit CAN-Kommunikation

Im ersten Szenario wird das Verhalten des P-Reglers unter Einfluss der CAN-Kommunikation simuliert, aber zunächst *ohne* Ausführungslatenzen. Diese werden in Unterabschnitt 6.1.3.3 berücksichtigt und mit dem Verhalten des `IDMControllers` verglichen. Ein Auszug der Geschwindigkeiten des simulierten, vorausfahrenden Fahrzeugs und des Ego-Fahrzeugs sowie die Distanz zwischen beiden sind in Abb. 6.5 illustriert. Ein Screenshot der integrierten Ausführung und Visualisierung der Simulation innerhalb von PREEvision® in einer neu hin-

zugefügten, integrierten *Ptolemy Run Control* Ansicht ist im Anhang in Abb. A.2 zu finden.



(a) Geschwindigkeit (Ausschnitt)

(b) Distanz

(c) P-Regler Beschleunigung

Abbildung 6.5: Simulierte Geschwindigkeiten (Auszug) des vorausfahrenden und des Ego-Fahrzeugs sowie die Distanz und die berechnete P-Regler Beschleunigung unter Einfluss von CAN-Kommunikation.

Diskussion Trotz der Verzögerung der beiden Sensorwerte des Radars und des Geschwindigkeitssensors durch die CAN-Kommunikation, wird die Beschleunigung durch den P-Regler korrekt berechnet und die Geschwindigkeit des `WheelControllerDE` folgt strikt der des vorausfahrenden Fahrzeugs. Dies ist auch durch die analog sinusförmig verlaufende Distanz zu erkennen. Die Verzögerung aufgrund der CAN-Kommunikation wurde zusätzlich an den Eingängen innerhalb des P-Reglers mit `TimedMonitor` Actors gemessen. Der `RadarSpeed` Frame wurde mit einer Verzögerung von $1,08\,ms$ aufgezeichnet, die exakt der

Verzögerung des 108 *Bit* großen Frames über beide CAN-Busse entspricht:

$$\begin{aligned} \Delta t_{RadarSpeed} &= \Delta t^{HS\text{-}CAN}_{RadarSpeed} + \Delta t^{LS\text{-}CAN}_{RadarSpeed} \\ \Delta t_{RadarSpeed} &= 108\,Bit \left(\frac{1}{500\,kbps} + \frac{1}{125\,kbps} \right) = 1,08\,ms \end{aligned} \tag{6.1}$$

Die Verzögerung des `VehicleSpeed` Frames fällt aufgrund von Kollisionen mit dem höher priorisierten `RadarSpeed` Frame größer aus, sodass dieser erst nach $1,944\,ms$ nach dem Versand vom P-Regler empfangen wird. Die Verzögerung ergibt sich durch folgenden Ablauf:

- Beide Frames werden zu jedem Sampling-Zeitpunkt $\tau = n \cdot t_s = n \cdot 0,1\,s$, $\forall n = 0, 1, 2, \ldots$ & $\tau \leq t_{max}$ gleichzeitig an den HS-CAN-Bus weitergeleitet, wobei t_{max} die maximal spezifizierte Simulationszeit ist.
- Der `RadarSpeed` Frame wird aufgrund seiner höher priorisierten ID bearbeitet und nach $\Delta t^{HS\text{-}CAN}_{RadarSpeed} = 0,216\,ms$ zum LS-CAN-Bus weitergeleitet.
  - Der `VehicleSpeed` Frame muss bis zum Ende der Transmission warten und wird nach $0,216\,ms$ erneut verschickt, sodass dieser nach $\Delta t^{HS\text{-}CAN}_{VehicleSpeed} = 0,432\,ms$ an den LS-CAN-Bus weitergeleitet wird.
- Der `RadarSpeed` Frame wird zum Zeitpunkt $t = \tau + 0,216\,ms$ vom LS-CAN-Bus arbitriert, versandt und wird schließlich zum Zeitpunkt $t = \tau + \Delta t_{RadarSpeed}$ vom P-Regler empfangen.
  - Der `VehicleSpeed` Frame trifft während der Transmission des `RadarSpeed` Frames zum Zeitpunkt $t = \tau + 0,432\,ms$ ein und muss bis zum Ende der Übertragung zum Zeitpunkt $t = \tau + 1,08\,ms$ warten, bis er vom LS-CAN-Bus bearbeitet werden kann.

Der `VehicleSpeed` Frame wird vom ACC-Controller empfangen nach:

$$\begin{aligned} \Delta t_{VehicleSpeed} &= \Delta t_{RadarSpeed} + \Delta t^{LS\text{-}CAN}_{VehicleSpeed} \\ \Delta t_{VehicleSpeed} &= 1,08\,ms + \frac{108\,Bit}{125\,kbps} = 1,944\,ms \end{aligned} \tag{6.2}$$

6.1.3.2 Simulation der *Intelligent Driver Model (IDM)*-Realisierung mit CAN-Kommunikation und Ausführungslatenzen

In diesem Szenario wird der P-Regler, welcher lediglich die Geschwindigkeit des vorausfahrenden Fahrzeugs verfolgt, durch einen adäquaten, adaptiven Abstandsregelautomat bzw. eine ACC-Funktion ersetzt, was durch einfaches Ändern des Mappings auf den realisierenden `IDMController` Block erfolgt.

Intelligent Driver Model Die Realisierung der ACC-Funktion basiert in diesem Fall auf dem mikroskopischen Fahrzeugfolgemodell IDM [165], welches auch in den Vorarbeiten [BBK+16, RBB+14] und [142] eingesetzt wurde. Das IDM hat eine unfallfreie Charakteristik und zeichnet sich mit einigen wenigen, aber intuitiven Parametern aus, mit denen unterschiedliche Fahrstile und Leistungsgrenzen des Fahrzeugs modelliert werden können. Durch geeignete Wahl der Parameter sind durchaus realistische Simulationen[3] durch Abgleich mit aufgezeichneten Fahrmanövern möglich, die auch das makroskopische Verhalten von Verkehrssituationen wie stockender Verkehr reflektieren [61, 62], sodass das IDM die Grundlage für in der Praxis eingesetzte ACC-Implementierungen darstellt [63, 64]. Das IDM besteht aus einer gekoppelten ODE, die die Beschleunigung aus einer Funktion der tatsächlichen Geschwindigkeit v, der Distanz s und der relativen Geschwindigkeit zum vorausfahrenden Fahrzeug Δv berechnet. Die relative Geschwindigkeit ist dabei als positive Annäherungsrate definiert [61, S. 15]: $\Delta v = v_i - v_{i-1}$, wobei $(i-1)$ das vorausfahrende Fahrzeug repräsentiert. In [166, Kap. 11.3] ist das IDM gegeben durch:

$$\begin{aligned} \dot{v} = \tilde{a}_{mic}(s, v, \Delta v) &= a \left[1 - \left(\frac{v}{v_0} \right)^{\delta} - \left(\frac{s^*(v, \Delta v)}{s} \right)^2 \right], \\ s^*(v, \Delta v) &= s_0 + max \left[0, \left(vT + \frac{v \Delta v}{2\sqrt{ab}} \right) \right]. \end{aligned} \tag{6.3}$$

Der Wunschabstand s^* ist dabei ein dynamischer Abstand, welcher das Annäherungsverhalten reflektiert und wird in [166, Kap. 11.3.4] als intelligente Bremsstrategie bezeichnet, die mithilfe der Modellparameter parametriert wird. Die Modellparameter werden in [166, Kap. 11.3.3] mittels folgender Standardsituationen, zwischen denen kontinuierliche Übergänge bestehen, veranschaulicht beschrieben:

[3] In [164] ist ein interaktiver Verkehrssimulator zu finden, mit welchem in verschiedenen Verkehrssituationen mit den IDM Modellparametern sowie zusätzlich mit Spurwechselverhalten experimentiert werden kann.

1. "Das *Beschleunigen* auf freier Strecke geschieht mit der Maximalbeschleunigung a, die beim Annähern an die Wunschgeschwindigkeit v_0 in einer durch δ beschriebenen Weise gegen Null geht. Je größer δ ist, desto später wird die Beschleunigung reduziert."
2. "Das *Folgefahren* geschieht mit einem durch die Folgezeit T charakterisierten Abstand zuzüglich eines Minimalabstandes s_0 bei stehendem Verkehr."
3. "Beim *Annähern* an langsamere oder stehende Fahrzeuge wird in Normalsituationen die komfortable Verzögerung b nicht überschritten."

Die gewählten IDM Modellparameter für die nachfolgende Modellierung und Simulation der `IDMController` Realisierung und deren Bedeutung sind in Tabelle 6.2 zusammengefasst.

Tabelle 6.2: Parameter der `IDMController` Realisierung. Die gewählten Werte der IDM Modellparameter orientieren sich an [166, S. 163].

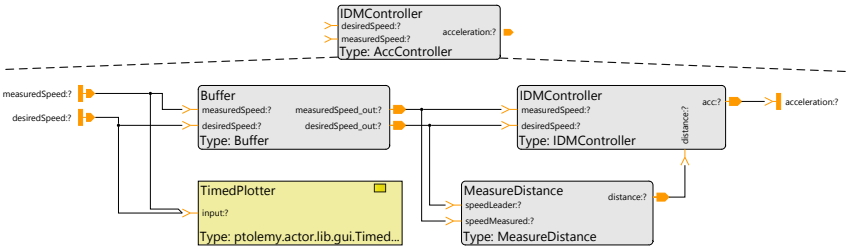
Parameter	Wert
IDM Parameter	
Maximale Beschleunigung a	$1,0\,m/s^2$
Komfortable Bremsverzögerung b	$1,5\,m/s^2$
Sicherheitszeitabstand (Folgezeit) T	$1,0\,s$
Minimaler Stauabstand s_0	$2\,m$
Wunschgeschwindigkeit v_0	$35\,m/s$
Beschleunigungsexponent δ	4
Zusätzliche Parameter	
Maximale Bremsverzögerung a_{min}	$-9,0\,m/s^2$
Initialer Abstand d	$200\,m$

Realisierung des `IDMControllers` Der `IDMController` Building Block vom Typ `AccController` in Abb. 6.6 (a) besteht zunächst aus drei weiteren hierarchischen Funktionsblöcken, während der `TimedPlotter` Actor zur Visualisierung der Geschwindigkeiten dient. Die hierarchischen Funktionsblöcke erfüllen dabei folgende Aufgaben:

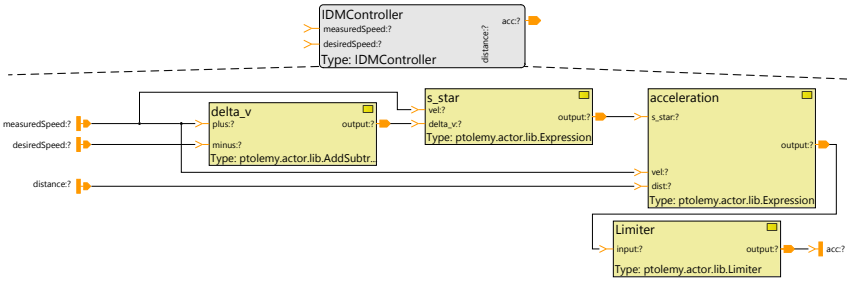
`Buffer`: Synchronisiert die empfangenen Geschwindigkeitswerte für die nachfolgenden Blöcke in Registern, die mit der globalen Samplingfrequenz von $10\,Hz$ betrieben werden.

`MeasureDistance`: Basierend auf den Geschwindigkeitswerten aus den Registern wird die Distanz zwischen beiden Fahrzeugen über einen Integrator berechnet. Die initiale Distanz beträgt $200\,m$.

`IDMController`: Realisiert die eigentliche ACC-Funktion in Form des IDM. Die interne Realisierung ist in Abb. 6.6 (b) zu sehen, wobei die Modellparameter des IDM sowie die zusätzlichen Parameter aus Tabelle 6.2 als generische Attribute auf dem Building Block modelliert sind, sodass sie von allen internen Blöcken referenziert werden können.



(a) Actor-orientierte Realisierung des Building Blocks vom Typ `AccController`.



(b) Actor-orientierte Realisierung des IDM.

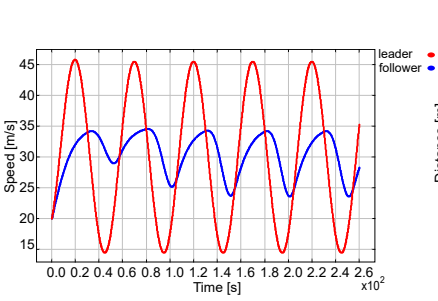
Abbildung 6.6: Hierarchische Realisierung der IDM-Berechnung.

Die IDM-Beschleunigung nach Gl. (6.3) wird mithilfe der `Expression` Actors `s_star` und `acceleration` berechnet, welche als Parameter mathematische Ausdrücke unter Verwendung der Expression-Sprache [136, Kap. 13] von PtII definieren. Im spezifizierten Ausdruck können sowohl Parameter des umgebenden Building Blocks als auch zusätzlich am `Expression` Actor angelegte Eingänge referenziert werden sowie vor- oder benutzerdefinierte mathematische Funktionen wie Potenz- oder Wurzelfunktionen. Am Ausgang des Actors wird die Lösung des mathematischen Ausdrucks bereitgestellt. Zur Berechnung des IDM wird zunächst die relative Geschwindigkeit bestimmt, die für die Berechnung des dynamischen Wunschabstandes s^* im Actor `s_star` benötigt wird. Die Beschleunigung wird sodann nach Gl. (6.3) im Actor `acceleration` berechnet und

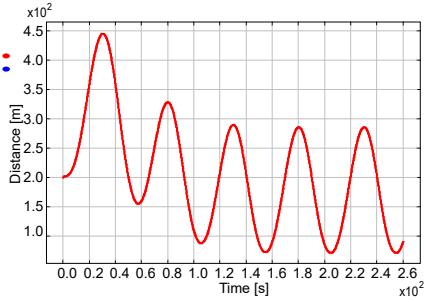
schließlich auf die maximale Beschleunigung a bzw. maximale Bremsverzögerung a_{min} begrenzt.

6.1.3.3 Diskussion

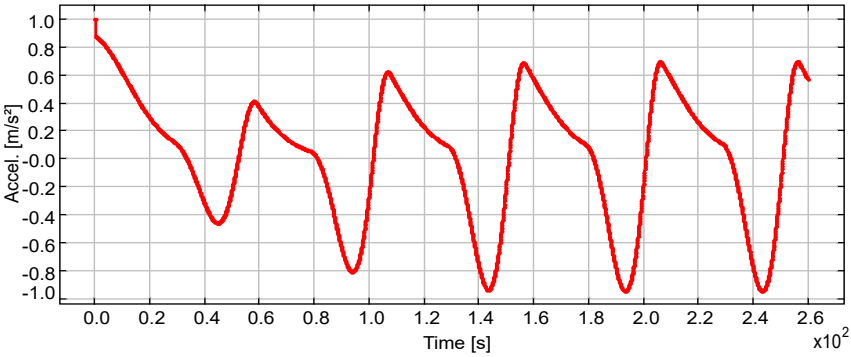
IDM Die Geschwindigkeiten des simulierten, vorausfahrenden Fahrzeugs und des Ego-Fahrzeugs sowie die Distanz zwischen beiden unter Einfluss der CAN-Kommunikation als auch der Ausführungslatenz von 100 ms des `WheelControllerDE` Blocks sind in Abb. 6.7 dargestellt.



(a) Geschwindigkeit



(b) Distanz



(c) IDM Beschleunigung

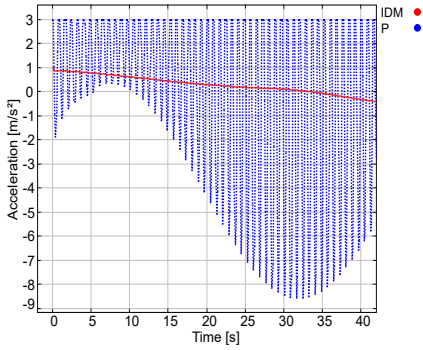
Abbildung 6.7: Simulierte Geschwindigkeiten des vorausfahrenden und des Ego-Fahrzeugs sowie die Distanz und die berechnete IDM-Beschleunigung unter Einfluss von CAN-Kommunikation und einer Ausführungslatenz des `WheelControllerDE`.

Es ist zu erkennen, dass das IDM ein robustes Verhalten gegenüber den Kommunikations- und Ausführungsverzögerungen zeigt und versucht, dem vorausfahrenden Fahrzeug in der Beschleunigungsphase bis zur parametrierten Wunschgeschwindigkeit v_0 zu folgen (Abb. 6.7 (a)). Aufgrund der geringeren max. Beschleunigung von $a = 1,0\,m/s^2$ kann das Ego-Fahrzeug dem vorausfahrenden Fahrzeug allerdings nicht bis auf die Wunschgeschwindigkeit v_0 folgen. Zudem entsprechen die gewählten IDM Modellparameter einer eher konservativen Fahrweise, u. a. verhält sich das IDM umso vorausschauender, je kleiner b ist [166, S. 163]. Des Weiteren ist eine exakte Erreichung der Wunschgeschwindigkeit v_0 nur auf freier Straße, ohne vorausfahrendes Fahrzeug oder Hindernissen, möglich [61, S. 21 f.].

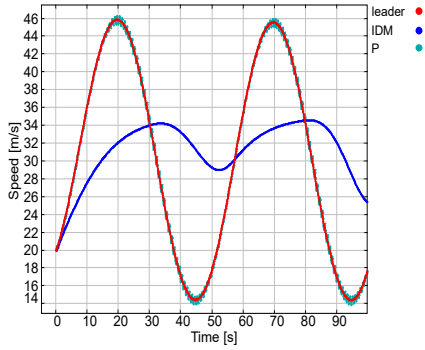
Beim Annähern des vorausfahrenden Fahrzeugs wird die Geschwindigkeit gemäß dem Wunschabstand s^* verringert, welcher sich aus dem Gleichgewichtsanteil $s_0 + vT$ und dem dynamischen Anteil $v\Delta v/(2\sqrt{ab})$ zusammensetzt. Da sich aufgrund der initialen Distanz der tatsächliche Abstand zunächst vergrößert und über dem dynamischen Wunschabstand liegt, wird die Geschwindigkeit in den ersten beiden Annäherungsphasen weniger verringert. Dies resultiert in einem Einschwingvorgang, bis der dynamische Wunschabstand in den darauffolgenden Beschleunigungs- und Annäherungsphasen schließlich erreicht wird (vgl. Abb. 6.7 (b)). Es ist außerdem zu sehen, dass in dieser Normalsituation des Annäherns die komfortable Verzögerung $b = 1,5\,m/s^2$ nie überschritten wird (vgl. Abb. 6.7 (c)).

Vergleich der beiden Realisierungen Hinsichtlich des Vergleichs der beiden Realisierungen des P-Reglers und des IDM, wird der P-Regler nun auch unter Einfluss der zusätzlichen Ausführungsverzögerung von $100\,ms$ simuliert, auf die der P-Regler im Rückkopplungspfad reagieren muss. Ein Ausschnitt der beiden berechneten Beschleunigungen und der Geschwindigkeiten sind in Abb. 6.8 (a) bzw. Abb. 6.8 (b) abgedruckt.

Während die P-Regler-Realisierung mit der gewählten Schleifenverstärkung von $K_p = 10$ unter Einfluss der CAN-Kommunikation beim Folgefahren noch strikt der vorausfahrenden Geschwindigkeit folgt, tritt durch die zusätzlichen Ausführungsverzögerungen vor allem in der Beschleunigung starkes Schwingverhalten auf. Dieses wird durch das Dynamikmodell des `WheelControllerDE` etwas geglättet, jedoch wirkt sich die Schwingung dennoch auf die eingestellte Geschwindigkeit aus. Im Gegensatz dazu stellt das IDM einen robusten und adaptiven Abstandsregelautomat dar, während der P-Regler lediglich versucht, der Geschwindigkeit des vorausfahrenden Fahrzeugs ohne Berücksichtigung eines Wunschabstands und einer Wunschgeschwindigkeit des Fahrers zu folgen.



(a) Beschleunigung (Ausschnitt)



(b) Geschwindigkeiten (Ausschnitt)

Abbildung 6.8: Vergleich der P-Regler- und IDM-Realisierung der ACC-Funktion.

6.2 Domänenspezifische Co-Simulation der ACC Applikation

Basierend auf der Evaluation der IDM-Realisierung der ACC-Funktion aus Abschnitt 6.1, wird in diesem Abschnitt das Verhalten der jeweiligen logischen Funktionen je durch ein werkzeugunabhängiges Verhaltensmodell mittels einer FMU ausgetauscht. Zunächst wird das Verhalten der ACC-Funktion in Form des IDM ausgetauscht und evaluiert. Somit wird eine domänenspezifische Co-Simulation des ACC mit dem integriert modellierten Verhalten durchgeführt. Anschließend werden die restlichen Funktionen durch eine entsprechende FMU in ihren dazugehörigen Building Blocks ersetzt. Dabei wird das Architekturmodell aus Abb. 6.1 wiederverwendet und das Verhalten der Funktionen wird durch einfaches Ändern des Mappings auf den dazugehörigen Building Block ausgetauscht[4].

Die Demonstration erfüllt dabei folgende Aufgaben:

- Einfacher Austausch des Verhaltens einer Funktion mit domänenspezifischer und werkzeugunabhängiger FMU.
- Demonstration der integrierten Co-Simulation zwischen dem integriert modellierten Verhalten sowie den FMUs und den Architekturaspekten aus Abschnitt 6.1.
- Verifizierung des ACC Verhaltens durch Vergleich der nativen IDM-Implementierung mit der FMU-Realisierung.

[4] In Abb. 6.1 beispielhaft durch den `IDM_FMU` Block dargestellt.

- Demonstration der integrierten Co-Simulation mit sukzessiv ausgetauschten FMUs und den Architekturaspekten aus Abschnitt 6.1.
- Vergleich der gesamten ACC Applikation mit der nativen und den FMU-basierten Realisierungen.

6.2.1 Realisierung des Verhaltens und Synthese

Erstellt wurde das den FMUs zugrunde liegende domänenspezifische Modell im kommerziellen Modelica [114] Quellwerkzeug Dymola 2018[5] zur Modellierung und Simulation von komplexen physischen Systemen. Die FMUs zur Co-Simulation (*Functional Mock-up Unit – Co-Simulation (FMU-CS)*) wurden dahingegen mithilfe der quelloffenen Plattform JModelica [117] exportiert, ein Werkzeug zur Kompilierung, Simulation und Optimierung von Modelica- und FMI-Modellen. Als Beispiel ist das realisierte Modelica-Modell der IDM-Realisierung, welches in eine o. g. FMU-CS kompiliert wird, im Anhang in Abb. A.3 zu finden. Analog existieren äquivalente Modelica-Modelle und FMUs für das Verhalten der `LeadingVehicle` und `WheelControllerDE` Blöcke in Abb. 6.1.

Das Verhalten der nativen IDM-Realisierung, in Form der beiden Building Blocks `MeasureDistance` und `IDMController` in Abb. 6.6 (a), ist durch eine FMU-CS gekapselt, die lediglich die notwendigen Schnittstellen (Ports) und Parameter nach außen bereitstellt. Die konkret eingesetzte FMU-CS `AccControllerCS_x64` ist in Abb. 6.10 (a) zu sehen. Diese wurde nach dem Prozess in Abb. 6.9 als entsprechender Funktionstyp in die Actor-Bibliothek importiert, instanziiert und enthält sämtliche Parameter aus Tabelle 6.2 sowie einige FMU-spezifische Parameter von PtII zur Editierung.

Der bereits modellierte `Buffer` Block wird aus Abb. 6.6 (a) wiederverwendet. Wichtig zu erwähnen ist an dieser Stelle, dass es sich, im Gegensatz zur nativen IDM-Realisierung in PtII, sowohl im ursprünglichen Modelica-Modell als auch speziell bei der exportierten FMU-CS um ein kontinuierliches Modell handelt, welches den Solver zur Lösung der ODEs integriert bereitstellt (vgl. Unterabschnitt 3.3.2.1). Da das IDM als gekoppelte ODE ausgedrückt wird (vgl. Gl. (6.3)), ist die Realisierung und Lösung als kontinuierliches Modell prädestiniert dafür. Nichtsdestotrotz sorgt der `Buffer` Block und der DE Director im synthetisierten Modell für eine synchrone fixed-step Co-Simulation mit diskreten Schnittstellen nach außen, um mit dem DE Director auf oberster Hierarchieebene kompatibel zu bleiben (vgl. Abb. 6.10 (b)).

[5] https://www.3ds.com/de/produkte-und-services/catia/produkte/dymola/, zuletzt aufgerufen am 27.09.2019.

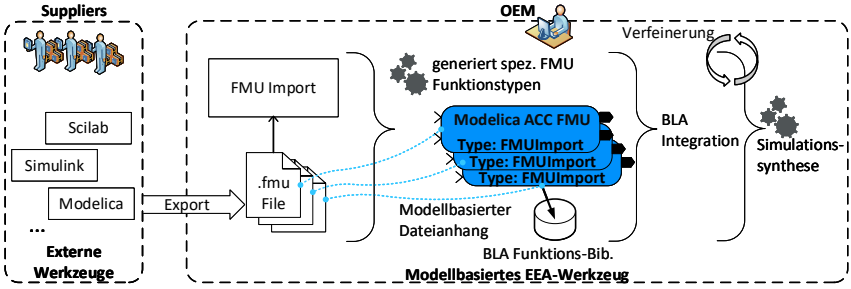
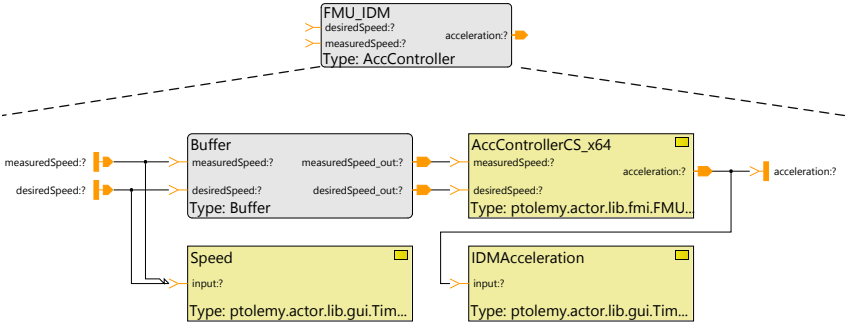


Abbildung 6.9: Prozess zur modellbasierten Integration von extern bereitgestellten, werkzeugunabhängigen FMUs als Funktionstypen in der ausführbaren BLA-Bibliothek zur domänenspezifischen Co-Simulation.

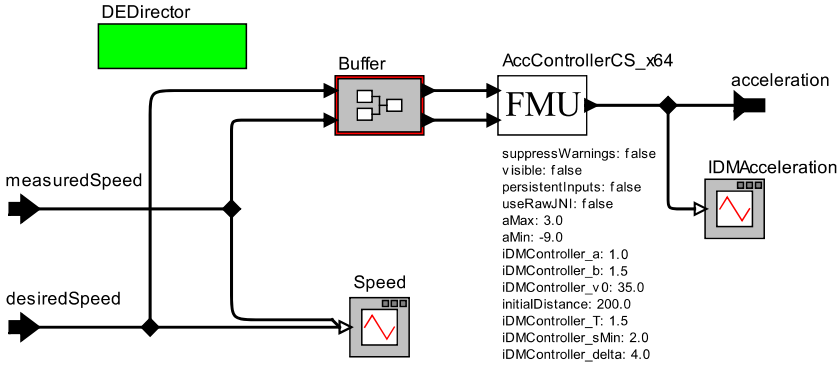
Im Gegensatz zum ACC-Funktion sind die Verhaltensmodelle der `Leading Vehicle` und `WheelControllerDE` Blöcke komplett durch je eine FMU-CS gekapselt. Die resultierenden, synthetisierten Modelle sind in Abb. 6.11 zu sehen. Während das native Verhalten des `LeadingVehicle` Blocks ebenfalls als kontinuierliches Modell realisiert ist (unter einem *Continuous* Director), verhält es sich mit der `WheelControllerCS_x64` FMU-CS ähnlich wie mit dem IDM-Verhalten. Diese sind nativ als DE Modell realisiert ist, stellen aber in der FMU-CS ein kontinuierliches Modell dar. Durch die eingesetzten, diskreten Integrator-Actors in den nativen Modellen ist daher eine bemerkbare Abweichung gegenüber der Co-Simulation mit den FMU-CS zu erwarten (vgl. Unterabschnitt 6.2.2.2).

Abschließend ist zu erwähnen, dass der in Abb. 6.11 (a) gezeigte SR Director mit seinem *period* Parameter ähnlich zum `Buffer` Block für eine synchrone fixed-step Co-Simulation der FMU mit einer Samplingfrequenz von 10 *Hz* sorgt. Mit dem Unterschied, dass sämtliche Ein- und Ausgänge einem globalen *Tick* folgend synchron gehalten werden und entweder einen Wert aufweisen oder explizit einen abwesenden Wert (engl. *absent*)[6].

[6]Befindet sich ein SR Director innerhalb eines DE Directors, wie es in den synthetisierten Modellen aufgrund des top-Level DE Directors der Fall ist, lehnt der SR Director die Ausführung der darin befindlichen Actors ab, wenn die Modellzeit nicht einem ganzzahligen Vielfachen n des *period* Parameters entspricht. Die `prefire()` Operation liefert dabei `false` zurück. Zwischen zwei Ticks $(n, (n+1)]$ eintreffende Tokens an ein oder mehreren Eingängen werden somit zum Zeitpunkt $(n+1)$ synchron vom SR Director verarbeitet und den verbundenen Actors zur Verfügung gestellt (über die `transferInputs()` Operation). Für nähere Details zur SR Semantik sei an dieser Stelle auf [136, Kap. 1.8] und [136, Kap. 5] verwiesen.

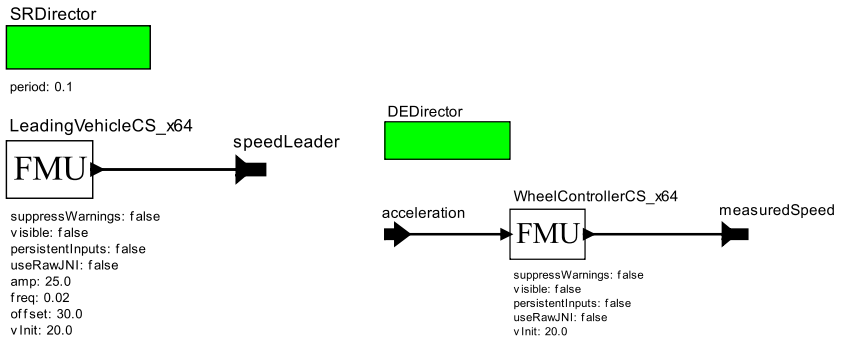


(a) Verhaltensmodell des ACC Controllers mithilfe einer Modelica-FMU zur Co-Simulation, die ein äquivalentes IDM-Modell implementiert.



(b) Synthetisiertes Modell des ACC Controllers. Der `Buffer` Block sorgt für eine synchrone fixed-step Co-Simulation der FMU mit einer Samplingfrequenz von 10 *Hz*.

Abbildung 6.10: Verhaltensmodell des ACC Controllers als FMU-CS (Abb. 6.10 (a)) und das resultierende, synthetisierte PtII Modell (Abb. 6.10 (b)).



(a) Synthetisiertes Modell des `Leading Vehicle` Blocks.

(b) Synthetisiertes Modell des `WheelController` Blocks.

Abbildung 6.11: Synthetisierte PtII Modelle der `LeadingVehicle` und `Wheel Controller` Blöcke, ausgetauscht durch FMU-CS Implementierungen.

6.2.2 Simulationsergebnisse und Diskussion

6.2.2.1 Simulationsergebnisse

Nachfolgend werden zunächst die Simulationsergebnisse der ACC Applikation durch Austausch der nativen IDM-Realisierung mit der o. g. aus Modelica exportierten FMU-CS präsentiert und verglichen. Dabei wird auch die Abweichung zwischen beiden Lösungen anhand der Kurvenverläufe der Beschleunigung und der Geschwindigkeiten analysiert. Im Anschluss daran werden wie in Unterabschnitt 6.2.1 beschrieben alle Verhaltensmodelle sukzessive durch entsprechende FMU-CS ausgetauscht und dieselben Messungen durchgeführt, um den weiteren Einfluss auf die Abweichung zu analysieren.

Die Abweichungen bzw. Fehler in Form des mittleren absoluten Fehlers (engl. *Mean Absolute Error (MAE)*) sowie des relativen Fehlers (engl. *Normalized Mean Absolute Error (NMAE)*) bezogen auf die native PtII Lösung werden dabei über folgende Formeln berechnet:

$$MAE = \frac{1}{|\mathbf{t_s}|} \sum_{i=1}^{|\mathbf{t_s}|} abs\left(y_i - \widehat{y}_i\right) \tag{6.4}$$

und

$$NMAE = \frac{MAE}{mean\,(abs\,(\mathbf{y}))} \tag{6.5}$$

Dabei ist t_s ein Sampling-Zeitpunkt, $|\mathbf{t_s}|$ die Anzahl aller Samples, y_i die Werte der PtII-Realisierung und $\widehat{y}_i$ die Werte der FMU-CS-Simulation.

Die Signalverläufe der nativen Realisierung verglichen mit jener, in der alle Funktionen durch FMU-CS ausgetauscht wurden, sind in Abb. 6.12 aufgetragen. In Tabelle 6.3 sind die Abweichungen der Kurvenverläufe der ACC Applikation mit sukzessiv ausgetauschten FMU-CS bezogen auf das integriert modellierte PtII Verhalten zusammengefasst.

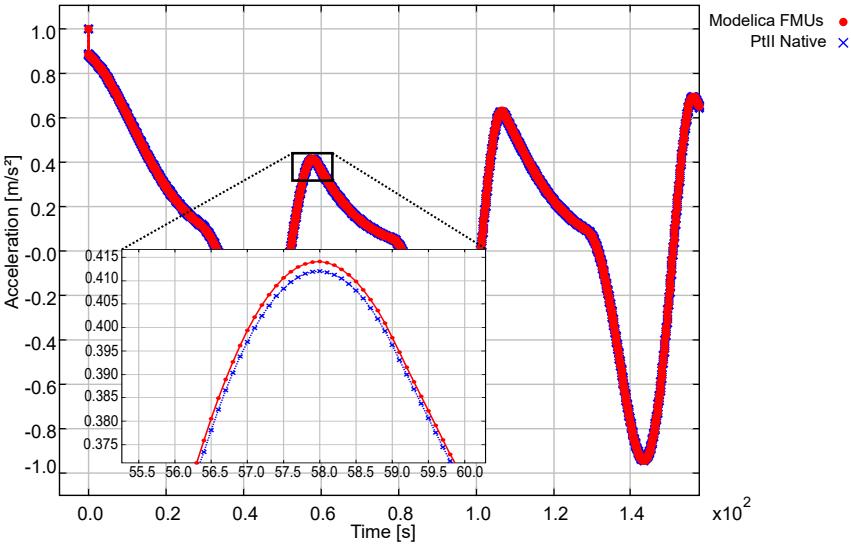
Tabelle 6.3: Abweichungen der Kurvenverläufe der ACC Applikation des nativen PtII Verhaltens zu sukzessiv ausgetauschten FMUs zur integrierten Co-Simulation.

	MAE			**NMAE**		
	Accel. $[m/s^2]$	**Ego Speed** $[m/s]$	**Leader Speed** $[m/s]$	**Accel.**	**Ego Speed**	**Leader Speed**
IDM FMU-CS	$8,293e^{-4}$	$4,809e^{-3}$	$0,0$	$2,125e^{-3}$	$1,585e^{-4}$	$0,0$
IDM & LeadingVehicle FMU-CS	$8,293e^{-4}$	$4,809e^{-3}$	$5,255e^{-7}$	$2,125e^{-3}$	$1,585e^{-4}$	$1,757e^{-8}$
Ausschl. FMU-CS	$1,595e^{-3}$	$1,490e^{-2}$	$5,255e^{-7}$	$4,087e^{-3}$	$4,911e^{-4}$	$1,757e^{-8}$

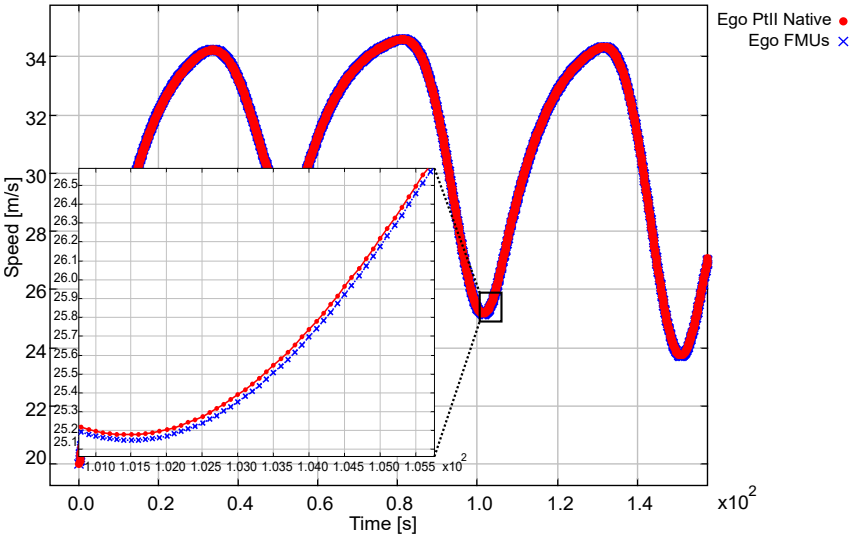
6.2.2.2 Diskussion

Wie Tabelle 6.3 zu erkennen bewirkt der Austausch der IDM Berechnung durch Co-Simulation mit der exportierten Modelica-FMU keine signifikante Abweichung zur diskreten PtII-Realisierung. Die relative Abweichung beträgt für die Beschleunigung ca. $0,21\,\%$ und für die durch das Dynamikmodell erzeugte Geschwindigkeit ca. $0,016\,\%$. Da das Verhalten für die Simulation der Geschwindigkeit des vorausfahrenden Fahrzeugs im ersten Fall nicht ausgetauscht wurde, beträgt die Abweichung korrekterweise $0,0\,\%$.

Tauscht man zusätzlich das Verhalten der Geschwindigkeit des vorausfahrenden Fahrzeugs durch die FMU-CS aus, ist ebenfalls noch keine signifikante Abweichung zu beobachten. Lediglich die relative Abweichung der Geschwindigkeit *LeaderSpeed* bewegt sich nun in der Größenordnung 10^{-8}, der Einfluss auf die Beschleunigungs- und Geschwindigkeitsberechnungen können somit vernach-



(a) Vergleich der nativ und FMU-CS berechneten IDM Beschleunigung (Ausschnitt).



(b) Vergleich der resultierenden Ego-Geschwindigkeiten mit den nativ und FMU-CS modellierten Funktionen (Ausschnitt).

Abbildung 6.12: Vergleich der nativ und FMU-CS berechneten IDM Beschleunigung sowie der daraus resultierenden Geschwindigkeiten.

lässigt werden. Die vernachlässigbar kleine Abweichung kann dadurch erklärt werden, dass das native PtII Verhalten wie die exportierte Modelica FMU-CS ebenfalls in der kontinuierlichen Domäne modelliert ist.

Ersetzt man schließlich das Dynamikmodell des `WheelControllerDE` Blocks durch die FMU-CS, tritt eine etwa doppelt so große Abweichung in der Beschleunigung von ca. $0,41\,\%$ auf. Auch die resultierende Geschwindigkeit *Ego Speed* erhöht sich auf ca. $0,049\,\%$, während der Fehler der Geschwindigkeit *LeaderSpeed* unverändert vernachlässigbar bleibt. Absolut betrachtet bewegen sich die mittleren absoluten Fehler jedoch in sehr kleinen Bereichen, zwischen ca. $0,00083\,m/s^2$ bzw. $0,0016\,m/s^2$ für die Beschleunigung und zwischen ca. $0,0048\,m/s$ bzw. $0,015\,m/s$ für die Ego-Geschwindigkeit im ersten bzw. dritten Fall. Für die resultierende Ego-Geschwindigkeit bedeutet das umgerechnet lediglich ca. $0,017\,km/h$ bzw. $0,054\,km/h$ mittlere Abweichung bei einem Dynamikbereich zwischen $20\,m/s = 72\,km/h$ und ca. $34,6\,m/s = 124,6\,km/h$. Die geringe absolute Abweichung wird in Abb. 6.12 durch den stark vergrößerten Bereich unterstrichen.

Die Fehlererhöhung lässt sich durch die eingangs erwähnte diskrete Modellierung der Integratoren in der nativen PtII-Realisierung erklären. Im ersten Fall wird der Integrator im IDM zur Bestimmung der Distanz in der Modelica FMU-CS kontinuierlich simuliert. Im dritten Fall wird zusätzlich der diskrete Integrator des Dynamikmodells durch das kontinuierliche FMU-CS Modell ersetzt. Der diskrete `Integrator` Actor von PtII führt lediglich eine lineare Interpolation der letzten beiden Events durch[7], während die FMU-CS zwischen zwei Iterationen der Co-Simulation, die durch den top-Level DE Director angestoßen werden, intern einen kontinuierlichen Solver mit variabler Schrittweite verwenden [117, 168]. Dieser variable Solver führt zu einem geringeren numerischen Fehler (vgl. Unterabschnitt 2.2.2.3). Zusätzlich kommt der Umstand hinzu, dass die beiden Blöcke `WheelController` und `FMU_IDM` durch die Rückkopplungsschleife direkt voneinander abhängen und so den diskreten Integrationsfehler verstärken.

Zusammenfassend lässt sich schließen, dass eine sukzessive Integration von FMU-CS durch einfaches Ändern des Mappings bei gleichzeitiger Berücksichtigung der Architekturaspekte möglich ist und somit die Wiederverwendung werkzeugunabhängiger Simulationsmodelle durch die verteilte Entwicklung gesteigert wird. Bezüglich der Co-Simulation der FMU-CS können die Schrittweiten jener individuell konfiguriert werden, bspw. über den genannten *period* Parameter des SR Directors. Der DE Director auf oberster Ebene sorgt für eine deterministische Ausführung des Modells. Die aspektorientierte Synthese der *Composite Actors*, in denen die FMU-CS gekapselt sind, erlaubt darüber hinaus

[7] Siehe https://icyphy.github.io/ptII-test/doc/codeDoc/ptolemy/domains/de/lib/Integrator.html, zuletzt aufgerufen am 07.10.2019.

die Berücksichtigung weiterer Architekturaspekte. In den präsentierten Fällen die Netzwerkkommunikation und Ausführungslatenzen. Zudem sind variable Schrittweiten individueller FMU-CS während der Co-Simulation möglich, wenn ein SR Director von einem DE Director gesteuert wird [136, S. 27].

Eine hybride Co-Simulation, bei der FMUs selbst diskrete Events erzeugen und verarbeiten können müssen, ist nicht betrachtet worden, da sich der Standard diesbezüglich noch in der Entwicklung befindet. Es existieren jedoch einige Aufwände in dieser Forschungsrichtung (vgl. [17, 18, 19, 29]), die in zukünftigen Arbeiten herangezogen werden können.

6.3 Ebenenübergreifende Modellierung der ACC Applikation

In diesem Abschnitt wird die in Abschnitt 6.1 untersuchte ACC Applikation um ein ebenenübergreifendes Verhaltensmodell erweitert und zudem eine 1-zu-1 Abbildung zwischen den logischen Funktionen der LA und den realisierenden Verhaltensmodellen auf der BLA Ebene umgesetzt, damit die dort beschriebenen Nachteile vermieden werden. Darüber hinaus werden in der Demonstration folgende Ziele verfolgt:

- Modellierung einer erweiterten ACC-Funktion bestehend aus einem Actor-orientierten und zustandsbasierten Verhaltensmodell.
- Untersuchung eines ebenenübergreifenden Verhaltensmodells mit bidirektionalen Datenabhängigkeiten zwischen der ACC-Funktion und der ausführenden ECU mithilfe der eingeführten Basisservice Interfaces.
- Untersuchung gemeinsam genutzter Funktionsressourcen durch die Synthese eines vom Funktions-Mapping abhängigen Multicore FIFO-Schedulings anhand erweiterter Ausführungsaspekten.
- Simulation der quasi-statischen Stromaufnahme der ausführenden Hardwarekomponenten anhand des ebenenübergreifenden Verhaltensmodells.

Der Einfluss von vernetzungsabhängiger Buskommunikation wird in dieser Fallstudie nicht betrachtet.

6.3.1 Modellierung der logischen Architektur

Das abgewandelte ACC-Modell der logischen Architektur zur Einhaltung der 1-zu-1 Abbildung zwischen den logischen Funktionen und den Verhaltensblöcken ist in Abb. 6.13 illustriert. Das Verhalten jeder logischen Funktion, bis auf die neu hinzugefügte `ACC_Testbench`, ist durch ein dem Funktionstyp hinterlegten State

Chart modelliert. Das Verhalten der `ACC_Testbench` Funktion ist Actor-orientiert dahingegen modelliert und ist nicht auf die Hardware abgebildet. Diese Funktion fungiert als Stimuli der Geschwindigkeiten für die beiden Sensorfunktionen sowie der Distanz für die ACC-Funktion. Die Werte werden in einer Feedback-Schleife mit einer globalen Sampling-Periode von 100 ms erzeugt. Die initialen Werte für die Geschwindigkeiten liegen im untersuchten Fall bei 15 m/s sowie bei einem initialen Abstand zwischen beiden Fahrzeugen von 200 m.

Jede der involvierten Funktionen, wiederum bis auf die `ACC_Testbench`, spezifiziert dabei die eingeführten Basisservice Interfaces, um einen bestimmen Betriebsmodus ihrer ausführenden Hardwarekomponente anzustoßen bzw. anzufragen. Die dazugehörigen Verhaltensblöcke sowie deren Mappings zu den LA Funktionen werden dabei durch die in Unterabschnitt 5.3.4 beschriebene Metrik automatisiert generiert.

6.3.2 Ebenenübergreifende Realisierung des Verhaltens

6.3.2.1 State Charts

Das State Chart der `ACC` Funktion ist in Abb. 6.14 zu finden, die dazugehörige Realisierung des State Charts des ausführenden Mikroprozessors als AUTOSAR-orientierter *Fixed ECU State Manager* [9] ist in Abb. 6.15 dargestellt. Die verbleibenden Realisierungen der Sensor- und Aktuatorfunktionen sind vergleichsweise trivial umgesetzt, da der Fokus auf der vom Betriebsmodus abhängigen Ausführung der ACC-Funktion liegt. Die Sensorfunktionen leiten die Stimuli an die ACC-Funktion weiter und die Aktuatorfunktion empfängt die berechnete Beschleunigung des ACC. Solange Werte empfangen werden, fordern die Funktionen den `run` Betriebsmodus der Sensor- und Aktuator-Hardwarekomponenten an.

Die ACC-Funktion berechnet die Beschleunigung nur, wenn der ECU State Manager den Betriebsmodus `run` zurückliefert, ansonsten verbleibt das State Chart im Zustand `initial`. Im entsprechenden orthogonalen `operate` Zustand wird die Beschleunigung anhand des IDM in einem Action-Ausdruck nach Gl. (6.3) berechnet und mittels der Zustandsvariablen ***aMax*** und ***aMin*** im parallelen Zustand begrenzt[8]. Das State Chart geht in den Zustand `freeRoad` über und fordert den `sleep` Zustand im ECU State Manager an, wenn der Radar kein Fahrzeug detektiert bzw. die Distanz über einem parametrierbaren Schwellwert liegt *und* sobald die Wunschgeschwindigkeit v_0 erreicht wurde. An dieser Stelle unterschei-

[8] Die Zustandsvariablen ***aMax*** und ***aMin*** entsprechen den konstanten Parametern a und a_{min} des IDM in Tabelle 6.2.

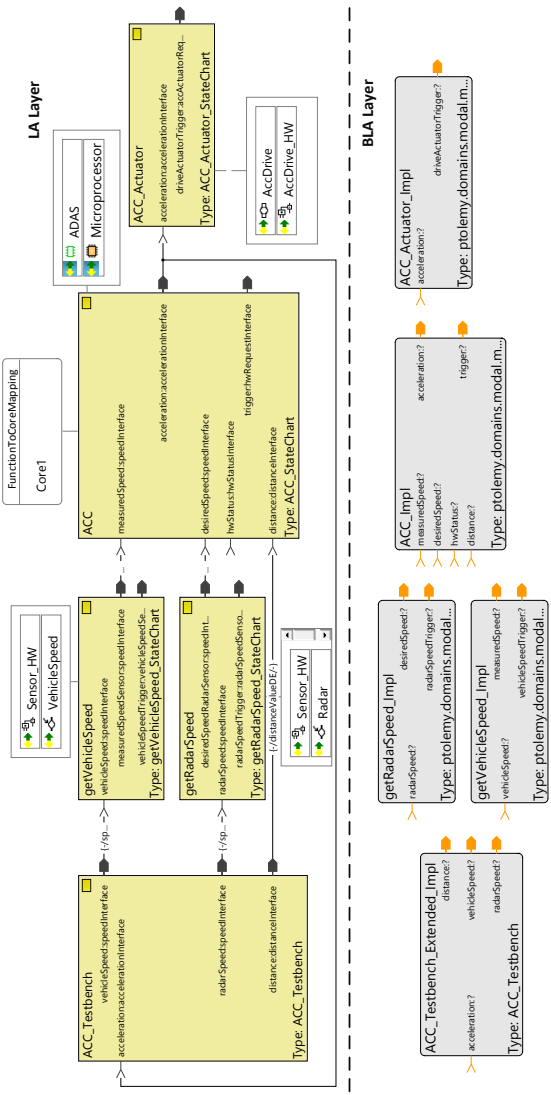


Abbildung 6.13: Modifizierte logische Architektur der ACC Applikation zur Berücksichtigung von 1-zu-1 Abbildungen mit den Verhaltensblöcken. Die Mappings zur Hardwarearchitektur sowie die eingeführten Core Sub-Mappings sind durch die annotierten Textboxen illustriert (Quelle: erweiterte Darstellung nach [BKB19c]).

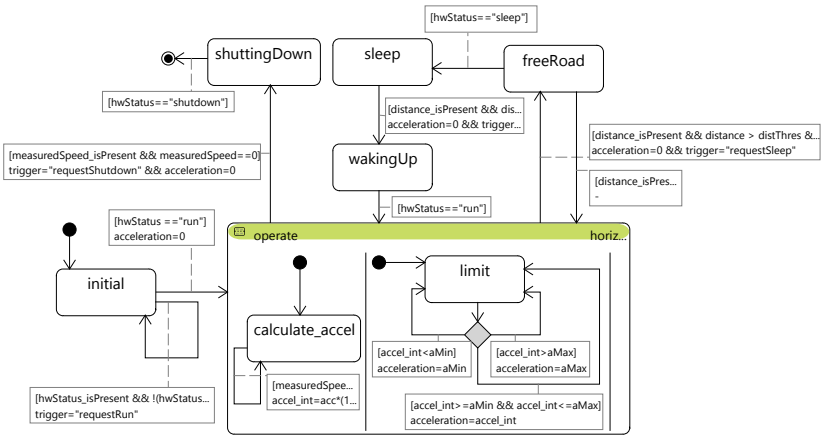


Abbildung 6.14: State Chart der `ACC` Funktion. Einige Guard- und Action-Ausdrücke sind aus Platzgründen nicht vollständig dargestellt (Quelle: modifizierte Realisierung nach [BKB19a]).

det sich die Realisierung von der in [BKB19a] vorgestellten, welche bereits in den `freeRoad` Zustand übergeht, sobald die Distanz den Schwellwert übertritt. Ein Aufweckprozess wird angestoßen, sobald der Radar ein neues Fahrzeug detektiert, d. h. sobald der Abstand unter dem eingestellten Schwellwert liegt und das State Chart in den Zustand `wakingUp` übergeht. Ein Herunterfahren der Hardware wird im ECU State Manager angefordert, wenn das Fahrzeug zum Stillstand kommt.

Der ECU State Manager reflektiert die möglichen Betriebszustände des Steuergeräts, die von der ACC Applikation über die Basisservice Interfaces angestoßen bzw. zurückgeliefert werden können. Zusätzlich referenziert das State Chart die Aufstart- und Nachlaufzeit Attribute *`startupTime`* und *`provisionTime`* der ECU sowie eine Zustandsvariable *`wakeupTime`* zur Abbildung einer Aufweckzeit. Die Nachlaufzeit ist ein Zeitintervall, in dem eine Hardwarekomponente aktiv bleibt, nachdem ihr Herunterfahren angefordert wurde, bspw. um einen erneuten Aufweckprozess anzustoßen.

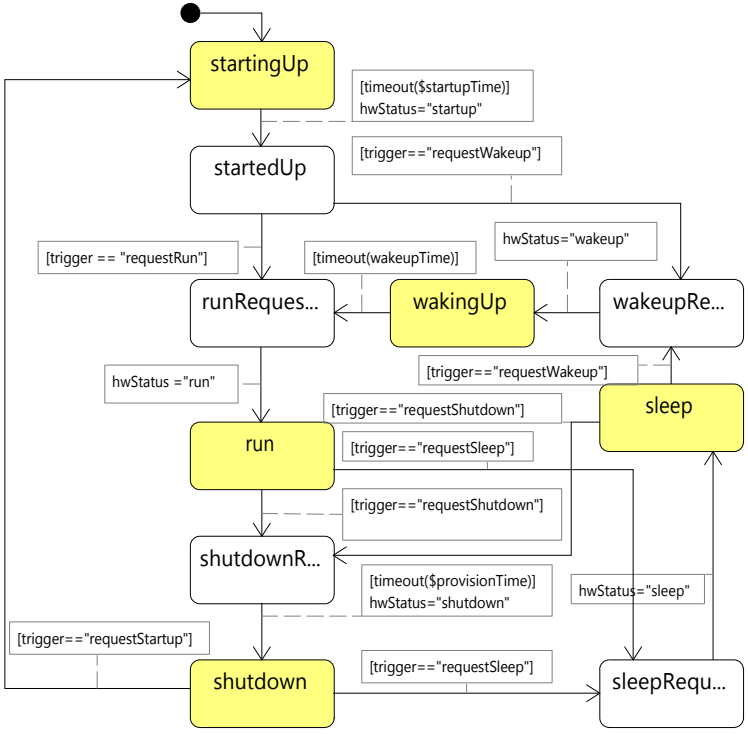


Abbildung 6.15: State Chart der ausführenden Hardware der `ACC` Funktion, welches den an AUTOSAR orientierten Fixed ECU State Manager [9] realisiert. Die Transitionen zu den in Gelb hinterlegten Zuständen spezifizieren dabei die quasi-statischen Stromaufnahmen im entsprechenden Betriebsmodus über ein Mapping der Strombeschreibungstypen (vgl. Abb. 5.11) (Quelle: [BKB19a]).

6.3.2.2 ACC Testbench

Wie in Unterabschnitt 6.3.1 eingeführt wird die Funktion `ACC_Testbench` Actor-orientiert umgesetzt, deren Actor-Modell auf oberster Hierarchieebene in Abb. 6.16 zu finden ist. Im Wesentlichen besteht die Testbench aus der Dynamik zur Umsetzung der vom ACC berechneten Beschleunigung in die Geschwindigkeit sowie aus drei weiteren hierarchischen Building Blocks, die folgende Aufgaben erfüllen.

- `speedProfile`: Gibt das Geschwindigkeitsprofil des vorausfahrenden Fahrzeugs vor.
- `Scenario`: Modelliert das Fahrszenario zur Evaluation der ACC Applikation, z. B. das Detektieren eines (neuen) vorausfahrenden Fahrzeugs.
- `Distance`: Berechnet die Distanz zwischen beiden Fahrzeugen und setzt diese ggf. dem Szenario folgend auf einen neuen Anfangswert.

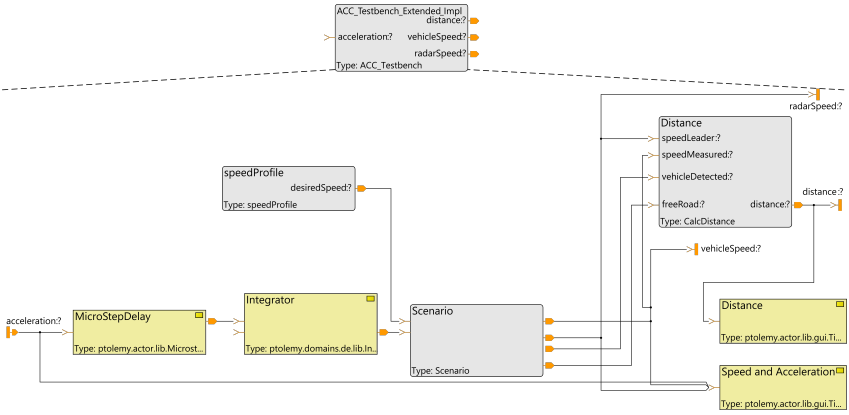


Abbildung 6.16: Actor-orientierte Modellierung der ACC Testbench.

6.3.3 Synthese

In diesem Abschnitt werden zwei Fallstudien präsentiert, zunächst ohne die Abbildung von Funktionen auf Rechenkerne. Die vom Mapping abhängige

Synthese und Simulation von erweiterten Ausführungsaspekten wird in Unterabschnitt 6.3.4.2 behandelt. Die gewählten Werte der Zustandsvariablen, ECU-Attribute sowie Actor-Parameter für die Simulation sind in Tabelle 6.4 zusammengefasst. Das resultierende, synthetisierte PtII-Modell ist in Abb. 6.17 illustriert. Die quasi-statische Stromaufnahmen der Hardwarekomponenten abhängig vom Betriebszustand sind dabei als Strombeschreibungstypen modelliert und den entsprechenden Transitionen im State Chart zugewiesen (vgl. Abb. 5.11).

Im synthetisierten PtII-Modell ist anhand der Icons der zusammengesetzten Actors die Realisierung der `ACC_Testbench` als Actor-orientiertes Modell illustriert, während die restlichen zusammengesetzten Actors Modal Models darstellen. Unter den Actors sind die in Tabelle 6.4 modellierten Parameter abgebildet und es ist die bidirektionale Verbindung zwischen dem Modal Model der ACC-Funktion `ACCImpl` und dem Modal Model der ausführenden `ACC ECU` über die Ports der Basisservice Interface `hwStatus` und `trigger` zu erkennen. Im Gegensatz dazu sind für die Sensor- und Aktuatorfunktionen lediglich unidirektionale Verbindungen zur Hardware modelliert (vgl. logische Architektur in Abb. 6.13). Des Weiteren sind während der Synthese zusätzliche Ports an den State Charts der Hardwarekomponenten erzeugt worden, die die quasi-statischen Strombeschreibungen beim Übergang in den entsprechenden Zustand als Output-Action ausgeben und über den Plotter-Actor `Currents` visualisieren.

Tabelle 6.4: Gewählte Parameterwerte für die ebenenübergreifende Simulation der ACC Applikation.

Parameter	Wert
ACC Funktion	
Maximale Beschleunigung a_{max}	$1.4\,m/s^2$
Maximale Bremsverzögerung a_{min}	$-9,0\,m/s^2$
Abstandsschwellwert d_{thres}	$200\,m$
Wunschgeschwindigkeitstoleranz v_0^{Tol}	$0,002\,m/s$
ACC ECU State Manager	
Aufstartzeit *startupTime*	$50\,ms$
Nachlaufzeit *provisionTime*	$200\,s$
Aufweckzeit *wakeupTime*	$10\,ms$
Stromaufnahme im Zustand `startingUp`	$1,5\,A$
—„— `run`	$1,0\,A$
—„— `wakingUp`	$0,5\,A$
—„— `sleep`	$0,2\,A$
—„— `shutdown`	$0,1\,A$
Radarsensor	
Nachlaufzeit *provisionTime*	$200\,s$
Stromaufnahme im Zustand `idle`	$0,1\,A$
—„— `run`	$0,5\,A$
Geschwindigkeitssensor	
Nachlaufzeit *provisionTime*	$200\,s$
Stromaufnahme im Zustand `idle`	$0,1\,A$
—„— `run`	$0,3\,A$
ACC Aktuator	
Nachlaufzeit *provisionTime*	$200\,s$
Stromaufnahme im Zustand `idle`	$0,1\,A$
—„— `startingUp`	$1,0\,A$
—„— `run`	$2,0\,A$
IDM[a]	
Wunschgeschwindigkeit v_0	$23\,m/s$
Maximale Beschleunigung a	$1,4\,m/s^2$
Komfortable Bremsverzögerung b	$2,0\,m/s^2$
Sicherheitszeitabstand (Folgezeit) T	$1,5\,s$
ACC Testbench	
Initialer Abstand d	$200\,m$
Abstandsschwellwert d_{thres}	$200\,m$
Initiale Geschwindigkeit v_{init}	$15\,m/s$
Stopp-Zeitpunkt des Fahrzeugs *vehicleStopTime*	$380\,s$

[a] Abweichend von Tabelle 6.2.

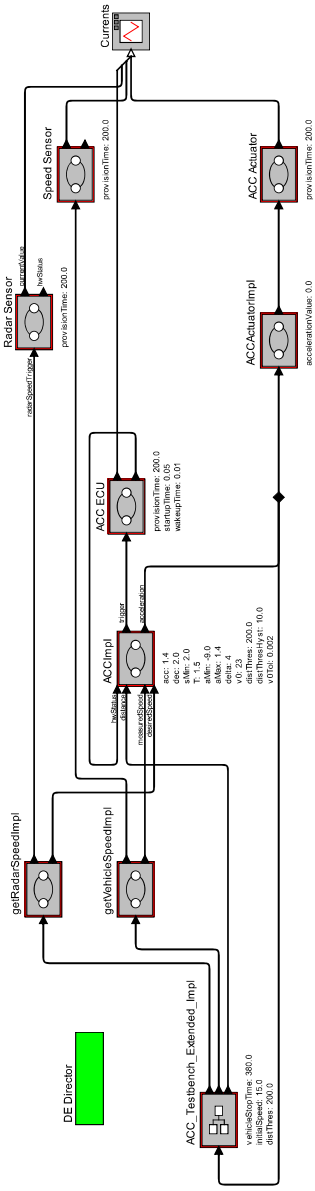


Abbildung 6.17: Synthetisiertes PtII-Modell der ebenenübergreifenden ACC Applikation, visualisiert in der grafischen Oberfläche Vergil.

6.3.4 Simulationsergebnisse und Diskussion

6.3.4.1 Ebenenübergreifende Simulation

Im oberen Plot von Abb. 6.18 sind die berechnete ACC Beschleunigung und die Geschwindigkeiten beider Fahrzeuge der ausgeführten Simulation der ebenenübergreifenden ACC Applikation zu sehen, während im unteren Plot die Distanz aufgetragen ist[9].

Bis 250 s folgt das ACC-geregelte Fahrzeug der Geschwindigkeit des vorausfahrenden Fahrzeugs bis maximal der Wunschgeschwindigkeit $v_0 = 23\,m/s$ und gemäß des dynamischen Wunschabstandes s^* des IDM, was sich in der aufgetragenen Distanz widerspiegelt. Zudem ist zu Beginn zu beobachten, dass die Aufstartzeit von 50 ms, in der die ACC-Funktion keinen Beschleunigungswert berechnen kann, keinen signifikanten Einfluss auf das Folgeverhalten hat. Die Beschleunigung bewegt sich im gesamten Zeitraums des Folgefahrens und Annäherns im Rahmen der parametrierten maximalen Beschleunigung a_{max} und der komfortablen Bremsverzögerung b.

Ab $t = 250\,s$ wird im Szenario durch das Setzen einer sehr hohen Geschwindigkeit und Distanz von $80\,m/s$ bzw. $3000\,m$ [61, S. 19] ein aus dem Sichtfeld des Radars verschwindendes Fahrzeug und damit eine freie Straße simuliert. Nach Erreichen der Wunschgeschwindigkeit v_0 abzgl. der Toleranz v_0^{Tol} geht die ACC-Funktion in den Zustand `freeRoad` über. Bei dieser Zustandstransition zum Zeitpunkt $t = 287,0\,s$ wird der `sleep` Betriebszustand der ECU angefordert und aktiviert.

Zum Zeitpunkt $t = 300\,s$ wird die Detektion eines einscherenden Fahrzeugs mit geringerer Geschwindigkeit in einer Distanz von $25\,m$ durch den Radarsensor simuliert, was ein Aufweckprozess der ECU anstößt und welcher nach den spezifizierten 10 ms abgeschlossen ist. Das ACC ist demnach zum Zeitpunkt $t = 300,01\,s$ erneut betriebsbereit und geht in den Zustand `operate` über. Aufgrund der erhöhten Geschwindigkeit relativ zum vorausfahrenden Fahrzeug von $\Delta v \approx (23 - 15)\,m/s = 8\,m/s$ und des geringen Abstands von $s = 25\,m$, wird eine kritische Situation provoziert, bei der die kinematisch notwendige

[9] Die Ausführung der integrierten Simulation in der *Ptolemy Run Control* Ansicht innerhalb von PREEvision® ist im Anhang in Abb. A.5 zu finden.

Bremsverzögerung [166, S. 164]

$$b_{kin} = \frac{v^2}{2s} = \frac{(23\,m/s)^2}{2 \cdot 25\,m} \approx 10,58\,m/s^2 \tag{6.6}$$

beträgt. Da diese deutlich über der parametrierten, komfortablen Bremsverzögerung von $b = 2,0 m/s^2$ liegt, wird das IDM eine kurzzeitig über der kinematisch notwendigen Bremsverzögerung berechnen, um die Situation unter Kontrolle zu bringen [166, S. 164] [61, S. 20].

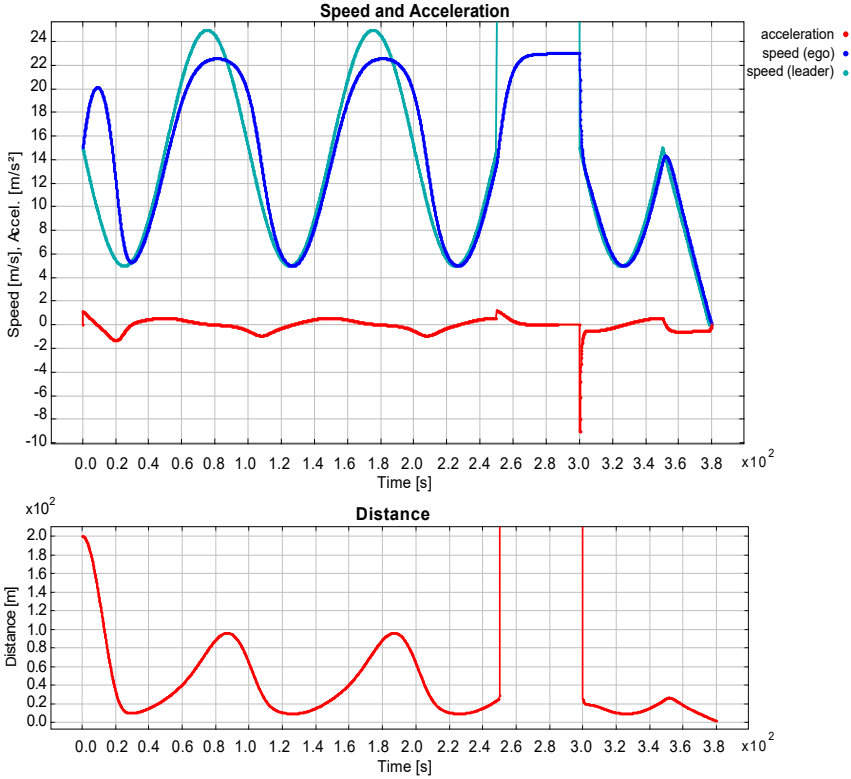


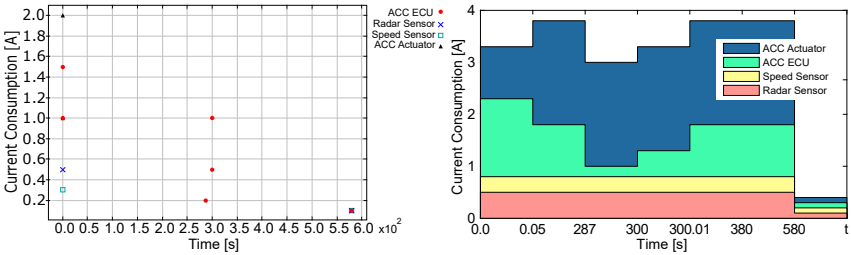
Abbildung 6.18: Simulationsergebnisse der ebenenübergreifenden ACC Applikation, stimuliert durch das Actor-orientiert modellierte Szenario der ACC Testbench.

Zum Zeitpunkt $t = 300\,s$ würde nach Gl. (6.3) eine IDM-Beschleunigung von $\tilde{a}_{mic}(25\,m,\ 23\,m/s,\ 8\,m/s) \approx -12,18\,m/s^2$ resultieren. Die aufgrund des einscherenden Fahrzeugs adaptierte Beschleunigung kann vom ACC-System, bedingt durch die Aufweckzeit, allerdings erst ein Sample verzögert bei $t = 300,1\,s$ berechnet werden. Zum Zeitpunkt $t = 300,1\,s$ wird diese zusätzlich gemäß der modellierten, physikalischen Limitierung auf die maximale Bremsverzögerung von $a_{min} = -9,0\,m/s^2$ begrenzt. Wie jedoch im folgenden Verlauf der Geschwindigkeit und der Distanz zu entnehmen ist, garantiert die vom IDM berechnete Beschleunigung aufgrund seiner kollisionsfreien Charakteristik weiterhin ein sicheres Annähern.

Ab $t = 350\,s$ bremst das Ego-Fahrzeug aufgrund der Verzögerung des vorausfahrenden Fahrzeugs ab, bis es bei $t = 380\,s$ bei einer Distanz nahe dem Minimalabstand s_0 zum Stillstand kommt. Das Szenario der ACC Testbench gibt dabei den Stillstand des Fahrzeugs durch Setzen der Geschwindigkeit auf null vor, was z. B. durch einen manuellen Bremsvorgang des Fahrers bei geringer Geschwindigkeit ausgelöst werden kann. Die ACC-Funktion setzt daher an dieser Stelle die Beschleunigung ebenso auf null.

In Abb. 6.19 ist die quasi-statische Stromaufnahme abhängig vom Betriebszustand der involvierten Hardwarekomponenten während der Simulation der ACC Applikation an den Schlüsselereignissen zu sehen, wobei die gewählten Stromwerte fiktiv angenommen wurden. Abb. 6.19 (a) zeigt dabei den während der Simulationsausführung aufgezeichneten Plot des `Currents` Actors in Abb. 6.17, während Abb. 6.19 (b) hingegen die kumulierte Stromaufnahme illustriert. Letztere wurde anhand der zurückgeführten Simulationsdaten durch den in Unterabschnitt 5.3.5.2 beschriebenen Listener erstellt. Die Daten wurden dabei innerhalb der PREEvision® Umgebung durch einen State Chart Observer in eine CSV-Datei geschrieben (vgl. die Rückführungssequenz in Unterabschnitt 5.4.2.2).

Dem Stromverlauf ist zu entnehmen, dass die Sensoren bis zu ihrem Herunterfahren operieren und einen Strom im `run` Zustand von $0,5\,A$ bzw. $0,3\,A$ aufnehmen. Im Gegensatz dazu konsumieren die ECU als auch der Aktuator während der Aufstartphase bis $50\,ms$ einen entsprechenden Strom, nach welcher die reguläre Betriebsstromaufnahme von $1,0\,A$ bzw. $2,0\,A$ folgt. Zum Zeitpunkt $t = 287,0\,s$ wird der `sleep` Zustand der ECU aktiv und die Stromaufnahme reduziert sich auf die modellierten $0,2\,A$ im Ruhemodus. Nach $300\,s$ wird der Aufweckprozess der



(a) Simulationsplot der Stromaufnahme der Hardwarekomponenten.

(b) Kumulierte Stromaufnahme, erstellt basierend auf den zurückgeführten Daten.

Abbildung 6.19: Quasi-statische Stromaufnahme der involvierten Hardwarekomponenten in der ebenenübergreifenden Simulation der ACC Applikation (Quelle: [BKB19c]).

ECU gestartet und für 10 ms die modellierten 0,5 A konsumiert, bis die ECU wieder voll betriebsbereit ist und die im operierenden Betriebszustand modellierten 1,0 A konsumiert werden. Nachdem das Fahrzeug zum Zeitpunkt $t = 380\,s$ zum Stillstand kommt, bleiben alle Hardwarekomponenten entsprechend der modellierten Nachlaufzeit von 200 s aktiv, bis sie schließlich nach 580 s herunterfahren und einen Ruhestrom von 0,1 A verbrauchen.

6.3.4.2 Multicore FIFO-Scheduling

Im Folgenden wird die Synthese und die Simulation von erweiterten Ausführungsaspekten von Funktionen, abhängig vom Mapping der Funktionen auf ihre ausführenden Rechenkerne, in Form eines FIFO-Schedulings demonstriert und untersucht. Im Speziellen werden die Ausführungsaspekte auf die synthetisierten Modal Models angewandt, welche wie in Unterabschnitt 5.3.7.4 beschrieben eine Erweiterung des ausführenden DE Directors notwendig machten.

Zur Evaluation wurde eine weitere Funktion zur logischen Architektur hinzugefügt, die durch ein State Chart im Verhalten verfeinert ist. Diese realisiert eine periodische Dummy-Funktion, welche sich selbst alle 100 ms triggert, sobald sie den `run` Zustand des ECU Managers empfängt. Verbindungen zu anderen Funktionen bestehen nicht, da der Fokus auf der Analyse des Schedulings mit der ACC-Funk-

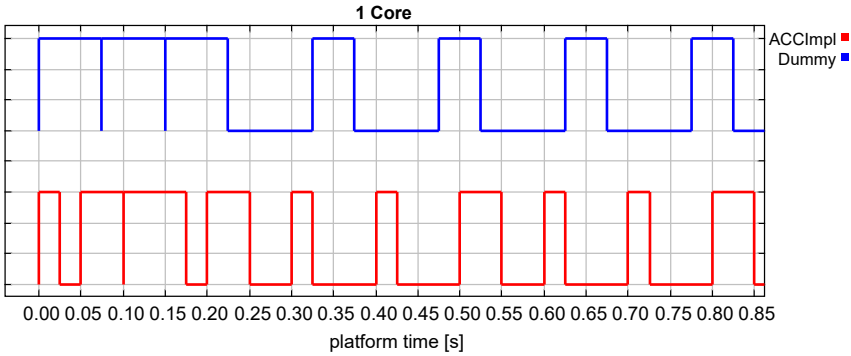
tion liegt. Die ACC- und die Dummy-Funktion spezifizieren auf der logischen Architektur jeweils eine maximale Ausführungszeit[10] von $t_L^{max}(ACC) = 25\,ms$ bzw. $t_L^{max}(Dummy) = 50\,ms$.

In Abb. 6.20 sind die Ergebnisse von zwei Simulationsausführungen dargestellt, bei welchen die beiden Funktionen mithilfe der eingeführten Funktion-zu-Core Sub-Mappings (1) einem gemeinsamen Rechenkern und (2) dedizierten Rechenkernen des ausführenden Mikroprozessors zugewiesen sind. Das synthetisierte PtII-Modell mit Ausführungsaspekt ist im Anhang in Abb. A.4 abgedruckt, wobei der darin abgebildete Ausführungsaspekt `Microprocessor` der ausführenden Einheit im Architekturmodell entspricht. Der Aspekt enthält demnach abhängig vom Funktions-Mapping ein oder zwei Kerne und führt nach Abb. 5.26 ein FIFO-Scheduling der Funktionen durch.

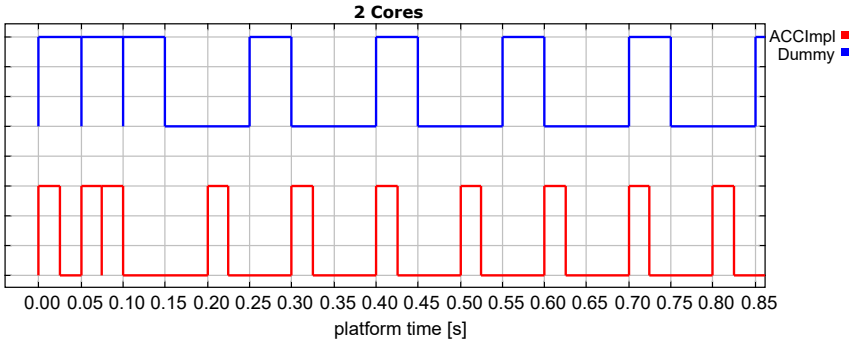
Die Verläufe der Ausführungszeiten einer Funktion in Abb. 6.20 repräsentieren dabei das Zeitintervall ab dem Zeitpunkt der Kontaktaufnahme zum Scheduler mit der Anfrage auf Ausführung, bis zum Zeitpunkt der vollständig abgeschlossenen Abarbeitung der Funktion mit der spezifizierten Ausführungszeit. Insbesondere ist zu beobachten, dass jedes von einer Funktion empfangene Event mit einem Zeitstempel (τ_1, n_1) genau einmal vom Scheduler verarbeitet wird, unabhängig davon, ob während der Ausführung der Funktion weitere, interne Events durch Transitionen mit (τ_1, n_m), $n_m > n_1$ ausgelöst werden. Dies verifiziert die in Unterabschnitt 5.3.7.4 beschriebene Vorgehensweise zur Simulation von Ausführungsaspekten mit Modal Models und die Einhaltung derer ursprünglichen Semantik.

Zusätzlich ist im Falle der gewählten Ausführungszeiten und Parameter in Tabelle 6.4 zu beobachten, dass beide Schedules bis auf die Aufstartphase der ACC-Funktion, durchführbar sind. Die Betriebsmodi `startingUp`, `startUp` und `run` vom ECU State Manager sorgen für die ersten drei Ausführungsverzögerungen beider Funktionen, wobei die Verarbeitung des `run` Betriebsmodus für den Übergang in den `operate` Zustand der beiden Funktionen sorgt. Allerdings kann ein Beschleunigungswert basierend auf den ersten Sensorwerten der Testbench zu den Zeitpunkten $t = n \cdot 0,1\,s$ für $n = 0,1$ nicht berechnet werden, da sich die Funktion noch im Zustand `initial` befindet.

[10] `LatencyTime` Attribut einer logischen Funktion, vgl. Abb. 5.26.



(a) Ausführungszeiten der zwei Funktionen auf einem gemeinsamen Kern.



(b) Ausführungszeiten der zwei Funktionen auf dedizierten Kernen.

Abbildung 6.20: Ausschnitt der Ausführungszeiten eines FIFO-Schedulings der ACC-Funktion und einer zyklischen Dummy-Funktion abhängig vom Mapping auf einen gemeinsamen oder dedizierte Rechenkerne (Quelle: [BKB19c]).

In beiden Fällen, dem gemeinsamen und bei dedizierten Rechenkernen, kann die Beschleunigung erst im übernächsten Zyklus ($t = 0,2\,s$) berechnet werden, wenn der Zustand `operate` bereits aktiv ist. In Fall (1) wird der `operate` Zustand nach $t = 0,175\,s$ bzw. in Fall (2) nach $t = 0,1\,s$ aktiv. Die gleichzeitig eintreffenden Sensorwerte bei $t = 0,1\,s$ in Fall (2) werden jedoch verworfen, da hier der Übergang in den Zustand `operate` stattfindet. Die Samples in beiden Fällen werden

ab $t = 0,2\,s$ alle $100\,ms$ rechtzeitig verarbeitet, während gleichzeitig die Dummy-Funktion alle $100\,ms$ ihren Ausführungswunsch durchführen kann.

Trotz des Einflusses von Ausführungsverzögerungen durch das Scheduling verhält sich die ACC Funktion in allen Phasen korrekt, auch in der kritischen Phase des einscherenden Fahrzeugs. Eine signifikante Abweichung in den Beschleunigungs-, Geschwindigkeits- und Abstandsverläufen gegenüber der rein funktionalen Simulation ohne Ausführungsaspekte wurde dabei nicht festgestellt.

6.4 Elektrische und leitungssatzsensitive Simulation

In diesem Abschnitt wird die elektrische und leitungssatzsensitive Simulation prototypisch demonstriert und umfasst folgende Fallstudien:

- Genauigkeits- und Performanzanalyse des neuen aspektorientierten QSS-Ansatzes zur verfeinerten Verhaltenssimulation mit elektrischen Schaltungen anhand eines Abwärtswandlers (engl. *buck converter*).
- Erweiterte elektrische Simulation des Abwärtswandlers in Kombination mit einem diskret PWM-geregelten Gleichstrommotor als dynamische Last.
- Leitungssatzsensitive Simulation der Spannungsversorgung eines Teil-Bordnetzes mithilfe der Modellierung von dynamischen und statischen Stromverbrauchern.

6.4.1 Abwärtswandler

6.4.1.1 Ebenenübergreifende Modellierung

Das ebenenübergreifende Modell zur verfeinerten Verhaltenssimulation der logischen Architektur mithilfe des Abwärtswandlers, welcher auf der elektrischen Hardware-Ebene modelliert wird, ist in Abb. 6.21 dargestellt.

Die logischen Funktionen `PowerSupply` und `PWM` sind jeweils 1-zu-1 auf ihre dazugehörigen Hardwarekomponenten abgebildet und stellen die Spannungsversorgung sowie ein PWM-Signal zur Steuerung der Spannung an die logische Aktuatorfunktion `eMotor` bereit. Das Verhalten der Spannungsversorgung und

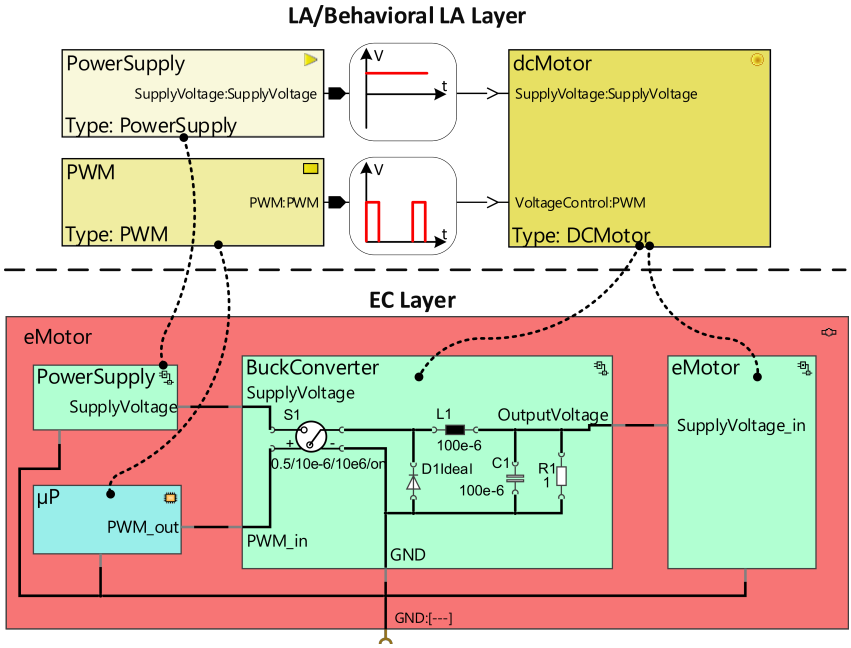


Abbildung 6.21: Logische Architektur und Verfeinerung auf der elektrischen Hardware-Ebene durch einen Abwärtswandler.

des PWM-Signals werden auf der BLA Ebene durch Signalquellen modelliert, welche `SmoothTokens` als stückweise stetige Signale versenden (vgl. Unterabschnitt 5.3.8.2). Das Verhalten des `eMotors` ist in diesem Fall nicht modelliert, sondern zeichnet die durch den Abwärtswandler erzeugte, anliegende Spannung auf.

Die logischen Ports der Spannungsversorgung und des PWM-Signals sind als analoge Ports konfiguriert und auf die Pins der dazugehörigen Hardwarekomponente bzw. des Mikroprozessors abgebildet (vgl. Unterabschnitt 5.2.5.2). Die Aktuatorfunktion wird dahingegen von den beiden Hardwaremodulen `eMotor` und `BuckConverter` realisiert. Letzterer sorgt für die verfeinerte, elektrische Modellierung und Simulation der am Motor ankommenden Spannung, indem die beiden analogen Ports `SupplyVoltage` und `VoltageControl` auf den Pin

`SupplyVoltage_in` des `eMotors` bzw. auf den Pin `PWM_in` des Abwärtswandlers abgebildet werden.

Der Abwärtswandler verwendet dabei einen idealen, spannungsgesteuerten Schalter `S1` sowie eine ideale Diode `D1Ideal`, die nach [112] als interner, idealer Schalter implementiert wurden. Zur Simulation der Schalter wurden in JSpice sowohl spannungsgesteuerte als auch interne Schalter hinzugefügt, welche das Modell in [92] implementieren[11]. Beide Arten von Schaltern repräsentieren dabei einen Widerstand R_S mit einem niederohmigen Wert R_{ON} oder einem hochohmigen Wert R_{OFF}, abhängig vom Schalterzustand. Die beiden Werte können dabei als Parameter der Schalter- und Diodenbauelemente im Architekturmodell konfiguriert werden. Der spannungsgesteuerte Schalter verfügt zusätzlich über eine Schwellenspannung, ab der der Schalter schließt bzw. öffnet (vgl. Abb. 6.21).

Zur Modellierung dieser Schalter auf der elektrischen Ebene wurden in PREEvision® entsprechende Bauelementtypen (engl. *Hardware Device Types*) angelegt. Die zusätzlich zu den passiven Bauelementen gewählten Parameter der Abwärtswandlerschaltung sind in Tabelle 6.5 zusammengefasst. Da keine der Hardwarekomponenten an externe Masseleitungen angeschlossen ist, werden sowohl die Signale der analogen Ports als auch die Bauelemente während der Synthese zur Referenzmasse von $0\,V$ referenziert.

Tabelle 6.5: Parameter der Abwärtswandlerschaltung. Die gewählten Werte orientieren sich an [112].

Parameter	Wert
Versorgungsspannung U_S	$24\,V$
Leitwiderstand des Schalters & der Diode R_{ON}	$10e^{-6}\,\Omega$
Sperrwiderstand —„— R_{OFF}	$10e^{6}\,\Omega$
Frequenz des PWM Signals f_{PWM}	$10\,kHz$
PWM Duty-Cycle D	$50\,\%$
Schwellenspannung des Schalters U_{thres}^{S1}	$0,5\,V$
High-Pegel des PWM Signals U_{PWM}	$1,0\,V$

[11] Die Modelle sind im Vergleich zu herkömmlichen SPICE-Schaltermodellen im Bereich einer Größenordnung schneller.

6.4.1.2 Aufbau der Simulationsbenchmark

Zur Genauigkeits- und Performanzanalyse wird die synthetisierte Simulation des Abwärtswandlers mit industriell etablierten und akademischen Werkzeugen verglichen. Diese umfassen folgende:

- *MATLAB/Simscape Power Systems R2015b* [103]: ein in der Industrie etabliertes Werkzeug zur Modellierung und Simulation von Regelungssystemen und elektrischen Versorgungssystemen unter Verwendung von ausgereiften, klassischen Solvern.
- *PowerDEVS* [13]: ein akademisches Werkzeug, das sich auf die Simulation von hybriden Systemen, basierend auf dem DEVS [197] Formalismus, konzentriert und die komplette Familie der QSS Solver implementiert.
- *LTSpice* [91]: ein von Linear Technology entwickelter Schaltungssimulator auf SPICE Basis, welcher auf die Simulation von Leistungsreglern wie DC/DC-Wandler optimiert ist.

Die äquivalenten MATLAB und PowerDEVS Modelle wurden auf Basis ihrer Demomodelle erstellt und auf die gewählten Parameter in Tabelle 6.5 angepasst. Für die Evaluation mit LTSpice wurde die während der Synthese generierte SPICE Netzliste wiederverwendet und die äquivalenten QSS Spannungs-/Stromquellen durch die Kapazität und Induktivität aus dem Stromlaufplan ersetzt.

Zur Analyse der Genauigkeit werden zwei Referenzsimulationen mit gleicher Simulationszeit ausgeführt, je eine mit MATLAB und PowerDEVS, bei welchen die Ausgangsspannung am Kondensator und der Spulenstrom aufgezeichnet werden. MATLAB verwendet dabei einen klassischen *ode23tb* Solver und PowerDEVS den LIQSS3 Solver, wobei beide Solver mit einer relativen Fehlertoleranz von $1e^{-9}$ konfiguriert werden. Um den Fehler zwischen beiden Simulationen bestimmen zu können, werden die von MATLAB erzeugten Samples an den Zeitpunkten der LIQSS3 Lösung linear interpoliert, da letztere ca. 55000 Samples produziert und MATLAB lediglich ca. 11800 Samples. Der daraus resultierende relative Fehler liegt dabei in der Größenordnung 10^{-5} für den Spulenstrom und im Bereich 10^{-6} für die Ausgangsspannung. Im Folgenden werden die Fehler der drei Realisierungen bei größerer Fehlertoleranz auf analoge Weise bestimmt.

Aufgrund der geringen Abweichung zwischen beiden Referenzlösungen werden die Fehler lediglich auf die LIQSS3 Lösung von PowerDEVS bezogen.

Des Weiteren gilt für alle drei Realisierungen, sofern nicht anders vermerkt, folgende Konfiguration:

- Das synthetisierte PtII-Modell und das PowerDEVS-Modell verwenden für alle Integratoren den QSS3 Solver.
- Die relative Fehlertoleranz sowie die absoluten und relativen Quanta ΔQ_i der QSS Integratoren werden auf $1e^{-3}$ eingestellt.
- Die Simulationsdauer beträgt $0,01\,s$ und die dafür benötigte Ausführungszeit wird über zehn Iterationen arithmetisch gemittelt, wobei während den Simulationen sämtliche Plots und Log-Ausgaben unterdrückt werden.
- Der mittlere absolute Fehler (MAE) und der relative Fehler bezogen auf die Referenzlösung (NMAE) werden über die Formeln (6.4) und (6.5) bestimmt. Darin sind y_i die Referenzwerte und $\hat{y}_i$ die Vergleichswerte.

6.4.1.3 Simulationsergebnisse und Diskussion

Abb. 6.22 zeigt die Signalverläufe der Ausgangsspannung und des Spulenstroms der Simulation des synthetisierten PtII-Modells.

In Abb. 6.23 (a) und Abb. 6.23 (b) sind die relativen Fehler der Ausgangsspannung und des Spulenstroms bezogen auf die Referenzlösung aufgetragen. Wie aus den beiden Abbildungen zu erkennen ist der relative Fehler beider Signalverläufe stets kleiner als die mit $1e^{-3}$ parametrierte Toleranz. Die relativen Fehler bewegen sich in der Größenordnung 10^{-4}.

In Tabelle 6.6 sind die Vergleichsergebnisse der eigenen Lösung und den gewählten Werkzeugen bzgl. des MAE- und NMAE-Fehlers sowie der jeweiligen Ausführungszeit zusammengefasst. Die PowerDEVS Lösung mit QSS3 Solver bietet dabei den geringsten Fehler, jedoch wirkt sich diese weit über der parametrierten Genauigkeit auf die Laufzeit aus.

Die eigens vorgestellte Lösung, in Tabelle 6.6 als *PtII QSS3* bezeichnet, bietet dahingegen einen etwas größeren Fehler, liegt jedoch signifikant unter dem parametrierten relativen Fehler von $1e^{-3}$ und liefert eine bessere Laufzeit.

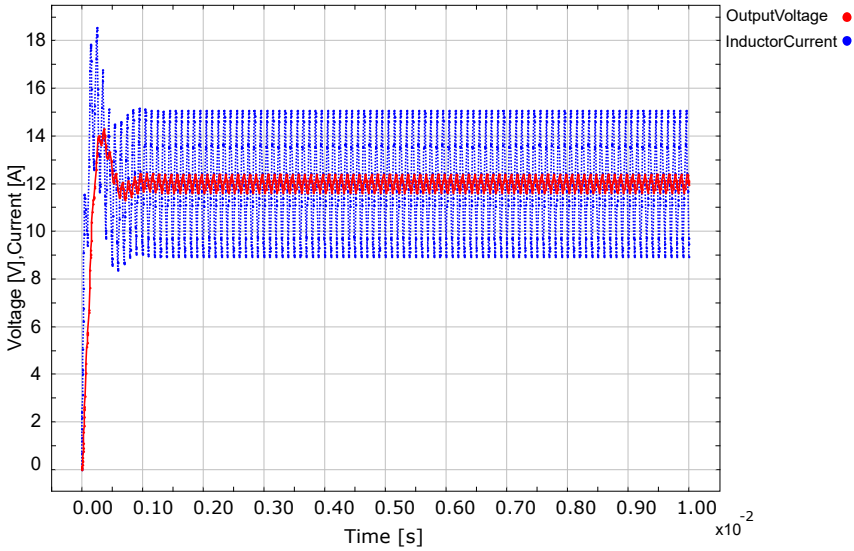
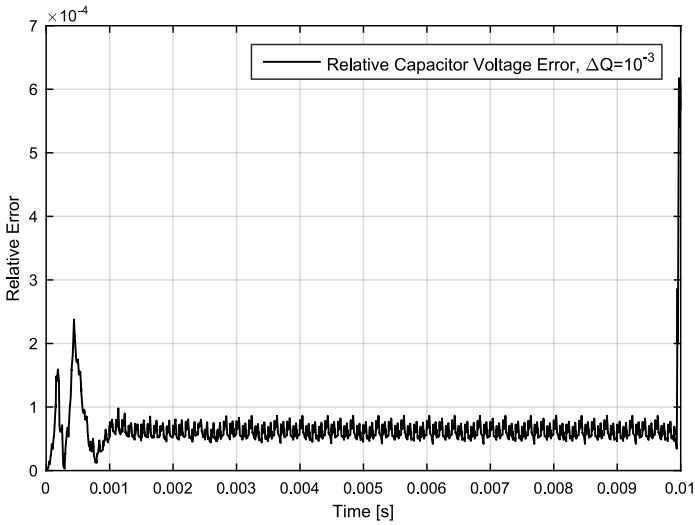


Abbildung 6.22: Ausgangsspannung und Spulenstrom der Simulation des synthetisierten PtII-Modells.

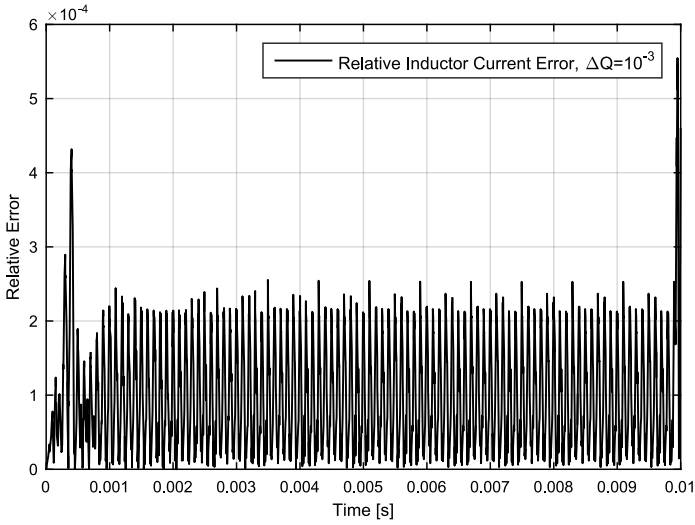
Tabelle 6.6: Vergleich der Performanz und Genauigkeit des Abwärtswandler des synthetisierten PtII Modells mit den gewählten Werkzeugen (Quelle: basierend auf [BB18]).

Werkzeug und Solver	**MAE**		**NMAE**		**Ausführungszeit**
	U(out) [V]	**I(L) [A]**	**U(out)**	**I(L)**	
PtII QSS3	$7,675e^{-4}$	$1,326e^{-3}$	$6,491e^{-5}$	$1,119e^{-4}$	182 ms[a]
—„—	„	„	„	„	328 ms[b]
PowerDEVS QSS3	$4,45e^{-4}$	$6,64e^{-4}$	$3,77e^{-5}$	$5,61e^{-5}$	416 ms
MATLAB ode23tb	$3,69e^{-3}$	$4,07e^{-3}$	$3,15e^{-4}$	$3,42e^{-4}$	422 ms
LTSpice mod. trap.	$8,46e^{-3}$	$2,95e^{-3}$	$7,13e^{-4}$	$2,47e^{-4}$	205 ms

[a] Simulation des Abwärtswandlers außerhalb des SPICE Aspekts.
[b] —„— innerhalb des SPICE Aspekts.



(a) Relativer Fehler der Ausgangsspannung.



(b) Relativer Fehler des Spulenstroms.

Abbildung 6.23: Relative Fehler der Ausgangsspannung und des Spulenstroms des Abwärtswandlers bezogen auf die Referenzlösung in PowerDEVS mit LIQSS3 Solver und einer relativen Fehlertoleranz von $1e^{-9}$ (Quelle: basierend auf [BB18]).

Verglichen mit den Simulationen in MATLAB bzw. LTSpice weist sie bei der Ausgangsspannung eine höhere Genauigkeit von ca. Faktor 4,9 bzw. etwa eine Größenordnung auf, für den Spulenstrom ca. den Faktor drei bzw. Faktor 2,2.

Beim Vergleich der Ausführungszeit werden zwei Arten der Ausführungszeit, bezogen auf die eigene Lösung, gemessen. Einerseits das synthetisierte Modell, bei welchem der Abwärtswandler innerhalb des generierten SPICE Aspekts simuliert wird. Andererseits wird aus Fairnessgründen die SPICE-Simulation aus dem Aspekt extrahiert, um den Overhead der Umleitung zum Aspekt zu vermeiden, da dieser bei den anderen Lösungen nicht auftritt. Interessanterweise sind die MATLAB und PowerDEVS Simulationen in etwa gleich schnell, obwohl PowerDEVS den für nichtkontinuierliche Signale effizienteren QSS3 Solver verwendet. Dies kann dadurch erklärt werden, dass QSS3 ein expliziter Solver ist, der Abwärtswandler aber steife Systemeigenschaften aufweist, die mit den LIQSS Verfahren deutlich effizienter zu lösen sind [112].

Trotz des verwendeten QSS3 Solvers in der eigenen Lösung ist die Simulation ohne Aspekt um etwa den Faktor 2,3 schneller als die MATLAB bzw. PowerDEVS Lösung. Das synthetisierte Modell mit SPICE Aspekt ist noch immer um ca. den Faktor 1,29 bzw. 1,27 schneller. Die Simulation ohne Aspekt liefert darüber hinaus eine um $23\,ms$ (ca. Faktor 1,13) schnellere Ausführung als die native SPICE Simulation in LTSpice, dessen Algorithmen auf Leistungsregler optimiert sind, bei gleichzeitig höherer Genauigkeit.

Die diskutierten Ergebnisse unterstreichen die Effizienz des vorgestellten Ansatzes aus der Kombination der QSS Verfahren und der integrierten, asynchronen SPICE Co-Simulation, welche zusätzlich die statischen Funktionen der QSS Integratoren implizit berechnet (vgl. Abb. 2.9 und Unterabschnitt 5.3.8.2). Somit wird die Anzahl der zu verarbeitenden Events zwischen Actors und Director reduziert. Der aspektorientierte Ansatz verringert zwar etwas die Performanz, erhöht allerdings die Modularität und sorgt für eine klare Trennung der verschiedenen Domänen, d. h. des Verhaltens der logischen und der verfeinerten elektrischen Architektur.

6.4.2 PWM-geregelter Gleichstrommotor

6.4.2.1 Erweiterte Verhaltensmodellierung

Im Folgenden wird die in Abb. 6.21 verwendete Funktion `dcMotor` mit Verhalten hinterlegt und implementiert ein Modell eines permanentmagneterregten, idealen Gleichstrommotors (DC-Motors), welches als Actor aus der PtII-Bibliothek wiederverwendet wird[12]. Zusätzlich wird die Funktion um einen Ausgangsport erweitert, welcher die momentane Drehzahl des Motors zur Verfügung stellt, bspw. über einen integrierten Sensor. Letztere wird in einer Feedback-Schleife in der Funktion `PWMController` verwendet, um die Drehzahl anhand eines proportionalen Reglers auf einen parametrierbaren Sollwert zu regeln. Der Regler passt den Duty Cycle des PWM-Ausgangssignals und damit die über den Abwärtswandler am Motor anliegende Spannung entsprechend an. Die modifizierte logischen Architektur und die des Verhaltens sind in Abb. 6.24 zu sehen. Die Mappings zu den Hardwarekomponenten und -Pins aus dem letzten Abschnitt bleiben bestehen.

Die Implementierung des prop. Reglers im top-level Block `PWMController_Impl` wurde dem Demomodell des DC-Motors von PtII entnommen[13] und auf die DE Domäne angepasst. Bei der Realisierung des top-Level Blocks `DCMotor_Impl` zur Abbildung des Motorverhaltens sind zusätzlich die folgenden Eigenschaften zu beachten:

1. Das Verhalten ist in der kontinuierlichen Domäne modelliert.
2. Die über das Motormodell simulierte Stromaufnahme repräsentiert einen dynamischen Stromverbraucher nach Unterabschnitt 5.2.5.2 und dient als zusätzliche Last in der elektrischen Simulation des Abwärtswandlers.
3. Die Samplingfrequenz der Drehzahl für den Übergang von der kontinuierlichen Domäne zum prop. Regler innerhalb des diskreten PWM-Controllers beträgt $20\,kHz$.

[12] Siehe https://github.com/icyphy/ptII/blob/master/ptolemy/domains/continuous/lib/DCMotor.xml, zuletzt aufgerufen am 03.02.2020.
[13] Vergleiche https://github.com/icyphy/ptII/tree/master/ptolemy/domains/continuous/demo/DCMotor, zuletzt aufgerufen am 03.02.2020.

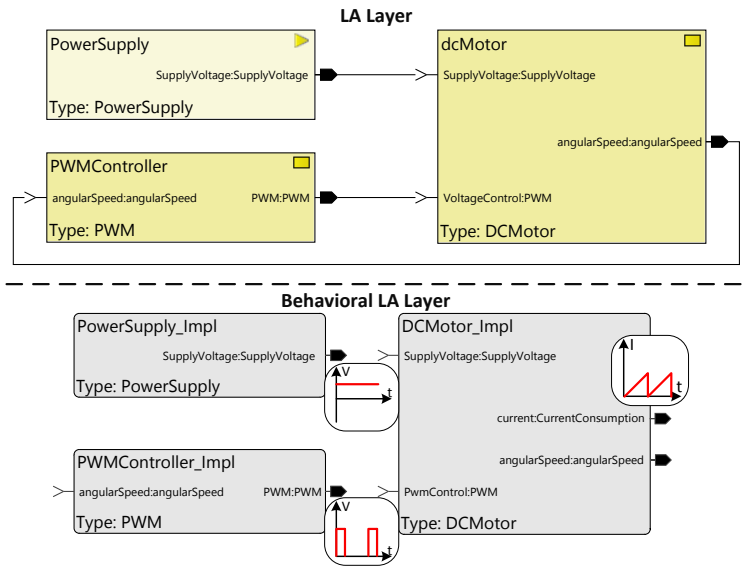


Abbildung 6.24: Modifizierte logische Architektur und Verhaltensebene zur erweiterten Simulation des Abwärtswandlers mit einem DC-Motor als Last. Die Spannung bzw. Drehzahl des Motors wird in einer prop. Regelschleife über das PWM-Signal angepasst.

Die Kenndaten des DC-Motors und weitere Parameter sind in Tabelle 6.7 zusammengefasst.

6.4.2.2 Simulationsergebnisse und Diskussion

In Abb. 6.25 und Abb. 6.26 sind die Simulationsergebnisse der kombinierten elektrischen, kontinuierlichen und diskreten Simulation des geregelten DC-Motors zu sehen. Abb. 6.25 zeigt die durch den Abwärtswandler am Motor angelegte Spannung sowie dessen Stromaufnahme, die gleichzeitig als Last für den Abwärtswandler in der elektrischen SPICE-Simulation dient. Der anfängliche Spannungs- und Stromanstieg bildet das Anlaufverhalten des idealen Motormodells bei voll angelegter Spannungsversorgung von 12 V (Duty Cycle ist gleich 1,0) ab und enthält aufgrund des Abwärtswandlers eine kurzzeitige Spannungs-

Tabelle 6.7: Kenndaten des DC-Motors und weitere Parameter. Die Motorkenndaten orientieren sich am Datenblatt in [105].

Parameter/Kenndaten	Wert
Trägheitsmoment des Rotors J_R	$0,867\,gcm^2$
Generatorkonstante k_e (Kehrwert von k_n)	$1/946\,V/min^{-1}$
Drehmomentkonstante k_M	$10,1\,mNm/A$
Wicklungswiderstand R	$8,29\,\Omega$
Wicklungsinduktivität L	$0,205\,mH$
Nenndrehmoment (max. kont. Last) M_N	$3,44\,mNm$
Nennstrom (max. kont. Strom) I_N	$349\,mA$
Anlaufstrom bei Nennspannung ($12\,V$) I_A	$1,45\,A$
Solldrehzahl für den prop. Regler n_p	$250\,min^{-1}$
Trägheitsmoment der Last J_L	$3,013\,gcm^2$
Versorgungsspannung U_S	$12\,V$
Sampling-Frequenz der Motordrehzahl f_s^n	$20\,kHz$

und Stromspitze, die über die Versorgungsspannung und den Anlaufstrom von $I_A = U_s/R = 12\,V/8,29\,\Omega = 1,45\,A$ (vgl. Tabelle 6.7) hinausgehen.

Der prop. Regler sorgt anschließend für eine sukzessive Verringerung des Duty Cycles des PWM-Signals und damit für eine Reduzierung der resultierenden Motorspannung durch den Abwärtswandler, bis sich ein stationärer Zustand mit etwa $n_N \approx 187\,min^{-1}$ und einem Duty Cycle $D \approx 25,2\,\%$ einstellt (vgl. Abb. 6.26). Die Abweichung von der Solldrehzahl n_p ist dabei typisch für einen prop. Regler.

Durch das angelegte Nenndrehmoment M_N ergibt sich für den idealen permanenterregten DC-Motor bei konstanter Spannung ein Nennstrom von [15, S. 39][49, S. 67 ff.] $I_N = M_N/k_M = 340,6\,mA$[14]. Für die Drehzahl gilt [15, S. 39]:

$$n = k_n \cdot U_{mot} - \left(\frac{30}{\pi} \cdot \frac{R}{k_M^2}\right) \cdot M \tag{6.7}$$

Umgestellt ergibt sich im stationären Zustand mit den Werten von M_N, n_N und den Motorkenndaten R, k_M und k_n eine Motorspannung von $U_{mot} \approx 3,02\,V$. Aufgrund des Abwärtswandlers entsteht jedoch ein Ripple um die Motorspannung und damit um die Drehzahl sowie den Nennstrom.

[14] Der in Tabelle 6.7 angegebene Nennstrom ist dahingegen jener gemessener Wert, welcher auch die Reibungsverluste mitberücksichtigt und daher zu einem höheren Strom führt.

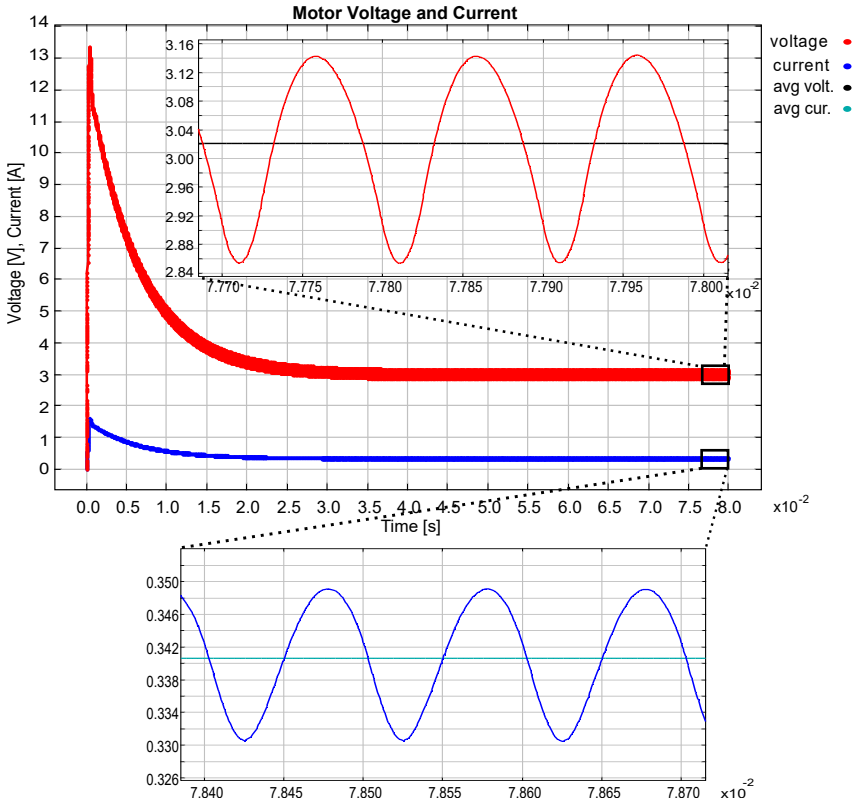


Abbildung 6.25: Motorspannung und -strom des geregelten Motors.

In Abb. 6.25 sind die Spannung und der Strom zusammen mit ihrem fortlaufenden Durchschnittswert vergrößert aufgetragen. Im stationären Zustand resultiert demnach im Mittel der berechnete Nennstrom von $I_N \approx 340,6\,mA$ sowie die Motorspannung von $U_{mot} \approx 3,02\,V$. Die Mittelung wird während der Simulation durch dedizierte Actors vom Typ `MovingAverage` fortlaufend bei je einer 30-fachen PWM-Frequenz von $300\,kHz$ über die letzten 3000 Werte vorgenommen, d. h. eine Mittelung über 100 Perioden mit je 30 Werten.

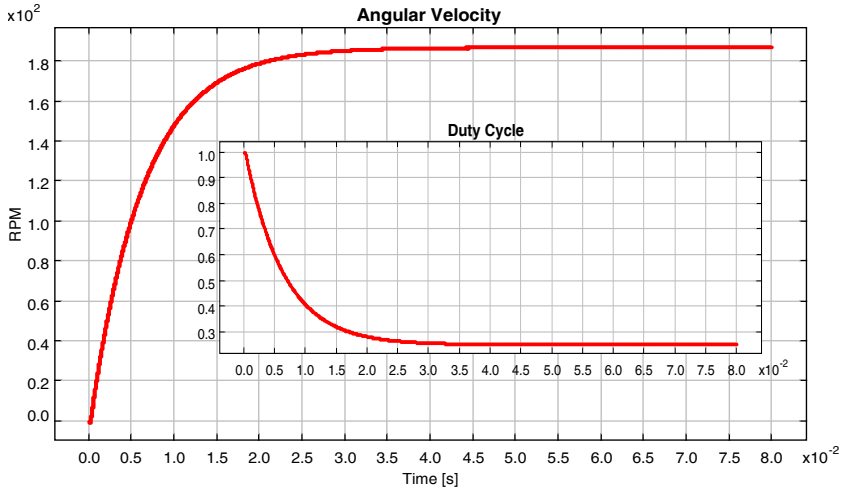


Abbildung 6.26: Motordrehzahl und Duty Cycle des PWM-Signals zur Ansteuerung des Abwärtswandlers.

6.4.3 Leitungssatzsensitive Simulation von Stromverbrauchern

6.4.3.1 Ebenenübergreifende Modellierung

In diesem Abschnitt wird eine Teilarchitektur des Bordnetzes untersucht, um die leitungssatzsensitive Synthese und Simulation der Spannungsversorgung unter Einfluss von statischen und dynamischen Stromverbrauchern zu demonstrieren. Das betrachtete E/E-Architekturmodell in PREEvision® ist aus Darstellungsgründen in Abb. 6.27 schematisch gezeigt. Das zugrunde liegende Topologiemodell des Fahrzeugs, in welchem die Steuergeräte, Sensoren und Aktuatoren etc. platziert werden, ist dem Demomodell *Elypsis* in der Version 7.0 von PREEvision® entnommen. Die Topologie wird dazu verwendet, um über den Leitungssatz-Router die Leitungslängen für die Berechnung der ohmschen Leitungswiderstände nach Gl. (5.1) zu bestimmen.

Auf elektrischer Ebene der Hardwarearchitektur repräsentiert das Modell einen EPS-Aktuator, welcher über einen konventionellen 12 V Generator mit einem modellierten Innenwiderstand von 1 $m\Omega$ versorgt wird. Beide Komponenten

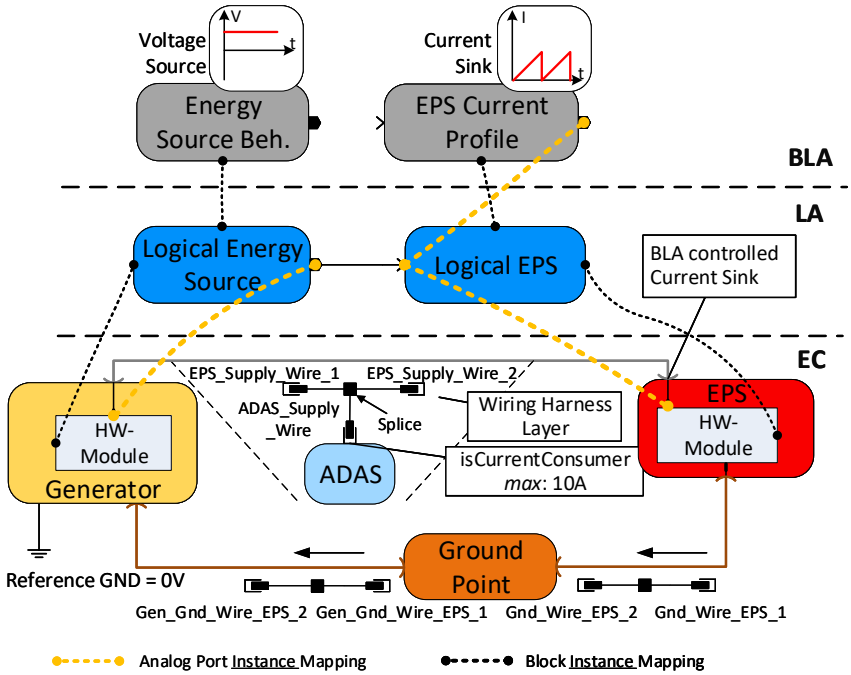


Abbildung 6.27: Schematische Darstellung der ebenenübergreifenden Architektur eines *Electric Power Steering (EPS)*-Aktuatornetzwerks zur leitungssatzsensitiven Simulation von Stromverbrauchern (Quelle: [BB18]).

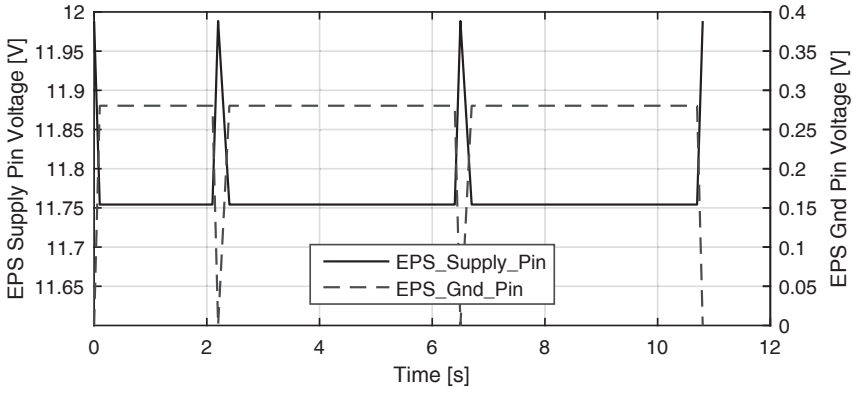
sind mit demselben Massepunkt verbunden. Zusätzlich sind die physikalischen Leitungen als Verfeinerung der schematischen, elektrischen Verbindungen zu sehen sowie die daran angeschlossenen Pins und Splices (vgl. *Wiring Harness Layer* Ausschnitt in Abb. 6.27). Dabei ist der Leitungstyp *AWG4* (*American Wire Gauge (AWG)*) allen Leitungen zugewiesen. Zusätzlich ist die ADAS ECU als statischer Stromverbraucher mit $10\,A$ modelliert, welche aber keinen Teil der logischen Funktionskette repräsentiert. Diese wird während der Synthese aber dennoch mit eingebunden, da sie an derselben elektrischen Verbindung zwischen Generator und Aktuator angeschlossen ist.

Die logische Architektur enthält die logischen Repräsentationen der beiden Hardwarekomponenten und sind aufeinander abgebildet. Über die eingeführten analogen Port-Mappings wird sowohl die leitungssatzsensitive Simulationssynthese des Verhaltens modelliert als auch der dynamische Stromverbrauch der logischen EPS-Funktion. Basierend auf dem Leitungssatz-Routing der physikalischen Leitungen durch die zugrunde liegende Topologie werden während der Simulationssynthese die ohmschen Leitungswiderstände des zugewiesenen Leitungstyps AWG4 bestimmt und in die SPICE-Netzliste integriert. Diese bewegen sich im Bereich von einigen hundert $\mu\Omega$. Pin-Übergangswiderstände sind nicht modelliert und werden somit vernachlässigt.

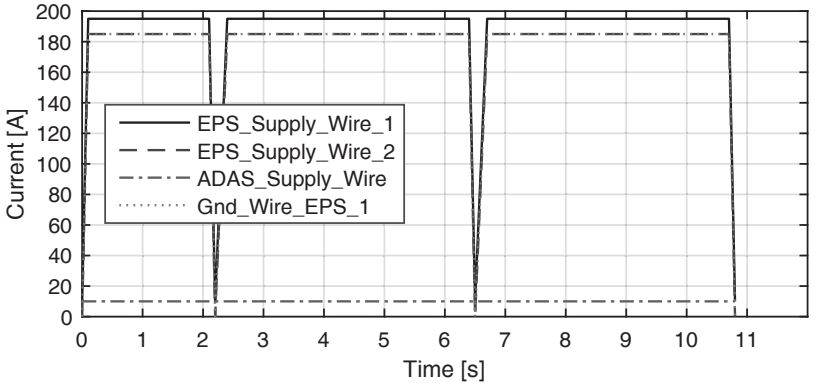
6.4.3.2 Simulationsergebnisse und Diskussion

Die aufbereiteten Simulationsergebnisse der leitungssatzsensitiven Simulation der Stromverbraucher sind in Abb. 6.28 abgedruckt. Abb. 6.28 (a) zeigt die Spannungsverläufe am Versorgungs- und Masse-Pin des Aktuators, abhängig von den Spannungsabfällen über den Innenwiderstand des Generators und die Leitungswiderstände. Ein Screenshot der integrierten Simulation und Visualisierung in der neuen Ptolemy Run Control Ansicht innerhalb von PREEvision® ist im Anhang in Abb. A.6 zu finden.

Die Spannungsabfälle werden zum großen Teil durch die Stromaufnahme des EPS-Aktuators verursacht, welche über das analoge Verhalten der logischen EPS-Funktion nachgebildet wird. Die Werte des Stromverbrauchs stammen aus real aufgezeichneten Testfahrten und wurden freundlicherweise in einer Spezifikation eines deutschen OEM zur Verfügung gestellt. Der Strom durch die Leitung `EPS_Supply_Wire_2` in Abb. 6.28 (b) von max. 185 A repräsentiert dieses Stromprofil und entspricht einem Einparkszenario aus dieser Spezifikation. Das zusätzliche ADAS Steuergerät nimmt den modellierten statischen Strombedarf von 10 A auf, der in Abb. 6.28 (b) durch die Leitung `ADAS_Supply_Wire` fließt und auf die Referenzmasse von 0 V bezogen ist, da keine Verbindung zum Massepunkt besteht. `Gnd_Wire_EPS_1` repräsentiert stellvertretend für die anderen Masseleitungen den Strom durch das Massenetz, da die Leitungswiderstände der Masseleitungen in Reihe geschaltet sind. Der vom Generator zu liefernde Strom, welcher durch die Leitung `EPS_Supply_Wire_1` fließt, beträgt in diesem



(a) Spannungen am Versorgungs- und Masse-Pin des EPS Aktuators.



(b) Ströme durch die Versorgungs- und Masseleitungen.

Abbildung 6.28: Leitungssatzsensitive Simulation des EPS-Aktuatornetzwerks (Quelle: [BB18]).

Fall 195 A und ergibt sich aus der Summe der Stromaufnahme des Aktuators und des ADAS Steuergeräts.

Der Anwendungsfall zeigt, dass mithilfe den in Unterabschnitt 5.2.5.2 und 5.2.5.3 eingeführten ebenenübergreifenden Modellierungserweiterungen bereits in frühen Phasen beginnend auf der logischen Architektur Untersuchungen bzgl. des Bordnetzes angestellt werden können. Dynamische Stromverbraucher können durch die eingeführte Verhaltensebene beliebige Stromprofile hinterlegen und der entsprechenden Hardwarekomponente zugewiesen werden. Über die ebenenübergreifenden Modellierungsmechanismen und die Simulationssynthese werden sowohl die durch die Topologie gerouteten Leitungen der Versorgungs- als auch Masseleitungen zwischen den abgebildeten Hardwarekomponenten in Form von ohmschen Widerständen automatisiert zur verfeinerten elektrischen Simulation der logischen Ebene berücksichtigt. Darüber hinaus werden auch Hardwarekomponenten als statische Verbraucher automatisiert berücksichtigt, die an das Versorgungsnetz auf elektrischer Ebene angeschlossen sind, jedoch kein Teil der logischen Architektur abbilden. Dies trägt zur deutlichen Reduzierung des Modellierungsaufwands auf logischer Ebene bei.

6.5 Metrikbasierte Analyse von Architekturvarianten

Nicht zuletzt wird in diesem Abschnitt das AMiL Konzept und der entwickelte Prototyp aus Unterabschnitt 5.4.3 demonstriert und evaluiert. Der Fokus liegt dabei auf der integrierten Analyse von Kommunikationslatenzen funktionaler Wirkketten, basierend auf dem integrierten Feedback der Simulationsdaten, sowie der daraus resultierenden Buslast, angewandt auf verschiedene Architekturvarianten mithilfe der variantensensitiven Simulationssynthese. Konkret werden in der nachfolgenden Fallstudie die CAN-Bus Kommunikationslatenzen zwischen Funktionen untersucht, sowohl unter Variation des Mappings der Funktionen auf ihre ECUs als auch unter Variation der Steuergeräte-Netzwerktopologie selbst. Durch die aspektorientierte Simulation der Netzwerkkommunikation und der eingeführten Latenzanalyse-Dekorierer lässt sich die Metrik jedoch auch auf andere Bustypen übertragen.

6.5.1 Aufbau der Wirkketten und Architekturvarianten

6.5.1.1 Wirkketten und Kommunikationsbeziehungen

Zur Evaluation wurden zum einen vier funktionale Wirkketten auf der logischen Architektur modelliert, die folgende Applikationen widerspiegeln:

- Spurhalteassistent (LDW, siehe Abb. 5.36).
- Adaptiver Abstandsregelautomat (ACC, vgl. Wirkkette in Abb. 6.1).
- *Predictive Powertrain Control (PPC)*: schätzt zusätzlich zum ACC die Geschwindigkeit basierend auf GPS-Daten.
- *Environment Perception (EP)*: erstellt ein virtuelles Umgebungsbild basierend auf Sensordaten von Radar, GPS, 4G und *Vehicle-to-X (V2X)* Kommunikation, um die Trajektorie des Fahrzeugs zu schätzen und den Lenkwinkel zu regeln.

Die einzelnen logischen Funktionen der Wirkketten beinhalten dabei kein detailliertes, funktionales Verhalten. Stattdessen wird ein abstraktes, ereignisbasiertes Verhalten verfolgt, welches basierend auf zyklischen Timing Constraints der Port-Prototypen automatisiert generiert wird (vgl. Unterabschnitt 5.3.4.2 und Abb. 5.22). Die LA Modelle der PPC und EP Wirkketten sind [Neu18] zu entnehmen.

Die Sender/Receiver-Schnittstelle jedes logischen Senderports spezifiziert genau ein Datenelement, das mit einem Signal oder einer Signalgruppe verknüpft ist. Das Signal bzw. die Signalgruppe der Schnittstelle ist dabei in einer PDU gekapselt, welche einem CAN-Frame zugeordnet ist. Abhängig von der aktiven Architekturvariante und der Abbildung einer Funktion auf ein Steuergerät, erstellt der Signal-Router für ein Signal bzw. eine Signalgruppe eine Signal-, PDU- und Frametransmission. Im Falle einer konventionellen Verbindung lediglich eine Signaltransmission.

Jeder CAN-Frame besteht dabei aus einem Daten-Payload von 64 *Bit*, zusätzlichen Header-/Trailer-Bits und besitzt eine CAN ID. Die Kommunikationsbeziehungen zwischen den einzelnen Funktionen, deren CAN-Frames sowie die aus den Timing Constraints abgeleitete Zykluszeit sind in Tabelle 6.8 zusammengefasst und gelten in dieser Fallstudie zu Vergleichszwecken für beide Architekturvari-

anten gleichermaßen. Jede Funktion, bis auf die `4G_V2X` Sensorfunktion, weist ein zyklisch generiertes Verhalten auf.

Das Verhalten der `4G_V2X` Sensorfunktion ist dahingegen auf der BLA explizit als sporadisches Verhalten modelliert. Das sporadische Verhalten zielt auf die Abstraktion der sich dynamisch ändernden Umgebung ab, mit der das Fahrzeug kommuniziert. Die Funktion sendet einen zyklischen Wert im Abstand von $100\,ms$[15] nur mit einer bestimmten Wahrscheinlichkeit, die im vorliegenden Falle zu $0,2$ gewählt wurde, um die Latenzen und Buslasten der restlichen zyklischen Sender nicht zu überdecken. Details zur Modellierung dieses Verhaltens sind in [Neu18] zu finden.

6.5.1.2 Architekturvarianten

Komplementär zu den Wirkketten wurden, inspiriert durch [73], zwei unterschiedliche Steuergeräte-Vernetzungstopologien modelliert, deren Kommunikationslatenzen und Buslasten basierend auf den Timings der Wirkketten untersucht werden sollen. Diese beinhalten eine klassische, verteilte Architektur sowie eine zentralisierte Architektur mit Domänen-Steuergeräten (engl. *Domain Control Units (DCUs)*) [158], welche ausschließlich über Bussysteme kommunizieren. Die beiden Netzwerkarchitekturen sind Abb. 6.29 (a) und Abb. 6.29 (b) zu entnehmen.

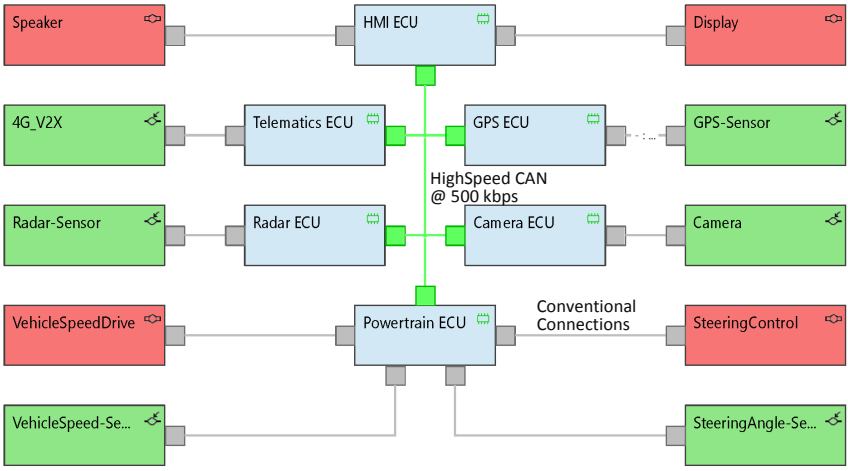
Die verteilte Architektur besteht aus mehreren ECUs, welche über einen gemeinsamen CAN-Bus mit einer Bitrate von $500\,kbps$ miteinander kommunizieren, während die Sensoren und Aktuatoren über konventionelle Verbindungen angebunden sind. Die zentralisierte Architektur besteht dahingegen aus zwei hochperformanten ECUs, welche über einen High-Speed CAN FD-Bus mit einer Bitrate von $10\,Mbps$ kommunizieren. Die Sensoren und Aktuatoren sind je über dedizierte Low-Speed CAN-Busse mit einer Bitrate von $125\,kbps$ mit den ECUs verbunden. Die Sensor- und Aktuatorfunktionen besitzen ein fixes Mapping auf ihre Hardwarekomponenten, während das Mapping der verarbeitenden Funktionen der Wirkketten auf ihre ECUs variiert.

[15] Eine Frequenz von $10\,Hz$ ist nach [143] ein typischer Wert für die Versendung von sog. *Beacons* über drahtlose V2X Kommunikation, welche grundlegende Daten des Fahrzeugzustands mitteilen. Dazu zählen Geschwindigkeit, Position oder Fahrtrichtung.

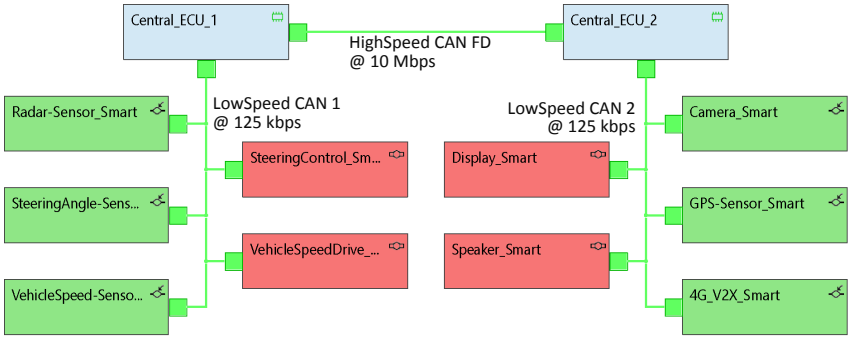
Tabelle 6.8: Zusammenfassung der Kommunikationsbeziehungen der Wirkketten (Quelle: [BNB19]).

CAN-Frame[a]	CAN ID	Zykluszeit [*ms*]	Sender Funktion	Empfänger Funktionen	Involvierte Wirkketten
ACC	0*x*120	50	ACC	SetWheelSpeed	ACC
Audio	0*x*121	100	Interpretation	Speaker	LDW
Buffer	0*x*123	100	Buffer	Comparator	PPC
Course	0*x*124	100	Course	NewCourse-Calc	EP
VehicleSpeed	0*x*125	25	GetCurrent-VehicleSpeed	ACC, LDW_Active, Buffer, ObstacleAna-lyzer	ACC, LDW, PPC, EP
Display	0*x*126	50	Interpretation	Display	LDW
Feature	0*x*127	25	Feature-Extraction	Interpretation	LDW
GPS	0*x*128	100	GPS	CalcTarget-Speed, Positioning	PPC, EP
4G_V2X	0*x*129	sporadisch	4G_V2X	Positioning	EP
Camera	0*x*130	20	Camera	Feature-Extraction	LDW
LDWAct	0*x*131	50	LDW_Active	Interpretation	LDW
Radar	0*x*132	25	LongDistance-Radar	ACC, Obstacle-Analyzer	ACC, EP
Obstacle	0*x*133	50	Obstacle-Analyzer	NewCourse-Calc	EP
Position	0*x*134	100	Positioning	NewCourse-Calc	EP
SteeringAngle	0*x*135	100	GetSteering-Angle	Course	EP
TargetSpeed-Interm	0*x*136	100	CalcTarget-Speed	Comparator	PPC
TargetSpeed-Out	0*x*137	50	NewCourse-Calc	SetWheelSpeed	EP
TargetSteering-Angle	0*x*139	50	NewCourse-Calc	SteeringAngle-Ctrl	EP
ThrottlePos	0*x*140	100	Comparator	ThrottlePos	PPC

[a] CAN FD-Frames im Basisformat besitzen bei Nutzdaten von ≤ 16 *Bytes* eine Header/Trailer-Länge von insg. 62 *Bits*, sonst 67 *Bits* (inkl. fixe Stuff-Bits im CRC-Feld) [57, 119].



(a) Verteilte Architektur.



(b) Zentralisierte Architektur.

Abbildung 6.29: Betrachtete Architekturalternativen zur Untersuchung der Kommunikationslatenzen von Wirkketten und damit zusammenhängenden Buslasten (Quelle: [BNB19]).

Um ein 150 % Modell zu erhalten, sind die Wirkketten initial auf beide Architekturvarianten abgebildet. Mithilfe des integrierten Variantenmanagements innerhalb von PREEvision® [175, Kap. 12.2.3.3] wurden zwei Architekturvarianten namens *DistributedHighLine* und *CentralizedHighLine* erstellt. Dazu sind in einem orthogonalen Variantenmodell u. a. mehrere sog. *Sets*, welche alle zu einer Architekturvariante dazugehörigen Artefakte aus dem 150 % Architekturmodell enthalten, aufgebaut worden. Jedes Set beinhaltet typischerweise einen bestimmten Aspekt der E/E-Architektur wie die logischen Wirkketten, die Hardwarearchitektur oder Kommunikationsartefakte wie Signale, Frames und Transmissionen. Ein Set stellt somit die Verbindung zum Variantenmodell her.

In Abb. 6.30 ist die Struktur des Variantenmodells gezeigt sowie die Verbindung zum tatsächlich aufgebauten Variantenmodell am Beispiel der *CentralizedHighLine* Architekturvariante. Ein *Konzeptraum* (engl. *Concept Space*) ist die Basis für die Erstellung von Varianten. Sets werden in bestimmten *Alternativen* referenziert, die wiederum Teil von sog. *Konzept- und Ausstattungsvorlagen* (engl. *Concept Template* und *Equipment Template*) sind. Ein Set kann dabei von mehreren Alternativen bzw. Vorlagen wiederverwendet werden, bspw. ein Set mit Funktionen, die in allen Varianten vorkommen. Eine Konzeptvorlage stellt dabei ein bestimmtes, technisches Konzept wie etwa die gezeigten Hardwarearchitekturen dar, während eine Ausstattungsvorlage verschiedene Fahrzeugkonfigurationen beinhaltet und unabhängig vom Konzeptraum vorliegt. Eine spezifische Architekturvariante ergibt sich sodann aus der Komposition von genau einer Ausstattungsvorlage mit genau einer Alternative sowie ein oder mehrerer Konzeptvorlagen mit je genau einer Alternative. Jede Komposition repräsentiert somit ein 100 % Modell. [175, Kap. 12.2.3.3]

Abb. 6.31 zeigt die Komposition der Alternativen der Konzeptvorlage *NetworkTopologies* und der Ausstattungsvorlage *TopologyEquipment* durch mehrere Sets. Um die Befüllung der Sets mit den dazugehörigen E/E-Artefakten zu erleichtern, sind sog. Propagationsregeln [175, Kap. 12.2.6] erstellt und angewandt worden. Diese erlauben es, ausgehend von bestimmten E/E-Artefakten, die assoziierten Artefakte automatisiert einem bestimmten Set hinzuzufügen. Im Anhang in Abb. A.7 ist eine beispielhafte Modellabfrageregel gezeigt, welche – ausgehend von einer abbildbaren logischen Funktion der aktiven Variante – das Mapping und die Process Unit einem Set hinzufügt, sofern die ECU ebenfalls Teil der aktiven Va-

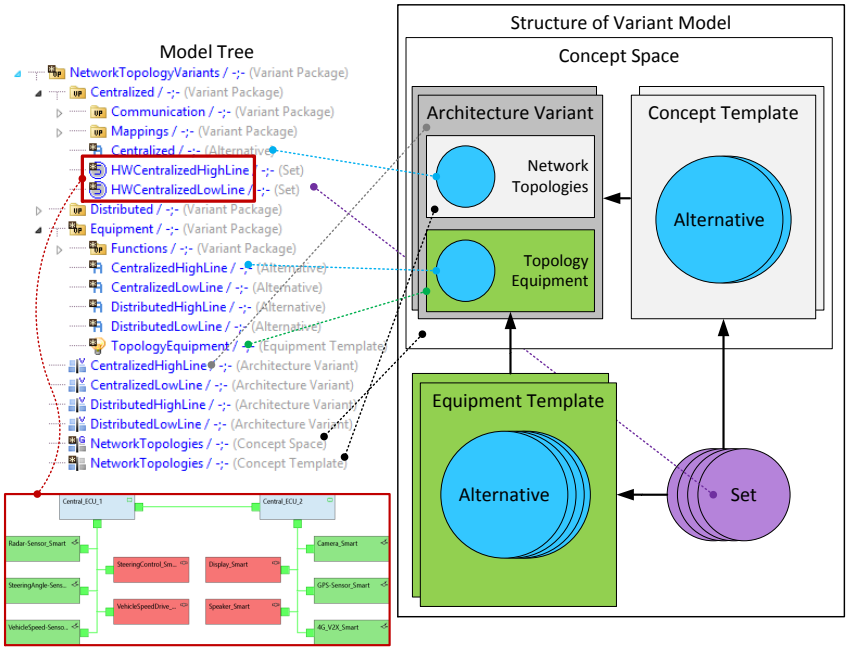


Abbildung 6.30: Struktur des Variantenmodells und die Verbindung zum E/E-Architekturmodell am Beispiel der zentralisierten Architektur (in Anlehnung an [175, S. 1810]).

riante ist. Sets, die mithilfe von Propagationsregeln befüllt wurden, enthalten in Abb. 6.31 den Präfix *Propagated*. Kommunikationsspezifische Sets, die in der jeweils aktiven Variante automatisch durch den Signal-Router befüllt werden, beginnen mit *Comm*.

6.5.2 Analyseergebnisse und Variantenvergleich

Bei der Ausführung des AMiL Prototyps zur Analyse und zum Vergleich der beiden Architekturvarianten wurde folgende Konfiguration angewandt:

- Das zyklische Verhalten der Funktionen auf der BLA wird auf Grundlage der Timing-Propagation der Senderports generiert.

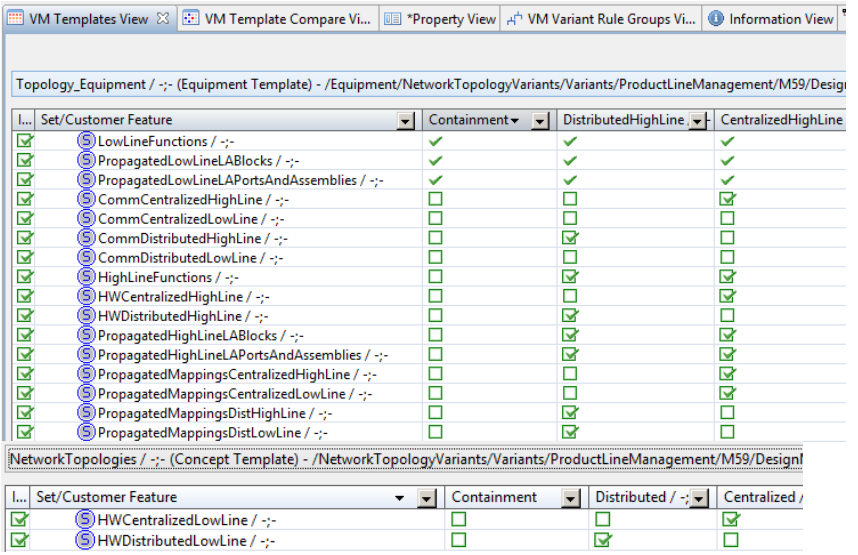


Abbildung 6.31: Komposition von Realisierungsalternativen durch mehrere Sets der Konzept- und Ausstattungsvorlagen *NetworkTopologies* bzw. *TopologyEquipment*.

- Die insgesamt elf verarbeitenden Funktionen der Wirkketten werden für jede Architekturvariante 20 Mal zufällig alloziert.
- Eine ECU der verteilten Architektur kann maximal zwei Funktionen allozieren.
- Ein Domänen-Steuergerät der zentralisierten Architektur kann maximal sechs Funktionen allozieren.

Ein Auszug der integrierten Ergebnistabelle der Analysemetrik innerhalb von PREEvision® ist im Anhang in Abb. A.8 zu finden. Die nachfolgend präsentierten Ergebnisse sind mithilfe der exportierten Excel-Tabelle weiter analysiert und aufbereitet worden.

Die maximalen Kommunikationslatenzen der individuellen Wirkketten sowie die Gesamtlatenz aller Wirkketten, abhängig vom Mapping der Funktionen auf die Steuergeräte der verteilten bzw. zentralisierten Architekturvariante, sind in

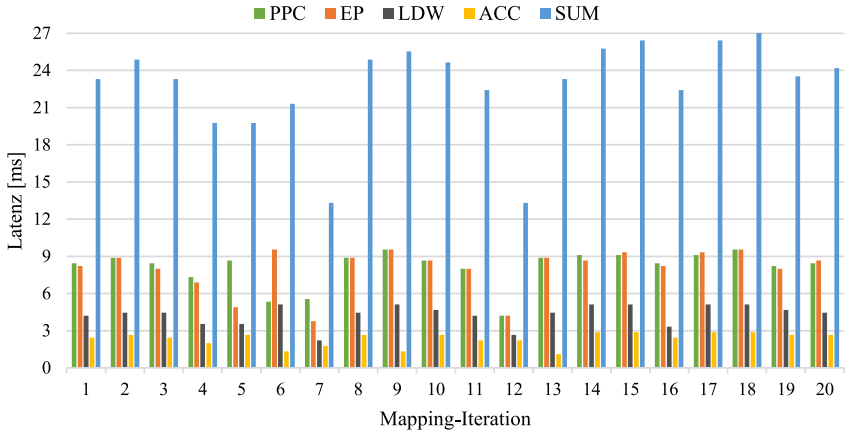
Abb. 6.32 (a) bzw. Abb. 6.32 (b) aufgetragen. Tabelle 6.9 und Tabelle 6.10 zeigen für jede Architekturvariante eine Zusammenfassung einiger Statistikwerte der maximalen Kommunikationslatenzen sowie der resultierenden Buslasten. Zusätzlich zum minimalen und maximalen Latenzwert werden die Differenz des Minimums und Maximums, die Standardabweichung und das arithmetische Mittel über die 20 Iterationen berichtet. Die Gesamtlatenz aller Wirkketten bezieht sich dabei auf genau eine bestimmte Mapping-Iteration, während die Latenzen der individuellen Wirkketten die Minima und Maxima über alle 20 Iterationen darstellen. Die maximalen Buslast-Werte beziehen sich auf jene Mapping-Iteration, in der die minimale und die maximale Gesamtlatenz aller Wirkketten aufgetreten sind.

6.5.3 Diskussion

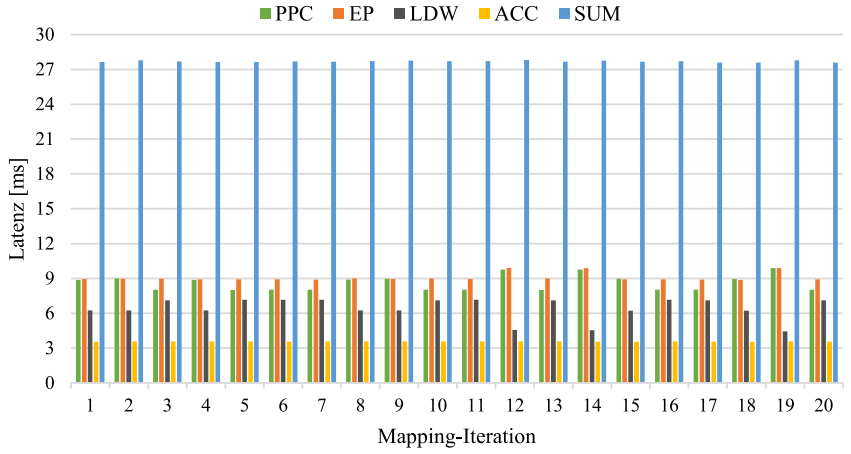
Bei den Latenzen der verteilten Architektur ist eine starke Variabilität und Abhängigkeit vom Mapping zu beobachten. Dies geht aus der Verteilung in Abb. 6.32 (a) sowie aus der Differenz von $13,76\,ms$ und der Standardabweichung von $3,72\,ms$ der Gesamtlatenz in Tabelle 6.9 hervor. Die ACC Wirkkette weist die geringste Latenz und Standardabweichung auf, da sie lediglich aus einer verarbeitenden Funktion besteht und der CAN-Frame `ACC` die höchste Priorität besitzt. Im Gegensatz dazu rufen die Wirkketten PPC und EP die größten Latenzen hervor, da sie einige CAN-Frames mit geringer Priorität versenden. Insbesondere besitzt die EP Wirkkette die längste Verarbeitungskette an Funktionen und die meisten CAN-Frames mit niedriger Priorität. Zusätzlich hat sie die höchste Standardabweichung, der sporadische `4G_V2X` CAN-Frame ist ein Grund dafür.

Tabelle 6.9: Zusammenfassung der Kommunikationslatenz- und Buslastanalyse der verteilten Architekturvariante (Quelle: basierend auf [BNB19]).

	Max. Latenz [*ms*]					**Max. Buslast** [%]
	PPC	EP	LDW	ACC	Σ	HighSpeed CAN
Min	4,22	3,77	2,22	1,11	13,32	6,44
Max	9,55	9,55	5,11	2,89	27,08	8,44
Δ	5,33	5,77	2,89	1,78	13,76	3,11
σ	1,41	1,70	0,82	0,54	3,72	0,92
Mittelwert	8,14	8,00	4,30	2,34	22,78	7,48



(a) Latenzen der verteilten Architektur.



(b) Latenzen der zentralisierten Architektur.

Abbildung 6.32: Maximale Kommunikationslatenzen der individuellen Wirkketten sowie die Gesamtlatenz aller Wirkketten abhängig vom Funktions-Mapping einer Architekturvariante (Quelle: basierend auf [BNB19]).

Betrachtet man die Buslast stellt man fest, dass diese mit einer Standardabweichung von 0,92 % relativ zur Latenz gesehen weitestgehend unabhängig vom Mapping bleibt. Daraus lässt sich ebenfalls schließen, dass die Buslast keine aussagekräftige Metrik dafür ist, um den Einfluss auf individuelle Wirkketten-Latenzen zu untersuchen. Im Gegenteil kann, wie mit obiger Diskussion der Latenzen dargelegt, eine signifikante Abweichung in der minimalen und maximalen Latenz auftreten. Diese Erkenntnis entspricht qualitativ den Beobachtungen in [159], bei denen zusätzliche Sender eine geringe Erhöhung der Buslast hervorrufen, die Latenzen jedoch stark variieren. Zudem ist darin zu beobachten, dass eine hohe Anzahl an sporadischen Frames bei konstanter zyklischer Last zu Verletzungen von Task-Deadlines führen kann.

Zusammenfassend lässt sich schließen, dass die verteilte Architektur ein hohes Optimierungspotential der minimalen und maximalen Latenz von ca. 50,8 % bzgl. der Funktionsverteilung bietet, während die Buslast lediglich um 2,00 % fällt.

Die zentralisierte Architektur führt im Vergleich generell zu höheren Latenzen, sowohl bei den individuellen Wirkketten als auch in der Gesamtlatenz. Ein Hauptgrund dafür sind die zusätzlichen Low-Speed CAN-Busse, über die die Sensoren und Aktuatoren mit den ECUs kommunizieren. Dafür weist diese Architekturvariante eine stabilere Latenzverteilung mit einer Standardabweichung der Gesamtlatenz von 0,07 *ms* auf. Sie ist somit nahezu konstant und unabhängig von der Funktionsverteilung. Grund dafür ist eine geringere Variabilität, da lediglich zwei ECUs für die Funktionsallokation zur Verfügung stehen.

Tabelle 6.10: Zusammenfassung der Kommunikationslatenz- und Buslastanalyse der zentralisierten Architekturvariante (Quelle: basierend auf [BNB19]).

	Max. Latenz [*ms*]					**Max. Buslast** [%]		
	PPC	EP	LDW	ACC	Σ	CAN FD	LowSpeed CAN 1	LowSpeed CAN 2
Min	7,99	8,88	4,44	3,55	27,59	0,30	14,21	8,88
Max	9,89	9,91	7,17	3,58	27,81	0,38	14,21	8,88
Δ	1,90	1,03	2,73	0,03	0,21	0,23	0	0
σ	0,65	0,34	0,90	0,01	0,07	0,06	0	0
Mittelwert	8,61	9,09	6,43	3,57	27,69	0,32	14,21	8,88

Bezüglich der Buslast lassen sich ähnliche Aussagen wie bei der verteilten Architektur treffen, mit dem Unterschied, dass sowohl die Latenzen als auch die Buslast mit Standardabweichungen von $0,07\,ms$ bzw. $0,06\,\%$ nahezu konstant bleiben. Die Buslast der beiden Low-Speed CAN-Busse ist exakt konstant, da die Sensor- und Aktuatorfunktionen ein fixes Mapping besitzen und nicht geändert werden. Das Optimierungspotential der Gesamtlatenz aller Wirkketten verringert sich bei dieser Variante aufgrund der geringeren Variabilität konsequenterweise auf ca. $0,79\,\%$.

Da der Fokus auf der automatisierten Analyse und dem Vergleich von Architekturvarianten liegt und weniger auf der Optimierung, wurde eine zufällige Allokationsstrategie gewählt. Die Limitierung auf 20 Iterationen zeigt jedoch bereits signifikante Unterschiede in den beiden Varianten, wie in den obigen Ausführungen dargelegt wurde. Darüber hinaus sind beliebig weitere Architekturvarianten denkbar, die untersucht und mit den diskutierten verglichen werden können. Zum Beispiel die Anbindung von Sensoren mit potenziell großen Datenmengen wie Kamera an den CAN FD-Bus mit einer maximalen Frame-Größe von 64 *Bytes* statt den hier konfigurierten 8 *Bytes* oder alternative CAN ID-Listen.

Abschließend ist hinsichtlich der Limitierung der CAN-Simulation zu bemerken, dass der CAN-Bus Aspekt von PtII kein bit-genaues Timing unterstützt und der CAN FD-Bus somit mit der fixen Bitrate der Datenphase von 10 *Mbps* simuliert wird, ohne den Bitraten-Wechsel zwischen Arbitrations- und Datenphase zu berücksichtigen. Aus diesem Grund wurde auch die Buslast bezogen auf die 10 *Mbps* berechnet, um die beiden Metriken auf derselben Grundlage konsistent zu halten.

6.6 Fazit

In diesem Abschnitt wird zunächst eine ganzheitliche Betrachtung durchgeführt, indem die entwickelten Methoden und Konzepte anhand ihrer demonstrierten Anwendungsfälle mit den systematisch abgeleiteten Anforderungen aus Abschnitt 4.3 abgeglichen werden. Anschließend folgt eine Diskussion über einen möglichen Transfer auf weitere E/E-Architektur-Metamodelle und Zielsimula-

toren sowie über die Erweiterbarkeit der vorliegenden Arbeit hinsichtlich der Berücksichtigung weiterer Architekturaspekte und Analysemethoden.

6.6.1 Gesamtbetrachtung bezüglich abgeleiteter Anforderungen

Zunächst werden gemäß Unterabschnitt 4.3.6 die Anforderungen in vier Kategorien eingeteilt und anschließend die darin befindlichen, individuellen Anforderungen den demonstrierten Anwendungsfällen gegenübergestellt. Dies ist in Tabelle 6.11 als Abdeckungstabelle zusammengefasst. Ein grünes Häkchen (✓) signalisiert dabei die Erfüllung der jeweiligen Anforderung durch die den gezeigten Anwendungsfällen zugrunde liegenden Methoden und Konzepte. Insgesamt kann mithilfe von Tabelle 6.11 festgestellt werden, dass alle Anforderungen erfüllt werden, entweder durch individuelle Anwendungsfälle oder durch deren Kombination. Ein Beispiel für eine kombinierte Abdeckung ist Anforderung **REQ07**, welche durch die beiden Demonstrationen der ebenenübergreifenden ACC Simulation und der elektrischen und leitungssatzsensitiven Simulation erfüllt wird. Simulator bezogene Anforderungen wurden bereits in Unterabschnitt 5.1.2 qualitativ begründet und werden durch die demonstrierten Anwendungsfällen ebenfalls untermauert.

Hinsichtlich der ersten Kategorie wird die integrierte und zur Architektur komplementäre Verhaltensspezifikation mittels mehreren Modellierungsparadigmen und Berechnungsmodellen über alle Anwendungsfälle gezeigt. Beginnend auf der logischen Architektur, aber insb. über mehrere Abstraktionsebenen hinweg und durch Verfeinerung von bestehenden Architektur- und Verhaltensmodellen, ohne die Schnittstellen anpassen zu müssen. Dies wurde über die diversen, eingeführten Mappings gelöst. Darüber hinaus sind alle Anwendungsfälle integriert simulierbar (vgl. **REQ01**–**REQ08**).

Anforderung **REQ09** wird zunächst durch die Synthese adressiert, indem alle Simulationsartefakte mit der UUID des ursprünglich assoziierten Architekturartefakts annotiert werden (vgl. Unterabschnitt 5.1.3 und 5.1.4). Die daraus resultierende Nachverfolgung von Modellartefakten und Simulationsdaten wurde konkret anhand der metrikbasierten Latenzanalyse von Wirkketten verschiedener Architekturvarianten demonstriert. In dieser wurde auch die automatische

Generierung des zyklischen Verhaltens, basierend auf den Timing Constraints der logischen Port-Prototypen, gezeigt (**REQ04**).

Kategorie zwei betrifft die domänen- und ebenenübergreifende Simulation. In allen Fällen wird das realisierungsunabhängige Verhalten der logischen Architektur durch realisierungsabhängige Details von modular entwickelten Architekturaspekten der darunterliegenden Ebenen automatisch verfeinert (**REQ10**). Die domänenübergreifende Simulation wird im Wesentlichen durch die Einbindung der Netzwerkkommunikation sowie der Elektrik, des Leitungssatzes und der Topologie als gekapselte, aspektorientierte Simulation charakterisiert (**REQ12**–**REQ14**). Die vertikale Interaktion der Funktionen mit Steuergeräten und die Einbindung von Ausführungsaspekten der ACC Applikation adressiert die ebenenübergreifende Simulation (**REQ12**).

Die letzten beiden Kategorien werden vollständig von den dazugehörigen Demonstrationen der Latenz- und Buslastanalyse von Architekturvarianten bzw. der ACC FMI Co-Simulation abgedeckt.

6.6.2 Übertragbarkeit und Erweiterbarkeit

6.6.2.1 Übertragbarkeit

Basierend auf der typischen Unterteilung einer E/E-Architektur in mehrere Abstraktionsebenen (vgl. Abb. 2.2), trägt die neu eingeführte BLA Ebene und deren eingeführten Mappings zu bestehenden Ebenen nach dem Schichtenmodell zur Übertragbarkeit auf andere ADLs wie die in Unterabschnitt 3.1.1 vorgestellte EAST-ADL bei. Gleichzeitig sind durch die separate BLA Ebene blockdiagrammbasierte Verhaltensmodelle anderer Actor-orientierter Zielsimulatoren durch Import von deren Bibliothek instanziierbar. Zur Simulationssynthese ist dann ein entsprechend angepasstes Back-End zu implementieren.

Zusätzlich werden durch die generischen Schnittstellen und Klassen-Templates der Softwarearchitektur (siehe Unterabschnitt 5.1.3, 5.1.4 und 5.3.8) die Portierung und Übertragbarkeit auf andere Metamodelle erleichtert, sowohl hinsichtlich des zugrunde liegenden E/E-Architekturmodells als auch des Zielsimulators (vgl. **REQ11**).

Tabelle 6.11: Abdeckung der in Unterabschnitt 4.3.6 spezifizierten Anforderungen durch die demonstrierten Anwendungsfälle.

Anforderungskategorie	REQ#	Demonstrierter Anwendungsfall				
		ACC Simulation	Ebenenüber-greifende ACC Simulation	EC/WH-sensitive Simulation	Latenz- und Buslastanalyse Arch.varianten	FMI Co-Simulation ACC
Integrierte Verhaltensspezifikation und Simulation	**REQ01**	✓	✓	✓	✓	✓
	REQ02	✓	✓	✓	✓	✓
	REQ03	✓	✓	✓	✓	✓
	REQ04				✓	
	REQ05	✓	✓	✓	✓	✓
	REQ06		✓			
	REQ07		✓	✓		
	REQ08	✓	✓	✓	✓	✓
	REQ09				✓	
Domänen- und ebenenübergreifende Simulation	**REQ10**	✓	✓	✓	✓	✓
	REQ11	✓	✓	✓	✓	✓
	REQ12	✓	✓		✓	✓
	REQ13		✓			
	REQ14			✓		
	REQ15	✓	✓	✓	✓	✓
Integrierte, statische und dynamische metrikbasierte Variantenbewertung	**REQ16**				✓	
	REQ17				✓	
	REQ18				✓	
	REQ19				✓	
	REQ20				✓	
Integration und Simulation von domänenspezifischen Modellen	**REQ21**					✓
	REQ22					✓
	REQ23					✓

Die auf AUTOSAR basierten Konzepte zur Modellierung der ebenenübergreifenden analogen und Basisservice Schnittstellen, welche zur Interaktion des logischen Verhaltens mit detailliertem Verhalten der Hardwarearchitektur dienen, trägt zur Wiederverwendung von standardisierten Formaten bei. Voraussetzung für den Einsatz von analogen Ports für die elektrische und leitungssatzsensitive Simulation ist, dass das zugrunde liegende Metamodell der E/E-Architektur den entsprechenden Detailgrad, in Form von elektrischen Bauelementen oder physikalischen Leitungen und deren Eigenschaften, aufweist.

Analoges zur standardisierten AUTOSAR Modellierung gilt für die Modellierung von UML-konformen State Charts. Die Erweiterung dieser mit den vorgestellten Schnittstellen und dem Actor-orientierten Verhalten steigert infolgedessen auch die Übertragbarkeit und Integration dieser Konzepte in weiteren Werkzeugen, die diesen Standards folgen.

6.6.2.2 Erweiterbarkeit

Anforderung **REQ12** wird im Wesentlichen durch die automatisch synthetisierte, aspektorientierte Simulation erfüllt. Sollen weitere oder detailliertere Architekturaspekte ergänzt werden, kann dies mithilfe der modularen und erweiterbaren Architektur der Interpreter-, Builder- und Dekorierer-Entwurfsmuster leicht umgesetzt werden. Dazu zählen bspw. weitere Bussysteme wie Ethernet, die – sobald sie in Ptolemy II als dedizierte Kommunikationsaspekte zur Verfügung stehen – durch die aspektorientierte Simulation des Netzwerks in der AMiL Metrik unmittelbar für eine erweiterte Latenzanalyse eingesetzt werden können. Des Weiteren können Details der Softwarearchitektur berücksichtigt werden. Dazu zählt u. a. ein prioritätsbasiertes Scheduling-Verfahren nach AUTOSAR Adaptive.

In den demonstrierten Anwendungsfällen sind mehrere Aspekte der domänen- und ebenenübergreifenden Simulation einzeln und teilweise kombiniert gezeigt worden. Jedoch sind diese auch beliebig miteinander kombinierbar. Grund dafür sind die modularen Schnittstellen (reguläre, analoge und Basisservice Ports) sowie die Modellierung auf separaten Ebenen, die über Mappings miteinander in Beziehung gesetzt und während der Synthese verschmolzen werden. Die aspektorientierte Simulation sorgt dabei für eine saubere, semantische Trennung der repräsentierten Domäne einer Ebene.

Darüber hinaus erhöht die Einbindung der FMI Schnittstelle als weiteres Standardformat die Wiederverwendung domänenspezifischer Simulationsmodelle unabhängig von deren Quellwerkzeug. Die modellbasierte Einbindung der FMU samt FMU-Datei als Funktionstyp steigert die Wiederverwendung in verschiedenen Subsystemen. Durch den einfachen Austausch des Verhaltens einer Funktion mit einer FMU wird so nicht nur die Erweiterbarkeit gesteigert, sondern trägt auch zur iterativen Integration des Verhaltens durch die verteilte Entwicklung bei.

Die Bewertung von Architekturvarianten kann durch das vorgestellte AMiL Konzept um beliebige, benutzerdefinierte Metriken erweitert werden. Hinsichtlich dynamischer Metriken ist dafür ein entsprechender Metrikblock zu implementieren sowie ein analysespezifischer Sink-Actor oder Listener auf Seiten des Simulators, der die im Metrikblock benötigten Simulationsdaten bereitstellt. Diese müssen vom *Simulation Executor* in der SDR abgelegt werden (siehe Unterabschnitt 5.4.2).

Eine Optimierung der EEA bzgl. mehrerer Metriken ist dadurch erweiterbar, dass bspw. statt dem zufälligen Vertauschen von Funktionen ein spezifischer Allokationsalgorithmus zur Optimierung der Latenzanalyse implementiert werden kann. Da das AMiL Konzept des Weiteren die gleichzeitige Betrachtung von statischen und dynamischen Metriken ermöglicht, können aber auch Algorithmen zur multikriteriellen Optimierung von Metriken, analog zur integrierten Latenz- und Buslastmetrik, erweitert werden. Ein Beispiel wäre die Betrachtung zusätzlicher, statischer Metriken wie Kosten, Gewicht oder Ressourcenverbrauch von zentralisierten Steuergeräten (vgl. [73, 123]). Jedoch kann im Vergleich zu diesen Arbeiten mit dem AMiL Ansatz eine multikriterielle Optimierung bzgl. statischer *und* dynamischer Metriken gleichzeitig erreicht werden, was zu einer globalen Betrachtung der EEA beiträgt. Die Algorithmen können dabei in dedizierten *EEA Modifikationsblöcken* (vgl. Unterabschnitt 5.4.1) umgesetzt werden.

6.7 Einordnung in verwandte Arbeiten und Abgrenzung

Ein ausführlicher Überblick über den Stand der Technik und Forschung wurde bereits in Kapitel 3 gegeben. An dieser Stelle werden im Folgenden einige weitere

Arbeiten diskutiert, die am ehesten verwandt zu der vorliegenden Arbeit sind. Anschließend folgt eine zusammenfassende Abgrenzung anhand einer Tabelle.

6.7.1 Beschreibung und Diskussion

Zu den architekturzentrierten Ansätzen zur Generierung von Simulationsmodellen basierend auf der EAST-ADL zählen [58, 101, 185, 186]. Ansätze, die sich auf Architekturen von allgemeinen CPS fokussieren werden in [137, 138, 182, 183] vorgestellt. Eine proprietäre Lösung ausgehend von logischen Architekturen genereller Multidomänen-Systeme wird in [168] beschrieben.

Rajhans et al. [138] schlagen eine CPS ADL vor, die es erlaubt, endliche Zustandsprozesse (engl. *Finite State Processes (FSP)*) oder lineare hybride Automaten (engl. *Linear Hybrid Automata (LHA)*) an Architekturkomponenten zu annotieren, um daraus eine Textdatei für ein dediziertes, externes Analysewerkzeug zu generieren. Somit wird eine integrierte Verhaltensspezifikation unterstützt. Allerdings wird ein typisches Schichtenmodell mit mehreren Abstraktionsebenen, wie es bei E/E-Architekturen zur Beherrschung der Komplexität gehandhabt wird, nicht adressiert. Die beiden Modellierungsformalismen des Verhaltens können zudem lediglich getrennt voneinander im jeweiligen Analysewerkzeug untersucht werden. Eine kombinierte Simulation verschiedener Formalismen in einem gemeinsamen Simulationsmodell wie in dieser Arbeit ist nicht möglich.

Ein generisches Framework, das sich mit der Komposition von heterogenen Modellen in Kombination mit einer Weiterentwicklung der CPS ADL beschäftigt, wird in [137] vorgestellt. Die Autoren verwenden semantische Mappings (vgl. Unterabschnitt 2.2.1) zwischen heterogenen Verifikationsmodellen unterschiedlicher Formalismen (z. B. hybride Automaten). Jedes Teilmodell einer Hierarchieebene ist dabei einer bestimmten Sicht auf die Architektur zugeordnet. Die Überprüfung der semantischen Konsistenz in externen Werkzeugen wird als zukünftige Arbeit angegeben, eine Rückführung der Analysedaten wird nicht erwähnt. Eine Betrachtung von Architekturvarianten und deren Bewertung bzgl. bestimmter Metriken werden nicht diskutiert. Der Fokus liegt mehr auf der formalen Analyse und Verifikation der semantischen Hierarchien eines CPS durch die Verifikationsmodelle und nicht auf einer einheitlich zugrunde liegenden Simulation. Beispielsweise erlaubt der Ansatz keine Verfeinerung von Architektursichten, um Verhalten von

spezifischen Simulationsmodellen integriert in der ADL zu verfeinern, wie es in der vorliegenden Arbeit möglich ist. Die in dieser Arbeit eingeführte BLA Ebene zur integrierten, Actor-orientierten Modellierung kann darüber hinaus mit State Charts und weiteren, realisierungsabhängigen Architekturaspekten gekoppelt werden kann.

Die EAST-ADL basierten Ansätze [58, 101, 185, 186] delegieren das Verhalten von Architekturartefakten über die `FunctionalBehavior` Metaklasse an extern definierte Simulationsmodelle (außerhalb des EEA Modells). Eine integrierte Verhaltensspezifikation wird somit von keinen der genannten Ansätze adressiert.

Die Autoren in [185, 186] verweisen das Verhalten auf externe SystemC Module, deren Dateien aus einer Bibliothek referenziert und über Modell-zu-Text-Konverter oder umgekehrt in das dazugehörige EAST-ADL bzw. SystemC Modell zur Simulation transformiert werden. In [185] wird die Integration von generiertem C-Code aus handelsüblichen Werkzeugen in die SystemC Module vorgeschlagen. Die C-Code Integration muss allerdings zunächst manuell erfolgen, bevor die SystemC Module als externe Verhaltensmodelle referenziert und simuliert werden können. State Charts werden in proprietären, externen Werkzeugen modelliert, um den o. g. C-Code zu generieren, können aber nicht integriert im Architekturmodell modelliert werden und nicht ebenenübergreifend interagieren. Während sich die Autoren in [185] auf die Abbildung von EAST-ADL auf SystemC-TLM [53] über alle Abstraktionsebenen der EAST-ADL konzentrieren, verwenden Weissnegger et al. [186] zusätzlich SystemC-AMS, um analoge Teile der EAST-ADL zu unterstützen, beschränken sich aber auf die Designebenen der EAST-ADL (vgl. Unterabschnitt 3.1.1). Marinescu et al. [101] schlagen die Simulation von EAST-ADL Funktionen auf der Funktions-Design-Architektur (FDA) vor, basierend auf verlinkten Simulink- und FMU-Modellen. Die Ebene der Hardware-Design-Architektur (HDA) wird dabei nicht berücksichtigt.

Eine integrierte Verhaltensmodellierung beginnend auf der logischen Architektur (vgl. Analysis Level der EAST-ADL), die durch ebenenübergreifende Mechanismen automatisiert mit den realisierungsabhängigen Architekturaspekten – Netzwerk, Elektrik, Leitungssatz, Ausführungsaspekte von Funktionen etc. – zu einer domänenübergreifenden Simulation angereichert wird, wird nicht in dem Umfang adressiert.

Weiss et al. [185] beschränken sich mit SystemC-TLM auf die digitale Domäne. Obwohl die Verwendung von SystemC-AMS das Potential zur Simulation von analogen Anteilen besitzt, wird eine elektrische und leitungssatzsensitive Simulation in [186] nicht diskutiert. Die Verwendung von QSS Methoden zur diskreten Simulation der analogen Teile unterscheidet die vorliegende Arbeit ebenfalls[16]. Ausführungsaspekte von Funktionen und die Generierung von abstraktem, ereignisbasiertem Verhalten basierend auf Timing Constraints, wird ebenfalls nicht adressiert. Bis auf die Arbeit von Marinescu et al. [101] wird zudem keine Einbindung von FMUs diskutiert, welche aber auf die proprietäre Werkzeugumgebung von Simulink zur Simulation angewiesen ist. Obwohl Timing Constraints der EAST-ADL in der Transformation durch Simulink-Entwurfsmuster berücksichtigt werden, ist dies nicht aspektorientiert gelöst. Ausführungsaspekte von Funktionen auf geteilten Ressourcen, oder ein mit der Hardware-Design Ebene interagierendes Verhalten, um bspw. Aufstartzeiten zu simulieren (vgl. Unterabschnitt 6.3.4), ist aufgrund der fehlenden Einbindung jener Ebene nicht möglich. Eine variantensensitive Synthese von Architekturmodellen sowie die Rückführung und Analyse der Simulationsdaten in weiteren Metriken wird von keinen der diskutierten Ansätze berücksichtigt.

Der Ansatz in [58] delegiert das Verhalten der EAST-ADL Funktionen auf Designebene ebenfalls an externe FMUs oder AUTOSAR SWCs. Letztere liegen in Form von C-Code mit dazugehörigen Wrapper-Funktionen vor. Beide Arten realisieren jeweils sog. Simulationsmodule. Die Module kommunizieren mit einer gemeinsamen API über die dort entwickelte *Adapt* Simulationsmiddleware. Eine variantensensitive Synthese ähnlich zu der vorliegenden Arbeit wird mittels des Variabilty Resolution Mechanismus der EAST-ADL erreicht (vgl. Unterabschnitt 3.4.3). Allerdings wird eine automatisierte Bewertung und ein Vergleich von Architekturvarianten bzgl. dynamischer Metriken durch Rückführung und Assoziation der Simulationsdaten mit den ursprünglichen Architekturartefakten nicht adressiert. Weitere Limitierungen im Vergleich zu dieser Arbeit sind nach [58, Kap. XIII] die fehlende Betrachtung von Netzwerkkommunikation und Ausführungszeiten.

[16] Die Verwendung von QSS Methoden zur Simulation von elektrischen Schaltungen wurde durch Lee et al. [87] inspiriert. Im Gegensatz dazu wird in dieser Arbeit jedoch eine elektrische Netzliste, basierend auf ihrer natürlichen, akausalen Schaltungsnotation im Architekturmodell, synthetisiert und nicht über ein kausales Actor-Netzwerk modelliert. Die Kapselung in einem dedizierten SPICE-Aspekt unterscheidet den vorliegenden Ansatz zusätzlich.

Die proprietäre Lösung in [168] verwendet das kommerzielle Werkzeug Simcenter System Synthesis, ein von Siemens PLM Software entwickeltes Werkzeug zur Systemarchitektur-Entwicklung, Systemintegration und für das Konfigurationsmanagement. Die Architektur ist dabei vergleichbar mit der logischen Architektur einer E/E-Architektur, deren Komponenten als sog. *Simulation Templates* dienen. Die Templates spezifizieren die Ports, Parameter und Variablen als Schnittstelle, die von einer dazugehörigen Simulationskomponente erfüllt werden muss. Simulationskomponenten können entweder Blackbox Verhaltensmodelle von Amesim[17] oder FMUs zur Co-Simulation bzw. zum Modellaustausch (engl. *Model Exchange*) sein. Letztere werden durch Mappings von Ports, Variablen und Parametern an ein Simulation Template gebunden. Der Fokus liegt dabei eher auf der Simulation von kontinuierlichen Modellen, wobei die Modelle zum Export der FMUs in der betrachteten Fallstudie in Dymola erstellt wurden.

Im Unterschied zu dieser Arbeit werden Port-Mappings nicht auf Ebene von Port-Prototypen vorgenommen, wodurch jede Simulationskomponente erneut instrumentiert werden muss. Port-Prototyp-Mappings werden in dieser Arbeit einmalig automatisiert generiert, unterschiedliche Realisierungen des Verhaltens können durch Block-Mappings ausgetauscht werden (vgl. Abschnitt 6.1 und 6.2). Eine integrierte Verhaltensspezifikation als Whitebox, in welcher die Interna eines Verhaltensblocks explizit Teil des Architekturmodells sind, sowie eine ebenenübergreifende Verhaltensmodellierung werden nicht diskutiert. Grund dafür sind die nicht näher erläuterte Hierarchisierung und Dekomposition von Simulation Templates. Eine standardisierte Modellierung der Schnittstellen, wie in dieser Arbeit basierend auf AUTOSAR, wird ebenso wenig adressiert wie realisierungsabhängige Aspekte in Abhängigkeit der Partitionierung. Dazu zählen die Netzwerkkommunikation und Ausführungsaspekte von Funktionen, aber auch die leitungssatzsensitive Simulation von elektrischem Verhalten oder die Betrachtung von Architekturvarianten.

Wan et al. [182, 183] stellen einen Ansatz auf Grundlage der *Functional Basis* Sprache für funktionale Beschreibungen vor und synthetisieren daraus Amesim Simulationsmodelle. Ein Verfeinerungsprozess kann, basierend auf den zur Verfügung

[17] Ein ebenfalls von Siemens PLM Software entwickeltes Werkzeug zur Multidomänen-Simulation von komplexen Systemen (siehe https://www.plm.automation.siemens.com/global/en/products/simcenter/simcenter-amesim.html, zuletzt aufgerufen am 11.10.2019).

stehenden Architektur-Templates[18], den gewählten Simulationskomponenten und den spezifizierten Schnittstellen, Verbesserungsvorschläge für das initiale funktionale Modell liefern. Der Fokus liegt jedoch erneut auf der Simulation von kontinuierlichen Modellen. Weitere E/E-Architekturaspekte wie Netzwerkkommunikation, Ausführungsaspekte von Funktionen und leitungssatzsensitive Simulation werden nicht diskutiert. Eine explizite Bewertung ganzer E/E-Architekturvarianten bzgl. dynamischer und statischer Metriken, wie mit dem AMiL Konzept, wird ebenfalls nicht adressiert. Die Varianten werden im Gegensatz zum AMiL-Ansatz implizit während der Synthese gebildet, basierend auf den spezifizierten Schnittstellen des funktionalen Modells und den verfügbaren Architektur-Templates.

AutoFOCUS 3 [6] ist ein integriertes Werkzeug zur durchgängigen, modellbasierten Entwicklung von eingebetteten Systemen. Der zugrunde liegende Modellierungsansatz basiert auf der FOCUS-Theorie [23] mit Schwerpunkt auf der Nachverfolgung von Anforderungen auf Softwarekomponenten, Modellierung von Softwarekomponenten, klassische MiL Simulation, formaler Verifikation, Code-Generierung sowie Mapping und Scheduling von Komponenten auf eine technische Hardwarearchitektur. Das optimierte Mapping und Scheduling ist auf spezifische, vordefinierte Plattformen[19] beschränkt, bspw. ein spezifischer Multicore-Prozessor, sowie auf eine generische Plattform auf ECU-Vernetzungsebene ohne interne Hardwarekomponenten. Im Allgemeinen werden detailliertere EEA-Ebenen wie Elektrik und Leitungssatz nicht adressiert und liegen außerhalb des Rahmens des Werkzeugs.

Ein Vorteil gegenüber den meisten existierenden Ansätzen (siehe Abschnitt 3.1) ist eine integrierte Verhaltensmodellierung und Simulation der logischen Softwarearchitektur mittels mehrerer Formalismen, inkl. FSMs, Tabellen, Code-Beschreibungen und Modi-Automaten. Letztere ähneln dem Aufbau von Modal Models und kann ein interessanter Anknüpfungspunkt in zukünftigen Arbeiten sein, da sich die Abbildungsvorschriften durch die Ähnlichkeit der beiden Formalismen relativ einfach gestalten könnten. Die Verhaltensmodellierung von einfachen FSMs ist allerdings nicht an standardisierten Notationen wie UML-konformen

[18] Eine Kombination aus ein oder mehreren Funktionen, z. B. ein Verbrennungsmotor, und dazugehörigen Architektur-Parametern. z. B. die Anzahl an Zylindern.

[19] Nach [6] wird unter einer Plattform Hardware und dazugehörige Software verstanden, z. B. Treiber und Betriebssystem.

State Charts orientiert. Möglichkeiten zur Modellierung von nebenläufigen und History-Zuständen werden nicht genannt und haben somit den Nachteil der Zustandsexplosion für komplexeres Verhalten (ähnliches lässt sich auf die anderen Modellierungsformalismen übertragen). Analog dazu sind die Schnittstellen nicht nach AUTOSAR standardisiert modellierbar. Ein Import von FMUs wird zwar unterstützt, generell wird eine domänen- und ebenenübergreifende Simulation der logischen Architektur mit der Hardwarearchitektur (Netzwerk, Elektrik, Leitungssatz) allerdings nicht adressiert. Ein Grund dafür ist die Beschränkung auf die zugrunde liegende Discrete-Time Semantik [6].

Ein weiterer Vorteil ist die integrierte Entwurfsraumexploration und Optimierung für das Mapping und Scheduling. Die Simulation eines abstrakten Multicore-Schedulings im Zusammenspiel mit dem Funktionsverhalten über mehrere ECUs (vgl. Abschnitt 6.3) wird jedoch nicht adressiert. Während sich die DSE auf Mapping-Varianten von Softwarekomponenten und deren Scheduling beschränkt, können weitere Metriken wie im AMiL Konzept genauso wenig berücksichtigt werden, wie die variantensensitive Simulation ganzer Architekturvarianten (vgl. Abschnitt 6.5).

6.7.2 Zusammenfassung

Ausgehend von den diskutierten Arbeiten und Werkzeugen in Kapitel 3 und Unterabschnitt 6.7.1, werden in Tabelle 6.12 die relevanten Arbeiten anhand der wichtigsten Eigenschaften zur Erfüllung der Anforderungen einer ganzheitlichen Betrachtung von E/E-Architekturen zusammengefasst.

Analog zur vorgestellten Methodik und den daraus abgeleiteten Anforderungen in Kapitel 4 ergibt sich die Aufteilung in Tabelle 6.12 in die drei Kategorien (1) modellbasierte Verhaltensspezifikation (als Voraussetzung für eine integrierte simulationsbasierte Herangehensweise), (2) die automatische Simulationssynthese und integrierte Ausführung von funktionalem Verhalten mit nicht-funktionalen Architekturaspekten in realisierungsnahen Abstraktionsebenen sowie (3) die integrierte metrikbasierte Variantenbewertung bzgl. statischer und dynamischer Metriken gleichzeitig. Unter den Kategorien befinden sich die jeweils wichtigsten Eigenschaften, die als Zusammenfassung der Anforderungen in Kapitel 4 anzusehen sind.

Tabelle 6.12: Abgrenzungstabelle mit den wichtigsten Eigenschaften zur modellbasierten Verhaltensspezifikation, Simulationssynthese und metrikbasierten Variantenbewertung. Eigenschaft wird ✓: vollständig unterstützt; ●: bedingt unterstützt; ✗: nicht unterstützt.

	Modellbasierte Verhaltensspez.				**E/E-Architektur-zentrierte Simulationssynth.**			**Integrierte metrikbasierte Variantenbewertung**	
	Integriert	**Hierarchisch**	**Ebenenübergr. Interaktion**	**Multi-Paradigm**	**Funktional + Arch.aspekte**	**FMI Co-Simulation**	**EC/WH-sensitive Simulation**	**Arch.varianten Simulationssyn.**	**Statische & dynamische Metrikanalyse**
Eigene Beiträge	✓	✓	✓	✓	✓	✓	✓	✓	✓
PREEvision® [175]	✓	✓	✗	✗	✗	✗	✗	●	●
EAST-ADL-basiert [101, 185, 186]	●	✗	✗	✗	●	✓	✗	✗	✗
MAENAD Projekt [97, 192]	✓	✗	✗	✗	●	✗	✗	●	●
AADL-basiert [36, 79, 148, 194]	✓	✓	●	✗	✗	✗	✗	✗	●
AutoFOCUS 3 [6]	✓	✓	✗	✓	●	✓	✗	✗	●
Wan et al. [182, 183]	✗	✗	✗	✗	●	✗	✗	●	✓
Rajhans et al. [137, 138]	✓	✓	✓	●	✗	✗	✗	✗	✗
Vansina et al. [168]	●	✗	✗	●	●	✓	✗	✗	✗
INTO-CPS Projekt [75, 76, 77]	●	✗	✗	✗	●	✓	✗	✗	●
Heavy-Road Projekt [58]	●	✗	✗	●	●	✓	✗	✓	✗
Capital Tool-Suite [107]	✗	✗	✗	✗	✗	✗	✓	●	●

Wie in Tabelle 6.12 zu erkennen, werden in den ersten beiden Kategorien eine oder mehrere Eigenschaften vollständig oder teilweise abgedeckt. Allerdings unterstützt keine Arbeit alle Eigenschaften, die für eine verzahnte modell- und simulationsbasierte Entwicklung von E/E-Architekturen über alle relevanten Abstraktionsebenen notwendig sind. Dazu zählt insb. die elektrische und leitungssatzsensitive Simulationssynthese. Aber auch die Kombination aus funktionaler und ebenenübergreifender Verhaltenssimulation – beginnend auf der logischen Architektur – mit automatisch synthetisierten Architekturaspekten aus verschiedenen Abstraktionsebenen wird nur bedingt unterstützt. Die Capital Tool-Suite ist auf das elektrische Design spezialisiert, lässt aber die übrigen Eigenschaften vermissen. Darüber hinaus wird eine hierarchische Verhaltensspezifikation mit mehreren, interagierenden Modellierungsparadigmen und Berechnungsmodellen von keinen der Arbeiten vollumfänglich unterstützt.

Hinsichtlich der integrierten, metrikbasierten Variantenbewertung unterstützen nur wenige Arbeiten sowohl eine variantensensitive Simulationssynthese als auch eine Bewertung hinsichtlich statischer und dynamischer Metriken gleichzeitig. Wan et al. bilden hier die Ausnahme mit der Einschränkung, dass keine expliziten Architekturvarianten bewertet werden können. Das INTO-CPS Projekt erlaubt eine Exploration bzgl. dynamischer Metriken. Jedoch bewegt sich die Variantenbewertung eher auf Algorithmenebene, bspw. zur Optimierung von Reglerparametern. Eine Variation ganzer E/E-Architekturvarianten bei gleichzeitiger Variation des Funktionsverhaltens und -Mappings werden nicht adressiert. Auf die gleichzeitige Betrachtung von statischen Metriken wird nicht näher eingegangen.

Zusammenfassend erfüllt keine der betrachteten Arbeiten alle aufgeführten Eigenschaften teilweise oder vollständig in den jeweiligen Kategorien (mit Ausnahme Rajhans et al. in der ersten Kategorie) und insb. nicht alle drei Kategorien übergreifend.

7 Schlussfolgerung und Ausblick

Im Folgenden werden die erarbeiteten Methoden, Werkzeuge und die Ergebnisse dieser Dissertation zusammengefasst sowie ein Ausblick auf mögliche aufbauende Arbeiten und Erweiterungen gegeben.

7.1 Zusammenfassung und Schlussfolgerungen

Mit den in dieser Arbeit entwickelten Methoden, Konzepte und Werkzeuge wird die Möglichkeit einer über alle relevanten Abstraktionsebenen von E/E-Architekturen verzahnten modell- und simulationsbasierten Entwicklung geschaffen. Diese ist typischerweise frühzeitig im Entwicklungsprozess angesiedelt, aber auch die Wiederverwendung und iterative Verfeinerung bestehender Architektur-Designs wird gestattet. Daraus resultiert eine integrierte, auf einem einheitlich zugrunde liegenden single-source Modell für Architektur und Verhalten, ausführbare Architekturspezifikation für eine integrierte, ganzheitliche Simulation und Analyse. Eine integrierte Herangehensweise hat den weiteren Vorteil, dass fehleranfällige Import/Export-Prozesse mit diversen Werkzeugketten reduziert, die verteilte Entwicklung gefördert und die Konsistenz des für alle Stakeholder einheitlich zugrunde liegenden Modells sichergestellt werden.

Aufbauend auf den systematisch abgeleiteten Anforderungen an eine verzahnte modell- und simulationsbasierte E/E-Architekturentwicklung, welche sich aus den Limitierungen des aktuellen Standes der Technik und Forschung sowie aus wichtigen Anwendungsfällen ergeben, wird mit dieser Arbeit zunächst eine Menge an Modellierungskonzepten zur *ebenenübergreifenden, ausführbaren Verhaltensspezifikation mittels mehrerer Modellierungsparadigmen* bereitgestellt. Dazu zählen Actor-orientierte Modelle, UML-konforme State Charts und akausale elektrische Schaltungen sowie deren Kombination. Begleitend wurden *Mechanismen zur automatischen Simulationssynthese für eine domänenübergreifende Simulation* des

zunächst realisierungsunabhängigen Funktionsverhaltens mit *realisierungsnahen und nicht-funktionalen Architekturaspekten* entwickelt.

Grundlage dafür ist ein schichtenorientierter Ansatz, der die typischen Abstraktionsebenen einer E/E-Architektur um eine Schicht namens *Behavioral Logical Architecture* ergänzt. Diese erlaubt es, statische Funktionen der logischen Architektur um ausführbares Verhalten zu verfeinern, welches zur Steigerung der Produktivität und Wiederverwendung aus einer ausführbaren Funktionsbibliothek instanziiert werden kann. Mithilfe einer Reihe komplementär eingeführter *Mappings* zwischen den E/E-Abstraktionsebenen sowie auf AUTOSAR basierten Schnittstellen in Form von *analogen Ports* und *Basisservice Interfaces*, wird die Verbindung zu ebenenübergreifendem Verhalten und nicht-funktionalen Architekturaspekten hergestellt. Diese werden innerhalb der Synthese in eine *ganzheitliche, aspektorientierte Simulation* transformiert, welche mehrere Berechnungsmodelle kombiniert und gleichzeitig eine modulare semantische Trennung der funktionalen und nicht-funktionalen Architekturaspekte gewährleistet. Dazu wurden geeignete *Abbildungs- und Synthesevorschriften* zwischen dem E/E-Architekturmodell und dem resultierenden Simulationsmodell des Zielsimulators abgeleitet. Zur integrierten Simulation dient das quelloffene, heterogene Modellierungs- und Simulationswerkzeug Ptolemy II.

Der schichtenorientierte Ansatz sowie die auf AUTOSAR basierten Schnittstellen tragen nicht nur zur Wiederverwendung von standardisierten Formaten und von bereits bestehenden E/E-Architekturmodellen bei, die einfach durch die integrierte Verhaltensmodellierung verfeinert werden können. In Kombination mit der Modularität und der generisch gehaltenen Softwarearchitektur der Simulationssynthese wird zudem die Portierbarkeit und Übertragbarkeit erleichtert, sowohl im Hinblick auf das zugrunde liegende Metamodell der E/E-Architektur als auch des Zielsimulators.

Die entwickelten Methoden und Konzepte zur ausführbaren Verhaltensmodellierung und Simulationssynthese wurden prototypisch im Werkzeug PREEvision® umgesetzt und integriert – ein in der Automobilindustrie etabliertes E/E-Architekturwerkzeug, welches zuvor nicht in der Lage war, als ausführbare Architekturspezifikation in Form einer Simulation zu fungieren. Ein bedeutender Mehrwert im Vergleich zu existierenden wissenschaftlichen Arbeiten und Entwicklungsumgebungen im Kontext von E/E-Architekturen ist die *durchgängige*

und integrierte Realisierung der modellbasierten Verhaltensspezifikation und Simulationssynthese über alle relevanten Abstraktionsebenen – beginnend auf der logischen Architektur. Werkzeugwechsel bzw. lange Werkzeugketten werden somit gezielt vermieden. Die Integration in ein industriell eingesetztes Werkzeug kann in der Serienentwicklung einen zusätzlichen Mehrwert bieten, insb. bei Verwendung der integrierten, umfassenden Datenbank, welche die verteilte Kollaboration unterstützt.

Im Rahmen der prototypischen Integration wurde nicht nur gezeigt, dass die parallel zu dieser Arbeit in PREEvision® hinzugefügte Verfeinerung von Architekturartefakten via UML-konformen State Charts mithilfe von erweiterten Abbildungsvorschriften mit geringem Aufwand der Simulationssynthese hinzugefügt werden konnten. Die neu eingeführte Verhaltensebene und ebenenübergreifenden Mechanismen erlaubten zudem eine kombinierte Simulation mit dem Actor-orientierten Verhalten. Dazu zählt die Interaktion mit State Chart Beschreibungen der Hardwarearchitektur unter Berücksichtigung nicht-funktionaler Eigenschaften wie zustandsabhängige Strombedarfe oder Aufstart- und Nachlaufverzögerungen.

Der entwickelte Prototyp wurde schließlich anhand mehrerer Anwendungsfälle demonstriert und evaluiert. Insgesamt konnte gezeigt werden, dass eine frühzeitige Simulation der E/E-Architektur über alle relevanten Abstraktionsebenen möglich ist. Verschiedenste Untersuchungen beginnend mit dem funktionalen Verhalten einer ACC Applikation wurden gezeigt. Über die Einbindung detaillierterer Abstraktionsebenen wurden der Einfluss von Netzwerkkommunikation, Ausführungszeit sowie Scheduling von Funktionen auf geteilten/dedizierten Rechenkernen auf die ACC Applikation untersucht. Durch die Berücksichtigung von elektrischen Eigenschaften und zustandsbasiertem Verhalten von ECUs via State Charts, werden zusätzlich ein Betriebsmodi bedingtes Funktionsverhalten sowie Strombedarfe über die Zeit untersucht und somit eine frühzeitige Abschätzung von Energiebedarfen unterstützt. Nicht zuletzt werden die Simulation von detaillierten elektrischen Schaltungen, welche das Verhalten auf logischer Architektur verfeinern, sowie die leitungssatzsensitive Simulation abhängig von den Verlegewegen durch die zugrunde liegende Fahrzeugtopologie ausführlich diskutiert. In diesem Kontext wurde zudem eine *neuartige, aspektorientierte Synthese zur DE-basierten elektrischen Simulation mithilfe einer Kombination aus QSS-Methoden*

und SPICE Netzlisten entwickelt, welche ausgehend von ihrer natürlichen, akausalen Notation im Architekturmodell automatisch die Simulation des logischen Verhalten verfeinern.

Da die verteilte und entkoppelte Entwicklung sowie die Wiederverwendung von Simulationsmodellen eine immer wichtigere Rolle spielen, wurde zudem die *modellbasierte Integration und Simulation von werkzeugunabhängigen FMUs der standardisierten FMI-Schnittstelle* in Kombination mit den nicht-funktionalen Kommunikations- und Ausführungsaspekten anhand der ACC Applikation demonstriert und evaluiert. Durch den mittels Mapping einfachen Austausch des Funktionsverhaltens mit einer FMU unter Beibehaltung der Schnittstellen, wird so nicht nur die Erweiterbarkeit gesteigert, sondern trägt auch zur standardisierten Wiederverwendung von domänenspezifischen Modellen und zur Entkopplung der Entwicklung zwischen Stakeholdern bei.

Ausgehend von den erarbeiteten Methoden und Werkzeuge zur integrierten Simulation, wurde darüber hinaus ein allgemeines Konzept namens *AMiL zur iterativen metrikbasierten Variantenbewertung von E/E-Architekturen* bzgl. statischer und *dynamischer* Metriken gleichzeitig entwickelt sowie ebenfalls prototypisch in PREEvision® umgesetzt und integriert. Anhand einer *variantensensitiven Simulationssynthese* und einer nachfolgend kombinierten Analyse aus quasi-statischer Buslast und dynamischer Kommunikationslatenz funktionaler Wirkketten konnte gezeigt werden, dass eine automatisierte und effiziente Evaluation ganzer Architekturvarianten bei gleichzeitiger Variation des Verhaltens frühzeitig im Entwicklungsprozess als Entscheidungshilfe dient.

Zusammenfassend präsentiert die vorliegende Dissertation ein umfassendes Paket aus neuen Methoden und Werkzeugen zur E/E-Architekturzentrierten, ebenenübergreifenden Verhaltensspezifikation sowie zur ganzheitlichen Simulation und metrikbasierten Evaluation frühzeitig im Entwicklungsprozess.

7.2 Ausblick

Im Wesentlichen sind durch die Dissertation ein grundlegendes Framework und eine Basis für Erweiterungen und Untersuchungen bzgl. zukünftiger E/E-Architekturen geschaffen worden. So wird sie bereits im Rahmen einer nachfol-

genden Promotionskooperation des ITIV und der Vector Informatik GmbH zu Forschungszwecken hinsichtlich der Weiterentwicklung der diskutierten Methoden und Konzepte sowie zur Untersuchung der Skalierbarkeit der integrierten Simulation im industriellen Umfeld eingesetzt.

Da der Fokus dieser Arbeit auf der methodischen Erarbeitung und konzeptionellen Umsetzung eines Prototyps lag, fällt die Modellkomplexität der E/E-Architektur beschränkt aus. Um verlässliche Aussagen hinsichtlich der Skalierung der Simulation bei steigender Modellkomplexität treffen zu können, sind Modelle aus der industriellen Produktreife notwendig. Dies stellt somit einen unmittelbaren Anknüpfungspunkt in der o. g. aufbauenden Industriepromotion dar.

Durch ein konkretes industrielles E/E-Architektur- und Verhaltensmodell können außerdem detaillierte Wechselwirkungen zwischen Fahrzeugfunktionen, unter Berücksichtigung der Architekturaspekte, untersucht werden. Vor diesem Hintergrund stellt zudem die modellbasierte Spezifikation von vorgeschriebenen Fahrszenarien und Testkriterien, z. B. nach UNECE [190], in Kombination mit der E/E-Architektur sowie einer realitätsnahen Umwelt- und Fahrdynamiksimulation eine wichtige Forschungsrichtung dar. Damit ist es möglich ein virtuelles Testfeld zur nachvollziehbaren Verifikation von automatisierten Fahrzeugfunktionen zu entwickeln. Aufbauend auf der vorliegenden Dissertation ist in diesem Kontext bereits eine Folgepublikation entstanden [NBHB20]. Die reproduzierbaren Simulationsergebnisse können später im Entwicklungsprozess zum Abgleich mit realen Fahrszenarien herangezogen werden.

Zur weiteren Steigerung der Wiederverwendung von Simulationsmodellen stellt die Weiterentwicklung in Richtung hybrider Co-Simulation des zukünftigen FMI Standards v3.0 ein interessanter Anknüpfungspunkt dar. In Kombination mit dem neuartigen DCP Protokoll kann bspw. eine verteilte Echtzeitsimulation von FMUs in Form von virtuellen ECU-Prototypen in späteren Entwicklungsphasen untersucht werden.

Im Hinblick auf die weitere Zentralisierung zukünftiger E/E-Architekturen ist die Einbindung und Simulation von Ethernet erstrebenswert. Die Netzwerkkommunikation war in dieser Arbeit jedoch bedingt durch die Bibliothek von Ptolemy II auf die Simulation des CAN-Bus beschränkt. Sobald ein passender Actor als Kommunikationsaspekt zur Verfügung steht, kann eine Ethernet-Simulation durch

die Modularität der Synthese mir geringem Aufwand ergänzt werden – dies gilt auch für alternative Bussysteme. In diesem Zusammenhang ist es auch möglich, weitere detailliertere Architekturaspekte mit in die Synthese einzubinden, wie z. B. ein prioritätsbasiertes Scheduling durch Interpretation der AUTOSAR Adaptive Softwarearchitektur.

Schließlich kann das AMiL Konzept als Ausgangspunkt für globale Optimierungen der E/E-Architektur bzgl. statischer *und* dynamischer Metriken durch entsprechende Algorithmen und Zielfunktionen, welche meist herstellerspezifisch sind, genutzt werden. Als Beispiel ist eine optimale Funktionsverteilung bzgl. des Schedulings, des Ressourcenverbrauchs und der zustandsabhängigen Stromaufnahme von Hardwarekomponenten zu nennen.

A Ergänzende Abbildungen

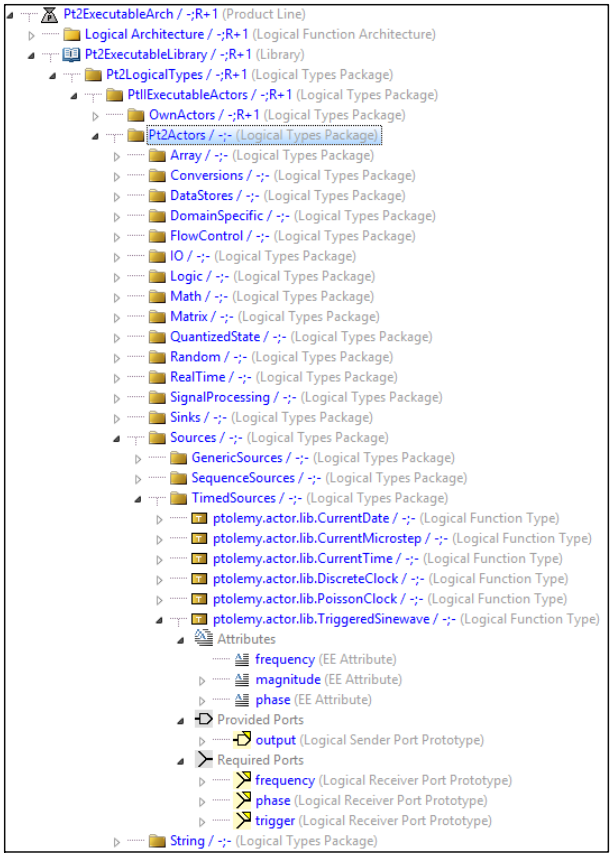


Abbildung A.1: Die durch die Metrik in Abb. 5.17 importierte PtII Actor-Bibliothek gemäß den Abbildungsvorschriften in Tabelle 5.1.

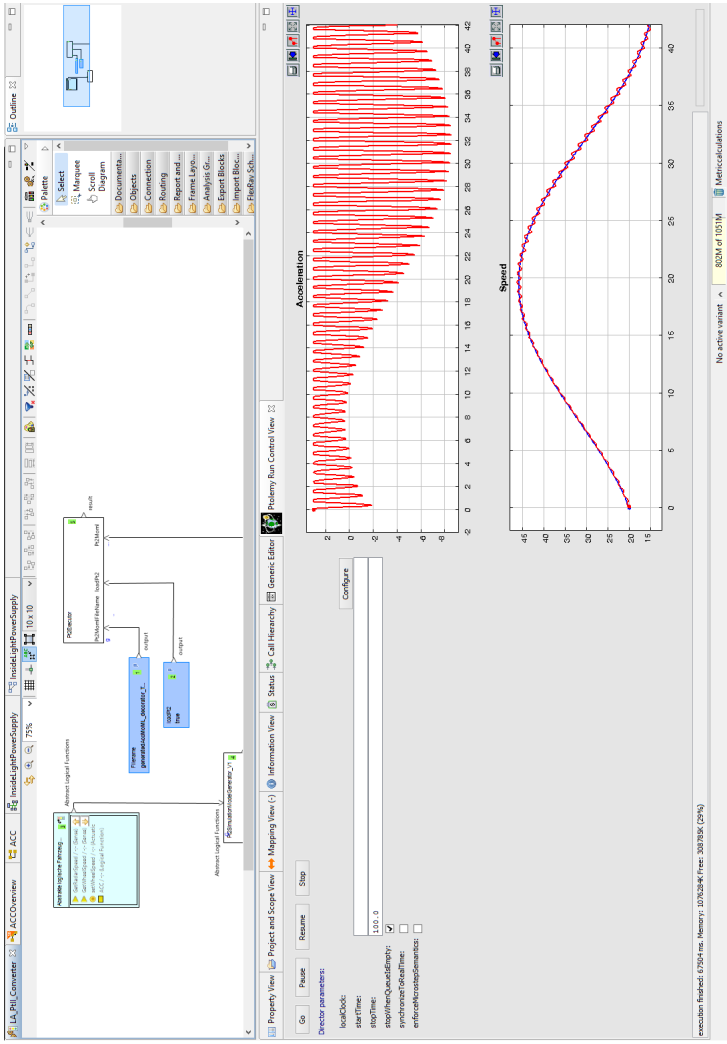


Abbildung A.2: Integrierte Ausführung und Visualisierung der ACC Simulation in der neu hinzugefügten *Ptolemy Run Control* Ansicht innerhalb von PREEvision®.

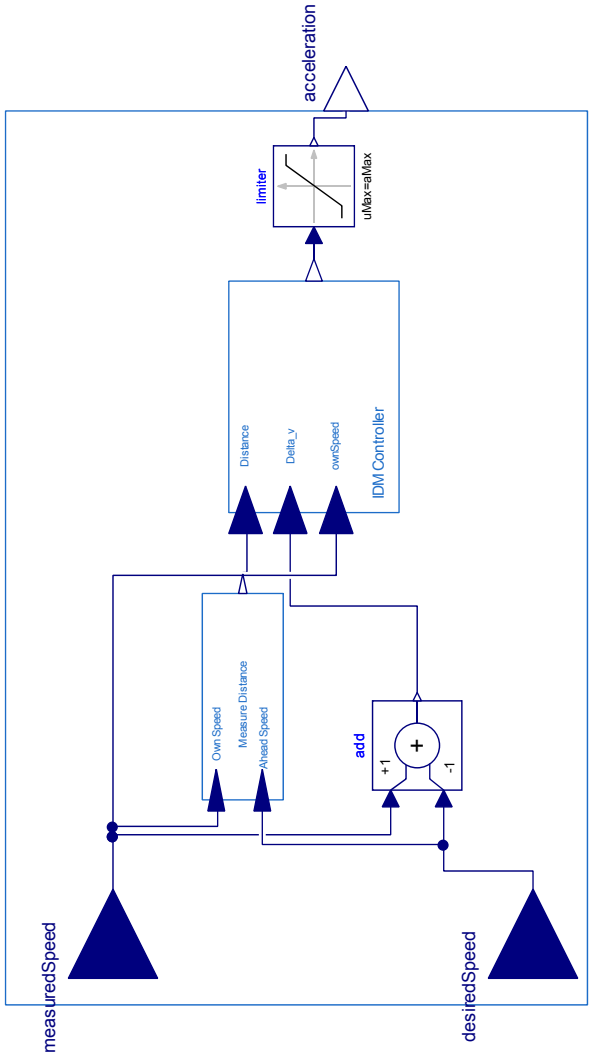


Abbildung A.3: Kontinuierliches Modelica-Modell des ACC Controllers, welches mittels JModelica als FMU-CS exportiert und in PREEvision® als Funktionstyp importiert worden ist. Die Gl. (6.3) des IDM ist als Modelica-Gleichung im Block `IDM Controller` implementiert.

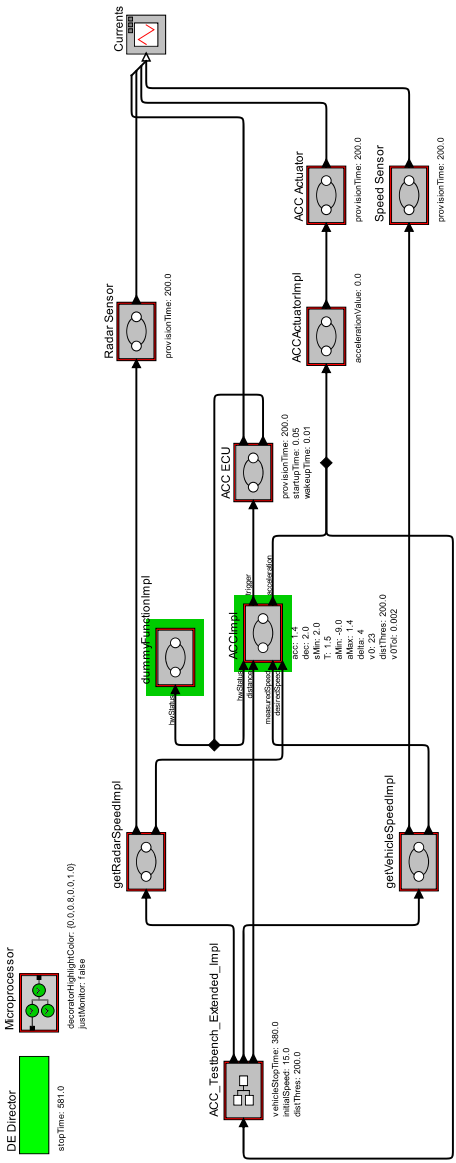


Abbildung A.4: Synthetisiertes PtII Modell der ebenenübergreifenden ACC Applikation mit Multicore-Ausführungsaspekt in Abhängigkeit des Funktions-Mappings.

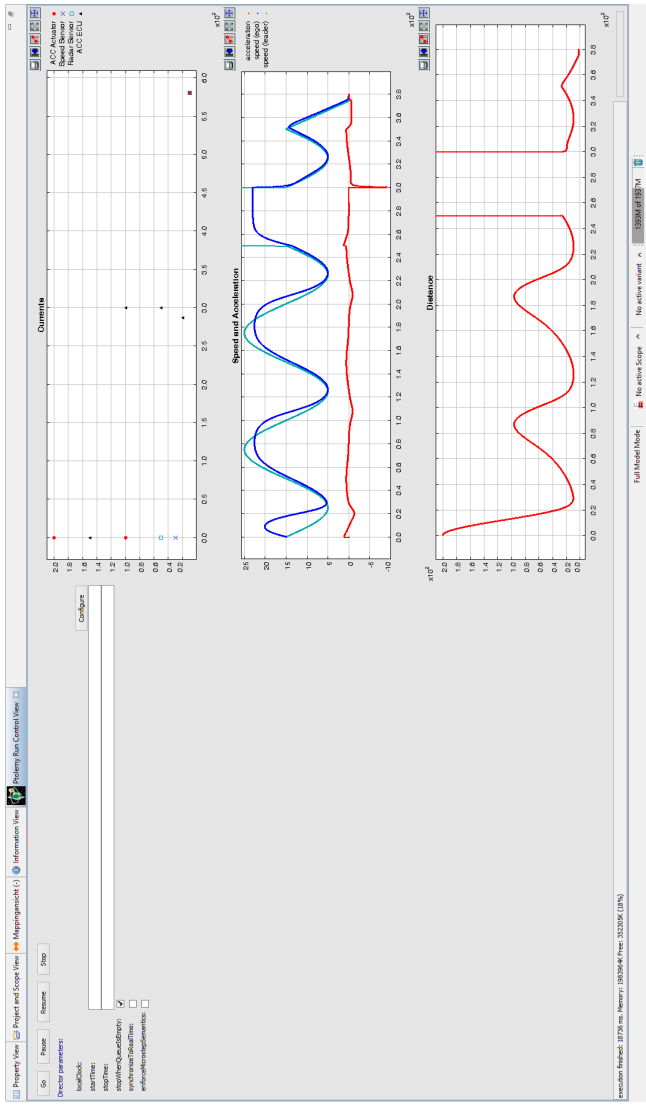


Abbildung A.5: Integrierte Ausführung und Visualisierung der ebenenübergreifenden ACC Simulation inkl. der von den Betriebsmodi abhängigen Stromaufnahme der Hardware in der *Ptolemy Run Control* Ansicht innerhalb von PREEvision®.

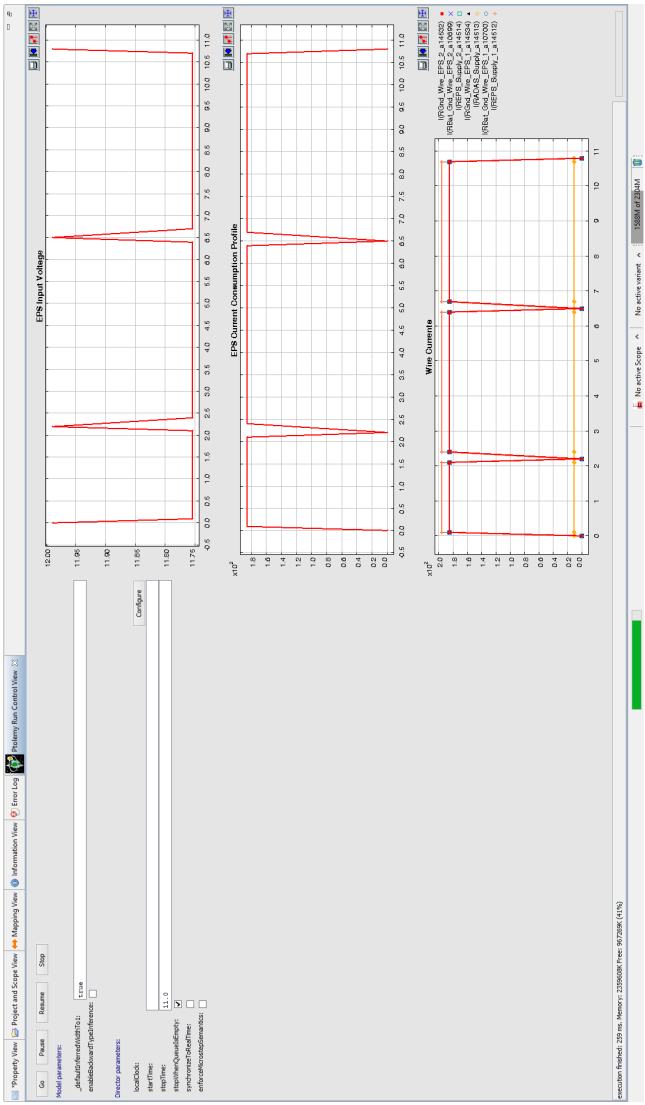


Abbildung A.6: Integrierte Ausführung und Visualisierung der leitungssatzsensitiven EPS-Simulation in der *Ptolemy Run Control* Ansicht innerhalb von PREEvision®. Es zeigt die Spannung am Versorgungspin des EPS Aktuators in Abhängigkeit von dessen Stromprofil sowie die Ströme durch die einzelnen Leitungen.

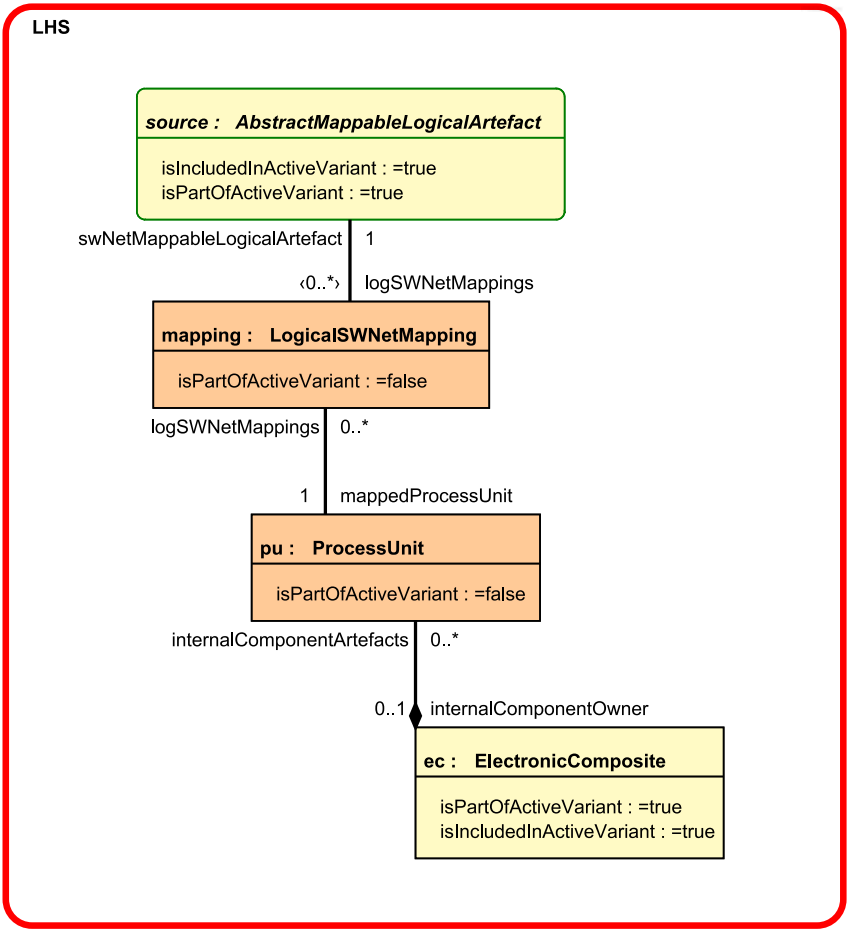


Abbildung A.7: Beispiel einer Modellabfrageregel zur Propagation von Mapping- und Process Unit-Artefakten einer logischen Funktion zu einem Set.

Result Table (160)

Input	Iteration	Mappings	Max. Activity Chain Latency	Activity Chain
CentralizedHighLine/...	1	Buffer -> Central_ECU_2_PU | calculateTargetSpeed -> Central_ECU_1_PU | Comparator -> Central_ECU_1_PU	0.008880000000004884	PPC
CentralizedHighLine/...	1	ACC_Controller -> Central_ECU_1_PU | New_Course_Calculator -> Central_ECU_1_PU | Obstacle_Analyzer -> Central_ECU_2_PU | Positioning -> Central_ECU_2_PU	0.003552000000044405	ACC_NEU
CentralizedHighLine/...	1	Current_Course -> Central_ECU_1_PU | New_Course_Calculator -> Central_ECU_1_PU | Obstacle_Analyzer -> Central_ECU_2_PU | Positioning -> Central_ECU_2_PU	0.008955600000728	UM
CentralizedHighLine/...	1	Feature_Extraction -> Central_ECU_1_PU | Interpretation -> Central_ECU_2_PU | LDW_Active -> Central_ECU_2_PU |	0.006241200000087776	LDW
CentralizedHighLine/...	2	Buffer -> Central_ECU_1_PU | calculateTargetSpeed -> Central_ECU_1_PU | Comparator -> Central_ECU_2_PU |	0.008993440000004925	PPC
CentralizedHighLine/...	2	ACC_Controller -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_2_PU | Obstacle_Analyzer -> Central_ECU_1_PU | Positioning -> Central_ECU_1_PU |	0.003577200000022786	ACC_NEU
CentralizedHighLine/...	2	Current_Course -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_2_PU | Obstacle_Analyzer -> Central_ECU_1_PU | Positioning -> Central_ECU_1_PU |	0.008988080000003342	UM
CentralizedHighLine/...	2	Feature_Extraction -> Central_ECU_1_PU | Interpretation -> Central_ECU_2_PU | LDW_Active -> Central_ECU_2_PU |	0.006241200000087776	LDW
CentralizedHighLine/...	3	Buffer -> Central_ECU_2_PU | calculateTargetSpeed -> Central_ECU_2_PU | Comparator -> Central_ECU_1_PU |	0.008029800000006304	PPC
CentralizedHighLine/...	3	ACC_Controller -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_1_PU | Obstacle_Analyzer -> Central_ECU_2_PU | Positioning -> Central_ECU_2_PU |	0.003577200000022786	ACC_NEU
CentralizedHighLine/...	3	Current_Course -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_1_PU | Obstacle_Analyzer -> Central_ECU_2_PU | Positioning -> Central_ECU_2_PU |	0.00896820000000531	UM
CentralizedHighLine/...	3	Feature_Extraction -> Central_ECU_1_PU | Interpretation -> Central_ECU_2_PU | LDW_Active -> Central_ECU_1_PU |	0.007104000000005328	LDW
CentralizedHighLine/...	4	Buffer -> Central_ECU_2_PU | calculateTargetSpeed -> Central_ECU_1_PU | Comparator -> Central_ECU_1_PU |	0.008880000000004884	PPC
CentralizedHighLine/...	4	ACC_Controller -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_2_PU | Obstacle_Analyzer -> Central_ECU_1_PU | Positioning -> Central_ECU_2_PU |	0.003577200000022786	ACC_NEU
CentralizedHighLine/...	4	Current_Course -> Central_ECU_1_PU | New_Course_Calculator -> Central_ECU_2_PU | Obstacle_Analyzer -> Central_ECU_1_PU | Positioning -> Central_ECU_2_PU |	0.008917800000006682	UM
CentralizedHighLine/...	4	Feature_Extraction -> Central_ECU_1_PU | Interpretation -> Central_ECU_2_PU | LDW_Active -> Central_ECU_2_PU |	0.006253800000006748	LDW
CentralizedHighLine/...	5	Buffer -> Central_ECU_1_PU | calculateTargetSpeed -> Central_ECU_2_PU | Comparator -> Central_ECU_1_PU |	0.007902000000005106	PPC
CentralizedHighLine/...	5	ACC_Controller -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_2_PU | Obstacle_Analyzer -> Central_ECU_1_PU | Positioning -> Central_ECU_2_PU |	0.003577200000022786	ACC_NEU
CentralizedHighLine/...	5	Current_Course -> Central_ECU_1_PU | New_Course_Calculator -> Central_ECU_2_PU | Obstacle_Analyzer -> Central_ECU_1_PU | Positioning -> Central_ECU_2_PU |	0.008917800000006682	UM
CentralizedHighLine/...	5	Feature_Extraction -> Central_ECU_2_PU | Interpretation -> Central_ECU_1_PU | LDW_Active -> Central_ECU_1_PU |	0.00715440000000811	LDW
CentralizedHighLine/...	6	Buffer -> Central_ECU_1_PU | calculateTargetSpeed -> Central_ECU_2_PU | Comparator -> Central_ECU_2_PU |	0.008029800000006304	PPC
CentralizedHighLine/...	6	ACC_Controller -> Central_ECU_1_PU | New_Course_Calculator -> Central_ECU_1_PU | Obstacle_Analyzer -> Central_ECU_1_PU | Positioning -> Central_ECU_1_PU |	0.003552000000044405	ACC_NEU
CentralizedHighLine/...	6	Current_Course -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_1_PU | Obstacle_Analyzer -> Central_ECU_1_PU | Positioning -> Central_ECU_1_PU |	0.008934400000004113	UM
CentralizedHighLine/...	6	Feature_Extraction -> Central_ECU_2_PU | Interpretation -> Central_ECU_1_PU | LDW_Active -> Central_ECU_2_PU |	0.007167000000006141	LDW
CentralizedHighLine/...	7	Buffer -> Central_ECU_1_PU | calculateTargetSpeed -> Central_ECU_1_PU | Comparator -> Central_ECU_2_PU |	0.008029800000006304	PPC
CentralizedHighLine/...	7	ACC_Controller -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_2_PU | Obstacle_Analyzer -> Central_ECU_1_PU | Positioning -> Central_ECU_2_PU |	0.003577200000022786	ACC_NEU
CentralizedHighLine/...	7	Current_Course -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_2_PU | Obstacle_Analyzer -> Central_ECU_1_PU | Positioning -> Central_ECU_2_PU |	0.008905200000000946	UM
CentralizedHighLine/...	7	Feature_Extraction -> Central_ECU_2_PU | Interpretation -> Central_ECU_1_PU | LDW_Active -> Central_ECU_1_PU |	0.007154400000063335	LDW
CentralizedHighLine/...	8	Buffer -> Central_ECU_1_PU | calculateTargetSpeed -> Central_ECU_1_PU | Comparator -> Central_ECU_1_PU |	0.008826000000020915	PPC
CentralizedHighLine/...	8	ACC_Controller -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_2_PU | Obstacle_Analyzer -> Central_ECU_2_PU | Positioning -> Central_ECU_2_PU |	0.003577200000022786	ACC_NEU
CentralizedHighLine/...	8	Current_Course -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_2_PU | Obstacle_Analyzer -> Central_ECU_2_PU | Positioning -> Central_ECU_1_PU |	0.008993440000001373	UM
CentralizedHighLine/...	8	Feature_Extraction -> Central_ECU_1_PU | Interpretation -> Central_ECU_1_PU | LDW_Active -> Central_ECU_2_PU |	0.006241200000001612	LDW
CentralizedHighLine/...	9	Buffer -> Central_ECU_1_PU | calculateTargetSpeed -> Central_ECU_2_PU | Comparator -> Central_ECU_2_PU |	0.008988080000003342	PPC
CentralizedHighLine/...	9	ACC_Controller -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_1_PU | Obstacle_Analyzer -> Central_ECU_1_PU | Positioning -> Central_ECU_2_PU |	0.003577200000022786	ACC_NEU
CentralizedHighLine/...	9	Current_Course -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_1_PU | Obstacle_Analyzer -> Central_ECU_1_PU | Positioning -> Central_ECU_2_PU |	0.008955600000000728	UM
CentralizedHighLine/...	9	Feature_Extraction -> Central_ECU_1_PU | Interpretation -> Central_ECU_2_PU | LDW_Active -> Central_ECU_2_PU |	0.006253800000006748	LDW
CentralizedHighLine/...	10	Buffer -> Central_ECU_2_PU | calculateTargetSpeed -> Central_ECU_2_PU | Comparator -> Central_ECU_2_PU |	0.008029800000006304	PPC
CentralizedHighLine/...	10	ACC_Controller -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_2_PU | Obstacle_Analyzer -> Central_ECU_1_PU | Positioning -> Central_ECU_1_PU |	0.003577200000022786	ACC_NEU
CentralizedHighLine/...	10	Current_Course -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_2_PU | Obstacle_Analyzer -> Central_ECU_2_PU | Positioning -> Central_ECU_1_PU |	0.009006000000002956	UM
CentralizedHighLine/...	10	Feature_Extraction -> Central_ECU_1_PU | Interpretation -> Central_ECU_1_PU | LDW_Active -> Central_ECU_2_PU |	0.007104000000005328	LDW
CentralizedHighLine/...	11	Buffer -> Central_ECU_1_PU | calculateTargetSpeed -> Central_ECU_2_PU | Comparator -> Central_ECU_2_PU |	0.008029800000006304	PPC
CentralizedHighLine/...	11	ACC_Controller -> Central_ECU_2_PU | New_Course_Calculator -> Central_ECU_1_PU | Obstacle_Analyzer -> Central_ECU_2_PU | Positioning -> Central_ECU_2_PU |	0.003577200000022786	ACC_NEU
CentralizedHighLine/...	11	Current_Course -> Central_ECU_1_PU | New_Course_Calculator -> Central_ECU_1_PU | Obstacle_Analyzer -> Central_ECU_2_PU | Positioning -> Central_ECU_2_PU |	0.008955600000000728	UM
CentralizedHighLine/...	11	Feature_Extraction -> Central_ECU_2_PU | Interpretation -> Central_ECU_1_PU | LDW_Active -> Central_ECU_1_PU |	0.00715440000000811	LDW
CentralizedHighLine/...	12	Buffer -> Central_ECU_2_PU | calculateTargetSpeed -> Central_ECU_1_PU | Comparator -> Central_ECU_1_PU |	0.009768000000004962	PPC

Abbildung A.8: Auszug der Ergebnistabelle der AMiL Metrikanalyse. Je Iteration einer Architekturvariante enthält eine Zeile das aktuelle Mapping, die maximale Wirkkettenlatenz und den Namen der Wirkkette.

Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

Abkürzungsverzeichnis

AADL	Architecture Analysis and Design Language
ACC	Adaptive Cruise Control
ACES	Autonomous Driving, Connectivity, Electrification and Smart Mobility
ADAS	Advanced Driver Assistance System
ADL	Architecture Description Language
AMiL	Architecture-Model-in-the-Loop
AMS	Analog/Mixed-Signal
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
AUTOSAR	Automotive Open System Architecture
AWG	American Wire Gauge
BLA	Behavioral Logical Architecture
BSW	Basic Software
CAN	Controller Area Network
CAN FD	CAN Flexible Data-Rate
CPS	Cyber-Physical System
CPU	Central Processing Unit
CT	Continuous-Time
DAE	Differential Algebraic Equation
DAG	Directed Acyclic Graph
DBC	Data Base CAN
DC	Direct Current

DCOP	Direct Current Operating Point
DCP	Distributed Co-Simulation Protocol
DCU	Domain Control Unit
DE	Discrete Event
DE	Datenelement
DEVS	Discrete Event System Specification
DSE	Design Space Exploration
DSL	Domain-Specific Language
DSP	Digital Signal Processor
EAST-ADL	Electronics Architecture and Software Technology - Architecture Description Language
EATOP	EAST-ADL Tool Platform
EC	Electric Circuit
ECU	Electronic Control Unit
EE	Elektrik/Elektronik
EEA	Elektrik/Elektronik-Architektur
EEA-ADL	Electric/Electronic-Architecture - Analysis Design Language
EFSM	Extended Finite State Machine
EP	Environment Perception
EPS	Electric Power Steering
ESP	Electronic Stability Program
FAA	Functional Analysis Architecture
FDA	Functional Design Architecture
FIBEX	Fieldbus Exchange Format
FIFO	First In First Out
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FMU-CS	Functional Mock-up Unit – Co-Simulation
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
FSP	Finite State Process

GRE	Gateway Routing Entry
GUI	Graphical User Interface
HDA	Hardware Design Architecture
HiL	Hardware-in-the-Loop
HLA	High-Level Architecture
IDM	Intelligent Driver Model
KBL	Kabelbaumliste
LA	Logische Architektur
LDF	LIN Description File
LDW	Lane Departure Warning
LHA	Linear Hybrid Automata
LIN	Local Interconnect Network
LIQSS	Linear Implicit Quantized-State System
M&S	Modellierung und Simulation
MAE	Mean Absolute Error
MARTE	Modeling and Analysis of Real-Time and Embedded Systems
MBD	Model-Based Design
MBSE	Model-Based Systems Engineering
MiL	Model-in-the-Loop
MNA	Modified Nodal Analysis
MoC	Model of Computation
MOF	Meta Object Facility
MoML	Modeling Markup Language
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MOST	Media Oriented Systems Transport
NMAE	Normalized Mean Absolute Error

ODE	Ordinary Differential Equation
OEM	Original Equipment Manufacturer
OSGi	Open Service Gateway initiative
PDU	Protocol Data Unit
PiL	Processor-in-the-Loop
PPC	Predictive Powertrain Control
PtII	Ptolemy II
PU	Processing Unit
QSS	Quantized-State System
QSS1	Quantized-State System 1. Ordnung
QSS2	Quantized-State System 2. Ordnung
QSS3	Quantized-State System 3. Ordnung
RTE	Runtime Environment
SDF	Synchronous Data Flow
SDR	Simulation Data Registry
SiL	Software-in-the-Loop
SMT	Satisfiability Modulo Theories
SPES	Software Platform Embedded Systems
SPICE	Software Process Improvement and Capability Determination
SPICE	Simulation Program with Integrated Circuit Emphasis
SR	Synchronous-Reactive
SWC	Software Component
SWT	Standard Widget Toolkit
SysML	Systems Modeling Language
UML	Unified Modeling Language
UNECE	United Nations Economic Commission for Europe
UUID	Universally Unique Identifier

V2X	Vehicle-to-X
VFB	Virtual Function Bus
ViL	Vehicle-in-the-Loop
V-Modell	Vorgehensmodell
VSA	Vehicle Systems Architect
WH	Wiring Harness
XML	eXtensible Markup Language

Glossar

CH(p)	Die Menge aller Kanäle auf einem Pfad $p \in P$ einer Wirkkette.
EV	Die Menge aller gesandten Events eines logischen Senderports.
P	Die Menge aller gültigen Pfade einer Wirkkette.
Abstrakte Semantik	Ein formaler Regelsatz, der die Ausführungskontrolle und Kommunikation von Actors sowie ein Zeitmodell spezifiziert [81].
Abstrakte Syntax	Struktur zur Darstellung eines Modells [136, S. 423].
Actor	Ausführbare, nebenläufige Softwarekomponenten, die über Ports Nachrichten austauschen [136, S. 11].
Analoger Port	Ein logischer Port, der genau ein Datenelement mit einem Applikations-Datentyp ELECTRICAL VOLTAGE besitzt, wird als analoger Port spezifiziert. Ein analoger Port kann einen beliebigen Signalverlauf annehmen.
Atomic Actor	Ein atomarer Actor, der nicht weiter verfeinert werden kann [136, S. 12]

Basisservice Interface An AUTOSAR orientierte Schnittstelle zur Kommunikation von Funktionen mit Diensten der Service Ebene. Dient zur ebenenübergreifenden Interaktion von Verhaltensbeschreibungen einer logischen Funktion und dazugehöriger Hardwarekomponente.

Berechnungsmodell Ein formaler Regelsatz zur eindeutigen Spezifikation der Semantik eines Modells [86].

Building Block Ein zusammengesetzter Block zur Hierarchisierung von logischen Funktionen innerhalb der LA [170, S. 848] bzw. von Actors innerhalb der BLA einer E/E-Architektur.

Composite Actor Ein hierarchischer Actor, der komplette Submodelle enthalten kann [136, S. 12].

Director Eine Implementierung eines Berechnungsmodells in Ptolemy II [136, S. 24].

Dynamische Metrik Eine Metrik, welche aufgrund von dynamischen Vorgängen in Abhängigkeit von Betriebszuständen, Funktionsverhalten oder komponentenspezifischem Betriebsverhalten mithilfe von Simulationsdaten bestimmt wird.

E/E-Architektur Eine E/E-Architektur beschreibt fahrzeugweit alle E/E-Systeme hinsichtlich derer Struktur und Interaktion, d. h. der Schnittstellen, der funktionalen, logischen, elektrischen sowie physikalischen Vernetzung, der Kommunikation, der elektrischen Versorgung sowie der Topologie [41, S. 233].

Gateway Ein Steuergerät, das an mehrere Bussysteme angeschlossen ist und Signal-Repräsentationen zwischen diesen transformiert [170, S. 1009].

Konkrete Semantik	Realisierung eines spezifischen Berechnungsmodells [81].
Konkrete Syntax	Spezifische Notation zur Darstellung einer abstrakten Syntax [136, S. 423].
Mapping	Ein Entwurfsmuster in der Softwareentwicklung, um Artefakte auf unterschiedlichen Abstraktionsebenen miteinander in Beziehung zu setzen [41, S. 35].
Metamodell	Modell einer Modellierungssprache oder abstrakten Syntax [59].
Metrik	Eine Berechnungsvorschrift, mit deren Hilfe die Bewertung einer E/E-Architektur bezüglich vorgegebener Bewertungskriterien in eine oder mehrere Kennzahlen durchgeführt wird [41, S. 98].
Modal Model	Ein hierarchisches FSM Modell, bei dem jeder Zustand ein Submodell einer beliebigen Domäne enthalten kann [89].
Modell	Eine physikalische, mathematische oder anderweitig logische Repräsentation eines Systems, Entität, Phänomens oder Prozesses [94].
Modellierung	Der Prozess zur Erstellung eines Modells [100].
Port	Eine Schnittstelle zum Datenaustausch zwischen Actors oder Funktionsblöcken [136, S. 13] [170, S. 849].
Simulation	Die Ausführung und Experimentierfähigkeit eines Modells über die Zeit [100].
Solver	Realisierung einer numerischen Integrationsmethode zur rechnergestützten Lösung von Differentialgleichungen [88, S. 165].

Statische Metrik	Eine Metrik, welche zu den globalen Kenngrößen gehört [153] und mit rein statischen Attributen einer E/E-Architektur berechnet werden kann.
Token	Ein typisiertes Datum, bspw. ein numerischer Wert, das zwischen Actors via Ports ausgetauscht wird [136, S. 52].
Vergil	Konkrete, visuelle Syntax zur Editierung von Ptolemy II Modellen [136, S. 39].
Wirkkette	Eine Wirkkette (engl. activity chain) repräsentiert einen Teil der gesamten logischen Funktionsarchitektur im Fahrzeug [170, S. 847].

Literatur- und Quellennachweise

[1] ADLER, N.: *Modellbasierte Entwicklung funktional sicherer Hardware nach ISO 26262*, Bd. 10 d. Reihe *Steinbuch Series on Advances in Information Technology*. KIT Scientific Publishing, Karlsruhe, 2015.

[2] ADLER, N., P. GRAF und K. D. MÜLLER-GLASER: *Model-Based Consistency Checks of Electric and Electronic Architectures against Requirements*. In: KIENZLE, J. (Hrsg.): *Models in Software Engineering*, S. 262–275, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[3] AKKAYA, I., P. DERLER, S. EMOTO und E. A. LEE: *Systems Engineering for Industrial Cyber-Physical Systems Using Aspects*. Proceedings of the IEEE, 104(5):997–1012, Mai 2016.

[4] ALAGAR, V. S. und K. PERIYASAMY: *Extended Finite State Machine*. In: *Specification of Software Systems*, S. 105–128. Springer London, London, 2011.

[5] APOSTU, S., O. BURKACKY, J. DEICHMANN, und G. DOLL: *Automotive software and electrical/electronic architecture: Implications for OEMs*. Techn. Ber., McKinsey Center for Future Mobility, Apr. 2019.

[6] ARAVANTINOS, V., S. VOSS, S. TEUFL, F. HÖLZL und B. SCHÄTZ: *AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems*. In: *Joint Proceedings of ACES-MB 2015 - Model-Based Architecting of Cyber-Physical and Embedded Systems and WUCOR 2015 - UML Consistency Rules (2015)*, S. 19–26, 2015.

[7] ATTARZADEH NIAKI, S. und I. SANDER: *Co-simulation of embedded systems in a heterogeneous MoC-based modeling framework*. In: *Industrial Embedded Systems (SIES), 2011 6th IEEE International Symposium on*, S. 238–247, Juni 2011.

[8] AUTOSAR CONSORTIUM: *Automotive Open System Architecture*. [online; https://www.autosar.org/about/, zuletzt aufgerufen am 12.06.2018], 2018.

[9] AUTOSAR CONSORTIUM: *AUTOSAR Classic Platform 4.3.1 Specifications*. [online; https://www.autosar.org/standards/classic-platform/classic-platform-431/, zuletzt aufgerufen am 12.06.2018], 2018.

[10] AZIZ, A. und A. PRAKASH: *Algorithms for Interviews*. Createspace, 1. Aufl., 2010.

[11] BARBIERI, G., P. DERLER, D. M. AUSLANDER, R. BORSARI und C. FANTUZZI: *Design of mechatronic systems through aspect and object-oriented modeling*. at - Automatisierungstechnik, 64(3):244–252, Jan. 2016.

[12] BASTIAN, J., C. CLAUSS, S. WOLF und P. SCHNEIDER: *Master for Co-Simulation Using FMI*. In: *Proceedings of the 8th Modelica Conference*, S. 115–120, 2011.

[13] BERGERO, F. und E. KOFMAN: *PowerDEVS: a tool for hybrid system modeling and real-time simulation*. SIMULATION, 87(1-2):113–132, 2011.

[14] BLOM, H., H. LÖNN, F. HAGL, Y. PAPADOPOULOS, M.-O. REISER, C.-J. SJÖSTEDT, D.-J. CHEN und R. KOLAGARI: *EAST-ADL - An Architecture Description Language for Automotive Software-Intensive Systems*. White Paper Version 2.1.12, EAST-ADL Association, 2013.

[15] BRAUN, J.: *Formelsammlung*. Verlag maxon academy, Sachseln, 2. Aufl., 2012.

[16] BRINGMANN, E. und A. KRÄMER: *Model-Based Testing of Automotive Systems*. In: *2008 1st International Conference on Software Testing, Verification, and Validation*, S. 485–493, Apr. 2008.

[17] BROMAN, D.: *Hybrid Simulation Safety : Limbos and Zero Crossings*. In: *Principles of Modeling : Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday*, Nr. 10760 in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, S. 106–121. 2018.

[18] BROMAN, D., C. BROOKS, L. GREENBERG, E. A. LEE, M. MASIN, S. TRIPAKIS und M. WETTER: *Determinate Composition of FMUs for Co-simulation*. In:

Proceedings of the Eleventh ACM International Conference on Embedded Software, EMSOFT '13, S. 2:1–2:12, Piscataway, NJ, USA, 2013. IEEE Press.

[19] BROMAN, D., L. GREENBERG, E. A. LEE, M. MASIN, S. TRIPAKIS und M. WETTER: *Requirements for Hybrid Cosimulation Standards*. In: *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, HSCC '15, S. 179–188, New York, NY, USA, 2015. ACM.

[20] BROOKS, C., T. H. FENG, E. A. LEE, X. LIU, S. NEUENDORFFER, N. SMYTH und Y. XIONG: *Expressions*. In: PTOLEMAEUS, C. (Hrsg.): *System Design, Modeling, and Simulation using Ptolemy II*, Kap. 13, S. 449–505. Ptolemy.org, 2014.

[21] BROOKS, C., E. A. LEE, X. LIU, S. NEUENDORFFER, Y. ZHAO und H. ZHENG: *Heterogeneous Concurrent Modeling and Design in Java (Volume 3: Ptolemy II Domains)*. Techn. Ber. UCB/EECS-2008-37, EECS Department, University of California, Berkeley, Apr. 2008.

[22] BROY, M., S. KIRSTAN, H. KRCMAR und B. SCHÄTZ: *What is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry?*. In: RECH, J. und C. BUNSE (Hrsg.): *Emerging Technologies for the Evolution and Maintenance of Software Models*, S. 343–369. IGI Global, 2012.

[23] BROY, M. und K. STØLEN: *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer-Verlag, Berlin, Heidelberg, 2001.

[24] BÜCS, R. L., L. MURILLO, E. KOROTCENKO, G. DUGGE, R. LEUPERS, G. ASCHEID, A. ROPERS, M. WEDLER und A. HOFFMANN: *Virtual Hardware-in-the-Loop Co-simulation for Multi-domain Automotive Systems via the Functional Mock-Up Interface*. In: DRECHSLER, R. und R. WILLE (Hrsg.): *Languages, Design Methods, and Tools for Electronic System Design: Selected Contributions from FDL 2015*, S. 3–28. Springer International Publishing, Cham, 2016.

[25] BURKACKY, O., J. DEICHMANN und J. P. STEIN: *Automotive software and electronics 2030*. Techn. Ber., McKinsey Center for Future Mobility, Juli 2019.

[26] BURKACKY, O., J. D. G. DOLL und C. KNOCHENHAUER: *Rethinking Car Software and Electronics Architecture*. Techn. Ber., McKinsey Center for Future Mobility, Feb. 2018.

[27] CELLIER, F. und E. KOFMAN: *Continuous System Simulation*. Springer, 2006.

[28] CHANG HO SUNG, J. H. H. und T. G. KIM: *Interoperation of DEVS Models and Differential Equation Models using HLA/RTI: Hybrid Simulation of Engineering and Engagement Level Models*. März 2009.

[29] CREMONA, F., M. LOHSTROH, D. BROMAN, E. A. LEE, M. MASIN und S. TRIPAKIS: *Hybrid co-simulation: it's about time*. Software & Systems Modeling, 18(3):1655–1679.

[30] DAVARE, A., D. DENSMORE, T. MEYEROWITZ, A. PINTO, A. SANGIOVANNI-VINCENTELLI, G. YANG, H. ZENG und Q. ZHU: *A Next-Generation Design Framework for Platform-based Design*. In: *DVCon 2007*, Feb. 2007.

[31] DERLER, P., E. A. LEE und A. SANGIOVANNI-VINCENTELLI: *Modeling Cyber-Physical Systems*. Proceedings of the IEEE (special issue on CPS), 100(1):13–28, Jan. 2012.

[32] DORI, D.: *Model-Based Systems Engineering with OPM and SysML*. Springer New York Heidelberg Dordrecht London, 2016.

[33] EAST-ADL ASSOCIATION: *EAST-ADL Domain Model Specification V2.1.12*. Techn. Ber., 2013.

[34] EKER, J., J. JANNECK, E. LEE, J. LIU, X. LIU, J. LUDVIG, S. SACHS, Y. XIONG und S. NEUENDORFFER: *Taming heterogeneity - the Ptolemy approach*. Proceedings of the IEEE, 91(1):127–144, 2003.

[35] EKI, H. und S. YONEKI: *Fail-operational EPS by distributed architecture*. In: PFEFFER, P. E. (Hrsg.): *5th International Munich Chassis Symposium 2014: chassis.tech plus*, S. 421–441. Springer Fachmedien Wiesbaden, Wiesbaden, 2014.

[36] FEILER, P. und D. GLUCH: *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley, 2012.

[37] FLOROS, X., F. BERGERO, F. CELLIER und E. KOFMAN: *Automated Simulation of Modelica Models with QSS Methods - The Discontinuous Case -*. In: *8th International Modelica Conference*, S. 657–667, März 2011.

[38] FOWLER, M. und R. PARSONS: *Domain-Specific Languages*. Addison Wesley, 2010.

[39] FRIEDL, M., A. KELLNER und L. WEINGARTNER: *Integration of domain-specific simulation models into descriptive system models by using SysML*. In: *2017 IEEE International Systems Engineering Symposium (ISSE)*, S. 1–5, Okt. 2017.

[40] GEAR, C.: *Simultaneous Numerical Solution of Differential-Algebraic Equations*. IEEE Transactions on Circuit Theory, 18(1):89–95, Jan. 1971.

[41] GEBAUER, D. J.: *Ein modellbasiertes, graphisch notiertes, integriertes Verfahren zur Bewertung und zum Vergleich von Elektrik/Elektronik-Architekturen*. Doktorarbeit, Karlsruher Institut für Technologie (KIT), 2016.

[42] GEHRING, R., J. FROSCHL, T. P. KOHLER und H. G. HERZOG: *Modeling of the automotive 14 V power net for voltage stability analysis*. In: *2009 IEEE Vehicle Power and Propulsion Conference*, S. 71–77, Sep. 2009.

[43] GÜHMANN, C.: *Model-Based Testing of Automotive Electronic Control Units*. In: *Proceedings of the 3rd Interantional Conference on Material Testing: Test 2005*, Nürnberg, Deutschland, Mai 2005.

[44] GIL, L. und M. RADETZKI: *SystemC AMS power electronic modeling with ideal instantaneous switches*. In: *Proceedings of the 2014 Forum on Specification and Design Languages (FDL)*, S. 1–8. IEEE, Okt. 2014.

[45] HALBACH, S., P. SHARER, S. PAGERIT, A. P. ROUSSEAU und C. FOLKERTS: *Model Architecture, Methods, and Interfaces for Efficient Math-Based Design and Simulation of Automotive Control Systems*. In: *SAE Technical Paper 2010-01-0241*. SAE International, Apr. 2010.

[46] HAMDANE, M. E., A. CHAOUI und M. STRECKER: *From AADL to Timed Automaton - A Verification Approach*. In: *International Journal of Software Engineering and Its Applications*, Bd. 7, S. 115–126, Juli 2013.

[47] HAREL, D. und B. RUMPE: *Meaningful modeling: what's the semantics of "semantics"?*. Computer, 37(10):64–72, Okt. 2004.

[48] HEMINGWAY, G., H. NEEMA, H. NINE, J. SZTIPANOVITS und G. KARSAI: *Rapid synthesis of high-level architecture-based heterogeneous simulation: a model-based integration approach*. SIMULATION, 88(2):217–232, 2012.

[49] HENNEBERGER, G.: *Electrical Machines I - Basics, Design, Function, Operation*. Manuskript zur Vorlesung Elektrische Maschinen I, RWTH Aachen, Institut für Elektrische Maschinen, März 2003.

[50] HO, C.-W., A. RUEHLI und P. BRENNAN: *The modified nodal approach to network analysis*. IEEE Transactions on Circuits and Systems, 22(6):504–509, Juni 1975.

[51] IEEE: *IEEE Standard VHDL Language Reference Manual*. IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002), S. c1–626, Jan. 2009.

[52] IEEE: *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)*. IEEE Std 1516.x-2010, Aug. 2010.

[53] IEEE: *IEEE Standard for Standard SystemC Language Reference Manual*. IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005), S. 1–638, Jan. 2012.

[54] IEEE: *IEEE Standard for Standard SystemC Analog/Mixed-Signal Extensions Language Reference Manual*. IEEE Std 1666.1-2016, S. 1–638, Jan. 2016.

[55] IEEE: *IEEE Standard VHDL Analog and Mixed-Signal Extensions*. IEEE Std 1076.1-2017 (Revision of IEEE Std 1076.1-2007), 2017.

[56] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO 26262: Road Vehicles - Functional Safety*. 1. Aufl., Aug. 2012.

[57] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO 11898-1:2015: Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling*. International Organization for Standardization, 2. Aufl., 2015.

[58] KAIJSER, H., H. LÖNN, P. THORNGREN, J. EKBERG, M. HENNINGSSON und M. LARSSON: *Towards Simulation-Based Verification for Continuous Integration and Delivery*. In: *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, Jan. 2018. hal-02156371.

[59] KARSAI, G., A. LANG und S. NEEMA: *Design patterns for open tool integration*. Software & Systems Modeling, 4(2):157–170, Mai 2005.

[60] KARSAI, G., J. SZTIPANOVITS, A. LEDECZI und T. BAPTY: *Model-integrated development of embedded software*. Proceedings of the IEEE, 91(1):145–164, Jan. 2003.

[61] KESTING, A.: *Microscopic Modeling of Human and Automated Driving: Towards Traffic-Adaptive Cruise Control*. Doktorarbeit, Fakultät Verkehrswissenschaften "Friedrich List", Technische Universität Dresden, 2008.

[62] KESTING, A. und M. TREIBER: *How Reaction Time, Update Time, and Adaptation Time Influence the Stability of Traffic Flow*. Computer-Aided Civil and Infrastructure Engineering, 23(2):125–137, 2008.

[63] KESTING, A., M. TREIBER und D. HELBING: *Enhanced Intelligent Driver Model to Access the Impact of Driving Strategies on Traffic Capacity*. Philosophical Transactions of the Royal Society A, 368:4585–4605, 2010.

[64] KESTING, A., M. TREIBER, M. SCHÖHOF und D. HELBING: *Adaptive cruise control design for active congestion avoidance*. Transportation Research Part C: Emerging Technologies, 16(6):668–683, 2008.

[65] KIENCKE, U. und L. NIELSEN: *Automotive Control Systems. For Engine, Driveline, and Vehicle*. Springer Verlag, 2. Aufl., 2005.

[66] KOFMAN, E.: *A Second-Order Approximation for DEVS Simulation of Continuous Systems*. 78(2):76–89, 2002.

[67] KOFMAN, E.: *Quantization-Based Simulation of Differential Algebraic Equation Systems*. SIMULATION, 79(7):363–376, 2003.

[68] KOFMAN, E.: *Discrete Event Simulation of Hybrid Systems*. SIAM Journal on Scientific Computing, 25(5):1771–1797, 2004.

[69] KOFMAN, E.: *A Third Order Discrete Event Simulation Method for Continuous System Simulation*. Latin American Applied Research, 36(2):101–108, 2006.

[70] KOFMAN, E.: *Relative Error Control in Quantization based Integration*. 39(3):231–237, 2009.

[71] KOFMAN, E. und S. JUNCO: *Quantized-state Systems: A DEVS Approach for Continuous System Simulation*. Trans. Soc. Comput. Simul. Int., 18(3):123–132, Sep. 2001.

[72] KRAMMER, M., K. SCHUCH, C. KATER, K. ALEKEISH, T. BLOCHWITZ, S. MATERNE, A. SOPPA und M. BENEDIKT: *Standardized Integration of Real-Time and Non-Real-Time Systems: The Distributed Co-Simulation Protocol*. In: *Proceedings of the 13th International Modelica Conference, 2019*, Bd. 157, S. 87–96, Regensburg, Germany, 2019. Modelica Association.

[73] KUGELE, S. und G. PUCEA: *Model-based Optimization of Automotive E/E-architectures*. In: *Proceedings of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis*, CSTVA 2014, S. 18–29, New York, NY, USA, 2014. ACM.

[74] KUHN, T., T. FORSTER, T. BRAUN und R. GOTZHEIN: *FERAL - Framework for Simulator Coupling on Requirements and Architecture Level*. In: *Proceedings of the 11th ACM/IEEE International Conference on Formal Methods and Models for Codesign*, MEMOCODE '13, S. 11–22, Washington, DC, USA, 2013. IEEE Computer Society.

[75] LARSEN, P., J. FITZGERALD, J. WOODCOCK, P. FRITZSON, J. BRAUER, C. KLEIJN, T. LECOMTE, M. PFEIL, O. GREEN, S. BASAGIANNIS und A. SADOVYKH: *Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project*. In: *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*, S. 1–6. IEEE, 2016.

[76] LARSEN, P., J. FITZGERALD, J. WOODCOCK, C. GAMBLE, R. PAYNE und K. PIERCE: *Features of Integrated Model-based Co-modelling and Co-simulation Technology*. In: *1st Workshop on Formal Co-Simulation of Cyber-Physical Systems*, Sep. 2017.

[77] LARSEN, P., C. THULE, K. LAUSDAHL, V. BANDUR, C. GAMBLE, E. BROSSE, A. SADOVYKH, A. BAGNATO und L. COUTO: *Integrated Tool Chain for Model-Based Design of Cyber-Physical Systems*. In: LARSEN, P., N. PLAT und N. BATTLE (Hrsg.): *The 14th Overture Workshop: Towards Analytical Tool Chains*, Bd. 4/28, S. 63–79. Aarhus University, Department of Engineering, Nov. 2016.

[78] LASNIER, G., J. CARDOSO, P. SIRON, C. PAGETTI und P. DERLER: *Distributed Simulation of Heterogeneous and Real-Time Systems*. In: *2013 17th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, S. 55–62, Okt. 2013.

[79] LASNIER, G., L. PAUTET, J. HUGUES und L. WRAGE: *An Implementation of the Behavior Annex in the AADL-Toolset Osate2*. In: *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, S. 332–337, Apr. 2011.

[80] LEE, E. und A. SANGIOVANNI-VINCENTELLI: *Component-based design for the future*. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, S. 1–5, März 2011.

[81] LEE, E. A.: *Heterogeneous Actor Modeling*. In: *Proceedings of the 9th ACM International Conference on Embedded Software*, EMSOFT '11, S. 3–12, New York, NY, USA, 2011. ACM.

[82] LEE, E. A.: *The Past, Present and Future of Cyber-Physical Systems: A Focus on Models*. Sensors, 15(3):4837–4869, 2015.

[83] LEE, E. A., X. LIU und S. NEUENDORFFER: *Classes and Inheritance in Actor-oriented Design*. ACM Trans. Embed. Comput. Syst., 8(4):29:1–29:26, Juli 2009.

[84] LEE, E. A. und D. G. MESSERSCHMITT: *Synchronous data flow*. Proceedings of the IEEE, 75(9):1235–1245, Sep. 1987.

[85] LEE, E. A. und S. NEUENDORFFER: *MoML - A Modeling Markup Language in XML Version 0.4*. Techn. Ber. UCB/ERL M00/12, University of California Berkeley, 2000.

[86] LEE, E. A., S. NEUENDORFFER und M. J. WIRTHLIN: *Actor-Oriented Design Of Embedded Hardware And Software Systems*. Journal of Circuits, Systems, and Computers, 12(3):231–260, 2003.

[87] LEE, E. A., M. NIKNAMI, T. S. NOUIDUI und M. WETTER: *Modeling and simulating cyber-physical systems using CyPhySim*. In: *2015 International Conference on Embedded Software (EMSOFT)*, S. 115–124, Okt. 2015.

[88] LEE, E. A. und S. A. SESHIA: *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*. MIT Press, 2. Aufl., 2017.

[89] LEE, E. A. und S. TRIPAKIS: *Modal Models in Ptolemy*. In: *Proceedings of the 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, S. 11–21. Linköping University Electronic Press, Okt. 2010.

[90] LEE, E. A. und H. ZHENG: *Leveraging Synchronous Language Principles for Heterogeneous Modeling and Design of Embedded Systems*. In: *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software*, EMSOFT '07, S. 114–123, New York, NY, USA, 2007. ACM.

[91] LINEAR TECHNOLOGY: *LTSpice XVII*. [online; http://www.linear.com/designtools/software/#LTspice, zuletzt aufgerufen am 04.01.2018], 2017.

[92] LITOVSKI, V., M. SAVIC und Z. MRCARICA: *Ideal switch model cuts simulation time*. IEEE Circuits and Devices Magazine, 22(4):16–22, Juli 2006.

[93] LIU, J., B. WU, X. LIU und E. A. LEE: *Interoperation of Heterogeneous CAD Tools in Ptolemy II*. 2(1):1–10, 2001.

[94] LOPER, M. L. und A. REGISTER: *Introduction to Modeling and Simulation*. In: LOPER, M. L. (Hrsg.): *Modeling and Simulation in the Systems Engineering Life Cycle: Core Concepts and Accompanying Lectures*, S. 3–16. Springer London, London, 2015.

[95] MA, Y., J. P. TALPIN und T. GAUTIER: *Interpretation of AADL Behavior Annex into Synchronous Formalism Using SSA*. In: *2010 10th IEEE International Conference on Computer and Information Technology*, S. 2361–2366, Juni 2010.

[96] MAENAD CONSORTIUM: *Analysis and Synthesis Concepts Supporting Engineering Scenarios*. Deliverable D3.2.1 V4.0, 2014.

[97] MAENAD CONSORTIUM: *Analysis Workbench*. Deliverable D5.2.1 V4.0, 2014.

[98] MANN, S.: *Anwendung von Kohäsions- und Kohärenzmetriken auf VEIA-Architekturmodell zur Bewertung von Produktlinien*. Techn. Ber., Fraunhofer Institut für Software- und Systemtechnik, 2008.

[99] MANNA, Z. und A. PNUELI: *Verifying hybrid systems*. In: GROSSMAN, R. L., A. NERODE, A. P. RAVN und H. RISCHEL (Hrsg.): *Hybrid Systems*, S. 4–35, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[100] MARIA, A.: *Introduction to Modeling and Simulation*. In: *Proceedings of the 29th Conference on Winter Simulation*, WSC '97, S. 7–13, Washington, DC, USA, 1997. IEEE Computer Society.

[101] MARINESCU, R., H. KAIJSER, M. MIKUCIONIS, C. SECELEANU, H. LÖNN und A. DAVID: *Analyzing Industrial Architectural Models by Simulation and Model-Checking*. In: ARTHO, C. und C. P. ÖLVECZKY (Hrsg.): *Formal Techniques for Safety-Critical Systems: Third International Workshop, FTSCS 2014, Luxembourg, November 6-7, 2014. Revised Selected Papers*, S. 189–205. Springer International Publishing, Cham, 2015.

[102] MATHEIS, J.: *Abstraktionsebenenübergreifende Darstellung von Elektrik/Elektronik-Architekturen in Kraftfahrzeugen zur Ableitung von Sicherheitszielen nach ISO 26262*. Doktorarbeit, Universität Karlsruhe (TH), 2010.

[103] MATHWORKS: *Simscape Power Systems*. [online; https://www.mathworks.com/products/simpower.html, zuletzt aufgerufen am 04.01.2018], 2018.

[104] MATHWORKS: *Simulink*. [online; http://www.mathworks.com/products/simulink/, zuletzt aufgerufen am 31.03.2018], 2018.

[105] MAXON MOTOR AG: *RE-max 17 Part Number 214897 Specifications*. [online; https://www.maxongroup.com/maxon/view/product/214897, zuletzt aufgerufen am 03.02.2020], 2020.

[106] MENDOZA CERVANTES, F.: *A Problem-Oriented Approach for Dynamic Verification of Heterogeneous Embedded Systems*. Doktorarbeit, Karlsruher Institut für Technologie (KIT), 2013.

[107] MENTOR GRAPHICS: *Capital Systems - Functional capture & multi domain implementation*. [online; http://s3.mentor.com/public_documents/datasheet/products/electrical-design-software/capital-systems.pdf, zuletzt aufgerufen am 19.06.2018], 2018. Datenblatt.

[108] MENTOR GRAPHICS: *Volcano™ Vehicle Systems Architect*. [online; http://s3.mentor.com/public_documents/datasheet/products/vnd/autosar-products/volcano-system-architect/VSA_Datasheet.pdf, zuletzt aufgerufen am 19.06.2018], 2018. Datenblatt.

[109] MEYER, J., J. HOLTMANN und M. MEYER: *Formalisierung von Anforderungen und Betriebssystemeigenschaften zur frühzeitigen Simulation von eingebetteten,*

automobilen Systemen. In: GAUSEMEIER, J., F. RAMMIG, W. SCHÄFER und A. TRÄCHTLER (Hrsg.): *8. Paderborner Workshop Entwurf mechatronischer Systeme*, S. 203–215, 2011.

[110] MICHAILIDIS, A., T. RINGLER, B. HEDENETZ und S. KOWALEWSKI: *Virtuelle Integration modellbasierter Fahrzeugfunktionen unter AUTOSAR*. ATZelektronik, 5(1):32–37, Feb. 2010.

[111] MIGONI, G., M. BORTOLOTTO, E. KOFMAN und F. E. CELLIER: *Linearly implicit quantization-based integration methods for stiff ordinary differential equations*. Simulation Modelling Practice and Theory, 35(Supplement C):118–136, 2013.

[112] MIGONI, G., E. KOFMAN, F. BERGERO und J. FERNÁNDEZ: *Quantization-based simulation of switched mode power supplies*. SIMULATION, 91(4):320–336, 2015.

[113] MIGONI, G., E. KOFMAN und F. CELLIER: *Quantization-based new integration methods for stiff ordinary differential equations*. Simulation: Transactions of the Society for Modeling and Simulation International, 88(4):387–407, 2011.

[114] MODELICA ASSOCIATION: *Modelica® - Unified Object-Oriented Language for Systems Modeling Language Specification Version 3.4*. [online; https://www.modelica.org/documents/ModelicaSpec34.pdf, zuletzt aufgerufen am 31.03.2018], 2017.

[115] MODELICA ASSOCIATION PROJECT DCP: *DCP Specification Document Version 1.0*. Modelica Association, 2019.

[116] MODELICA ASSOCIATION PROJECT FMI: *Functional Mock-up Interface for Model Exchange and Co-Simulation Version 2.0*. Modelica Association, 2014.

[117] MODELON AB: *JModelica.org User Guide Version 2.2*. [online; https://jmodelica.org/downloads/UsersGuide.pdf, zuletzt aufgerufen am 27.09.2019], 2018.

[118] MOLTER, T.: *JSpice*. [online; https://github.com/knowm/jspice, zuletzt aufgerufen am 02.01.2018], 2017.

[119] MUTTER, A.: *CAN FD and the CRC issue*. CAN Newsletter, 1:4–10, März 2015.

[120] NAGEL, L. und D. PEDERSON: *SPICE (Simulation Program with Integrated Circuit Emphasis)*. Techn. Ber. UCB/ERL M382, EECS Department, University of California, Berkeley, Apr. 1973.

[121] NEGHINA, M., C.-B. ZAMFIRESCU, P. LARSEN, K. LAUSDAHL und K. PIERCE: *A Discrete Event-first Approach to Collaborative Modelling of Cyber-Physical Systems*. In: FITZGERALD, J., P. TRAN-JØRGENSEN und T. ODA (Hrsg.): *The 15th Overture Workshop: New Capabilities and Applications for Model-based Systems Engineering*, S. 116–129. Newcastle University, Sep. 2017.

[122] NICKEL, U., J. MEYER und T. KRAMER: *Wie hoch ist die Performance?*. Automobil-Elektronik, 3:36–38, Juni 2010.

[123] OBERGFELL, P., S. KUGELE und E. SAX: *Model-Based Resource Analysis and Synthesis of Service-Oriented Automotive Software Architectures*. In: *Proceedings of the IEEE/ACM 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2019.

[124] OBJECT MANAGEMENT GROUP (OMG): *UML Profile for MARTE: Modeling and Analysis of Real-time and Embedded Systems Vesion 1.1*. [online; http://www.omg.org/spec/MARTE/1.1/PDF, zuletzt aufgerufen am 05.12.2017], 2011. Dokument Nr.: formal/2011-06-02.

[125] OBJECT MANAGEMENT GROUP (OMG): *Meta Object Facility Core Specification Version 2.5.1*. [online; https://www.omg.org/spec/MOF/2.5.1/PDF, zuletzt aufgerufen am 10.04.2018], 2015. Dokument Nr.: formal/16-11-01.

[126] OBJECT MANAGEMENT GROUP (OMG): *Systems Modeling Language SysML® Version 1.4*. [online; http://www.omg.org/spec/SysML/1.4/PDF, zuletzt aufgerufen am 05.12.2017], 2015. Dokument Nr.: formal/2015-06-03.

[127] OBJECT MANAGEMENT GROUP (OMG): *Unified Modelling Language Version 2.5*. [online; http://www.omg.org/spec/UML/2.5/PDF, zuletzt aufgerufen am 05.12.2017], 2015. Dokument Nr.: formal/2015-03-01.

[128] OBJECT MANAGEMENT GROUP (OMG): *Meta-Modeling and the OMG Meta Object Facility (MOF)*. [online; https://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf, zuletzt aufgerufen am 10.04.2018], März 2017. White Paper.

[129] OSGi Alliance: *Listeners Considered Harmful: The "Whiteboard" Pattern*. [online; https://www.osgi.org/wp-content/uploads/whiteboard1.pdf, zuletzt aufgerufen am 09.01.2019], Aug. 2004. Technical Whitepaper Revision 2.0.

[130] OSGi Alliance: *The Dynamic Module System for Java*. [online; https://www.osgi.org, zuletzt aufgerufen am 21.12.2018], 2018.

[131] Otten, S., J. Bach, C. Wohlfahrt, C. King, J. Lier, H. Schmid, S. Schmerler und E. Sax: *Automated Assessment and Evaluation of Digital Test Drives*. In: Zachäus, C., B. Müller und G. Meyer (Hrsg.): *Advanced Microsystems for Automotive Applications 2017*, S. 189–199. Springer International Publishing, Cham, 2018.

[132] Pohl, K., M. Broy, H. Daembkes und H. Hönninger (Hrsg.): *Advanced Model-Based Engineering of Embedded Systems*. Springer International Publishing, 1. Aufl., 2016.

[133] Pohl, K., H. Hönninger, R. Achatz und M. Broy (Hrsg.): *Model-Based Engineering of Embedded Systems*. Springer-Verlag Berlin Heidelberg, 1. Aufl., 2012.

[134] Prawitz, S.: *Homologation - ADAS: Assistenzsysteme digital homologieren*. [online; https://www.automobil-industrie.vogel.de/adas-assistenzsysteme-digital-homologieren-a-842598/pdf/, zuletzt aufgerufen am 18.11.2019], Juli 2019. Automobil Industrie.

[135] Prawitz, S.: *Simulation - CAE bei VW: „2025 wird jeder Ingenieur virtuelle Berechnungen durchführen"*. [online; https://www.automobil-industrie.vogel.de/cae-bei-vw-2025-wird-jeder-ingenieur-virtuelle-berechnungen-durchfuehren-a-883681/?cmp=nl-99&uuid=C30013F6-3ED4-427C-BBA8-5F3C711C13C8, zuletzt aufgerufen am 18.11.2019], Nov. 2019. Automobil Industrie.

[136] Ptolemaeus, C. (Hrsg.): *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.

[137] Rajhans, A., A. Bhave, I. Ruchkin, B. H. Krogh, D. Garlan, A. Platzer und B. Schmerl: *Supporting Heterogeneity in Cyber-Physical Systems Architectures*. IEEE Transactions on Automatic Control, 59(12):3178–3193, Dez. 2014.

[138] RAJHANS, A., S.-W. CHEN, B. SCHMERL, D. GARLAN, B. H. KROGH, C. AGBI und A. BHAVE: *An Architectural Approach to the Design and Analysis of Cyber-Physical Systems*. Electronic Communications of the EASST, 21:10, 2009.

[139] RASKIN, J.-F.: *An Introduction to Hybrid Automata*. In: HRISTU-VARSAKELIS, D. und W. S. LEVINE (Hrsg.): *Handbook of Networked and Embedded Control Systems*, S. 491–517. Birkhäuser Boston, Boston, MA, 2005.

[140] REEDY, J. und S. LUNZMAN: *Model Based Design Accelerates the Development of Mechanical Locomotive Controls*. In: *SAE 2010 Commercial Vehicle Engineering Congress*. SAE International, Okt. 2010.

[141] REICHMANN, C.: *Grafisch notierte Modell-zu-Modell-Transformationen für den Entwurf eingebetteter elektronischer Systeme*. Doktorarbeit, Universtität Karlsruhe (TH), 2005.

[142] ROTH, C.: *Parallele und kooperative Simulation für eingebettete Multiprozessorsysteme*. Doktorarbeit, Karlsruher Institut für Technologie (KIT), 2015.

[143] SAE INTERNATIONAL: *SAE Standard for Dedicated Short Range Communications (DSRC) Message Set Dictionary*. Nov. 2009. Standard J2735.

[144] SAE INTERNATIONAL: *SAE Architecture Analysis and Design Language (AADL) Annex Volume 1: Annex A: Graphical AADL Notation, Annex C: AADL Meta-Model and Interchange Formats, Annex D: Language Compliance and Application Program Interface, Annex E: Error Model Annex*, Apr. 2011. Standard AS5506/1.

[145] SAE INTERNATIONAL: *Architecture Analysis & Design Language (AADL)*, Sep. 2012. Standard AS5506B.

[146] SAE INTERNATIONAL: *SAE Architecture Analysis and Design Language (AADL) Annex Volume 1: Annex A: ARINC653 Annex, Annex C: Code Generation Annex, Annex E: Error Model Annex*, Sep. 2015. Standard AS5506/1A.

[147] SAE INTERNATIONAL: *Architecture Analysis & Design Language (AADL)*, Jan. 2017. Standard AS5506C.

[148] SAE INTERNATIONAL: *SAE Architecture Analysis and Design Language (AADL) Annex Volume 2: Annex B: Data Modeling Annex, Annex D: Behavior Model Annex, Annex F: ARINC653 Annex*, Feb. 2019. Standard AS5506/2.

[149] SANDER, I. und A. JANTSCH: *System modeling and transformational design refinement in ForSyDe [formal system design]*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 23:17–32, 2004.

[150] SANGIOVANNI-VINCENTELLI, A. und M. DI NATALE: *Embedded System Design for Automotive Applications*. Computer, 40(10):42–51, 2007.

[151] SAX, E.: *Automatisiertes Testen Eingebetteter Systeme in der Automobilindustrie*. Hanser Verlag, München, 2008.

[152] SCHIERZ, T., M. ARNOLD und C. CLAUSS: *Co-simulation with Communication Step Size Control in an FMI Compatible Master Algorithm*. In: *Proceedings of the 9th Modelica Conference*, S. 205–214, 2012.

[153] SCHÄUFFELE, J.: *E/E Architectural Design and Optimization using PREEvision*. In: *SAE Technical Paper 2016-01-0016*. SAE International, Apr. 2016.

[154] SCHÄUFFELE, J. UND ZURAWKA, T.: *Automotive Software Engineering*. Springer Vieweg, Wiesbaden, 6. Aufl., 2016.

[155] SHOKRY, H. und M. HINCHEY: *Model-Based Verification of Embedded Software*. Computer, 42(4):53–59, Apr. 2009.

[156] SPEEDS CONSORTIUM: *The SPEEDS Project*. [online; https://web.archive.org/web/20160110053702/http://www.speeds.eu.com/, zuletzt aufgerufen am 03.07.2018], 2008.

[157] SPORER, H., G. MACHER, E. ARMENGAUD und C. KREINER: *Incorporation of Model-Based System and Software Development Environments*. In: *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, S. 177–180, Aug. 2015.

[158] STOLZ, W., R. KORNHAAS, R. KRAUSE und T. SOMMER: *Domain Control Units - the Solution for Future E/E Architectures?*. In: *SAE Technical Paper 2010-01-0686*. SAE International, Apr. 2010.

[159] STREICHERT, T. UND TRAUB, M.: *Elektrik/Elektronik-Architekturen im Kraftfahrzeug*. Springer, Berlin, Heidelberg, 2012.

[160] SUNG, C. und T. G. KIM: *Framework for Simulation of Hybrid Systems: Interoperation of Discrete Event and Continuous Simulators Using HLA/RTI*. In: *Principles of Advanced and Distributed Simulation (PADS), 2011 IEEE Workshop on*, S. 1–8. Juni 2011.

[161] SZTIPANOVITS, J. und G. KARSAI: *Model-integrated computing*. Computer, 30(4):110–111, Apr. 1997.

[162] TOLVANEN, J.-P. und S. KELLY: *Defining Domain-specific Modeling Languages to Automate Product Derivation: Collected Experiences*. In: *Proceedings of the 9th International Conference on Software Product Lines*, SPLC'05, S. 198–209, Berlin, Heidelberg, 2005. Springer-Verlag.

[163] TRAUB, M., A. MAIER und K. L. BARBEHÖN: *Future Automotive Architecture and the Impact of IT Trends*. IEEE Softw., 34(3):27–32, Mai 2017.

[164] TREIBER, M.: *Microsimulation of Road Traffic Flow*. [online; http://www.traffic-simulation.de/, zuletzt aufgerufen am 15.07.2019].

[165] TREIBER, M., A. HENNECKE und D. HELBING: *Congested traffic states in empirical observations and microscopic simulations*. Phys. Rev. E, 62:1805–1824, Aug. 2000.

[166] TREIBER, M. und A. KESTING: *Verkehrsdynamik und -simulation*. Springer-Verlag Berlin Heidelberg, 1. Aufl., 2010.

[167] UNIVERSITÄT LEIPZIG: *Domänenspezifische Sprachen*. IT-Radar, 2, [online; http://it-radar.eu/serendipity/uploads/reports/2_2007.pdf, zuletzt aufgerufen am 18.06.2018], Okt. 2007.

[168] VANSINA, N., B. LOYER und Y. OGATA: *Managing Heterogeneous Simulations Using Architecture-Driven Design*. In: *Proceedings of the 2nd Japanese Modelica Conference*, Mai 2018.

[169] VARGA, A. und R. HORNIG: *An overview of the OMNeT++ simulation environment*. In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, S. 60:1–60:10, ICST, Brussels, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[170] VECTOR INFORMATIK GMBH: *PREEvision® Manual Version 8.5*, 2017.

[171] VECTOR INFORMATIK GMBH: *PREEvision® Metamodell Version 8.5*, 2017.

[172] VECTOR INFORMATIK GMBH: *Lernmodul AUTOSAR: AUTOSAR Konzept*. [online; https://elearning.vector.com/mod/page/view.php?id=289, zuletzt aufgerufen am 09.01.2019], 2018.

[173] VECTOR INFORMATIK GMBH: *Lernmodul AUTOSAR: AUTOSAR Schichtenmodell*. [online; https://elearning.vector.com/mod/page/view.php?id=290, zuletzt aufgerufen am 09.01.2019], 2018.

[174] VECTOR INFORMATIK GMBH: *Lernmodul AUTOSAR: Schnittstellendefinitionen in AUTOSAR*. [online; https://elearning.vector.com/mod/page/view.php?id=291, zuletzt aufgerufen am 09.01.2019], 2018.

[175] VECTOR INFORMATIK GMBH: *PREEvision® Manual Version 9.0*, 2018.

[176] VECTOR INFORMATIK GMBH: *PREEvision® Metamodell Version 9.0 Alpha*, 2018.

[177] VECTOR INFORMATIK GMBH: *AUTOSAR Adaptive Platform*. [online; https://www.vector.com/de/de/know-how/technologien/autosar/autosar-adaptive/, zuletzt aufgerufen am 09.01.2019], 2019.

[178] VLACH, J. und K. SINGHAL: *Computer Methods for Circuit Analysis and Design*. Springer, 2. Aufl., 1993.

[179] VOIGT, K.-I.: *Industrielles Management - Industriebetriebslehre aus prozessorientierter Sicht*. Springer Verlag, 2008.

[180] WACHENFELD, W. und H. WINNER: *Die Freigabe des autonomen Fahrens*. In: MAURER, M., J. C. GERDES, B. LENZ und H. WINNER (Hrsg.): *Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte*, S. 439–464. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[181] WALLA, G., D. GABRIEL, A. BARTHELS, F. RUF, H. U. MICHEL und A. HERKERSDORF: *ITE-Sim: A simulator and power evaluation framework for electric/electronic architectures*. In: *2012 IEEE Vehicle Power and Propulsion Conference*, S. 869–874, Okt. 2012.

[182] WAN, J., A. CANEDO und M. A. AL FARUQUE: *Functional Model-Based Design Methodology for Automotive Cyber-Physical Systems*. IEEE Systems Journal, 11(4):2028–2039, Dez. 2017.

[183] Wan, J., N. Rashid, A. Canedo und M. A. A. Faruque: *Concept Design: Modeling and Synthesis from Requirements to Functional Models and Simulation*. In: Al Faruque, M. A. und A. Canedo (Hrsg.): *Design Automation of Cyber-Physical Systems*, S. 3–20. Springer International Publishing, Cham, 2019.

[184] Waszecki, P., M. Lukasiewycz, A. Masrur und S. Chakraborty: *How to Engineer Tool-chains for Automotive E/E Architectures?*. SIGBED Rev., 10(4):6–15, Dez. 2013.

[185] Weiss, G., M. Zeller, D. Eilers und R. Knorr: *Approach for Iterative Validation of Automotive Embedded Systems*. In: *Models 2010 ACES-MB Workshop Proceedings*, S. 69–83, 2010.

[186] Weissnegger, R., M. Schuss, C. Kreiner, M. Pistauer, K. Römer und C. Steger: *Simulation-based Verification of Automotive Safety-critical Systems Based on EAST-ADL*. Procedia Computer Science, 83:245–252, 2016. The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016) / The 6th International Conference on Sustainable Energy Information Technology (SEIT-2016) / Affiliated Workshops.

[187] Wekerle, T. und C. Blume: *Smart Systems Engineering (SmartSE) - uniformed approach for exchange of functional and behavior models for simulation using Functional Mockup Interface (FMI)*. In: *International Symposium On Tools And Methods of Competitive Engineering*, Mai 2018.

[188] Winner, H., S. Hakuli, F. Lotz und C. Singer (Hrsg.): *Handbuch Fahrerassistenzsysteme*. 3. Aufl., 2015.

[189] Wood, M., C. Knobel, N. Garbacik, D. Wittmann, M. O'Brien, S. Syguda, U. Dannebaum, S. Liu, J. Weast, T. Wiltschko, B. Dornieden et al.: *Safety First for Automated Driving (SaFAD)*. [online; https://www.aptiv.com/docs/default-source/white-papers/safety-first-for-automated-driving-aptiv-white-paper.pdf, zuletzt aufgerufen am 09.12.2019], Juli 2019. White Paper.

[190] Working Party on Automated/Autonomous and Connected Vehicles (GRVA): *Proposal for the Future Certification of Automated/Autonomous*

Driving Systems. Techn. Ber. ECE/TRANS/WP.29/GRVA/2019/13, United Nations Economic Commission for Europe (UNECE), 2019.

[191] WOZNIAK, E., C. MRAIDHA, S. GERARD und F. TERRIER: *A Guidance Framework for the Generation of Implementation Models in the Automotive Domain*. In: *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, S. 468–476. IEEE Computer Society, Sep. 2011.

[192] WOZNIAK, E., S. TUCCI-PIERGIOVANNI, C. MRAIDHA und S. GERARD: *An Integrated Approach for Modeling, Analysis and Optimization of Systems whose Design Follows the EAST-ADL2/AUTOSAR Methodology*. SAE International Journal of Passenger Cars - Electronic and Electrical Systems, 6(1):276–286, Apr. 2013.

[193] WÜTHERICH, G., N. HARTMANN, B. KOLB und M. LÜBKEN: *Die OSGi-Service-Platform: eine Einführung mit Eclipse Equinox*. dpunkt Verlag, 2008.

[194] XU, J., Z. YANG, Z. HUANG, Y. ZHOU, C. LIU, L. XUE, J. BODEVEIX und M. FILALI: *Hierarchical Behavior Annex: Towards an AADL Functional Specification Extension*. In: *2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, S. 1–11, Okt. 2018.

[195] YANG, Z., K. HU, D. MA und L. PI: *Towards a formal semantics for the AADL behavior annex*. In: *2009 Design, Automation Test in Europe Conference Exhibition*, S. 1166–1171, Apr. 2009.

[196] YILDIZ, A.: *A MNA-based unified ideal switch model for analysis of switching circuits*. Journal of Circuits, Systems and Computers, 22(6):12, Juli 2013.

[197] ZEIGLER, B., T. KIM und H. PRAEHOFER: *Theory of Modeling and Simulation*. Wiley Interscience, New York, 1976.

[198] ZEIGLER, B., T. KIM und H. PRAEHOFER: *Theory of Modeling and Simulation*. Academic Press, 2. Aufl., 2000.

[199] ZEIGLER, B. P. und J. S. LEE: *Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment*. In: SISTI, A. F. (Hrsg.): *Enabling Technology for Simulation Science II*, Bd. 3369 d. Reihe *Proceedings of the SPIE*, S. 49–58, Aug. 1998.

Betreute studentische Abschlussarbeiten

[Buc14] BUCIUMAN, MARIUS-FLORIN: *Entwurf, Modellierung und Evaluation eines Car-2-X Message Filter zur Minimierung von Buslasten und -latenzen in modernen E/E-Architekturen*. Masterarbeit ID-1911, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2014.

[Edu13] EDUGURU, MOHAN VAMSI: *Simulation of 2G / 3G Inter-Radio-Access-Technology (INTERRAT) Scenarios on a Virtual Platform*. Masterarbeit ID-1737, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2013.

[Kam14] KAMM, SIMON: *Entwurf und Implementierung einer Zynq-Hardware-Schnittstelle zur Kommunikation mit dem Modellierungstool Ptolemy II*. Bachelorarbeit ID-1904, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2014.

[Kam18] KAMM, SIMON: *Konzept, Implementierung und Evaluation einer ebenenübergreifenden Verhaltensmodellierung mittels State-Charts und deren integrierte Simulation in modellbasierten E/E-Architekturen*. Masterarbeit ID-2364, Vector Informatik GmbH, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2018.

[Koz15] KOZAREV, ATANAS: *Konzept, Implementierung und Evaluation eines Car-2-X Message Evaluation Moduls*. Bachelorarbeit ID-2024, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2015.

[Mer18] MERKER, JANINA: *Konzept zur Bewertung von statischen E/E-Architekturmodellen durch verknüpfte Verhaltensbeschreibungen im Fahrzeugentwicklungsprozess*. Masterarbeit ID-2361, Porsche AG, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2018.

[Neu18] NEUBAUER, KEVIN UWE: *Entwicklung und Evaluation einer Model-in-the-Loop Simulation zur iterativen Optimierung von automobilen E/E-Architekturen*. Masterarbeit ID-2334, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2018.

[Pup15] PUPLINKHUISEN, TIM: *Realisierung eines drahtlosen Sensornetzes mit optimiert zentraler/dezentraler Datenkommunikation*. Masterarbeit ID-2016, SAP AG, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2015.

[Ras13] RASTETTER, ROUVEN: *Portierung von PtidyOS auf ein Multiprozessorsystem zur echtzeitfähigen Ausführung verteilter Tasks*. Bachelorarbeit ID-1687, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2013.

[Red14] REDER, SIMON: *Adaptiver Algorithmus und Tool-Chain zur Beschleunigung von SystemC auf Many-Core Architekturen*. Masterarbeit ID-1824, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2014.

[Rei15] REICHERT, ALEXANDER: *Konzept, Implementierung und Evaluation eines Car2X-Message-Evaluation-Moduls in einem HW/SW Co-Design*. Bachelorarbeit ID-2035, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2015.

[Sch16] SCHADE, FLORIAN: *Design, Integration, and Evaluation of a Misbehavior Detection Concept for Vehicular Ad-Hoc Networks*. Masterarbeit ID-2148, Daimler AG, Sunnyvale, USA, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2016.

[vP17] POBLOTZKI, PATRICK VON: *Entwicklung eines Verfahrens zur Portierung von optimierten Gestenbibliotheken*. Bachelorarbeit ID-2358, Elmos Semiconductor AG, Karlsruher Institut für Technologie, Institut für Technik der Informationsverarbeitung, 2017.

Eigene Publikationen

Konferenz- & Journalbeiträge

[BB18] BUCHER, H. und J. BECKER: *Electric Circuit- and Wiring Harness-Aware Behavioral Simulation of Model-Based E/E-Architectures at System Level*. In: *2018 IEEE International Systems Engineering Symposium (ISSE)*, Seiten 1–8. IEEE, Oktober 2018.

[BBK+15] BUCHER, H., F. BUCIUMAN, A. KLIMM, O. SANDER und J. BECKER: *A V2X Message Evaluation Methodology and Cross-Domain Modelling of Safety Applications in V2X-enabled E/E-Architectures*. In: *Proceedings of the 8th EAI International Conference on Simulation Tools and Techniques*, SIMUtools '15, Seiten 71–78. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), August 2015.

[BBK+16] BUCHER, H., F. BUCIUMAN, A. KLIMM, O. SANDER und J. BECKER: *A V2X Message Evaluation Methodology and Cross-Domain Modelling of Safety Applications in V2X-enabled E/E-Architectures*. EAI Endorsed Transactions on Security and Safety, 16(8), Dezember 2016.

[BBO+15] BRITO, A. V., H. BUCHER, H. OLIVEIRA, L. F. COSTA, O. SANDER, E. U. K. MELCHER und J. BECKER: *A Distributed Simulation Platform using HLA for Complex Embedded Systems Design*. In: *2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Seiten 195–202. IEEE, Oktober 2015.

[BKB19a] BUCHER, H., S. KAMM und J. BECKER: *Cross-Layer Behavioral Modeling and Simulation of E/E-Architectures using PREEvision and Ptolemy II*. In: DURAK, U. (Herausgeber): *ASIM-Workshop Grundlagen und Methoden in Modellbildung und Simulation/Simulation Technischer Systeme.*, ARGESIM Reports, Seiten 7–12. ARGESIM, Wien, 2019. ARGESIM Report AR57, ASIM Mitteilung AM170.

[BKB19b] BUCHER, H., S. KAMM und J. BECKER: *Cross-Layer Behavioral Modeling and Simulation of E/E-Architectures using PREEvision and Ptolemy II*. Simulation Notes Europe, 29(2):73–78, Juni 2019. Selected ASIM GMMS/STS Postconference Publication.

[BKB19c] BUCHER, H., S. KAMM und J. BECKER: *Cross-layer Behavioral Modeling and Simulation of E/E-Architectures Using PREEvision and Ptolemy II [Extended Version]*. In: *Proceedings of the 2019 Summer Simulation Conference*, SummerSim '19, Seiten 18:1–18:12, San Diego, CA, USA, 2019. Society for Computer Simulation International.

[BKSB15] BUCHER, H., A. KLIMM, O. SANDER und J. BECKER: *Power Estimation of an ECDSA Core applied in V2X Scenarios using Heterogeneous Distributed Simulation*. In: *2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Seiten 187–194. IEEE, Oktober 2015.

[BNB19] BUCHER, H., K. NEUBAUER und J. BECKER: *Automated Assessment of E/E-Architecture Variants Using an Integrated Model- and Simulation-Based Approach*. Nummer 2019-01-0111 in *SAE Technical Paper*. SAE International, April 2019.

[BRB17] BUCHER, H., C. REICHMANN und J. BECKER: *An Integrated Approach Enabling Cross-Domain Simulation of Model-Based E/E-Architectures*. Nummer 2017-01-0006 in *SAE Technical Paper*. SAE International, März 2017.

[DPA+17] DERRIEN, S., I. PUAUT, P. ALEFRAGIS, M. BEDNARA, H. BUCHER, C. DAVID, Y. DEBRAY, U. DURAK, I. FASSI, C. FERDINAND, D. HARDY, A. KRITIKAKOU, G. RAUWERDA, S. REDER, M. SICKS, T. STRIPF, K. SUNESEN, T. TER BRAAK, N. VOROS und J. BECKER: *WCET-aware parallelization of model-based applications for multi-cores:*

The ARGO approach. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, Seiten 286–289. IEEE, März 2017.

[NBHB20] NEUBAUER, K., H. BUCHER, B. HAAS und J. BECKER: *Model-Based Development and Simulative Verification of Logical Vehicle Functions Using Executable UN/ECE Regulations [in press]*. In: *Proceedings of the 2020 Summer Simulation Conference*, SummerSim '20, San Diego, CA, USA, 2020. Society for Computer Simulation International.

[OBM+15] OLIVEIRA, H., A. V. BRITO, E. U. K. MELCHER, H. BUCHER, J. ARAÚJO und L. DUENHA: *Power-Aware Design of Electronic System Level using Interoperation of Hybrid and Distributed Simulations*. In: *Proceedings of the 28th Symposium on Integrated Circuits and Systems Design*, SBCCI '15, Seiten 18:1–18:7. ACM, August 2015.

[RBB+14] ROTH, C., H. BUCHER, A. BRITO, O. SANDER und J. BECKER: *A Simulation Tool Chain for Investigating Future V2X-based Automotive E/E Architectures*. In: *Proceedings of the 7th European Congress on Embedded Real Time Software and Systems (ERTS^2)*, Seiten 1739–1748, Februar 2014.

[RBR+13a] ROTH, C., H. BUCHER, S. REDER, F. BUCIUMAN, O. SANDER und J. BECKER: *A SystemC modeling and simulation methodology for fast and accurate parallel MPSoC simulation*. In: *Proceedings of the 28th Symposium on Integrated Circuits and Systems Design*, SBCCI '13, Seiten 1–6. IEEE, September 2013.

[RBR+13b] ROTH, C., H. BUCHER, S. REDER, O. SANDER und J. BECKER: *Improving parallel MPSoC simulation performance by exploiting dynamic routing delay prediction*. In: *2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, Seiten 1–8, Juli 2013.

[RKB+19] REDER, S., F. KEMPF, H. BUCHER, J. BECKER, P. ALEFRAGIS, N. VOROS, S. SKALISTIS, S. DERRIEN, I. PUAUT, O. OEY, T. STRIPF, C. FERDINAND, C. DAVID, P. ULBIG, D. MUELLER und U. DURAK: *Worst-Case Execution-Time-Aware Parallelization of Model-Based Avionics Applications*. Journal of Aerospace Information Systems, 16(11):521–533, 2019.

[RMB+18] REDER, S., L. MASING, H. BUCHER, T. TER BRAAK, T. STRIPF und J. BECKER: *A WCET-aware parallel programming model for predictability enhanced multi-core architectures*. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2018*, Seiten 943–948. IEEE, März 2018.

[RRB+14] ROTH, C., S. REDER, H. BUCHER, O. SANDER und J. BECKER: *Adaptive Algorithm and Tool Flow for Accelerating SystemC on Many-Core Architectures*. In: *2014 17th Euromicro Conference on Digital System Design (DSD)*, Seiten 137–145, August 2014.

[RRB+15] REDER, S., C. ROTH, H. BUCHER, O. SANDER und J. BECKER: *Adaptive Algorithm and Tool Flow for Accelerating SystemC on Many-core Architectures*. Microprocess. Microsyst., 39(8):1063–1075, November 2015.

[RRE+12] ROTH, C., S. REDER, G. ERDOGAN, O. SANDER, G. ALMEIDA, H. BUCHER und J. BECKER: *Asynchronous Parallel MPSoC Simulation on the Single-chip Cloud Computer*. In: *2012 International Symposium on System-on-Chip (SoC)*, Seiten 1–8, Oktober 2012.