# Skalierbarkeit einer Szenarien- und Template-basierten Simulation von Elektrik/Elektronik-Architekturen in reaktiven Umgebungen

Zur Erlangung des akademischen Grades eines

**DOKTORS DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)**

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

**DISSERTATION**

von

**M.Sc. Kevin Neubauer**

geb. in Mosbach

Tag der mündlichen Prüfung:
08.02.2022

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

**Skalierbarkeit einer Szenarien- und Template-basierten Simulation von Elektrik/Elektronik-Architekturen in reaktiven Umgebungen**

1. Auflage: Dezember 2022

# Kurzfassung

Die Automobilindustrie befindet sich in einem Wandel. Zukünftige Fahrzeuge sind elektrisch, autonom, vernetzt, werden geteilt und regelmäßig aktualisiert. Die Auswirkung davon ist ein starkes Wachstum der Software in zukünftigen Fahrzeugen, das vor allem auf die Implementierung von autonomen Fahrerverhalten und herstellerspezifischen Betriebssystemen zurückzuführen ist. Zur sicheren Ausführung dieser Software werden leistungsstarke Zentralrechner benötigt. Daneben führen ein steigender Bedarf an Sicherheitsmechanismen gegen Cyberangriffe, der Einzug von Leistungselektronik und die notwendige Gewährleistung der Ausfallsicherheit zu einem Anstieg der Komplexität bei der Entwicklung von automobilen *Elektrik/Elektronik-Architekturen (E/E-Architekturen)*. Im Bereich der Leistungselektronik liegt dies etwa an der benötigten Realisierung einer galvanischen Trennung zwischen Hochvolt- und Niedervoltnetz [177], um die Unversehrtheit der Insassen zu gewährleisten. Außerdem erfordert der Einsatz von permanenterregten Synchronmaschinen die sichere Auslegung und das Design entsprechender Schaltungen zur Ansteuerung. Cyberangriffe erfordern hingegen Mechanismen zur Abwehr und Gewährleistung der Informationssicherheit. Dazu zählen präventive Firewalls oder proaktive Angriffserkennungssysteme [4]. Eine Ausfallsicherheit wird dagegen durch Komponenten- oder Informationsredundanz ermöglicht. Um entsprechende Ausfallmaßnahmen einzuleiten, kann zusätzlich die Implementierung eines entsprechenden Monitorings nötig sein [235]. Im Zuge des Wandels wachsen die E/E-Architektur-modelle und weisen einen höheren Vernetzungsgrad auf. Dadurch haben E/E-Architekten mehr Designentscheidungen zu treffen, wobei Lösungen mehr Freiheitsgrade aufweisen und Auswirkungen schwieriger zu beurteilen sind. Jedoch müssen frühestmöglich im Entwicklungsprozess überprüfbar richtige Entscheidungen getroffen werden. Die Einführung frühzeitiger Tests in zukünftigen Zulassungsprozessen gibt dieser Anforderung ein weiteres Gewicht.

In existierenden Arbeiten wurde gezeigt, dass eine in E/E-Architekturentwicklungswerkzeugen integrierte Simulationen einen Mehrwert für E/E-Architekten bei der frühzeitigen Findung von Designentscheidungen bietet. In dieser Arbeit werden dagegen die Grenzen der Skalierbarkeit einer solchen Simulation untersucht. Dies geschieht mithilfe von industriell relevanten Anwendungsfällen. Ein bestehender Ansatz zur automatisierten Synthese von Simulationsmodellen aus PREEvision-E/E-Architekturmodellen wird dabei unter Berücksichtigung der Anforderungen bei großmaßstäblichen Modellen erweitert und angepasst. Hierzu werden zunächst Simulatoren hinsichtlich ihrer Eignung für einen Einsatz im industriellen Umfeld untersucht. Dies erfolgt anhand in der Arbeit definierten Auswahlkriterien sowie mithilfe von synthetischen und skalierbaren Benchmarks. Im Anschluss werden Konzepte untersucht, welche die Erhöhung der Skalierbarkeit einer E/E-Architektursimulation adressieren. Zu den Aspekten der Skalierbarkeit gehören neben der Performanz auch die Anwendbarkeit und die Validierbarkeit, welche von der Emergenz generierter Modelle beeinflusst werden.

Als Lösung werden in dieser Arbeit ausführbare Szenarienmodelle zur zustandsabhängigen Generierung von Stimuli und der reaktiven Evaluierung von Signalwerten verwendet. Durch deren Schnittstellen können gezielt die für einen Anwendungsfall relevanten Modellkomponenten der E/E-Architektur identifiziert werden, welche in Summe das sogenannte „System of Interest“ bilden. Auf diese Weise kann die Simulationsmodellgröße reduziert werden. Darüber hinaus werden parametrisierbare, pre-validierte und performanzoptimierte Teilmodelle, sogenannte „Templates“, bei der Generierung verwendet. Neben einer manuellen Zuweisung der Templates zu E/E-Architekturmodellkomponenten über die in dieser Arbeit verwendeten *Template And Layer Integration Architecture (TALIA)*, haben spezifische Komponenten auf der Leistungssatzebene, wie Batterien, Stecker oder Kabel, bereits Standard-Templates zugewiesen. Simulationsmodelle können dadurch ohne manuelle Verhaltensmodellierung und zugehörige Validierung generiert werden. Damit Standard-Templates verwendet werden können, wird eine Hardware-zentrierte Abbildung verfolgt. Die physikalische E/E-Architektur aus der Realität bildet dabei die Grundlage für die generierten Simulationsmodelle. Softwaremodelle werden ergänzend über die Modelle der Steuergeräte bzw. *Electronic Control Units (ECUs)* integriert. Ebenso sind die Szenarienmodelle nach der Generierung ein Teil der Simulationsmodelle. Damit findet die Integration unterschiedlicher E/E-Architekturebenen statt, wodurch hybride Simulationsmodelle entstehen.

Für die Evaluation werden Anwendungsfälle für Simulationen aus möglichen Designentscheidungsfragen abgeleitet und anhand definierter Kriterien für die weitere Betrachtung ausgewählt. Designentscheidungsfragen ergeben sich beim Technologieentscheid, der Dimensionierung von Komponente oder bei Optimierungen. Die Anwendungsfälle bestimmen das benötigte Testmodell, bestehend aus dem zu evaluierenden System of Interest und dem Prüfstandmodell, realisiert als Szenariomodell. Da das Testmodell die Basis des Simulationsmodells bildet und damit dessen Komplexität bestimmt, lässt sich anhand der Anwendungsfälle die Skalierbarkeit der E/E-Architektursimulation beurteilen. Insbesondere wird in dieser Arbeit der Einfluss emergenter Modelleigenschaften auf die Skalierbarkeit untersucht.

# Abstract

The automotive industry is undergoing a transformation. Future vehicles will be electric, autonomous, connected, shared and updated regularly. This leads to a strong increase of the software size in future vehicles, particularly because of implementing autonomous driving behaviors and manufacturer-specific operating systems. As a result, high-performance computers are needed. Together with the introduction of power electronics, security mechanisms against cyber-attacks, and the need to guarantee fail-safety, they increase the complexity in the development of automotive electric/electronic (E/E) architectures. Regarding power electronics, the design of a galvanic isolation between high-voltage and low-voltage networks, to ensure occupant safety, is one example of this increasing complexity [177]. Another one is the safe development of control circuits for permanent magnet synchronous motors. Concerning cyberattacks, defense mechanisms that range from preventive firewalls to proactive attack detection systems must be developed [4]. Referring to fail-safety, the realization of component or information redundancy, as well as the implementation of monitoring systems for error detection, are necessary [235]. As a result, E/E architecture models are growing and exhibit a higher degree of interconnectedness. In turn, E/E architects have to make more design decisions, in which solutions have an increased degree of freedom and their impact is more difficult to assess. However, verifiable and correct decisions have to be made as early as possible in the development. The importance in terms of safety and security is further emphasized by the introduction of early tests in future vehicle approval processes.

In existing work, it has been shown that simulations, integrated in E/E architecture development tools, can early support E/E architects with design decisions. This work, in contrast, uses industrially relevant use cases to determine the scalability of such simulations. Hereto, an existing approach of an automated synthesis of simulation models from PREEvision E/E architecture models is extended and adapted with respect to the requirements of large-scale models. Moreover, different simulators are compared regarding their suitability for the use in an industrial environment. For this purpose, an evaluation is performed on the basis of defined criteria and both synthetic and scalable benchmarks. Subsequently, concepts to master the scalability challenges of simulating E/E architectures are investigated. They address performance issues as well as increasing the applicability and validability of such simulations, which depends on the emergent characteristics of generated models.

One aspect of the proposed solution in this work is the use of executable scenario-models to generate state-dependent stimuli and evaluate signals in a reactive manner. The interfaces of a scenario-model can be used to determine the relevant model components of the E/E architecture. They form the so-called system of interest, in order to reduce the size of the simulation models. Another aspect is formed by parameterizable, pre-validated and performance-optimized sub-models, called templates, that are used for the generation. They can be manually assigned to E/E architecture model components via an introduced *Template And Layer*

*Integration Architecture (TALIA)*. In addition, default-templates are initially linked to E/E architecture model components on the power net level, such as batteries, connectors or cables. Consequently, simulation models can be generated without manual behavior modeling and its validation. To realize the latter, hardware-centric models of the physical E/E architecture form the basis of the generation and are complemented with software-models, integrated into ECUs, and the scenario-models. As this approach integrates various E/E architecture layers, hybrid simulation models are the outcome.

For the evaluation, use cases for E/E architecture simulations are derived from E/E architects' design decision issues, such as technology decisions, component dimensioning or optimization. They determine the test model, consisting of the system of interest to be evaluated and a test bench model (implemented as a scenario-model). As the test model is the basis of the simulation model and thus determines its complexity, the use cases are utilized to assess the scalability of the generated E/E architecture simulations. In particular, the influence of emergent model properties on scalability is investigated in this work.

# Vorwort

*„Ohne Ratgeber sind Pläne zum Scheitern verurteilt;*
*aber wo man gemeinsam überlegt, hat man Erfolg.“* - Sprüche 15:22 (HFA)

In unserem Leben begegnen wir vielen Ratgebern, doch nur wenige und ganz besondere bringen uns voran. Sie helfen uns dabei unser Potenzial zu entfalten und stärken uns. Wir vertrauen ihnen. Bei diesen Ratgebern möchte ich mich im Folgenden bedanken.

Ein ganz besonders herzlicher Dank geht an Prof. Dr. Dr. h. c. Jürgen Becker, der diese Arbeit erst ermöglicht hat, sowie an Dr. Clemens Reichmann, Dr. Max Kramer und Dr. Leonard Masing. Die Handlungsfreiheiten und das entgegengebrachte Vertrauen habe ich stets genossen und geschätzt. Außerdem danke ich Prof. Bernhard Bauer von der Universität Augsburg, ebenso herzlichst, für die Übernahme des Korreferats. Zahlreiche Diskussionen, kritische Fragen und entscheidende Ratschläge haben maßgeblich zur Qualität dieser Arbeit beigetragen.

Als wissenschaftlicher Mitarbeiter am Institut für Technik der Informationsverarbeitung (ITIV) des Karlsruher Instituts für Technologie (KIT) und Software Development Engineer bei der Vector Informatik GmbH konnte ich einerseits viel lernen und hatte andererseits die Möglichkeit mein Wissen an Studierende weiterzugeben. An dieser Stelle möchte ich mich bei Florian Schade für die angenehme Zusammenarbeit bei der Betreuung der Lehrveranstaltung „Labor Schaltungsdesign“ bedanken. Außerdem bedanke ich mich bei all meinen betreuten Studierenden, die mit ihren Abschlussarbeiten einen Beitrag zu dieser Dissertation geleistet haben.

Ebenso geht ein großes Dankeschön an all meinen Kollegen für ihr Mitwirken an zielführenden und inspirierenden Gesprächen sowie für die vielen schönen Momente auch abseits der Arbeit.

Meiner Familie, der Familie Caetano da Silva und meinen Freunden danke ich sehr für ihre Unterstützung sowie ihrem Verständnis für die wenig verbrachte gemeinsame Zeit in den letzten vier Jahren. Meinen Eltern, Uwe Neubauer und Bettina Heckmann, danke ich insbesondere für ihren stetigen Zuspruch. Meiner Verlobten, Natália Caetano da Silva, danke ich sehr für ihren unermüdlichen Rückhalt. Sie hatte für jede Situation einen passenden Rat und des Öfteren zum Erhalt meiner Gesundheit beigetragen. Obrigado!

Sorocaba, im Dezember 2022
Kevin Neubauer

# Inhaltsverzeichnis

# 1. Einleitung

Die Fahrzeuge der Zukunft sind elektrisch, autonom, vernetzt, werden geteilt und stetig aktualisiert. Zur Realisierung befindet sich die Automobilindustrie in einem Wandel. Vor allem *Elektrik/Elektronik-Architekturen (E/E-Architekturen)* sind davon betroffen. Die Elektrifizierung führt zum Einzug von Leistungselektronik und Hochvoltsystemen in moderne Fahrzeuge, wobei entsprechende Überwachungs- und Schutzsysteme für Insassen nötig werden [307]. Durch das autonome Fahren werden auch bisher mechanische Komponenten, wie die Lenkung (Steer-by-Wire), die Schaltung (Shift-by-Wire) oder die Bremsanlage (Brake-by-Wire) elektrifiziert. Dabei muss eine entsprechende Ausfallsicherheit durch Komponenten- oder Informationsredundanz sowie die Erkennung von Fehlern während des Systembetriebs durch Überwachung bzw. Monitoring gewährleistet sein [232, 235]. Der Softwareumfang wird, u.a. durch die Implementierung des Fahrerverhaltens, voraussichtlich um den Faktor 10000 ansteigen [309]. Ein Teil dieser Software wird dabei in der Cloud ausgeführt [309]. Vernetztes Fahren, Carsharing und Updates *Over the Air (OTA)* erfordern eine Kommunikation des Fahrzeugs mit seiner Umwelt, wodurch jenes zu einem mobilen, *cyber-physischen System (CPS)* wird. Bis 2025 sollen 273 Millionen vernetze Fahrzeuge auf den Straßen Europas unterwegs sein [153]. Dies erhöht jedoch die Gefahr von Cyberangriffen. Im Zeitraum von 2010 bis 2020 waren 32,94 % aller Cyberangriffe auf Fahrzeuge mit Anbindung an Cloud-Server zurückzuführen, welche von 2019 bis 2020 um 73 % gestiegen sind [285]. Daher wird der Einsatz entsprechender Angriffserkennungs- oder Angriffsverhinderungssysteme immer wichtiger.

## 1.1. Motivation

Durch den zuvor beschriebenen Wandel steigt die Komplexität von automobilen E/E-Architekturen. Die Komplexität wird an der Anzahl der Komponenten (*Hardware (HW)* und *Software (SW)*) sowie deren Vernetzungsgrad gemessen. Hinzu kommen zahlreiche Varianten, die durch den Individualisierungswunsch der Kunden entstehen und durch eine Konfiguration beim Kauf möglich sind. Für die Entwickler von E/E-Architekturen, im Folgenden als *E/E-Architekten* bezeichnet, bedeutet diese Komplexität einen größeren Spielraum bei Designentscheidungen, wobei es zunehmend schwieriger wird, die Auswirkung einzelner Entscheidungen auf das Gesamtsystem zu beurteilen. Basis für die Entscheidungen bilden Anforderungen, die vom jeweiligen Einsatzland abhängen und sich durch Anforderungsmanagement während der Entwicklung ändern können. Das Treffen der dazu passenden Entscheidungen beruht hierbei vorwiegend auf der Erfahrung der E/E-Architekten, d.h. zum Teil auch auf Bauchentscheidungen [219], und ist damit an Experten gebunden. Dabei gibt es Experten für unterschiedliche E/E-Domänen, wie z.B. für das Leitungssatz- oder Kommunikationsdesign. Ein Ausfall solcher Experten kann vor allem in frühen Entwick-

lungsphasen zu Verzögerungen in der gesamten Entwicklung führen. Dies führt wiederum zu erhöhten Kosten, da Entscheidungen früh getroffen werden sollten, weil die Kosten der Entwicklung mit der Zehnerregel vom Anfang des Entwicklungsprozesses bis zur Produktion immer weiter ansteigen [234]. Neben dem Zeitpunkt beeinflusst auch die Qualität der Entscheidung, welche z.B. von der verwendeten Technologie abhängt, die Entwicklungskosten. Ziel ist es also möglichst früh eine hohe Qualität, d. h. eine Überstimmung des Ergebnisses (Produkt) mit den Anforderungen, zu erreichen [97]. Zur Qualitätsverbesserung während der Entwicklung werden nach [42] üblicherweise statische Methoden, wie das *Quality Function Deployment (QFD)*, welches auf Kundenwünschen basiert, und die Design-*Fehlermöglichkeits- und Einflussanalyse (FMEA)*, welche auf Produktkonzepten oder Konstruktionsunterlagen basiert, angewandt. Design-FMEA wird aus gesammelten Erfahrungen durch begangene Fehler abgeleitet und eignet sich daher nur bedingt für die Beurteilung neuer Konzepte oder der Auswirkung einzelner Designentscheidungen auf das Gesamtsystem [42]. Daher werden modellbasierte Berechnungen und Tests benötigt. Tests sind ein fester Bestandteil im V-Modell (siehe Unterabschnitt 3.1.4) und geben den E/E-Architekten Rückmeldung zu ihren Entscheidungen. Sie gehören zu den manuellen Methoden (siehe Unterabschnitt 3.2.3) und werden ferner in statische und dynamische Tests unterteilt. Bei statischen Tests wird das zu testende System nicht ausgeführt. Dazu zählen z. B. Reviews. Bei dynamischen Tests werden das System, oder Teile davon, real oder virtuell, d.h. simulativ, ausgeführt. Simulation bietet nach [303] die Möglichkeit Fehler und Verhaltensweisen eines Systems, die erst bei dessen Ausführung auftreten, in frühen Entwicklungsphasen zu erfassen, sofern es die Testabdeckung erlaubt. So können die Auswirkungen von Designentscheidungen frühzeitig untersucht und abgeschätzt werden. Ebenso wird die Testzeit verkürzt und eine Untersuchung extremer Randfälle ermöglicht, was mit realen Tests aufgrund des Aufwands oder der Sicherheit nicht umsetzbar ist [303]. Angesichts dessen schlägt auch die *Organisation Internationale des Constructeurs de Automobiles (OICA)*[1] eine frühzeitige Integration der Simulation während der Entwicklung von Fahrzeugen zur Typenzulassung vor, wie in Abbildung 1.1 dargestellt.

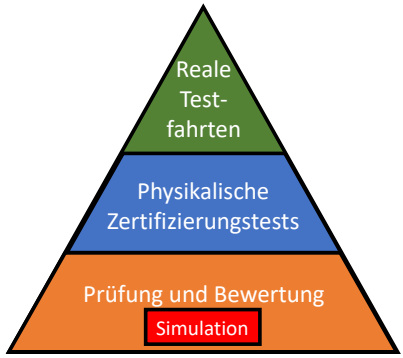



Abbildung 1.1.: OICA-3-Säulen-Ansatz (Quelle: [199]).

[1] Internationale Automobilherstellervereinigung.

In der ersten Phase (**Prüfung und Bewertung**) soll Simulation vordergründig zur Beurteilung von sicherheitsrelevanten Konzepten eingesetzt und durch Audits ergänzt werden. Diese Phase hat den größten Umfang. In der anschließenden Phase (**Physikalische Zertifizierungstests**) soll die Simulation anhand von realen Tests kalibriert und überprüft werden. Zum Abschluss der Zertifizierung sollen **Reale Testfahrten** mit minimalem Umfang stattfinden. [199]

Simulation bietet eine Möglichkeit, die Bedürfnisse von E/E-Architekten bei der Bewertung ihres Designs zur Entscheidungsfindung zu erfüllen. Zudem werden aktuell gesetzliche Grundlagen geschaffen, die eine Integration von Simulation im Entwicklungsprozess verpflichtend vorsehen.

## 1.2. Zielsetzung und Abgrenzung

Anknüpfend an die vorausgegangenen Beobachtungen beschäftigt sich diese Arbeit mit dem Einsatz von Simulation zur Bewertung von E/E-Architekturen. Hierzu soll mithilfe von industriell relevanten Anwendungsfällen die Skalierbarkeit von solchen Simulationen untersucht werden. Dabei sollen die Aspekte Anwendbarkeit, Performanz und Validierbarkeit untersucht sowie der Nutzen für E/E-Architekten bewertet werden. Aufbauend auf dem Ansatz einer automatisierten Synthese von Simulationsmodellen aus PREEvision-E/E-Architekturmodellen nach [43], sollen Konzepte aus dem Stand der Technik zur Erhöhung der Skalierbarkeit verglichen, integriert und bewertet werden. Das verwendete Abbildungskonzept soll dabei angepasst und erweitert werden. Außerdem soll der eingesetzte Simulator Ptolemy II innerhalb eines Vergleichs mit Alternativen bewertet und ggf. ersetzt werden. Dazu werden Kriterien festgelegt und synthetische Benchmarks verwendet. Erweiterungen sollen insbesondere die Problematik der Modellbildung und Validierung von ebenenübergreifenden E/E-Architektursimulationen mit hybridem Charakter adressieren. Dies schließt die Beachtung emergenter Modelleigenschaften ein, die bei einer Komposition des Simulationsmodells aus Teilmodellen auftreten können. Des Weiteren sollen Umweltaspekte und Fahrdynamik berücksichtigt werden sowie eine einfache Bildung reaktiver Prüfstandmodelle möglich sein.

## 1.3. Vorgehen und Aufbau dieser Arbeit

In Kapitel „Grundlagen“ wird zunächst das zum Verständnis der Arbeit relevante Wissen aufbereitet. Dabei werden folgende Themen behandelt: *Model Based Systems Engineering (MBSE)*, Simulation, Skalierbarkeit und die verwendeten Systeme.

In Kapitel „Stand der Technik“ werden aktuelle und zukünftige Veränderungen von E/E-Architekturen beschrieben. Diese sind vorrangig zur Identifikation von Anwendungsfällen für eine E/E-Architektursimulation relevant. Danach werden Entwicklungsprozesse sowie die Möglichkeiten zur Verifikation von E/E-Architekturen beschrieben. Des Weiteren werden Methoden zur Erhöhung der Skalierbarkeit vorgestellt und diskutiert. Aufbauend darauf findet die Einordnung und Abgrenzung dieser Arbeit statt.

Kapitel „Forschungsfragen und Methodik“ beinhaltet die konkreten Herausforderungen und Forschungsfragen, die diese Arbeit beantworten soll. Außerdem wird die Methodik bzw. das wissenschaftliche Vorgehen erläutert.

Kapitel „Vergleich und Auswahl eines Simulators“ beschreibt die Anforderungen bei der Auswahl eines geeigneten Simulators mit Hinblick auf seine Skalierbarkeit und seinem Einsatz im industriellen Umfeld. Die Konzeption und Ausführung synthetischer sowie skalierbarer Benchmarks ist ebenfalls Bestandteil dieses Kapitels.

Für den ausgewählten Simulator wird, unter Berücksichtigung der Aspekte der Skalierbarkeit, in Kapitel „E/E-Architekturmodellbasierte Simulation“ eine Abbildung konzeptioniert. Des Weiteren wird deren Implementierung beschrieben.

Zur Evaluation werden in Kapitel „Anwendungsfallbasierte Evaluation der Konzepte und Simulatoren“ entsprechende Anwendungsfälle beschrieben und ausgewählt. Diese bilden die Grundlage von anschließend ausgeführten Simulationen, mit denen die implementierten Konzepte und die Skalierbarkeit evaluiert werden.

# 2. Grundlagen

## 2.1. Modellbasierte Systementwicklung

Ein Modell ist ein abstrahiertes Abbild der Realität und besitzt nach Stachowiak [241] drei Merkmale, welche sich in Abbildung 2.1 wiederfinden:

1. **Das Abbildungsmerkmal**: Ein **Modell** repräsentiert ein natürliches oder künstliches Urbild (**Original**), mit dem es nicht identisch ist.
2. **Das Verkürzungsmerkmal**: Modelle sind eine Abstraktion und enthalten nur die für einen bestimmten Zweck relevanten Details, d. h. es werden bei der **Abbildung** nicht relevante (**präterierte**) Details vernachlässigt. Es können aber auch zusätzliche (**abundante**) Eigenschaften ergänzt werden.
3. **Das pragmatische Merkmal**: Modelle sind ihren Originalen nicht per se eindeutig zugeordnet und erfüllen eine Ersetzungsfunktion nur für bestimmte Subjekte, zu einer bestimmten Zeit und *„unter Einschränkung auf bestimmte gedankliche oder tatsächliche Operationen“* [241].

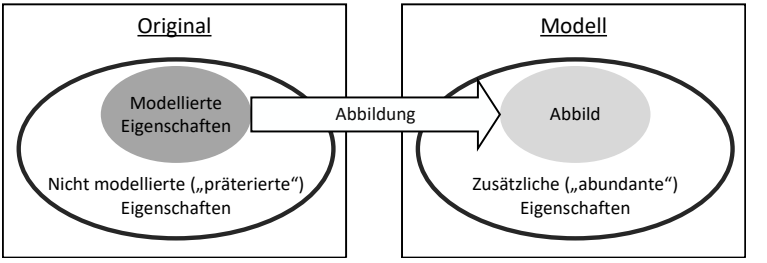



Abbildung 2.1.: Original, Abbildung und Modell (Quelle: eigene Darstellung nach [114]).

Modelle können grafisch, mathematisch oder physikalisch beschrieben werden und dienen nach Hart [110] u. a. zur:

- Verbesserung des Verständnisses
- Unterstützung bei Entscheidungsfindungen
- Erklärung, Kontrolle und Vorhersage von Ereignissen

Modelle bilden die Grundlage der modellbasierten Systementwicklung bzw. des MBSE. Darunter wird nach dem *International Council on Systems Engineering (INCOSE)* der *„[...] formalisierte Gebrauch der Modellierung zur Unterstützung bei Systemanforderungen, [sowie] bei dem Design, der Analyse und Verifizierungs- und Validierungsaktivitäten [von Systemen bezeichnet], die in der Kon-*

*zeptionsphase beginnen und während der Entwicklung bis zu späteren Lebenszyklusphasen fortgesetzt werden“* [127] verstanden.

In dieser Arbeit wird der Modellbegriff für digitale Computermodelle verwendet. Seit den 1960er-Jahren habe diese nach [111] einst verwendete Dokumente im Zuge des MBSE ersetzt. Sie erleichtern seither die Zusammenarbeit, erlauben die Beherrschung komplexer Systeme und ermöglichen sowohl eine bessere Systemqualität als auch schnellere Entwicklungszeiten. [111]

### Systemmodelle

Ein System bildet eine geschlossene und von der Umgebung durch Schnittstellen (Ein- und Ausgänge) abgegrenzte Einheit aus intern interagierenden Komponenten [253]. Der Zustand der Komponenten beeinflusst den Zustand des Systems. Heterogenität zwischen den Komponenten oder eine bestimmte, erforderliche Sichtweise können dabei verschiedene Modellarten zur Beschreibung erfordern. Abbildung 2.2 zeigt zur Verdeutlichung das vereinfachte **Systemmodell** eines Autos mit seinen unterschiedlichen Modellen.

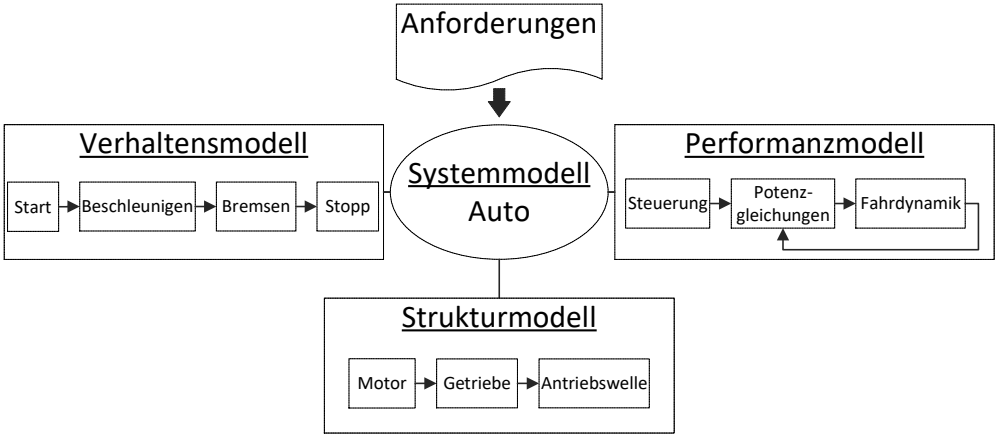



Abbildung 2.2.: Beispiel eines Systemmodells (Quelle: eigene Darstellung nach [92]).

Nach [195, 143, 252] sind die Modellarten folgendermaßen definiert: **Strukturmodelle** werden zur Modellierung statischer, zeitunabhängiger Zusammenhänge verwendet. Dabei wird vordergründig der Datenfluss modelliert und es bleiben sowohl die Struktur als auch die Beziehungen zwischen den Modellelementen konstant. **Verhaltensmodelle** legen dagegen den Fokus auf die Modellierung dynamischer, zeitabhängiger Aspekte. Hier wird vordergründig der Kontrollfluss innerhalb von Systemen modelliert. Eine Sonderform der Verhaltensmodelle sind **Performanzmodelle**. Sie beschreiben im Gegensatz zu Verhaltensmodellen nicht *was* ausgeführt wird, sondern *wie*. [195, 143, 252]

Es wird deutlich, dass die Anwendung des Modells, die Funktion seiner Komponenten und deren Beziehungen definiert sein müssen. Im digitalen Entwurf werden zur Beschreibung der unterschiedlichen Modellarten entsprechende Modellierungssprachen verwendet, die in Unterabschnitt 2.1.1 beschrieben sind.

**Emergenz von Systemen** *„Emergenz bezeichnet das Auftauchen von Systemzuständen, die nicht durch die Eigenschaften der beteiligten Systemelemente erklärt werden können.“* [244]. Ein emergentes Verhalten von Systemen kann durch die Komposition eines Systems aus Komponenten mit einem definierten Verhalten entstehen, wobei sich das Gesamtverhalten nicht vollständig aus dem definierten Verhalten seiner Komponenten ableiten lässt und neue Eigenschaften aufweisen kann. Oft lässt sich dieses Verhalten auf eine lückenhafte Spezifikation des Gesamtsystems zurückführen [117].

### 2.1.1. Modellierungssprachen und Metamodelle

*„Modellierungssprachen stellen [...] das Vokabular und die Grammatik zur Verfügung, die benötigt werden, um Sachverhalte [...] in Modellen abbilden zu können.“* [88]. Zur Definition einer solchen Sprache bietet es sich an, die Sprache selbst als Mittel zur Beschreibung, also als Metasprache, zu nutzen. Hierzu dienen Metamodelle, durch die Modellierungssprachen in ihrer eigenen Sprache oder mit einer geeigneten Notation definiert werden können [114]. Nach der *Object Management Group (OMG)* ist ein Metamodell *„[...] ein Modell, das die Modellierung seiner selbst modelliert. “* [194]. Metamodelle bilden die Grundlage von Modelltransformationen, womit Quellmodelle über Transformationsregeln in Zielmodelle überführt werden können. Modellierungssprachen sind in MBSE-Werkzeugen, wie PREEvision (siehe Unterabschnitt 2.4.1), implementiert.

#### UML

Nach [143] ist die *Unified Modeling Language (UML)* ein standardisiertes, allgemein verwendbares und seit 1997 in der Version 1.0 veröffentlichtes Beispiel für eine Modellierungssprache. Sie stellt Diagramme und deren Bestandteile zur Modellierung von statischen und dynamischen Aspekten zur Verfügung. [143]

### 2.1.2. Elektrik/Elektronik-Architekturen

Die E/E-Architektur ist ein eingebettetes, verteiltes System. Der elektrische Teil umfasst Komponenten zur Energieversorgung, -Verteilung, -Umwandlung und physikalischen Kommunikation. Dazu zählen Leitungen, passive Aktoren oder Sensoren mit linearer Kennlinie. Der elektronische Teil umfasst Komponenten zur Steuerung des elektrischen Stroms durch Schaltungen, die mit nicht-linearen Bauteilen, wie Halbleiter, aufgebaut sind. Hierbei werden i.d.R. Mikroprozessoren in Steuergeräten bzw. *Electronic Control Units (ECUs)* eingesetzt, die in Software realisierte Funktionen ausführen. Da an dem Entwurf einer E/E-Architektur noch

weitere Aspekte berücksichtigt werden müssen und unterschiedliche Fachbereiche sowie Interessenvertreter beteiligt sind, haben sich für die strukturierte Beschreibung entsprechende Ebenenmodelle der E/E-Architektur etabliert. Eine allgemeine Form ist nach [246] in Abbildung 2.3 dargestellt. Die Schichten bzw. Ebenen der Architektur, zwischen denen auch Abhängigkeiten bestehen, entsprechen dabei den spezifischen Sichten der Interessenvertreter. Sie werden in den folgenden Absätzen nach [246] beschrieben.

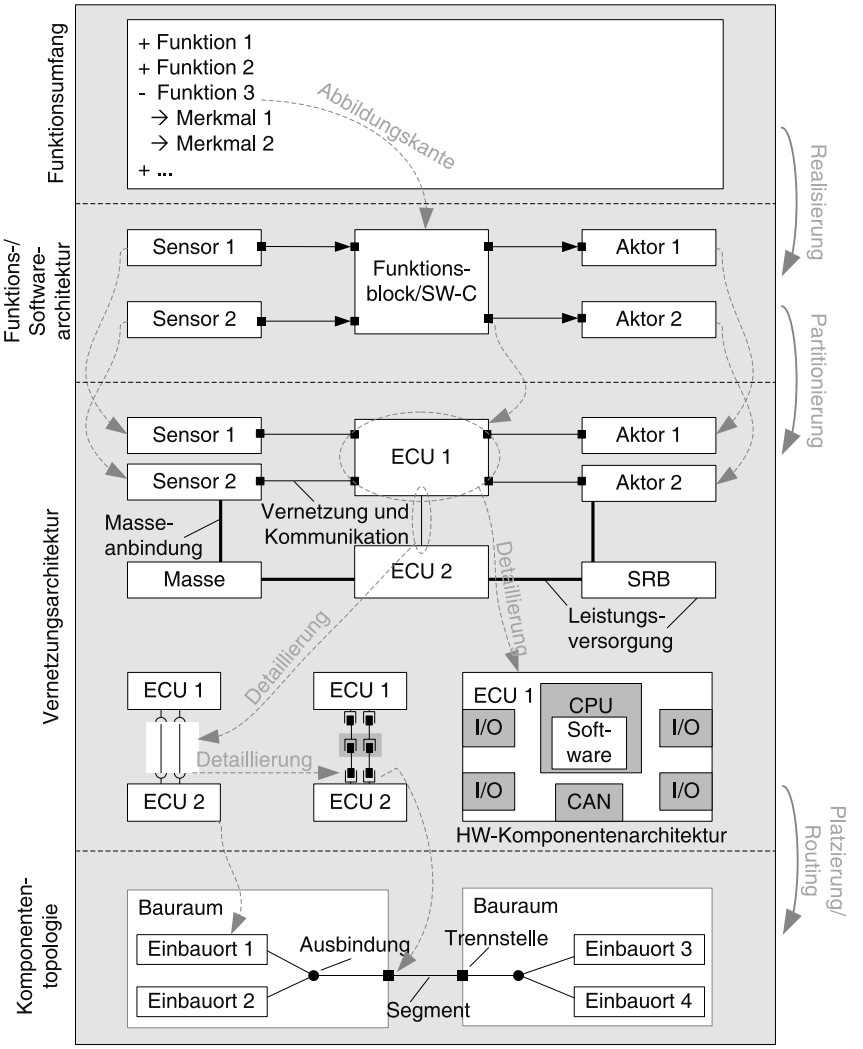



Abbildung 2.3.: Systemebenen einer E/E-Architektur (Quelle: [246]).

Die oberste Ebene bildet der **Funktionsumfang**, der die vom Kunden erlebbaren Funktionen bzw. Kundenanforderungen als Liste mit detaillierten Merkmalen zusammenfasst. [246]

Diese textuelle Auflistung wird auf der Ebene (**Funktions-/Softwarearchitektur**) in ein technologieunabhängiges Blockschaltbild übersetzt, welches dem *Eingabe-Verarbeitung-Ausgabe-Prinzip (EVA-Prinzip)*, vertreten durch **Sensoren**, **Funktionen** und **Aktoren**, folgt. Die Verbindungen zwischen den Ein- und Ausgängen der Blöcke entsprechen den Datenabhängigkeiten. Im späteren Verlauf des Entwicklungsprozesses kann ein Funktionsblock einer Softwarekomponente (**SW-C** bzw. *Software Component (SWC)*) entsprechen. [246]

Auf der **Vernetzungsarchitekturebene** werden die Funktionen bzw. die SWCs den dort modellierten Hardwarekomponenten (*Hardware Components (HWCs)*) zugewiesen. HWCs sind z.B. **Sensoren**, **ECUs** oder **Aktoren** und physikalisch im realen Fahrzeug vorhanden. Daher müssen die logischen Datenabhängigkeiten aus der Funktions-/Softwarearchitektur bei der Zuordnung über physikalische Kommunikationsmedien abgebildet werden. Neben der **Kommunikation** wird auf dieser Ebene auch die **Leistungsversorgung** modelliert. Verfeinerungen der HWCs (**HW-Komponentenarchitektur**) und eine **Detaillierung** der Verbindungen zur Kommunikation und Versorgung sind ebenfalls Bestandteil dieser Ebene. [246]

Auf der **Komponententopologieebene** werden schließlich die HWCs den im Fahrzeug verfügbaren **Bauräumen** zugeordnet und die Leitungen entsprechend vorgegebener Verlegewege geroutet. Hierdurch erfolgt eine weitere Verfeinerung des Designs, da entsprechend notwendige **Trennstellen** modelliert bzw. eingeplant werden müssen. [246]

### 2.1.3. Cyber-physische Systeme

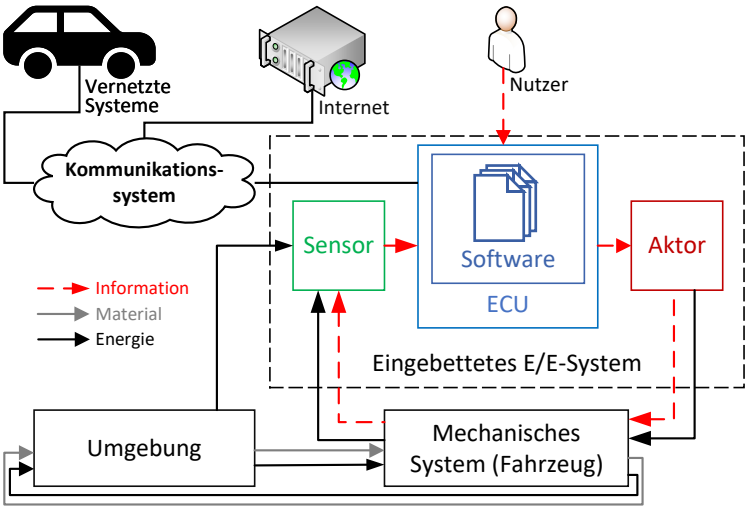



Abbildung 2.4.: Cyber-physisches System (Quelle: eigene Darstellung nach [41, 314]).

Eine E/E-Architektur (**Eingebettetes E/E System**) bildet zusammen mit dem **Fahrzeug**, in welches sie eingebettet ist, und dessen **Umgebung** ein mechatronisches System. Besteht darüber hinaus über ein **Kommunikationssystem** (z.B. Cloud) eine Verbindung zu anderen **Systemen** bzw. dem **Internet**, so handelt es sich um ein cyber-physisches System (siehe Abbildung 2.4).

Charakteristisch für mechatronische Systeme ist neben dem aus der Funktions-/Softwarearchitektur bekannten Daten- bzw. **Informationsfluss** eine weitere Differenzierung von **Material**- (bzw. Stoff-) und **Energiefluss**. Der Materialfluss modelliert nach [41, 314, 57] kein Signal, sondern bildet tatsächlich die physikalische Leitung von Materie bzw. Stoffen in beliebigen Aggregatzuständen ab. Der Energiefluss modelliert den Austausch von Energie zwischen den beteiligten Komponenten. Dazu zählen neben der elektrischen Energie auch weitere Energieformen, wie die Bewegungsenergie oder chemische Energie. Eine Umwandlung, analog zur Verarbeitung beim Datenfluss, erfolgt innerhalb der Systemkomponenten. [41, 314, 57]

## 2.2. Simulation

Simulation leitet sich vom lateinischen Wort *Simulatio* ab und bedeutet *„Schein, Täuschung"*. Nach Kopacek [149] und der Richtlinie 3633 des *Vereins Deutscher Ingenieure (VDI)* lautet eine allgemeine Definition: *„Simulation ist die Nachbildung eines dynamischen Prozesses, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind."*[149]. Charakteristisch ist hierbei die Verwendung eines Modells, wodurch sich die Simulation von einer Berechnung unterscheidet [104]. Aus technischer Sicht ist eine Simulation die Implementierung und Ausführung eines Modells über der Zeit und schließt die in Abbildung 2.5 dargestellten Komponenten ein.

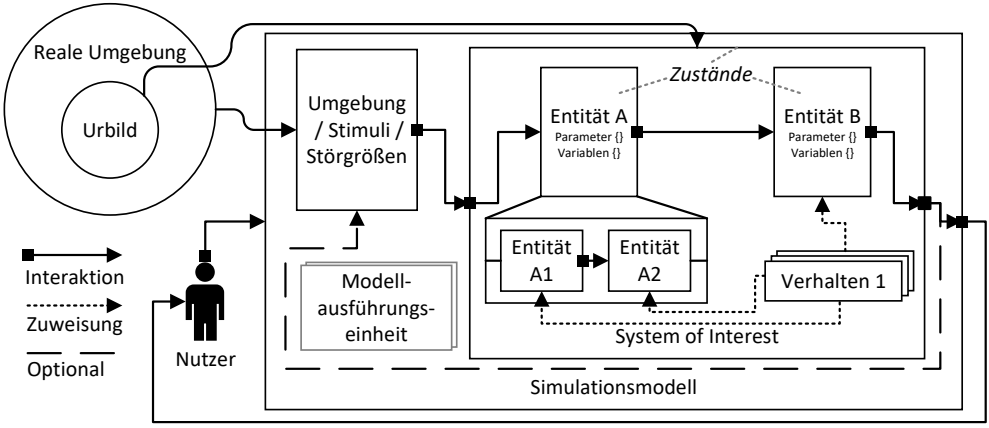



Abbildung 2.5.: Komponenten einer Simulation (Quelle: eigene Darstellung unter Verwendung von [167, 241, 215]).

Hier befindet sich zunächst auf der linken Seite das reale **Urbild**, welches in einer realen Umgebung eingebettet ist. Durch die Abstraktion des Urbilds und die Festlegung der Sys-

temgrenzen wird das **System of Interest** *(SOI)*, also das relevante System, definiert. Auf die E/E-Architektur bezogen entspricht das SOI gerade dem Teil der E/E-Architektur, der für die Simulation eines Anwendungsfalls benötigt wird. An den Systemgrenzen sind auch die Schnittstellen definiert, d.h. die Ein- und Ausgänge (►/ ■), über die das SOI mit der **Umgebung** oder dem **Nutzer** interagiert. Der Nutzer ist für die Konfiguration und Überwachung des Simulationsmodells zuständig. Im Folgenden werden die einzelnen Komponenten von Simulationsmodellen nach [167, 215, 28] erläutert:

**Entitäten:** repräsentieren die Komponenten eines Systems oder Prozesses, wie z.B. Sensoren. Sie werden in Form von Blöcken modelliert und können, je nach Abstraktion der Urbild-Komponente, unterschiedliche Detaillierungsebenen umfassen. Oft wird bei der Modellierung von Entitäten ein hierarchischer Ansatz gewählt, d.h. das eine Entität durch weitere Entitäten verfeinert werden kann. Des Weiteren besitzen Entitäten **Parameter** und **Variablen**, die Eigenschaften bzw. **Zustände** repräsentieren. Der Zustand des gesamten Simulationsmodells ist dabei ein n-Tupel aus allen Zuständen der Entitäten. [167, 215, 28]

**Verhalten:** Entitäten besitzen ein Verhalten, um mit Aktionen oder Aktivitäten auf bestimmte Ereignisse reagieren zu können. Eine Aktivität ist einer Reihe aufeinanderfolgender Aktionen. Das Verhalten kann darüber hinaus vom Zustand der Entität oder des Gesamtsystems abhängen. Der Zustand einer Entität wird durch die Werte all ihrer Variablen zu einem bestimmten Zeitpunkt definiert. Das Verhalten wird durch den Anwender entweder grafisch mit Bibliothekselementen modelliert und dann automatisch in ausführbaren Code übersetzt oder direkt in Form von Algorithmen implementiert. Aktivitäten, die nur innerhalb des SOI ablaufen, werden *endogen* bezeichnet, während Aktivitäten, die in der Umgebung ablaufen und das SOI beeinflussen, als *exogen* bezeichnet werden. [167, 215, 28]

**Interaktionen:** Entitäten stehen mit weiteren Entitäten oder der Umgebung durch Ein- und Ausgänge in Verbindung. Sie können durch **Interaktionen**, wie das Senden von Ereignissen oder Nachrichten, miteinander kommunizieren und gegenseitig ihren Zustand bzw. das davon abhängige Verhalten beeinflussen. [167, 215, 28]

**Modellausführungseinheit:** Zustände und das Verhalten der Entitäten ändern sich mit der Zeit. Zur Ausführung der Simulation wird eine *Modellausführungseinheit (MAE)*[1] benötigt. Verschiedene Implementierungen sind in Unterabschnitt 2.2.4 beschrieben. In hybriden Modellen kann es mehrere Modellausführungseinheiten für ein Modell geben. [167, 215, 28]

**Umgebung:** Systeme sind i.d.R. in eine Umgebung eingebettet, welche Auswirkung auf ihr Verhalten hat. Im sogenannten „Systems Thinking“ ist jedes System ein Subsystem eines anderen Systems. Die Umgebung kann dabei **Stimuli** oder **Störgrößen** vorgeben. Optional passt die Umgebung reaktiv ihre Stimuli auf Zustandsänderungen des SOI an. [167, 215, 28]

### 2.2.1. Prozess und Anwendung im Systems Engineering

Simulation wird im MBSE nach [104, 173] in folgenden Situationen angewendet:

- wenn mathematische Berechnungen an ihre Grenzen stoßen, weil das Problem so komplex ist, dass es sich nicht mit einfachen Gleichungen darstellen lässt.

[1] Alternative Bezeichnung: Model Execution Engine.

- wenn das Verhalten eines realen Systems für Designentscheidungen vorausgesagt werden soll und ein realer Test nicht möglich, zu teuer oder zu zeitaufwendig ist.
- zur Veranschaulichung, wenn ein virtueller Prototyp für Systeme entwickelt werden soll, die es in der Realität noch nicht gibt.
- um das Verhalten eines Systems unter verschiedenen Randbedingungen zu verstehen oder im Zuge einer Zusammenarbeit mit Kollegen zu erklären.
- zur Optimierung des Systemverhaltens.

Der generelle Prozess der Simulation ist nach [23, 167] in Abbildung 2.6 abgebildet.

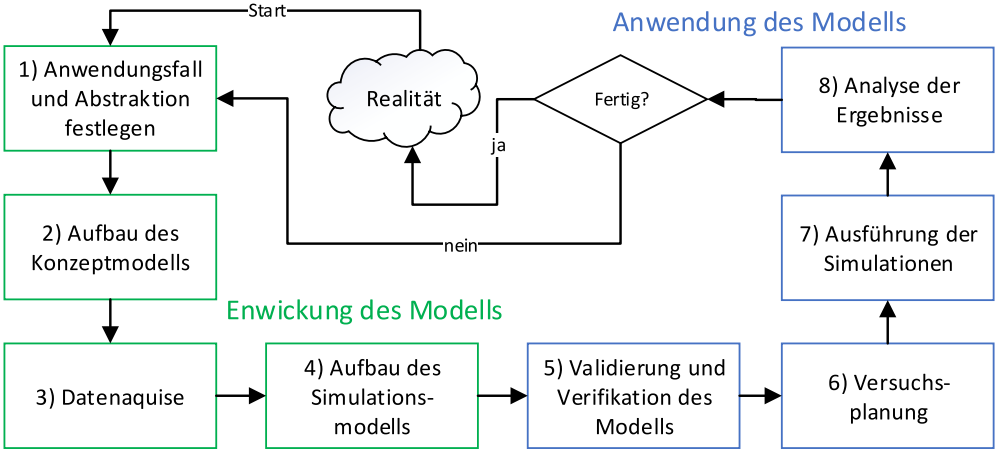



Abbildung 2.6.: Modellierung und Simulationsprozess (Quellen: [23, 167]).

Die Schritte werden in den folgenden Absätzen nach [23, 167] erläutert:

Ausgehend von der **Realität**, also dem Urbild, wird im ersten Schritt die Abstraktion für das Modell festgelegt. So werden nur die für einen **Anwendungsfall** relevanten Aspekte im Konzeptmodell berücksichtigt. Hierbei wird die Struktur der Komponenten innerhalb des Systems und das SOI durch Bestimmung der Systemgrenzen festgelegt. Außerdem werden die Anforderungen spezifiziert. [23, 167]

Im zweiten Schritt steht der **Aufbau des Konzeptmodells** an. Hier werden zunächst die Anforderungen in ein formales oder nicht-formales[1] Modell übersetzt. Dabei sollte die grobe Struktur des Systems vorhanden sein. Ziele sind etwa ein gemeinsames Verständnis bei der Kollaboration und die Aufdeckung von Inkonsistenzen oder Fehlern. [23, 167]

Im dritten Schritt steht die Sammlung von Daten (**Datenakquise**) an, die entweder als Eingabedaten dienen oder für die Beschreibung und Konfiguration des Verhaltens von Entitäten benötigt werden. Damit kann das Modell iterativ verfeinert werden. [23, 167]

[1] Zum Beispiel Zeichnungen.

Im vierten Schritt entsteht aus dem anfänglichen Strukturmodell das **Simulationsmodell**. Dabei wird definiert, welche Art der Modellausführung für die Umsetzung verwendet wird. Relevante Modellausführungsarten sind in Unterabschnitt 2.2.4 aufgeführt. [23, 167]

Die **Validierung und Verifikation** des Simulationsmodells findet im fünften Schritt statt. Erstgenannte umfasst die Prüfung, ob die im ersten Schritt definierten Anforderungen an das Simulationsmodell erfüllt werden und die Ausführung des Modells funktioniert. In der Praxis werden dazu entsprechende Unit-Tests geschrieben und Debug-Tools verwendet. Die Validierung umfasst hingegen die Prüfung, ob das entstandene Simulationsmodell die Realität ausreichend genau für den Anwendungsfall repräsentiert. [23, 167]

Im sechsten Schritt werden die für den Anwendungszweck passenden Versuche definiert (**Versuchsplanung**). Dies schließt auch die Definition der Stimuli, die Konfiguration der zeitlichen Auflösung und die Festlegung der Anzahl an Durchgängen mit ein. [23, 167]

Somit sind die Vorbereitungen abgeschlossen und es kann mit der **Ausführung der Simulationen** im siebten Schritt begonnen werden. Die zuvor definierten, unterschiedlichen Testfälle bzw. Szenarien werden nun ausgeführt. [23, 167]

Im achten Schritt erfolgt die **Analyse der Ergebnisse**. Sind sie verwertbar, so ist der Prozess abgeschlossen. Falls nicht, so werden alle Schritte iterativ verbessert. [23, 167]

#### Methoden zur Validierung von Simulationsmodellen

Zunächst werden im Folgenden Begriffe eingeführt, die im Zusammenhang mit der Validierung von Simulationsmodellen verwendet werden.

Die **Wiedergabetreue** beschreibt, wie gut ein Modell der Realität entspricht, d.h. wie gut *„eine Simulation den Zustand und das Verhalten eines realen Objekts, Merkmals oder Zustands reproduziert“* [202]. Die Wiedergabetreue wird dabei vorwiegend durch den Modellierer bei der Wahl der Abstraktion festgelegt.

Die **Genauigkeit** bezeichnet die Abweichung eines durch Simulation erhaltenen Ergebnisses von einem in der Realität gemessenen [275].

Die **Präzision** bezeichnet die Abweichung der Ergebnisse einer Simulation bei mehrfacher Ausführung mit gleichen Parametern [275].

Ein Simulationsmodell ist **valide**, wenn die Ergebnisse für den jeweiligen Anwendungszweck ausreichend genau sind und dadurch eine Entscheidungsgrundlage gebildet werden kann. Mögliche Methoden zur Validierung sind nach [231, 179, 216] beschrieben:

- **Animation und Betriebsgrafiken:** Die Ergebnisse der Simulation, Zustände von Entitäten oder Messdaten werden grafisch über der Zeit dargestellt, sodass visuell geprüft werden kann, ob die Ergebnisse plausibel [1] sind. Dies kann durch den Modellierer selbst oder durch einen Experten erfolgen.
- **Vergleich mit anderen Modellen:** Die Ausgaben der Simulation werden mit denen von bereits validierten Modellen verglichen.

[1] D.h. nachvollziehbar bzw. theoretisch möglich.

- **Ereignis-Gültigkeit:** Die Ereignisse in der Simulation werden mit den Ereignissen in der Realität abgeglichen.
- **Feste Werte:** Für Eingangsvariablen, interne Variablen und Parameter werden konstante Werte genutzt, um die Ausgangsvariablen mit einfachen Rechnungen zu prüfen.
- **Interne Gültigkeit:** Es werden die Ausgangsvariablen von mehreren Durchläufen miteinander zur Konsistenzprüfung verglichen (Beurteilung der Präzision).
- **Parameteranalyse:** Eingangsvariablen und interne Parameter werden verändert, um die Auswirkung auf das Verhalten und die Ausgangsvariablen zu untersuchen.
- **Traces:** Das Verhalten von ausgewählten Entitäten wird nachverfolgt und auf Plausibilität geprüft.

### Versuchsplanung

Mittels Versuchsplanung soll allgemein der Testaufwand eingeschränkt werden. In Bezug auf die Simulation bedeutet das eine Einschränkung der Anzahl notwendiger Ausführungskonfigurationen und damit der Anzahl der Ausführungen. Eine Ausführungskonfiguration definiert u.a. das zu verwendende Simulationsmodell, dessen Zustand und dafür vorgesehene Stimuli. Dabei resultiert jede Ausführungskonfigurationen aus einem Testfall. Ein Testfall ist nach dem *International Software Testing Qualifications Board (ISTQB)*: „*Eine Menge von Eingaben, Ausführungsvorbedingungen, erwarteten Ergebnissen und Ausführungsnachbedingungen, die für ein bestimmtes Ziel oder eine bestimmte Testbedingung entwickelt wurden, z. B. um einen bestimmten Programmpfad oder die Einhaltung einer bestimmten Anforderung zu überprüfen.*“ [128]. Testfälle ergeben sich damit aus den Anforderungen.

**Statistische Versuchsplanung** Die statistische Versuchsplanung hilft nach [159] eine Menge von Ausführungskonfigurationen der Simulation zu bestimmen, mit denen die Anzahl der Simulationsausführungen eingeschränkt werden kann. Dazu werden stochastische Methoden verwendet. Eine Strategie ist es z.B. diejenigen Eingaben zu identifizieren, die bei einer Änderung die größte Änderung der Ausgaben bewirken. [159]

**Szenarien** Szenarien bieten eine Alternative zur statistischen Versuchsplanung. Mehrere Testfälle werden hier aus den Anwendungsfällen abgeleitet und kombiniert ausgeführt.

„*Ein Szenario beschreibt ein konkretes Beispiel für die Erfüllung bzw. Nichterfüllung eines oder mehrerer Ziele. Es konkretisiert dadurch eines oder mehrere Ziele. Ein Szenario enthält typischerweise eine Folge von Interaktionsschritten und setzt diese in Bezug zum Systemkontext.*“ [212].

Szenarien erlauben die Einbeziehung von Kontextinformationen bei der Validierung. Mithilfe von Szenarien können Akteure entstandene Fehler oder Missverständnisse sowohl im SOI als auch in dessen Kontext aufdecken. [212]

Abbildung 2.7 zeigt nach [212] die verschiedenen Ebenen von Szenarien und die Komponenten, die ein Szenario beschreiben. Sie werden im Folgenden nach [212] erläutert.

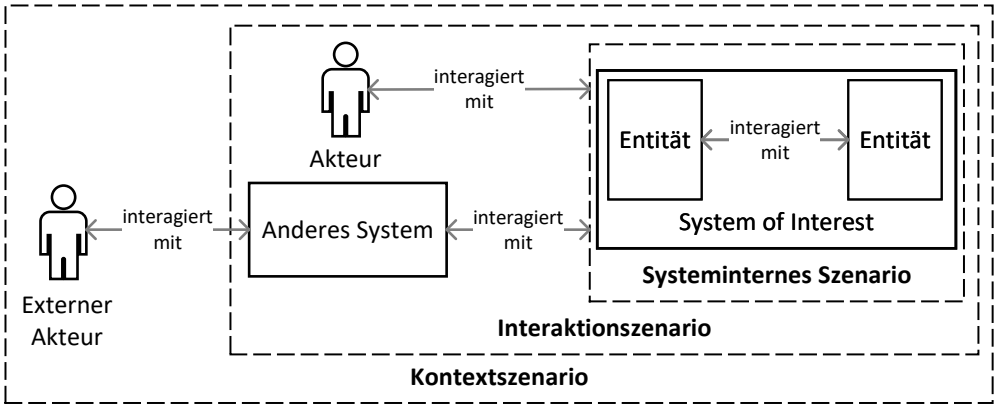



Abbildung 2.7.: Systeminterne, Interaktions- und Kontextszenarien (Quelle: eigene Darstellung nach [212]).

**Systeminterne Szenarien** beschränken sich auf die internen Interaktionen des SOI ohne seine Umgebung. **Interaktionsszenarien** ziehen zusätzlich direkte Interaktionen mit **Akteuren** und **anderen Systemen** aus der Umgebung mit ein. **Kontextszenarien** betrachten zusätzlich noch indirekte Interaktionen mit **externen Akteuren**. Akteure und andere Systeme werden in der Literatur auch als *Agenten* bezeichnet und stellen Entitäten dar, die mit dem SOI interagieren. Sollen Szenarien (simulativ) ausführbar sein, so benötigen sowohl das SOI als auch die beteiligten Akteure ein definiertes Verhalten und entsprechende Zustandsvariablen. Außerdem muss die Reihenfolge der Interaktionen, also der Kontrollfluss, modelliert sein. [212]

*Interaktionsfolgen (IAFs)* können unterschiedliche Ziele verfolgen, die nach [212] zu unterschiedlichen Ausprägungen von Szenarien führen. Diese Szenarien sind im Folgenden nach [212] aufgeführt und beschrieben:

- **Positive Szenarien** werden durch IAFs beschrieben, die zur Erfüllung der Anforderungen führen. Die IAFs **negativer Szenarien** führen zur Nichterfüllung der Anforderungen.
- **Hauptszenarien** sind positive Szenarien und beschreiben durch IAFs den typischen Anwendungsfall eines Systems.
- **Alternativszenarien** stellen eines Varianz des Hauptszenarios dar.
- **Ausnahmeszenarien** sind negative Szenarien und beschreiben innerhalb von IAFs ein Ereignis, das die Erfüllung der Anforderungen verhindert.
- **Missbrauchsszenarien** beschreiben mittels IAFs den Missbrauch eines Systems.
- **Instanzszenarien** beschreiben ganz konkret, welche Agenten an einem Szenario beteiligt sind und wie die Ein- und Ausgaben aussehen.
- **Typenszenarien** abstrahieren Agenten, Ein- und Ausgaben und können dadurch eine Menge von Instanzszenarien beschreiben.

### 2.2.2. Klassifikation von Simulationsmodellen

Simulationsmodelle können nach [16] als *statisch* oder *dynamisch*, *deterministisch* oder *stochastisch* und *kontinuierlich* oder *diskret* klassifiziert werden:

Bei einem **statischen** Modell wird das System nur zu einem einzigen Zeitpunkt modelliert, d.h. die Zeit schreitet nicht voran. Im Gegensatz dazu ändert sich der Zustand bei **dynamischen** Simulationsmodellen mit der Zeit. [16]

**Deterministische** Modelle verwenden im Vergleich zu **stochastischen** Simulationsmodellen keine Zufallsvariablen, d.h. sie haben bei einer bestimmten Eingabemenge immer die gleiche Ausgabemenge. Da stochastische Simulationen bei zufälligen Eingaben auch zufällige Ausgaben liefern, repräsentieren sie nur bedingt die realen Eigenschaften eines Modells, weswegen die Ergebnisse einer Simulation nur statistische Schätzwerte sind. Monte-Carlo Simulationen sind stochastisch und statisch. [16]

Bei **diskreten** Modellen ändert sich der Zustand des Systems nur zu festen Zeitpunkten, während er sich bei **kontinuierlichen** Simulationsmodellen stetig bzw. kontinuierlich ändern kann. Daneben gibt es auch **hybride** Simulationsmodelle, welche kontinuierliche und diskrete Modellanteile enthalten. [16]

### 2.2.3. Modellierungsmethoden

Zur Modellierung von Simulationen gibt es drei Hauptansätze:

1. **Systemdynamik (SD):** Die Methode wird nach [106] vor allem zur Modellierung, Simulation und Analyse in sozioökonomischen Systemen verwendet. Es werden kausale Rückkopplungsschleifen, Wirkungsverzögerungen, Bestandsgrößen, Flussgrößen und Nichtlinearitäten erfasst. Zur Modellierung werden u.a. Rückkopplungsdiagramme, Systemstrukturdiagramm oder Flussdiagramme verwendet. [106]
2. **Discrete Event System Specification (DEVS):** Ursprünglich wurde diese Methode nach [22, 286] als Formalismus zur Modellierung und Simulation ereignisdiskreter Systeme entwickelt. Die Beschreibung des Systemverhaltens erfolgt dabei auf zwei Ebenen. Auf der unteren Ebene wird das Verhalten atomarer Entitäten als Abfolge deterministischer Zustandsübergänge beschrieben. Damit ist beschrieben, wie die Entitäten auf eingehende Ereignisse reagieren, d.h. welche Ausgaben (Ereignisse) davon abhängig erzeugt werden. Auf der oberen Ebene werden die Entitäten (Komponenten) zu einem Netzwerk gekoppelt. Die Verbindungen geben an, wie sich die Komponenten gegenseitig beeinflussen. Durch Quantisierung können außerdem auch kontinuierliche Systeme beschrieben werden. [22, 286]
3. **Agentenbasierte Modellierung (ABM):** Ein Agent ist die Entität eines Simulationsmodells. Jedem Agenten ist nach [64] ein Verhaltensmodell mit möglichen Verhaltensalternativen zugewiesen, welche anhand des Systemzustands selbstständig vom Agenten mittels Bewertung ausgewählt werden. Die Zeit wird bei der Modellierung in Perioden unterteilt. Zustandsübergänge laufen innerhalb dieser Perioden ab. [64]

### 2.2.4. Modellausführungsarten

Die Ausführung des Modells über der Zeit ist als Algorithmus in der MAE implementiert (siehe Abbildung 2.5). Je nach Simulationsart und dem Anwendungsfall erfordert dies unterschiedliche Implementierungen und Konfigurationen. Dies umfasst einerseits das Zeitmodell (kontinuierlich oder diskret) und andererseits das Festlegen der Ausführungsreihenfolge (Scheduling) der Entitäten. Beide Aspekte sind miteinander verzahnt, da viele Zustandsänderungen in einer Simulation zwar zeitgleich ablaufen, also parallel, aber ein Prozessor sie nur begrenzt parallel berechnen kann.

#### Kontinuierliche Simulation

Kontinuierliche Zeitmodelle werden vor allem dann verwendet, wenn eine möglichst große Wiedergabetreue erforderlich ist. Eine Voraussetzung dabei ist nach [28], dass das reale System genau erforscht oder beobachtbar ist. Dies ist bspw. bei physikalischen Systemen der Fall. Die Abhängigkeit eines Systems von der Zeit kann mathematisch durch Funktionen und deren Ableitungen beschrieben werden. Die Lösungen an gewünschten Zeitpunkten werden dann durch sog. Löser bzw. „Solver“ bestimmt, welcher Teil der MAE ist. [28]

Einen Überblick über die in der Praxis eingesetzten Lösungsverfahren zeigt Abbildung 2.8. Es wird hierbei zunächst zwischen *wertkontinuierlichen* Verfahren und *wertdiskreten* Verfahren unterschieden.

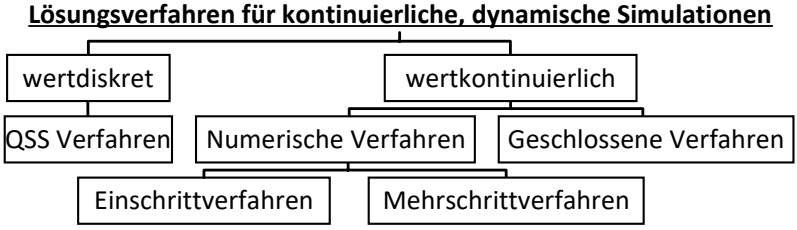



Abbildung 2.8.: Einordnung der Lösungsverfahren für kontinuierliche Simulationen.

**Wertkontinuierliche Verfahren** Die einfachste Form der wertkontinuierlichen Verfahren bilden nach [28] **geschlossene Verfahren**. Die Lösung lässt sich dabei unter Verwendung von Grundrechenarten (+,-,*,/) bestimmen. Ein Beispiel dafür ist:

$$f(t) = 2 * t \tag{2.1}$$

Dabei ist f(t) die gesuchte Lösung in Abhängigkeit von der Zeit t. Geschlossene Verfahren spielen aufgrund ihrer Einschränkung auf Grundrechenarten beim praktischen Einsatz in Simulatoren jedoch keine Rolle. [28]

Die meisten zeitabhängigen Systeme sind deshalb durch Differenzialgleichungen beschrieben, welche von gewöhnlicher, partieller oder beider Natur sein können [167]. Alle Arten der Differenzialgleichungen können dabei nach [28] auf eine einheitliche, kanonische Form gebracht werden, nämlich die Zustandsraumdarstellung:

$$x'(t) = f(x(t), u(t), t) \text{ mit } x(t_0) = x_0 \tag{2.2}$$

$$y(t) = g(x(t)) \tag{2.3}$$

$x(t)$, $u(t)$ und $y(t)$ sind n-dimensionale Vektoren, welche die Zustands-, Eingangs- und Ausgangsvariablen repräsentieren. Die Zustandsraumdarstellung ist die Ausgangsbasis für **numerische Verfahren** zur Lösung des Gleichungssystems. [28]

Zu beachten ist nach [24], dass bei numerischen Verfahren die exakte Lösung nur angenähert wird. Deshalb ist die Angabe der Näherungsgenauigkeit eines Lösungsverfahrens bei seinem Einsatz für einen bestimmten Anwendungsfall relevant. Diese wird durch den lokalen Fehler innerhalb eines Integrationsschritts oder den globalen Fehler als Summe aller lokalen Fehler bestimmt. Üblicherweise wird hierbei der Fehler eines Verfahrens bzw. die sogenannte „Konsistenz“ in der O-Notation angegeben:

$$O(h^p) \tag{2.4}$$

Dabei ist $h$ eine Schrittweite > 0 und $p = \nu - 1$ die Konsistenzordnung, welche sich aus der lokalen Fehlerordnung $\nu$ bestimmt. Je genauer das Ergebnis sein soll, desto höher muss die erforderliche Ordnung des Lösungsverfahrens und desto kleiner die Schrittweite gewählt werden. Höhere Ordnungen und kleinere Schrittweiten führen jedoch generell zu einer längeren Rechenzeit. Neben der Ordnung unterscheiden sich die Lösungsverfahren weiter darin, ob eine feste oder variable Zeitschrittweite gewählt wird und ob es sich um ein Ein- oder Mehrschrittverfahren handelt. Bei **Einschrittverfahren** wird in jedem Zeitschritt das Ergebnis ausgehend von der letzten bestimmten Näherung berechnet. Vertreter davon sind z.B. das Eulerverfahren und das Runge-Kutta-Verfahren. Bei **Mehrschrittverfahren** werden zusätzlich auch Näherungen bzw. Stützpunkte aus weiter zurückliegenden Zeitschritten verwendet. Vertreter davon ist bspw. das Adams-Bashforth Verfahren. [24]

**Wertdiskrete Verfahren** Bei dem numerischen *Quantized-State-System-Verfahren* (**QSS-Verfahren**) wird der Zustandsvektor x(t) quantisiert. Aus Gleichung 2.2 folgt nach [22]:

$$x'(t) \approx f(\mathbf{q}(t), u(t), t) \tag{2.5}$$

$q(t)$ ist der quantisierte Zustandsvektor. Bei der Wahl der Quantisierungsfunktion ist zu beachten, dass Zustände am Ende nicht mit unendlicher Frequenz geändert werden, wozu sich Quantisierungsfunktionen mit Hysterese eigenen [22]. Durch die Quantisierung kann die Ausführung kontinuierlicher Simulationen beschleunigt und insbesondere parallelisiert werden [87].

### Diskrete Simulation

Diskrete Zeitmodelle können nach den in Abbildung 2.9 dargestellten Grundformen der Zeitsteuerung unterteilt werden. Bei der diskreten, dynamischen Simulation wird zwischen wertdiskreten (qualitativen) und zeitdiskreten Ansätzen unterschieden.

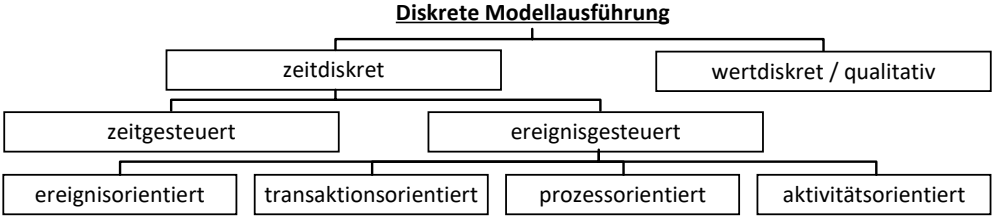



Abbildung 2.9.: Zeitsteuerungsformen der diskreten Simulation (Quelle: eigene Darstellung nach [207, 203]).

**Wertdiskreter Ansatz** Beim wertdiskreten/qualitativen Ansatz sind die Zustandsvariablen nach [49] diskretisiert, d.h. sie können nur zuvor festgelegte Werte annehmen. Die Zeitschritte sind dabei ebenso qualitativ, womit die Einheit der Zeit kein Quantum. Bekannte Vertreter von qualitativen Modellen sind *Finite State Machines (FSMs)*, zu Deutsch Zustandsautomaten, oder Petrinetze. [49]

**Zeitdiskrete Ansätze** Die zeitdiskreten Ansätze unterteilen sich in zeitgesteuerte und ereignisgesteuerte. Der Unterschied ist in Abbildung 2.10 dargestellt.

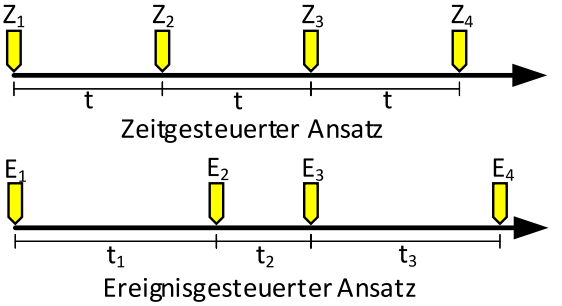



Abbildung 2.10.: Zeitgesteuerter und ereignisgesteuerter Ansatz (Quelle: eigene Darstellung nach [298]).

Beim **zeitgesteuerten Ansatz** ist die Zeitskala nach [207, 176] äquidistant (**t**) aufgeteilt, wobei sich eine Zustandsänderung des Modells nur an fixen Zeitpunkten ($\boldsymbol{Z_i}$) ergeben kann. Die Her-

ausforderung hierbei ist die Festlegung der Schrittweite. Eine zu klein gewählte führt zu unnötiger Totzeit, während eine zu groß gewählte zum Verpassen von Ereignissen führen kann. Als *Totzeit* wird die ereignislose Zeit zwischen zwei Zustandsänderungen bezeichnet. [207, 176]

Beim **ereignisgesteuerten Ansatz** werden nach [207, 173] keine festen Zeitpunkte durchlaufen, da nur Ereignisse ($E_i$) zu Zustandsänderungen führen. Ein Ereignis führt zur Zustandsänderung einer Entität oder des Systems und hat einen Ereigniszeitpunkt. Die globale Simulationszeit schreitet dann von Ereigniszeitpunkt zu Ereigniszeitpunkt ($t_i$) voran, d.h. nicht zwingend in äquidistanten Schritten. [207, 173]

Die in Abbildung 2.9 aufgelisteten ereignisgesteuerte Ansätze werden im Folgenden nach [173, 207, 203, 160, 157] unter Beachtung der *Ablaufkontrollen*[1] beschrieben:

- **Ereignisorientiert:** Zustandsänderungen in den Modellen können nur zu Ereigniszeitpunkten stattfinden. *Aktivitäten* sind eine Folge von Ereignissen. Ein Modell besteht aus *statischen Entitäten*, die permanent vorhanden sind und änderbare Zustandsvariablen besitzen, sowie *dynamischen Entitäten*, welche entweder die Ereignisse als *Ereignisroutinen* repräsentieren oder temporäre Objekte darstellen, die dynamisch erzeugt oder vernichtet werden können. Ereignisroutinen fügen der **Ereignisliste** neue und in der Zukunft geplante Ereignisse hinzu oder entfernen bereits eingeplante (vgl. Abbildung 2.11). Innerhalb der Ablaufkontrolle wird die Ereignisliste sequenziell abgearbeitet. Ereignislose Totzeiten werden übersprungen. Im Vergleich zu den anderen Ansätzen ist dieser meist am performantesten.

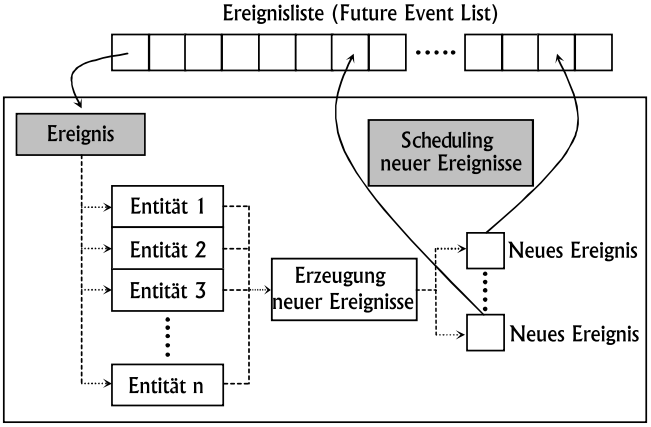



Abbildung 2.11.: Ablauf des Schedulings beim ereignisorientierten Ansatz (Quelle: Modifikation von [160]).

- **Prozessorientiert:** Ein *Prozess* bezeichnet die auf eine bestimmte Entität bezogene Folge von Ereignissen. Er kann entweder aktiv sein, wenn er ausgeführt wird und Zustandsänderungen auslöst, oder inaktiv. Die Simulationszeit schreitet nur in inaktiven Phasen voran, um die Dauer des Prozesses zu simulieren. Ein Prozess kann von einer aktiven Phase in eine in-

[1] Algorithmen, die zur Modellausführung in der MAE implementiert werden können.

aktive übergehen und zu einem späteren Zeitpunkt reaktiviert werden. Dann fährt er an der Stelle fort, an der zuvor aufgehört hat. Innerhalb der Ablaufkontrolle gibt es eine Ereignisliste, in der die Prozesse nach ihren Ausführungszeitpunkten geordnet sind. Ein Hauptprozess arbeitet diese Liste ab und aktiviert, reaktiviert oder terminiert die einzelnen Prozesse.

- **Transaktionsorientiert:** Zur Modellierung werden dynamische Komponenten, *Transaktionen* genannt (→), und statische Komponenten, *Knoten* genannt (**A**,**B**,**C**,**D**,**E**), verwendet. Knoten bilden einen Graphen, durch den die Transaktionen fließen. Wenn Transaktionen einen Knoten traversieren, können sie um $t_i$ verzögert werden und eine Zustandsänderung bewirken (vgl. Abbildung 2.12). Transaktionen und Knoten besitzen Zustandsvariablen und beeinflussen sich gegenseitig. Ein Aufeinandertreffen von ihnen wird als Ereignis bezeichnet. Innerhalb der Ablauflaufkontrolle werden Ereignisse in einer Ereignisliste nach Prioritäten geordnet abgearbeitet.

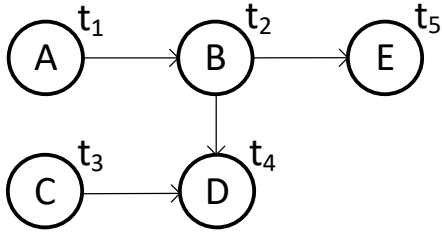



Abbildung 2.12.: Transaktionsorientierter Ansatz (Quelle: eigene Darstellung nach [173]).

- **Aktivitätsorientiert**: Eine **Aktivität** beschreibt einen Zustandsübergang, der durch eine Dauer, ein Anfangs- und ein Endereignis charakterisiert ist. Innerhalb der Ablaufkontrolle werden Ereignisse ($E_i$), die eine Aktivität auslösen, in einer Ereignisliste eingeplant. Dazu werden zum entsprechenden Zeitpunkt alle Bedingungen, die durch ein Ereignis eine Aktivität auslösen können, überprüft. Da eine Aktivität weitere Aktivitäten auslösen kann, werden auch die von einer Aktivität abhängigen Aktivitäten eingeplant. Die Simulationszeit schreitet immer entsprechend der Aktivität mit der geringsten Dauer voran. Nach dem Voranschreiten beginnt die Einplanung neuer, aktivitätenauslösender Ereignisse erneut. Jede Aktivität verbraucht dabei Simulationszeit. Der Unterschied zwischen Aktivität, **Prozess** und Ereignis ist in Abbildung 2.13 zusammengefasst.

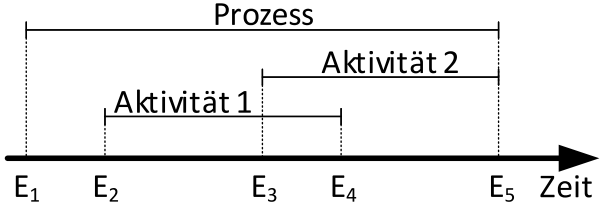



Abbildung 2.13.: Aktivitätsorientierter Ansatz (Quelle: eigene Darstellung nach [157, 203]).

#### Hybride Simulation

Der Begriff „*Hybride Simulation (HS)*“ hat sich im Laufe der Zeit gewandelt. Nach [79] wurde zu Beginn eine kombinierte Simulation aus diskreten und kontinuierlichen Modellanteilen darunter verstanden. Heute ist vor allem der kombinierte Einsatz von mindestens zwei der üblichen Modellierungsmethoden (DEVS, SD und ABM) damit gemeint [79]. Ähnliche Ausdrücke in der Literatur für HS lauten *Multimethoden-*, *paradigmenübergreifende* oder *kombinierte Simulation* [187]. Eine allgemeingültige Definition für den Begriff gibt es nicht. In dieser Arbeit ist HS jedoch im klassischen Sinne zu verstehen, d.h. wenn bei einer Simulation diskrete und kontinuierliche Modellausführungsarten kombiniert werden. Dies ist häufig der Fall, wenn ein System aus mehreren funktionalen Domänen besteht. CPS und insbesondere E/E-Architekturen sind Vertreter dieser Systeme. Bei ihren Simulationen wird auch von „Multi-Domänen-Simulationen“ gesprochen.

Bei der Kombination unterschiedlicher Modellierungsarten für eine HS werden nach [34, 89] folgende Arten unterschieden und beschrieben:

- **Sequentiell**: Modelle unterschiedlicher Modellierungsmethoden werden nacheinander und jeweils in ihrer entsprechenden Ausführungsart ausgeführt. Die Ausgabe eines Modells wird zur Eingabe des Nachfolgenden.
- **Anreichernd**: Eine dominante Modellierungsmethode bestimmt die Ausführungsart und wird mit weiteren Modellierungsmethoden angereichert.
- **Interaktion**: Für jede Modellierungsmethode werden Teilmodelle genutzt, die in einem gemeinsamen Modell kombiniert und jeweils mit ihrer entsprechenden Ausführungsart ausgeführt werden. Sie interagieren hierbei zyklisch miteinander.
- **Integration**: Es handelt sich um ein nahtloses Gesamtmodell, bei dem es unmöglich ist, eine Modellierungsmethode von der anderen zu unterscheiden.

### 2.2.5. Parallele und verteilte Simulation

Unter **paralleler Simulation** versteht man die Ausführung einer Simulation auf einem Mehrkernprozessorsystem oder lokalem Computer-Cluster. Von **verteilter Simulation** wird gesprochen, wenn eine Simulation auf mehreren, geografisch verteilten Computersystemen ausgeführt wird, die über ein *Local Area Network (LAN)* oder das Internet miteinander verbunden sind. In beiden Fällen wird das Ziel einer Ausführungsbeschleunigung der Simulation verfolgt. Das zentrale Problem hierbei ist nach [94] die Partitionierung, also die Aufteilung der Simulation auf mehrere Rechenkerne oder Rechner. Voneinander kausal abhängige Teilergebnisse müssen nämlich zeitlich synchronisiert sein, um die Korrektheit der Simulation zu gewährleisten. Hierzu werden entsprechende Synchronisierungsalgorithmen innerhalb der Zeitsteuerung benötigt. [94]

### Definitionen von Zeit in der Simulation

Die (logische) **Simulationszeit** ist nach [93] die abstrakte Zeit in der Simulation, die verwendet wird, um die reale bzw. **physikalische Zeit** zu simulieren. Die Zeit, die während der Ausführung der Simulation in der Realität vergeht, wird als **Simulationslaufzeit**[1] bezeichnet. [93]

### Definition von Kausalität

Als **Kausalität** wird nach [239] der Zusammenhang zwischen Ursache und Wirkung bezeichnet. Wenn ein Eingang eines Simulationsmodells von einem seiner Ausgänge abhängt, wird dieser Zusammenhang als **Kausalitätsschleife** bezeichnet [215].

### Synchronisierungsalgorithmen für diskrete Zeitausführungsformen

Bei diskreten Simulationen wird nach [93] zwischen *konservativen* und *optimistischen* Algorithmen unterschieden. Sie werden im Folgenden nach [93] beschrieben:

**Konservative Algorithmen** verfolgen das Ziel bei einer parallelen/verteilten Simulation exakt die gleichen Ergebnisse zu liefern, wie bei der Ausführung auf einem einzelnen Kern. Die Entitäten eines Simulationsmodells implementieren *Logische Prozesse (LPs)* und tauschen mit einem Zeitstempel versehene Ereignisse aus. Ein früher Vertreter der konservativen Algorithmen ist der *Chandy/Misra/Bryant-Algorithmus (CMB-Algorithmus)* [50]. Hierbei wird nach [93] die Ausführung eines LPs solange geblockt, bis feststeht, dass kein Ereignis mehr mit einem Zeitstempel empfangen wird, der zeitlich vor dem Zeitstempel des am nächsten zu sendenden Ereignisses liegt. Zur Realisierung werden die Ereignisse nach aufsteigender Reihenfolge ihrer Zeitstempel sortiert und versendet. Jedem Eingang eines LPs ist ferner eine *First-In-First-Out-Warteschlange (FIFO-Warteschlange)* zugewiesen. Sobald die Warteschlange an einem der Eingänge des LPs leer ist, blockt der LP. Dadurch wird die Ankunft eines Ereignisses ausgeschlossen, dessen Zeitstempel zeitlich vor dem Zeitstempel des vom LP am nächsten zu senden geplanten Ereignisses liegt. Dieser Umstand kann zu einem *Deadlock* führen, wenn die beteiligten LPs sich in einer Schleife befinden und sich gegenseitig blocken. Daher werden zu jedem Zeitschritt an allen Ausgängen der LPs Nachrichten gesendet. Gibt es kein Ereignis, so werden Nachrichten ohne Inhalt (Null-Nachrichten), aber mit der voraussichtlichen Sendezeit des nächsten Ereignisses im Zeitstempel, gesendet. Die Differenz zwischen der Simulationszeit beim Versand einer Null-Nachricht und der Zeit im Zeitstempel dieser Null-Nachricht wird als *Lookahead* bezeichnet. Ist der Lookahead zu klein gewählt, kann es in Schleifen zum sogenannten *Lookahead-Creep* kommen, einer Laufzeiterhöhung durch zu viele verschickte Null-Nachrichten. Um den Lookahead-Creep zu vermeiden, gibt es weitere, konservative Algorithmen, die durch eine globale Ereignisliste die exakte Bestimmung des jeweils optimalen Lookaheads erlauben und dadurch Null-Nachrichten vermeiden. Ein weiterer Vertreter der konservativen Algorithmen ist das YAWNS Protokoll [191]. Hier wird ein synchroner Algorithmus mit globalen Synchronisierungspunkten verwendet. [93]

[1] Wird auch Wanduhrzeit (engl. „Wallclock time“) genannt.

Bei **optimistischen Algorithmen** werden nach [93] Fehler in Kauf genommen und durch Wiederholungen (Rollback) beseitigt. Hierzu zählt etwa der *Time-Warp-Algorithmus* [138].

#### Parallele Ausführung kontinuierlicher Simulationen

Bei kontinuierlichen Simulationen gibt es ebenfalls Wege eine Simulation auf mehrere Recheneinheiten aufzuteilen, d.h. Gleichungssysteme verteilt/parallel zu berechnen. Man unterscheidet dabei Ansätze basierend auf dem *Mehrfachschießverfahren*, der *Waveform-Relaxation*, der *Domain Decompensation*, dem *Mehrgitterverfahren* und der *Zeitparallelen Methode* [197]. QSS-Verfahren sind ebenfalls parallelisierbar [87].

#### Einschränkungen

Einer erhöhten Ausführungsgeschwindigkeit bei der Parallelisierung steht nach [7, 46] vorwiegend die auftretende Latenz bei der nötigen Synchronisation zwischen den Recheneinheiten entgegen. Außerdem müssen durch Schleifen verursachte kausale Abhängigkeiten (vgl. CMB-Algorithmus) entsprechend behandelt werden, was zu einem weiteren Zeitaufwand führt [93].

#### Co-Simulation

Eine Co-Simulation ist nach [245] definiert als die „ *[..] koordinierte Ausführung von zwei oder mehr Modellen [...] in verschiedenen Ausführungsumgebungen.*“ [245]. Ausführungsumgebungen können z.B. Simulatoren sein. Nach [245] werden einzelne Teilmodelle im Gegensatz zur parallelen/verteilten Simulation unabhängig voneinander entwickelt und implementiert. Die Realisierung einer Co-Simulation erfolgt entweder **spezifisch**, durch eigens entwickelte Co-Simulationsschnittstellen (**Interfaces**), oder **generisch**, durch die Verwendung von Standards[1], die ein Rahmenwerk (**Framework**) dafür anbieten (vgl. Abbildung 2.14).

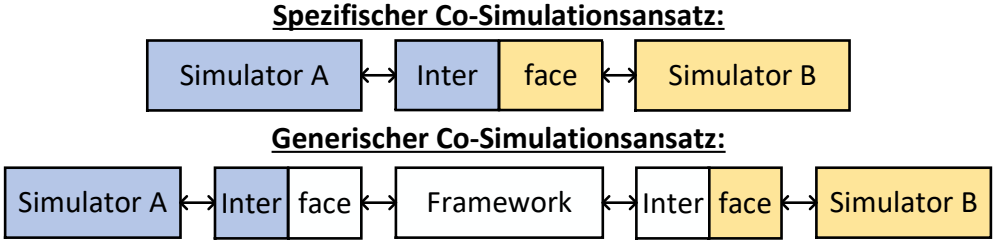



Abbildung 2.14.: Spezifische und generische Co-Simulation (Quelle: eigene Darstellung nach [245]).

[1] Zum Beispiel der *High-Level-Architecture-Standard (HLA-Standard)* [126] oder der *Functional-Mockup-Interface-Standard (FMI-Standard)* [183].

Bei beiden Varianten muss sowohl der Datenaustausch als auch die Zeitsynchronisation zwischen den Simulatoren koordiniert werden. Beim spezifischen Ansatz wird ein Simulator als Hauptsimulator definiert (*Master*), der diese Zeitsynchronisation implementiert und so das Fortschreiten der Zeit in den anderen Simulatoren koordiniert. Beim generischen Ansatz wird das Co-Simulation-Framework zum Master. [245]

## 2.3. Skalierbarkeit

Der Begriff „Skalierbarkeit“ ist in der Literatur nicht eindeutig definiert. Dies wurde bereits 1990 in [116] festgestellt, wobei gängige Definitionen verglichen wurden und der neue Versuch einer Definition scheiterte. Zehn Jahre später werden in [30] die Charakteristiken der Skalierbarkeit und deren Auswirkungen auf die Performanz beschrieben. Allgemein beschreibt [30] Skalierbarkeit als *„[...] die Fähigkeit eines Systems, sich an eine wachsende Anzahl von Elementen oder Objekten anzupassen, wachsende Arbeitsumfänge performant zu verarbeiten und/oder erweiterungsfähig zu sein.“*. Ferner werden nach [30] folgende Arten der Skalierbarkeit unterschieden und beschrieben:

- **Lastskalierbarkeit:** Die Fähigkeit eines Systems, ohne Verzögerung und unnötigem Ressourcenverbrauch, bei leichter, mittlerer oder hoher Last zu funktionieren.
- **Raumskalierbarkeit:** Der Speicherbedarf eines Systems steigt mit zunehmender Anzahl an Elementen nicht auf ein unvertretbares Maß.
- **Raum-Zeit-Skalierbarkeit:** Ein System funktioniert noch ordnungsgemäß, auch wenn die Anzahl seiner umfassenden Objekte um viele Größenordnungen steigt.
- **Strukturskalierbarkeit:** Die Implementierung eines Systems behindert nicht, oder nicht innerhalb eines Zeitrahmens, das quantitative Wachstum seiner umfassenden Objekte.

Nach [171] wird des Weiteren noch zwischen *administrativer*, *funktionaler*, *geografischer* und *Generationsskalierbarkeit* differenziert. All diese Unterscheidungen liefern mögliche Dimensionen, in deren Abhängigkeit die Skalierbarkeit gemessen werden kann.

### 2.3.1. Begriffe

Ein Aspekt des Skalierbarkeitsbegriffs in dieser Arbeit ist die Performanz. Zur Beurteilung werden benötigte Rechenzeiten und der Speicherbedarf eines Systems in Abhängigkeit der Anzahl der zu verarbeitenden Elemente pro Zeit und den dafür zur Verfügung stehenden Ressourcen untersucht. Dafür benötigte Begriffe werden in den folgenden Abschnitten definiert.

#### Scaleup

Für die Skalierbarkeit (*Scaleup*) S folgt nach [63, 192]:

$$S = \frac{t_{Antwort}\,(1 \cdot P, 1 \cdot R)}{t_{Antwort}\,(N \cdot P, N \cdot R)} \tag{2.6}$$

Es gilt $t_{Antwort}$ = Antwortzeit, $P$ = Problemgröße, $R$ = Anzahl der Ressourcen und $N$ = Multiplikationsfaktor.

Die Performanz lässt sich damit nun dahingehend untersuchen, ob größere Probleme durch das Hinzufügen weiterer Ressourcen schneller ($S > 1$) , genauso schnell ($S = 1$) oder langsamer ($S < 1$) gelöst werden. Ist $S >= 1$ erfüllt, dann *skaliert* die Lösung. Zur Bestimmung der Grenzen der Skalierbarkeit lässt sich **S** in Abhängigkeit vom Multiplikationsfaktor **N** untersuchen, wie in Abbildung 2.15 dargestellt.

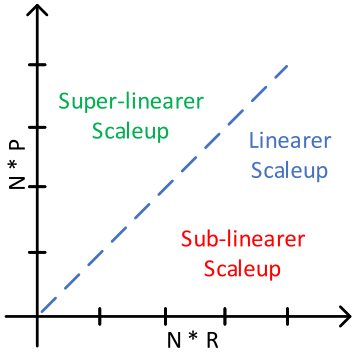



Abbildung 2.15.: Grenzen der Skalierbarkeit.

Solange S im **super-linearen** oder **linearen Bereich** liegt, skaliert die Lösung.

Kann nur P, nicht aber R geändert werden, lässt sich aus Gleichung 2.6 folgern, dass S höchstens proportional zu P steigen darf, wenn die Lösung skalieren soll. Für den von der Problemgröße abhängenden Scaleup $S_p$ gilt demnach:

$$S_p = \frac{N \cdot t_{Antwort}(1 * P)}{t_{Antwort}(N * P)} >= 1 \quad \text{bzw.} \quad S_p = \frac{N \cdot Speicherverbrauch(1 * P)}{Speicherverbrauch(N * P)} >= 1 \tag{2.7}$$

Die Skalierbarkeit kann des Weiteren durch zusätzliche Randbedingungen beschränkt sein. Bei Simulationen, die in Echtzeitzeit ablaufen müssen, wie bspw. bei einer Kopplung der Simulation mit einem realen System, muss die logische Zeit mindestens so schnell wie die physikalische Zeit voranschreiten. Damit die Lösung skaliert, muss Folgendes für den *zeitabhängigen Scaleup* $S_t$ erfüllt sein:

$$S_t = \frac{\text{Logische Zeit}}{\text{Physikalische Zeit}} >= 1 \tag{2.8}$$

Daneben lassen sich weitere messbare Grenzen der Skalierbarkeit definieren, etwa in Abhängigkeit von der Genauigkeit oder des Energieverbrauchs.

Mit $S_p^*$ und $S_t^*$ seien ferner die reziproken Werte für $S_p$ und $S_t$ bezeichnet. Sie geben eine Einschätzung darüber, wie nahe sich ein System an der Grenze der Skalierbarkeit befindet. $S_p^*$ und $S_t^*$ müssen jeweils $<= 1$ sein, wenn ein System skalieren soll.

#### Speedup

*„Der Speedup ist ein allgemeines Maß zur Bestimmung, in welchem Umfang die Leistungsfähigkeit von Computersystemen durch eine bestimmte Optimierung verbessert wird“* [217].

Nach [192] ist der *Speedup* ferner definiert als:

$$Speedup = \frac{t_{Antwort}(\text{ursprüngliches System})}{t_{Antwort}(\text{optimiertes System})} \tag{2.9}$$

Es gilt $t_{Antwort}$ = Antwortzeit eines Programms.

Der Speedup gibt damit an, um welchen Faktor eine optimierte Lösung im Vergleich zur ursprünglichen schneller ist. So kann z.B. die Beschleunigung beim Wechsel von einer seriellen auf eine verteilte/parallele Ausführung bestimmt werden.

#### Effizienz

Die Effizienz ist nach [53] definiert durch:

$$Effizienz = \frac{Speedup}{Speedup_{max}} \tag{2.10}$$

Um den maximal möglichen Speedup zu ermitteln, wird die vom Computersystem abhängige Grenze $Speedup_{max}$ benötigt. Methoden zur Bestimmung dieser Grenzen werden im Nachfolgenden Unterabschnitt 2.3.2 beschrieben.

### 2.3.2. Grenzen des Speedups

Zur Erhöhung der Ressourcen sind nach [3, 171] zwei Möglichkeiten beschrieben:

1. **Horizontale Skalierung**: Die Rechenlast wird über mehrere Maschinen (Ressourcen), wie z.B. Server, verteilt. Auf diesen Maschinen können unterschiedliche Betriebssysteme laufen. Ein weiterer Begriff dafür ist *„scaling out“*.
2. **Vertikale Skalierung**: Einer einzelnen Maschine werden leistungsfähigere Komponenten hinzugefügt bzw. schwächere Komponenten ersetzt. In der Regel läuft auf dieser Maschine ein einziges Betriebssystem. Ein weiterer Begriff hierfür ist *„scaling up“*.

In der Vergangenheit setzte eine vertikale Skalierung im Vergleich zu einer horizontalen normalerweise keine Anpassungen zur Parallelisierung der Software voraus. Jedoch ist die vertikale Skalierung durch erzielbare Taktraten begrenzt, was in den 2000er Jahren zum Einzug

von Mehrkernprozessoren geführt hat. „*Das Gratisessen ist vorbei*“ [249], womit die für Softwareentwickler kostenlose Steigerung des Speedups durch schnellere Taktraten gemeint ist. Eine Steigerung des Speedups durch eine horizontale Skalierung ist jedoch ebenfalls begrenzt, was bereits 1967 in „Amdahls Gesetz“ [7] begründet wurde:

$$Speedup_{max} = \frac{1}{r_s + \frac{r_p}{n}} \tag{2.11}$$

Es gilt hierbei $r_s + r_p = 1$, wobei $r_s$ den sequenziellen und $r_p$ den parallelen Anteil eines Programms darstellt, der auf $n$ Recheneinheiten aufgeteilt wird.

Es folgt daraus, dass der Speedup bei einer zunehmenden horizontalen Skalierung durch den sequenziellen Anteil in der Software begrenzt wird. Außerdem führt die Datenverwaltung, die bei der Synchronisation entsteht, zu einem weiteren, sequenziellen Anteil [7]. Dabei wird die Problemgröße jedoch als konstant angenommen. Den Spezialfall, bei dem der nicht parallelisierbare Teil eine von der Problemgröße unabhängige und konstante Laufzeit hat, beschreibt das „Gustafson-Gesetz“ [218]. Nach [108, 218] folgt damit:

$$Skalierter\ Speedup_{max} = \frac{r_s + r_p(P,1))}{r_s + r_p(P,n)} \tag{2.12}$$

Es gilt $P$ = Problemgröße und $n$ = Anzahl der Recheneinheiten.

### 2.3.3. Methoden zur Untersuchung der Skalierbarkeit

Zur Untersuchung der Performanz eines Systems werden, je nach Anwendungsfall, die Antwortzeit oder der Speicherverbrauch benötigt. Es ist zu beachten, dass die Ergebnisse von der Konfiguration des Systems und den Eingabedaten abhängen. Die Performanz eines Systems lässt sich nach [136] auf zwei Arten bestimmen, nämlich *modellbasiert* oder durch *Messungen*. Bei der Auswahl der Methode kommt es auf verschiedene Faktoren, wie die benötigte Genauigkeit des Ergebnisses, die zur Verfügung stehende Zeit, den aktuellen Entwicklungsstand des Systems oder das Kostenbudget an [136]. In den folgenden Abschnitten werden die einzelnen Methoden genauer vorgestellt.

#### Modellbasierte Performanzuntersuchung

Modelle können nach [178] zu jedem Entwicklungsstand eingesetzt werden und sind verhältnismäßig günstig. Die Genauigkeit ist aber, im Vergleich zu einer Simulation oder Messung, geringer. Es handelt sich viel mehr um eine Vorhersage der Performanz. Zur Modellierung werden mathematische *Performanzmodelle* der Software verwendet. Dazu gehören Warteschlangennetze, stochastische Petrinetze oder stochastische Prozessalgebren. In der Regel können Softwaremodelle in Performanzmodelle transformiert werden, wenn das zugrunde liegende Metamodell entsprechende Annotationen unterstützt. Zu diesen Annotationen gehören nach [21] Ressourcenanforderungen, Eingabeparameter oder das Benutzerverhalten. [178, 21]

Performanzmodelle, die rein mathematisch berechnet werden können, werden als *analytische Modelle* bezeichnet [136]. Wenn analytische Modelle aufgrund ihrer Genauigkeit nicht ausreichen, weil die Architektur vernachlässigt wird, kann ein entsprechender Performanz-Simulator verwendet werden [178]. Ein Beispiel dafür ist der „Palladio Software Architecture Simulator“ [222].

### Messbasierte Performanzuntersuchung

Messungen können erst dann durchgeführt werden, wenn ein Prototyp vorhanden ist. Die Ergebnisse von Messungen liegen zwar näher an der Realität als die von modellbasierten Methoden, weißen dafür aber eine höhere Varianz auf. [136]

In den nachfolgenden Abschnitten werden Methoden zur Messung der Performanz beschrieben, welche die Messgrößen Laufzeit und Speicherbedarf umfassen.

**Einfache Methoden** Messungen der Laufzeit können im einfachsten Fall mit einer Stoppuhr erfolgen, wenn das zu untersuchende Zeitintervall groß genug ist und kein Zugriff auf den Quellcode besteht. Für ein Messintervall werden sowohl ein Start- als auch ein Endpunkt festgelegt. Das kann entweder ein Ereignis oder ein Zustand sein. Ein Messintervall kann so etwa zwischen dem Einschalten eines Computersystems (Startereignis) und dem anschließenden Erreichen des Startbildschirms (Endzustand) liegen. Falls Zugriff auf den Quellcode besteht, ist eine (automatisierte) Laufzeitermittlung vorzuziehen, da das Ergebnis genauer ist. Eine Möglichkeit hierzu ist die Differenzbildung der physikalischen Zeit zwischen zwei Code-Stellen.

Der Speicherverbrauch kann analog bestimmt werden. Besteht Zugriff auf den Quellcode, kann der Verbrauch beim Erreichen entsprechender Programmstellen zur Prüfung ausgegeben werden. Anderenfalls können Bordmittel des Betriebssystems verwendet werden.

Insgesamt unterliegen diese Methoden entsprechenden Varianzen und Toleranzen, sodass die Messungen wiederholt durchzuführen sind, um aussagekräftige Ergebnisse zu erhalten. Ist das Zeitintervall jedoch sehr groß, dann relativieren sich die Abweichungen und die Anforderungen an die Messgenauigkeit sinken. In diesem Fall kann auf Wiederholungen verzichtet werden, da sie die Genauigkeit nicht signifikant erhöhen.

**Performanz-Profiling** Performanz-Profiling bezieht sich in dieser Arbeit auf die Verwendung einer Software, genannt *Profiler*. Damit lassen sich sowohl Laufzeiten und Speicherverbrauch bestimmen als auch gezielt Schwachstellen (*Hotspots*) identifizieren. In den meisten Fällen arbeiten Profiler Sample-basiert, d.h. die Ausführung der Anwendungssoftware wird periodisch unterbrochen, um die relevanten Zustandsinformationen der Software zu speichern [151]. Die Genauigkeit dieser Zeitmessung lässt sich über ein Konfidenzintervall bestimmen [151].

**Ereignis-Tracing** Beim Ereignis-Tracing werden nach [151] Ereignisse und entsprechende Informationen bei der Ausführung eines Programms aufgezeichnet. Im Vergleich zum Performanz-Profiling lässt sich die Laufzeit in Abhängigkeit des Kontrollflusses untersu-

chen. Somit können hier Schwachstellen im Programmablauf, wie der Aufruf zu vieler Schleifendurchgänge, aufgedeckt werden. [151]

### 2.3.4. Benchmarking

Nach [243] ist „*ein Benchmark [...] ein Test oder eine Reihe von Tests, die dazu dienen, die Performanz eines Computersystems mit der Performanz anderer zu vergleichen.*“. Daneben lässt sich auch die Performanz eines Systems bei unterschiedlichen Konfigurationen vergleichen. Für den Vergleich werden nach [162] zwei Strategien beschrieben:

- **Konstante Problemgröße:** Vergleich der benötigten Zeit oder des Speicherverbrauchs bei der Lösung einer festen Problemgröße.
- **Konstante Zeit:** Vergleich der maximal lösbaren Problemgröße bei fester Zeit.

Je nach Problemart werden nach [151, 162] die folgenden Benchmark-Typen definiert:

- **Synthetische Benchmarks:** Es werden künstliche Probleme verwendet, die die Eigenschaften einer Gruppe von Anwendungsfällen nachbilden. Sie werden eingesetzt, um die Grenzen eines Systems unter verschiedenen Bedingungen zu überprüfen.
- **Microbenchmarks:** Es werden kleine Probleme verwendet, um gezielt nur einen spezifischen Teil eines Systems hinsichtlich seiner Performanz zu bewerten.
- **Kernelbenchmarks:** Kleine und verallgemeinerbare Probleme decken die Untersuchung der Teile eines Systems ab, die den größten Teil der gesamten Ausführungszeit benötigen.
- **Anwendungsbenchmarks:** Es wird ein reales Problem untersucht.

## 2.4. Zugrundeliegende Softwaresysteme

Im Folgenden werden die in dieser Arbeit genutzten Software-Werkzeuge beschreiben.

### 2.4.1. PREEvision

*PREEvision* ist ein Werkzeug der „*Vector Informatik GmbH*“, das zum modellbasierten Design von verteilten, eingebetteten Systemen im automobilen Umfeld konzipiert ist und insbesondere die Modellierung von E/E-Architekturen ermöglicht [290]. Ein gemeinsames Datenmodell bildet die Basis einer integrierten und durchgängigen Entwicklung, wodurch zugleich die Datenkonsistenz bei der Kollaboration gewährleistet wird. Einzelne Entwicklungsschritte, die i.d.R. unterschiedliche Berufsrollen betreffen, haben jeweils eigene Ebenen zugewiesen. Diese Ebenen sind in Abbildung 2.16 dargestellt. Für diese Arbeit relevante Ebenen werden im Folgenden nach [291] erläutert.

Die oberste Ebene umfasst die *Produktziele*. Hierbei werden sowohl Kundenfunktionen (**Customer Features**), die der Kunde bzw. Nutzer des entwickelten Fahrzeugs wahrnehmen

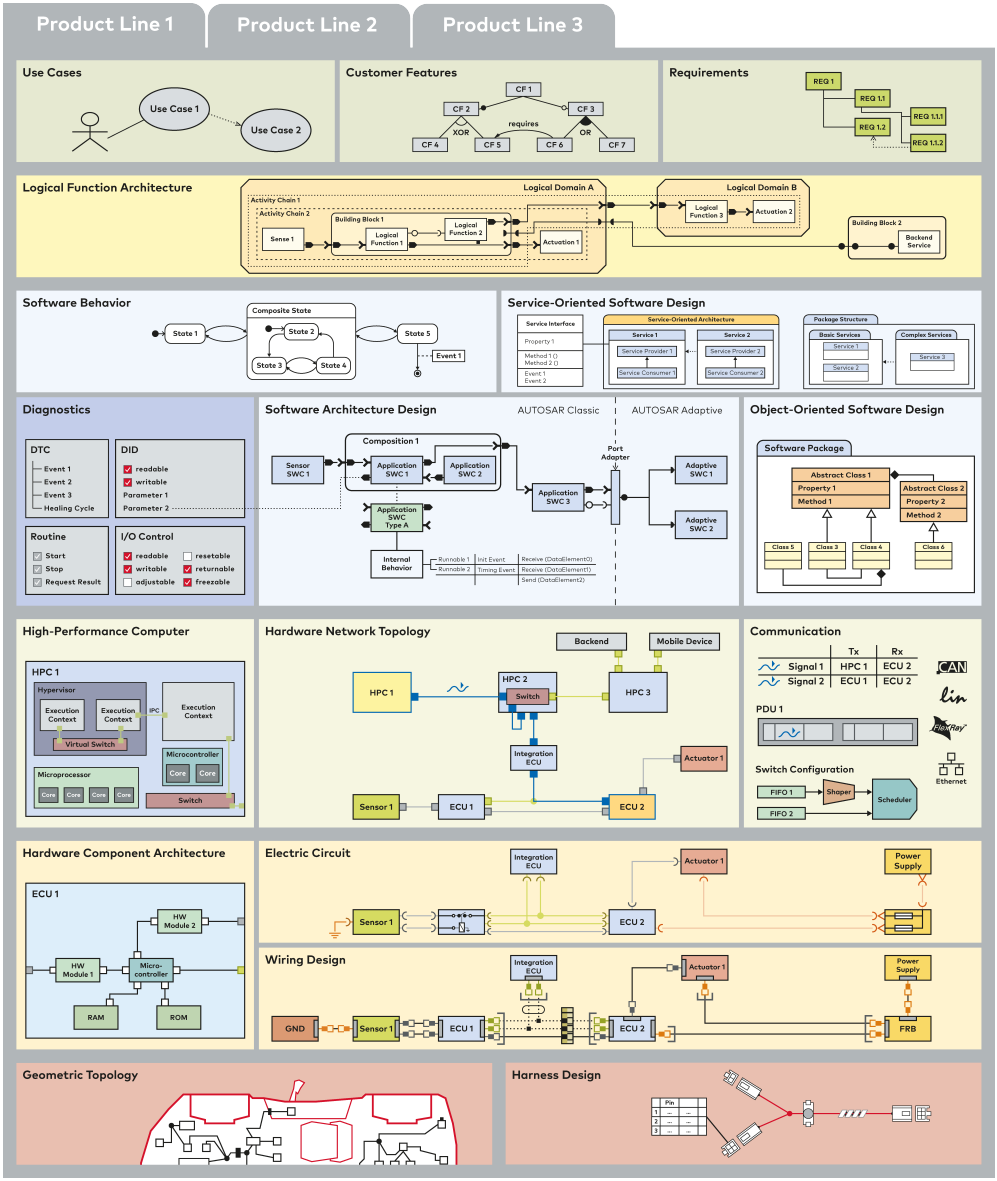



Abbildung 2.16.: PREEvision-Ebenen der Version 10.0 (Quelle: [290]).

kann, als auch die Anforderungen (**Requirements**), mit technischen Details und Einschränkungen zur genauen Umsetzung, modelliert. Zusätzlich lassen sich hier Bedingungen für das integrierte *Variantenmanagement* annotieren, z.B. wenn eine Funktion zu ihrer Umsetzung zwingend eine andere erfordert. Ebenso lassen sich Anwendungsfälle in Form von Anwendungsfalldiagrammen (**Use Cases**) modellieren. [291]

Die Ebene *Logische Architektur (LA)* (**Logical Function Architecture**) dient zur Modellierung von Konzepten, die aus den Anforderungen abgeleitet sind. Es werden dazu Blockschaltbilder genutzt, welche dem EVA-Prinzip folgen. Als Artefakte[1] stehen im Wesentlichen Sensorblöcke (**Sense**), zur Modellierung der Eingabebereitstellung, Funktionsblöcke (**Logical Function**), zur Modellierung der Verarbeitung, und Aktorblöcke (**Actuation**), zur Modellierung der Ausführung, bereit. Teilmodelle lassen sich zu Bausteinen (**Building Block (BB)**) zusammenfassen. Ferner werden Umgebungsfunktionsblöcke (nicht dargestellt) verwendet, um die Umgebung und ihre Beziehungen mit den E/E-Architekturkomponenten zu modellieren. Sie verwenden spezielle Schnittstellen, da sie kein Teil der eigentlichen E/E-Architektur sind. LA-Artefakte sind implementierungsunabhängig und lassen sich, mit Ausnahme der Umgebungsfunktionen, entweder als HWC oder als SWC realisieren. Mithilfe von *Mappings* werden ferner Beziehungen zwischen Artefakten modelliert, die auf unterschiedlichen Ebenen liegen. So ist eine durchgängige Änderungsnachverfolgung gewährleistet. [291]

Die Ebene *Software-Architektur (SA)* (**Software Architecture Design**) ist kompatibel mit dem Standard *AUTomotive Open System ARchitecture (AUTOSAR)* und dient zur Block-basierten Modellierung von Software. Als Artefakte werden **SWCs** und entsprechende Schnittstellen verwendet. Insbesondere lassen sich Trigger-Schnittstellen modellieren, wenn eine SWC die Ausführung einer anderen auslöst. SWCs werden in der Regel aus logischen Funktionen abgeleitet und auf den Recheneinheiten von ECUs ausgeführt. [291]

Auf der Ebene *Hardware-Architektur (HA)* werden die physikalischen E/E-Komponenten in Blockschaltbildern modelliert. Hierzu werden vorrangig die Artefakte **ECU**, **Sensor** und Aktor (**Actuator**) verwendet. In Netzwerk-Diagrammen (**Hardware Network Topology**) lassen sich die Verbindungen der Artefakte ohne Beachtung der konkreten elektrischen Leitungen modellieren. Durch automatisierte Synthese lassen sich daraus Schaltplandiagramme (**Electric Circuit**) erzeugen, wobei die elektrischen Leitungen in Abhängigkeit der zuvor gewählten Verbindungstypen erzeugt werden. Ferner wird in diesen Diagrammen die Stromversorgung modelliert, welche Batterien (**Power Supply**), Sicherungen und Massestellen (**GND**) einschließt. Die letzte Verfeinerung erfolgt in Leitungssatzdiagrammen (**Wiring Design**), die automatisiert aus den Schaltplandiagrammen erzeugt werden können. Hier kommen weitere Details dazu, wie *Steckertypen*, *Spleiße*, *Trennstellen* oder die Bündelung von Einzelleitungen zu *Kabeln*. Trennstellen werden abhängig von der Verortung der daran angeschlossenen Komponenten im Fahrzeug benötigt. Sie werden beispielsweise beim Leitungsübergang zwischen Türen und Karosserie eingesetzt, um den Austausch der Türen zu ermöglichen, ohne dafür Leitungen durchtrennen zu müssen. Ihre Modellierung findet auf der *Geometrieebene* (**Geometry Topology**) in Geometriediagrammen statt. Darüber hinaus erlauben die Verortungsinformationen auch die Berechnung von Leitungslängen. [291]

[1] Bezeichnung für Modellkomponenten in PREEvision.

Verfeinerungen lassen sich mit Komponentenmodellen (z.B. **Hardware Component Architecture**, **Composition** oder **Building Block**) sowie durch Verhaltensmodellierung mit FSMs auf den Ebenen HA, SA (vgl. **Software Behavior**) und LA umsetzen. [291]

Neben einer integrierten Prozess- und Teamunterstützung, inklusive des Testdaten-Managements, bietet PREEvision Unterstützung bei der Überwachung der funktionalen Sicherheit sowie Mechanismen zur Verwaltung von Änderungen, Fehlern und Freigaben. PREEvision ist unabhängig von einem gewählten Entwicklungsprozess einsetzbar. [291]

**Das PREEvision-Metrik-Framework** Innerhalb des PREEvision-Metrik-Frameworks können *PREEvision-Metriken (PMs)* entwickelt und berechnet werden. PMs sind nach [291] Algorithmen, die in Blockdiagrammen definiert und kombiniert werden. Sie ermöglichen es E/E-Architekturen zu bewerten, indem sie Ergebnisse zum Abgleich mit den Anforderungen oder mit anderen Varianten liefern [291]. Abbildung 2.17 zeigt das Prinzip zur Ausführung einer PM. Über einen **Eingabe**-Block (hier stellvertretend für die spezifischen Blöcke dargestellt) werden Daten oder Artefakte für die verarbeitenden Elemente der PM bereitgestellt. Eingaben können im einfachsten Fall *Parameter* in Form von Strings sein. Sie werden konkret mit Parameter-Blöcken modelliert. Des Weiteren können Artefakte aus dem gesamten E/E-Architekturmodell (*EEA*) als Eingabe verwendet werden. Hierzu werden *Modellkontextblöcke* als Eingabe-Block verwendet.

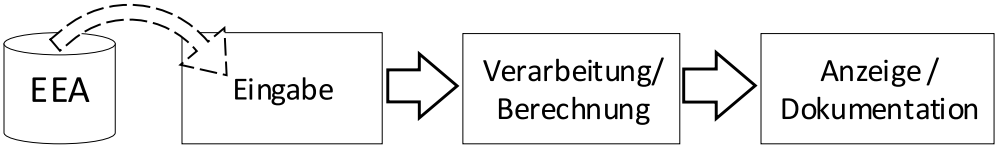



Abbildung 2.17.: Prinzip der Ausführung einer PREEvision-Metrik.

Die **Verarbeitung/Berechnung** erfolgt mit entsprechend verknüpften, spezifisch dafür vorgesehenen Blöcken. Diese nehmen die Eingabedaten auf, führen Berechnungen durch und leiten die Ergebnisse an einen daran angeschlossenen Block zur weiteren Verarbeitung, der Anzeige oder zur Dokumentation weiter. Durch sogenannte *Berechnungsblöcke* kann insbesondere eigener Java-Code implementiert werden. Hierbei kann das EEA-Modell durch Modifikation, Hinzufügen oder Löschen von Artefakten verändert werden.

**Anzeigen** können mit spezifischen Blöcken, z.B. als Tabellen oder Diagramme, realisiert werden. Ferner ist eine Aufnahme der Ergebnisse in generierte Dokumente möglich.

### 2.4.2. Ptolemy II

*Ptolemy II* [280] ist ein quelloffenes und in Java programmiertes Simulationsframework, welches an der „University of Berkeley“ entstanden ist. Die Modellbildung basiert auf sogenannten **Actors**. Sie sind nach [281, 215] als parallel ausführende Softwarekomponenten definiert, deren Instanzen in Form von Blöcken modelliert werden (vgl. Abbildung 2.18).

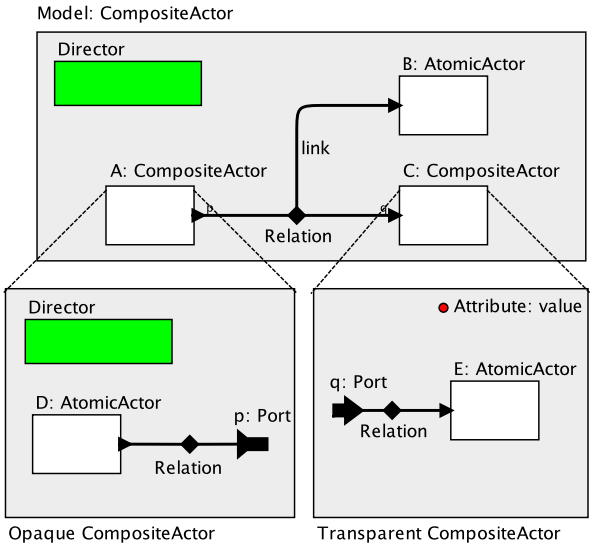



Abbildung 2.18.: Modellierungsansatz in Ptolemy II (Quelle: [215]).

Miteinander verbunden ergeben sie das Modell und besitzen, analog zu den in Abbildung 2.5 vorgestellten Entitäten, Verbindungen (**Relation**), Zustandsvariablen (●), Parameter sowie Ein- und Ausgänge (**Ports**). Über Ports können Nachrichten mit anderen Actors ausgetauscht werden, welche in der Datenstruktur *Token* gekapselt sind. Sowohl Tokens als auch Ports sind einem bestimmten Datentyp zugewiesen, um die Datenkonsistenz zu gewährleisten. Außerdem besitzt jeder Token einen Zeitstempel, dem der Sendezeitpunk des Tokens oder der Zeitpunkt dessen letzter Veränderung zugewiesen ist.

Wie zudem in Abbildung 2.18 dargestellt, können Actors in verschiedenen Hierarchieebenen verfeinert werden, d.h. zu Teilmodellen (**CompositeActor**) zusammengefasst werden. Jeder CompositeActor besitzt zudem lokale Zustandsvariablen. Ihnen kann außerdem ein **Director** zugewiesen werden. Wird einem CompositeActor kein Director zugewiesen, so wird der Director von der nächsthöheren Hierarchieebene geerbt. Directors stellen die Implementierung verschiedener Modellausführungsarten dar (vgl. Unterabschnitt 2.2.4) und bilden damit die MAE. Die Hierarchie erlaubt ferner die Ausführung und Bildung hybrider Simulationsmodelle in der Form Interaktion.

Eine Besonderheit bilden hierarchisch verfeinerbare FSMs, genannt *ModalModel*. Ein ModalModel enthält eine FSM, deren Modell sich im Wesentlichen aus Zuständen, Zustandsübergängen mit Booleschen Bedingungen (*Guards*) sowie Variablen für Eingänge, Ausgänge und lokalen Parametern zusammensetzt. Zustände innerhalb eines ModalModels werden sequenziell ausgeführt und können entweder mit einem ModalModel oder einem CompositeActor verfeinert werden. [281, 215]

Eine Auflistung relevanter Directors und deren Zusammenhang zu den in Unterabschnitt 2.2.4 gezeigten Modellausführungsarten ist in Tabelle 2.1 abgebildet.

Tabelle 2.1.: Ptolemy II Directors und entsprechende Modellausführungsart.

| **Ptolemy II Director** | **Modellausführungsart** |
|---|---|
| Continuous Time | wertkontinuierlich |
| Discrete Event | ereignisorientiert |
| Synchronous Reactive | zeitgesteuert |
| ModalModel, Synchronous Dataflow | wertdiskret/qualitativ |

### 2.4.3. MATLAB

*MATLAB* ist nach [8] ein Softwarepaket der Firma *The MathWorks, Inc.*. Das Basismodul stellt eine Ein- und Ausgabefunktion zur Programmablaufsteuerung und Berechnung mathematischer Funktionen bereit. Zur Eingabe der Befehle wird eine Hochsprache genutzt, die ebenfalls MATLAB heißt. Des Weiteren bietet das Basismodul diverse Visualisierungsmöglichkeiten für die Ergebnisse. Es kann außerdem nach [32] mittels *Toolboxen* erweitert werden, welche jeweils bestimmte Bereiche der Ingenieurswissenschaften abdecken. Die größte Stärke liegt nach [210] in der Berechnung von Vektoroperationen, da der Kern daraufhin optimiert ist. Darauf weist auch der historisch bedingte Name „MATrix LABoratory“ hin. [8, 32, 210]

Die folgenden Abschnitte behandeln die in dieser Arbeit verwendeten Toolboxen.

#### Simulink

*Simulink* ist nach [8] eine MATLAB-Toolbox, die das Basismodul um den modellbasierten Entwurf zur Simulation oder Analyse von dynamischen Systemen erweitert. Dazu werden Funktionsblöcke über Ein- und Ausgänge in Form eines Signalflussdiagramms miteinander verknüpft. Signale können dabei während der Ausführung der Simulation an beliebigen Stellen abgegriffen und entsprechend dargestellt oder im *Matlab-Workspace*, z.B. für eine weitere Aufbereitung, gespeichert werden. Als Matlab-Workspace wird der zentrale Speicherort für in MATLAB erstellte oder importierte Daten bzw. Variablen bezeichnet [272]. Entstandene Modelle können als *.mdl* oder *.slx* Datei abgespeichert werden. Zur Modellierung selbst können die Funktionsblöcke entweder einer Bibliothek entnommen werden oder vom Nutzer selbst in Form von System-Funktionen (*S-Functions*) implementiert werden. Dazu kann Code mittels der Sprachen MATLAB, C, C++ oder Fortran geschrieben werden. [8]

Eine Verfeinerung, bzw. eine Zusammenfassung zu Teilmodellen, ist durch die Verwendung von *Subsystem*-Blöcken möglich (siehe Abbildung 2.19). Des Weiteren kann Simulink durch *Blocksets* um weitere Funktionen ergänzt werden. Dazu gehören z.B. die in dieser Arbeit verwendeten Blocksets *Stateflow*, *Simscape* und *SimEvents*. Je nach deren Kombination können hybride Simulationen der Formen anreichernd oder Interaktion entstehen, wie Abbildung 2.19

anhand eines Beispiels verdeutlicht. Über die Entität **Globaler Solver** wird der Solver ausgewählt, d.h. diejenige MAE, welche der Simulation übergeordnet ist. Die Auswahl einer zeitdiskreten MAE (*discrete*) bedingt hierbei, dass sich keine Blöcke im Simulationsmodell befinden, die aufgrund ihrer Implementierung eine kontinuierliche Zeit benötigen. Wird dagegen eine zeitkontinuierliche MAE gewählt, können auch diskrete Modellanteile in die Ausführung integriert werden. In diesem Fall handelt es sich um die anreichernde Art der hybriden Simulation.

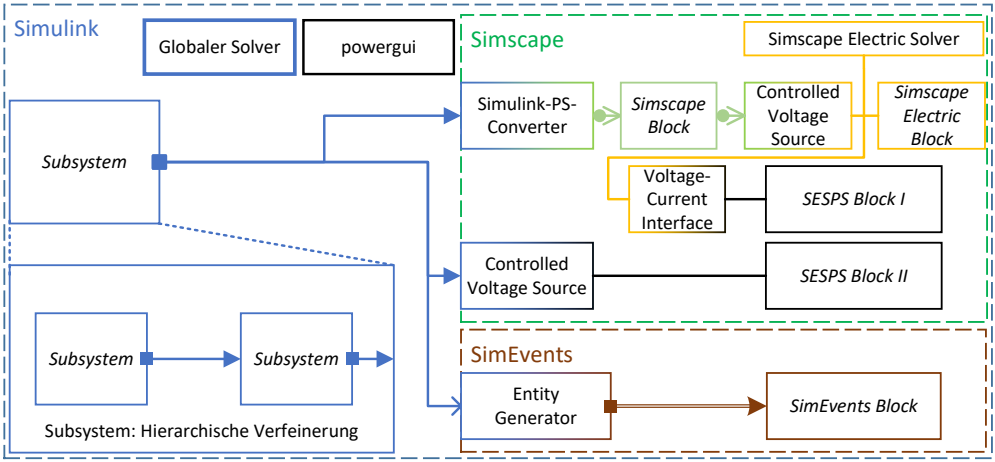



Abbildung 2.19.: Hybride Simulation in Simulink.

Für eine Integration von **Simscape**-Komponenten sind Konvertierungsblöcke (**Simulink-PS-Converter**) nötig, welche Simulink-Signale (blau) in physikalische Signale (grün) wandeln. Für elektrische Netze (orange dargestellt) ist eine weitere Konvertierung der physikalischen Signale in elektrische nötig. Die geschieht z.B. durch den Block **Controlled Voltage Source** in die Hinrichtung und über Block *Voltage Sensor* in die Rückrichtung. Jedem elektrischen Netz muss ferner ein eigener Solver in Form des Blocks **Simscape Electric Solver** zugewiesen werden. Sollen zusätzlich noch *Simscape-Electrical-Specialized-Power-Systems-Komponenten (**SESPS**-Komponenten)* an das elektrische Netz angeschlossen werden, so ist eine weitere Konvertierung, z.B. per **Voltage-Current Interface**, nötig. Werden also Simscape- und SESPS-Komponenten gemischt, sind zwei Konvertierungen in jede Richtung nötig. Eine direkte Konvertierung von Simulink- zu SESPS-Komponenten ist jedoch ebenfalls möglich. Dazu wird z.B. ebenfalls der Block **Controlled Voltage Source** aus der SESPS-Bibliothek verwendet. Die Ausführung der SESPS-Komponenten wird zentral durch den Block **powergui** koordiniert. Dabei wird für alle SESPS-Komponenten ein entsprechender Code generiert. Eine hybride Simulation mit Simulink- und Simscape-Komponenten entspricht der Art Interaktion.

Für die Integration von **SimEvents**-Blöcken sind ebenfalls entsprechende Komponenten nötig, die Signale (blau) in Ereignisse (braun) konvertieren und ggf. wieder zurück.

### Simscape

*Simscape* ist ein Blockset zur Erweiterung von Simulink. Es erlaubt die Modellierung und Simulation von physikalischen Systemen, mit mechanischen, elektrischen, hydraulischen oder pneumatischen Komponenten [32, 269]. Im Gegensatz zu konventionellen Simulink-Blöcken, welche mathematische Operationen repräsentieren und mit Signalen arbeiten, werden durch Simscape-Blöcke physikalische Komponenten und deren physikalische, nicht gerichtete Verbindungen repräsentiert. Durch die unterschiedlichen Sichtweisen müssen die Simscape-Modelle entsprechend in ein Simulink-Modell integriert werden. Hierzu werden aus dem Simscape-Modell automatisch die entsprechenden mathematischen Gleichungen erzeugt [83]. Schnittstellen zwischen den Modellen bilden Konvertierungsblöcke, die in beide Richtungen existieren (vgl. Abbildung 2.19). Eine doppelte Konvertierung ist dabei zu vermeiden, da dadurch *algebraische Schleifen* entstehen, welche die Laufzeit verlängern oder zum Abbruch der Simulation führen können [268]. Eine direkte Verbindung von Simulink- und Simscape-Blöcken wird ferner durch unterschiedliche Anschlüsse verhindert. Simulink verwendet gerichtete *In-* und *Outports*, während Simscape-Blöcke ungerichtete *Connection Ports* verwenden.

### SimEvents

*SimEvents* ist ein Blockset zur Erweiterung von Simulink, um ereignisgesteuerte Systeme zu simulieren. Mit SimEvents-Blöcken lassen sich sowohl rein prozessorientierte Simulationen als auch hybride Simulationen in Verbindung mit Simulink-Blöcken realisieren [47]. Bei der rein prozessorientierten Simulation mit SimEvents werden Ereignisse erzeugt, verarbeitet und am Ende zerstört, um den belegten Speicher wieder freizugeben. Ein Ereignis wird als *Entity* definiert. Eine Entity kann, analog zu Tokens in Ptolemy II (vgl. Unterabschnitt 2.4.2), als Datenstruktur realisiert werden, welche (individuelle) Attribute zugewiesen sind. Jedem Attribut muss hierbei mindestens ein Name und ein Wert zugewiesen sein. Entities werden innerhalb von Blöcken des Typs *Entity Generator*, zyklisch oder basierend auf einem extern angelegten Triggersignal, erzeugt und innerhalb von Blöcken des Typs *Entity Server* verarbeitet. Für die Verarbeitung werden Matlab-Skripte verwendet. Die Manipulation oder Auswertung von Entity-Attributen kann über eine entsprechende Matlab-Syntax bewirkt werden.

### Stateflow

Stateflow ist nach [8] ein Blockset zur Erweiterung von Simulink, um die grafische Modellierung und Simulation endlicher Zustandsautomaten bzw. FSMs zu realisieren. Im Gegensatz zu reinen Simulink-Modellen wird ein Stateflow-Modell ereignisgesteuert ausgeführt. Zustände sind damit nicht wertkontinuierlich, sondern können entweder *aktiv* oder *inaktiv* sein. Vor der Ausführung wird aus dem Stateflow-Modell automatisiert eine S-Function erzeugt und der dadurch entstandene C-Code kompiliert. [8]

Wie in Abbildung 2.20 dargestellt, wird ein **Stateflow-Diagramm (Chart)** als ein Subsystem in die Simulink-Umgebung eingebettet. Zur Kommunikation mit der Umgebung besitzen die

Charts Ein- und Ausgänge (*Inports* ▶ bzw. *Outports* ■). Diesen ist über zugehörige **Eingangsvariablen** und **Ausgangsvariablen** jeweils ein Datum (*Data*), ein Ereignis (*Event*) oder eine Nachricht (*Message*) zugeordnet. Eine Nachricht ist eine Folge von Ereignissen [270]. Zusätzlich besitzt ein Chart **lokale Variablen**, die das Chart nicht verlassen. Daten, Ereignisse und Nachrichten werden vorwiegend für die Auswertung von **Zustandsübergängen (Transitions)** genutzt. Ein Zustandsübergang kann dabei durch ein Ereignis bzw. eine Nachricht (↯) oder durch Erfüllung einer Booleschen Bedingung (*Condition:* **[]**) ausgelöst werden. Zur Überprüfung der Bedingungen werden Daten genutzt, die dazu einen entsprechenden Datentyp, z.B. Integer, zugewiesen haben. Ein Beispiel für eine solche Bedingung ist „$a > 5$“, wobei a das Datum ist. Als ein Ereignis können auch zeit-logische Ausdrücke (*Temporal Logic*) definiert werden. Durch den Befehl `after(20,sec)` wird bspw. ein Zustandsübergang nach 20 Sekunden ausgelöst. Ausgehend von einem aktiven Zustand mit mehreren ausgehenden Zustandsübergängen erfolgt die Überprüfung der Bedingungen zyklisch und geordnet nach einer jedem Übergang zugewiesenen Priorität.

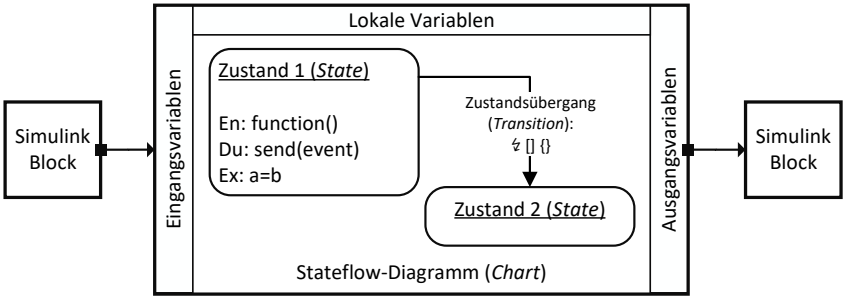



Abbildung 2.20.: Grundelemente eines Stateflow-Diagramms.

Auf einem Zustandsübergang können ferner Aktionen (*Actions:* {}) festgelegt werden, die dann ausgeführt werden, wenn der Zustandsübergang durch eine zutreffende Bedingung oder ein Ereignis ausgelöst wird. Mittels Aktionen lassen sich weitere Ereignisse auslösen, Daten zuweisen (vgl. **a=b**) oder Funktionen (vgl. **function()**) aufrufen. Aktionen können auch innerhalb von Zuständen, durch eine entsprechende Syntax (vgl. `send(event)`), definiert werden. Hier findet eine Unterscheidung statt, ob die Aktion beim Eintritt (*Entry:* **En**), während dem Verharren im Zustand (*During:* **Du**) oder beim Verlassen (*Exit:* **Ex**) ausgeführt wird. Durch Funktionen kann die Ausführung von *internen* Subsystemen, d.h. dem Chart zugeordnete, und von *externen* Subsystemen angestoßen werden.

Neben den beschriebenen UML-konformen [195] Zuständen gibt es in Stateflow noch hierarchisch verfeinerbare Zustände (*Substates*) und spezielle Simulink-Zustände (*Simulink States*). Letztgenannte führen, solange ihr Zustand aktiv ist, ein intern modelliertes Simulink-Modell aus. Des Weiteren können mehrere Charts als Subcharts in einem übergeordneten Chart zusammengefasst und parallel ausgeführt werden.

### Matlab Engine API for Java

Mit dem *Matlab Engine Application Programming Interface for Java (ME-API-J)* können Java Programme nach [264] synchron oder asynchron mit Matlab kommunizieren. Neben dem Starten und Beenden von Matlab aus Java Code heraus, ist es möglich Befehle an Matlab zu senden. Dadurch können Funktionen, wie das Starten der Simulation, aufgerufen und Variablen in beide Richtungen ausgetauscht werden. Ferner lassen sich Ergebnisse aus dem Matlab-Workspace abrufen und dahin schreiben. Weitere Befehle, die zum Erstellen von Modellen benötigt werden, sind in Tabelle 2.2 zusammengefasst:

Tabelle 2.2.: Basisbefehle zur Modellerstellung mit der Matlab Engine API for Java.

| **Matlab Befehl** | **Beschreibung** |
|---|---|
| **new_system**('Modellpfad + Modellname') | Erstellt ein neues System. |
| **load_system**('Modellpfad + Modellname') | Lädt ein bestehendes System in den Speicher, ohne den grafischen Editor zu öffnen. |
| **save_system**('Modellpfad + Modellname') | Speichert ein System. |
| **open_system**('Modellpfad + Modellname') | Öffnet ein System im grafischen Editor. |
| **add_block**('Bibliothekspfad','Modellpfad+Blockname') | Fügt Block / Zustandsdiagramm hinzu. |
| **delete_block**('Bibliothekspfad','Modellpfad+Blockname') | Löscht Block / Zustandsdiagramm. |
| **set_param**('Modellpfad+Blockname' , 'Parametername', 'Parameterwert') | Modifiziert einen bestehenden Parameter. |
| **add_line**('Modellpfad + Blockname','Modelpfad + Eingangsblockname + Eingangsportindex','Modelpfad + Ausgangsblockname + Ausgangsportindex'); | Fügt eine Leitung zwischen einem Eingangs- und einem Ausgangsport hinzu. |
| **delete_line**('Modellpfad + Blockname",'Modelpfad + Eingangsportindex','Modelpfad + Ausgangsportindex'); | Löscht die Leitung zwischen einem Eingangs- und einem Ausgangsport. |

Sie können in die Matlab-Kommandozeile eingegeben oder über die ME-API-J mit der Methode `eval()` an Matlab übermittelt werden. Aktuell ist die Datenmenge von Arrays auf 2 Gb und die Version des *Java Development Kits (JDK)* auf 8 begrenzt. [264]

## 2.4.4. OpenDS

*OpenDS* ist nach [68] ein quelloffener Fahrsimulator, der in Java geschrieben ist und auf der *JMonkeyEngine* [135] aufbaut. Der Simulator unterstützt darüber hinaus den *openDRIVE* Standard [12], sodass Straßen und dazugehörige Metadaten, wie Fahrspuren oder Kreuzungen, standardisiert im *.xodr*-Format, importiert werden können. Zur Szenarienbeschreibung, für den Ablauf und die Konfiguration der Fahrsimulation, sind die nach [172] beschriebenen vier *Extensible-Markup-Language-Dateien (XML-Dateien)* nötig:

- **scene.xml:** Hier werden die an der Simulation beteiligten *statischen* Straßenobjekte und ihre Eigenschaften festgelegt.
- **scenario.xml:** Enthält die *dynamischen* Objekte als Teil der Umgebung. Dies sind z.B. Verkehrsteilnehmer, die entweder eine zugeordnete *openDRIVE* Straße entlang fahren, oder einer Wegpunktliste folgen, die in der .xml definiert ist.

- **interaction.xml:** Hier werden *Bedingungen* und dadurch ausgelöste Aktionen zwischen den Entitäten definiert.
- **settings.xml:** Enthält noch weitere Einstellungen zur *Konfiguration* des Simulators selbst, welche das eigentliche Szenario nicht betreffen.

### 2.4.5. VisualVM

*VisualVM* ist nach [296] ein Werkzeug zur visuellen Anzeige von detaillierten Laufzeitinformationen einer Java-Anwendung, die auf einer *Java Virtual Machine (JVM)* ausgeführt wird. Die Informationen werden dabei von JDK-internen Werkzeugen gesammelt und mit VisualVM aufbereitet. Zu den Hauptfunktionen gehört einerseits die Monitor-Funktion, die es z.B. erlaubt den Speicherbedarf, geladene Klassen und aktive Threads einer Java-Anwendung zu überwachen. Andererseits bietet VisualVM einen Profiler zur Analyse der CPU Nutzung, z.B. die anteilige Laufzeit einzelner Methoden, und des Speicherverbrauchs einzelner Datentypen. [296]

## 2.5. Verwendete Hardware

In diesem Abschnitt werden durch Tabelle 2.3 und Tabelle 2.4 die Spezifikationen der zur Leistungsuntersuchung verwendeten Systeme aufgelistet. Es handelt sich dabei um handelsübliche Klapprechner, die u.a. für den Einsatz in Unternehmen entwickelt wurden. Sie erfüllen beide die in [292] aufgeführten Systemanforderungen von PREEvision.

Tabelle 2.3.: Spezifikation von System A.

| | |
|---|---|
| Hersteller | Lenovo |
| Typ | T480 / 20L6-S01W00 |
| Prozessor | Intel Core i5-8250U (4x 1,60 GHz (bis 3,4 GHz) / 6 MB Cache / 15 W) |
| RAM | 16 Gb DDR4 |
| Betriebssystem | Windows 10 1809 (17763.1935) |

Tabelle 2.4.: Spezifikation von System B.

| | |
|---|---|
| Hersteller | Lenovo |
| Typ | T460p / 20FXS05500 |
| Prozessor | Intel Core i7-6700HQ (4x 2,60 GHz (bis 3,5 GHz) / 6 MB Cache / 45 W) |
| RAM | 16 Gb DDR4 |
| Betriebssystem | Windows 10 21H1 (19043.1202) |

# 3. Stand der Technik

Dieses Kapitel beleuchtet die aktuellen Randbedingungen und Herausforderungen bei der Entwicklung von automobilen E/E-Architekturen. Hierzu werden aktuelle Entwicklungs- und Evaluationsmethoden vorgestellt und diskutiert. Ferner werden Methoden zur Generierung, Validierung und Laufzeitbeschleunigung von Simulationen analysiert.

## 3.1. E/E- Architekturentwicklung

Zur modernen E/E-Architekturentwicklung gehören entsprechende Prozesse, die die Anforderungen nach sicheren und zuverlässigen Fahrzeugen mit einem guten Preis-/Leistungsverhältnis abdecken [208]. Demgegenüber steht eine entsprechende Komplexität, die durch einen stetig wachsenden Funktionsumfang verursacht wird.

In den nächsten Abschnitten werden die Charakteristiken moderner und zukünftiger E/E-Architekturen dargestellt. Ergänzend dazu werden aktuelle Trends und Herausforderungen aufgegriffen und deren Einfluss auf gegenwärtige und zukünftige Entwicklungsprozesse herausgearbeitet. Im Anschluss werden Prozesse und Werkzeuge zur Entwicklung von E/E-Architekturen beschrieben sowie deren Limitierungen aufgezeigt.

### 3.1.1. Wandel der Paradigmen

In den Anfängen der E/E-Entwicklung gab es nach [233, 309] noch wenige elektrisch oder elektronisch umgesetzte Funktionen, weswegen jeder einer so realisierten E/E-Funktion ein eigenes Steuergerät zugewiesen war, wie Abbildung 3.1 zeigt (**Modular**).
Im Laufe der Zeit hat die Anzahl an Funktionen vor allem durch Innovationen im Bereich Sicherheit und Komfort zugenommen, womit auch die Anzahl der Steuergeräte gestiegen ist. Allerdings führte dies auch zu einem Anstieg der Kosten und dem Gewicht des Fahrzeugs, weswegen Ende der 1970er-Jahre eine zunehmende Integration mehrere Funktionen auf einer ECU stattfand (**Funktionale Integration**). Die Anzahl an ECUs stieg dabei weiter. Seit Beginn der 2010er-Jahre ist eine Erhöhung der ECU-Zahl aber immer schwieriger geworden, da der physikalische Bauraum im Fahrzeug weitestgehend ausgereizt ist. Dennoch stieg die Anzahl der Funktionen weiter, was vor allem auf den Einzug von *Fahrerassistenzsystemen (FAS)* zurückzuführen ist. Doch auch die E/E-Domänen *Infotainment* und *Komfort* trugen ihren Teil zu einer fortschreitenden Integration bei.
Heute ist ein Ende des Funktionsanstiegs nicht absehbar und aufgrund der aktuellen Trends, ändern sich die Anforderungen an die E/E-Architektur weiter. [233, 309]

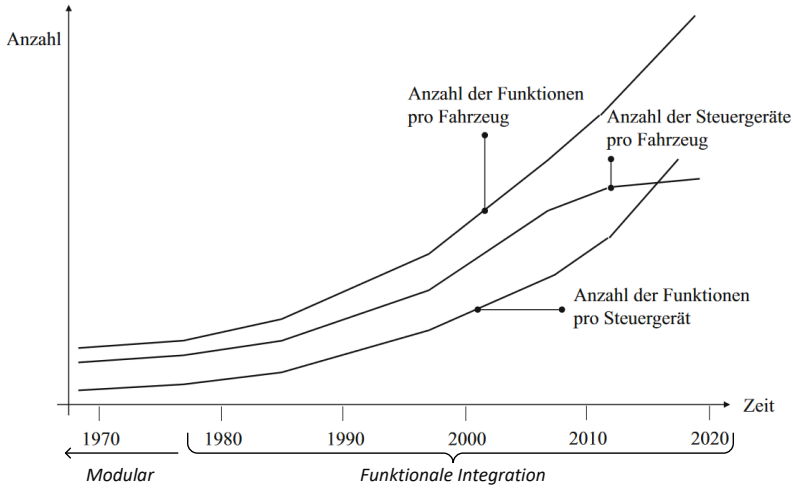



Abbildung 3.1.: Anzahl der Steuergeräte im Fahrzeug (Quelle: Modifikation von [233] mit Informationen aus [309]).

### 3.1.2. Herausforderungen und Entwicklung des Automobilsektors

Im Folgenden werden die fünf Megatrends nach [155] vorgestellt und ihre Herausforderungen beim Design zukünftiger E/E-Architekturen beschrieben.

#### Elektrifizierung

„*Klimaschutz, knapper werdende fossile Brennstoffe und ein erhöhter Mobilitätsbedarf durch steigende Bevölkerungszahlen [...]*“ [293] sind Gründe für die bereits wahrzunehmende und weiter steigende Elektrifizierung der Fahrzeuge [150]. Dazu zählen neben PKWs auch Nutzfahrzeuge. Des Weiteren sind ein höherer Wirkungsgrad und eine bessere Wartbarkeit der Elektromotoren im Vergleich zu Verbrennungsmotoren ausschlaggebend für die Elektrifizierung. Durch diesen Trend wird Leistungselektronik, d.h. Elektromotoren, Batterien und zugehörige Regler, ein Teil der E/E-Architektur. Hochspannungssysteme müssen dabei von Niederspannungssystemen galvanisch getrennt sein, um die Insassen vor Stromschlägen zu schützen. Zur Einhaltung der Trennung werden Isolationsspannungsüberwachungssysteme benötigt [307].

#### Autonomes Fahren

Höhere Sicherheit, weniger Staus oder reinere Luft: Die Gründe für autonomes Fahren sind vielfältig. Es handelt sich dabei um einen Trend, der die Automobilindustrie noch in dieser und der nächsten Dekade beschäftigen wird. Hauptmerkmale hiervon sind der steigende Umfang der Software und eine Erhöhung der Hardwareleistung im Automobil. Funktionen, die

klassischerweise der menschliche Fahrer übernommen hat, werden stückweise zu Bestandteilen E/E-Architektur. Das autonome Fahren wird dabei schrittweise über FAS-Funktionen mit steigendem *Society-of-Automotive-Engineers-Level (SAE-Level)* [240] eingeführt. Dadurch steigt die Zahl der Softwarekomponenten und mit ihr auch die Anforderungen an die zugrundeliegende Hardware. Besonders Algorithmen zur notwendigen Objekterkennung basieren auf künstlicher Intelligenz und erfordern leistungsfähige Rechenplattformen, wie *High Performance Computer (HPC)*. Außerdem wird ein passender Mix aus Sensoren zur Umgebungserkennung benötigt. Ferner erfolgt eine für das autonome Fahren erforderliche Elektrifizierung bisher mechanisch realisierter Komponenten. Dazu zählen die Steuerung (*Steer-by-Wire*), die Schaltung (*Shift-by-Wire*) oder die Bremsen (*Brake-by-Wire*), was allgemein als *X-By-Wire* zusammengefasst wird [118]. Da hierbei eine entsprechende Ausfallsicherheit gewährleistet sein muss, werden einerseits eine *Komponenten- oder Informationsredundanz* und andererseits eine Erkennung von Fehlern während des Systembetriebs durch Überwachung (*Monitoring*) nötig [235].

#### Vernetzung, Shared Mobility und OTA-Updates

Die Vernetzung zukünftiger Fahrzeuge untereinander und mit ihrer Umgebung ist essenziell, um das autonome Fahren, die gemeinsame Nutzung von Fahrzeugen (*Shared Mobility*), drahtlose Softwareaktualisierungen (*OTA-Updates*) und Upgrades der mit neuen Kundenfunktionen zu ermöglichen. E/E-Architekturen benötigen dazu flexible Software bzw. Betriebssysteme sowie entsprechende Schutzmaßnahmen vor unbefugten Zugriffen [309, 67]. Durch eine Virtualisierung von ECUs und den Einsatz von AUTOSAR Adaptive kann diese Flexibilität erreicht werden [309]. Hypervisor helfen dabei sicherheitskritische Systeme zu isolieren [17], während *Intrusion Detection Systems (IDSs)* Manipulationen zur Laufzeit erkennen und falls möglich korrigieren [P4].

### 3.1.3. Folgerungen für zukünftige E/E-Architekturen

Aufgrund der vorgestellten Trends verändert sich die E/E-Architektur in Zukunft grundlegend, wie Abbildung 3.2 darstellt. Nach [309] folgt auf die **Funktionale Integration** (siehe Unterabschnitt 3.1.1) eine **Zentralisierung**. Hauptmerkmal davon ist der Einsatz von zentralen HPCs, welche auf leistungsstarker Mikroprozessortechnik basieren. Sie können dadurch, im Vergleich zu klassischen ECUs, komplexere Algorithmen ausführen. Jedoch werden klassische ECUs noch weiterhin für die Ausführung von Aufgaben mit harten Echtzeitanforderungen benötigt, da sie Befehle deterministisch abarbeiten. Zentralisierung bedeutet ferner eine Verschmelzung bzw. Fusion von bisher, aus historischen Gründen, getrennten Domänen. Dies beschleunigt Entscheidungen, da eine Koordination zwischen vielen Steuergeräten wegfällt und zugleich die Komplexität reduziert wird. Möglich wird dies insbesondere durch eine Virtualisierung mittels Hypervisor auf den zentralen HPCs, wodurch sich virtuelle ECUs unabhängig voneinander, aber dennoch integriert betreiben lassen [166]. Ferner werden nach [309] bisher domänenspezifisch eingesetzte Zonen-ECUs schrittweise durch domänenunabhängige Zonen-ECUs ersetzt. Eine Zone definiert hierbei einen geometrischer Bereich innerhalb des Fahrzeugs. Auf Hardwaresystemebene werden E/E-Architekturen durch die Zentralisierung mit HPCs und **Domänenfusion** einfacher, kleiner, flexibler und leichter. Auch ist eine einfache-

re Trennung von Hardware und Software in der Entwicklung möglich. Der Softwareumfang wird in Bezug auf den belegten Speicherplatz um den Faktor 10000 ansteigen, wobei ein Teil davon zukünftig in der **Cloud** ausgeführt werden wird. [309, 166]

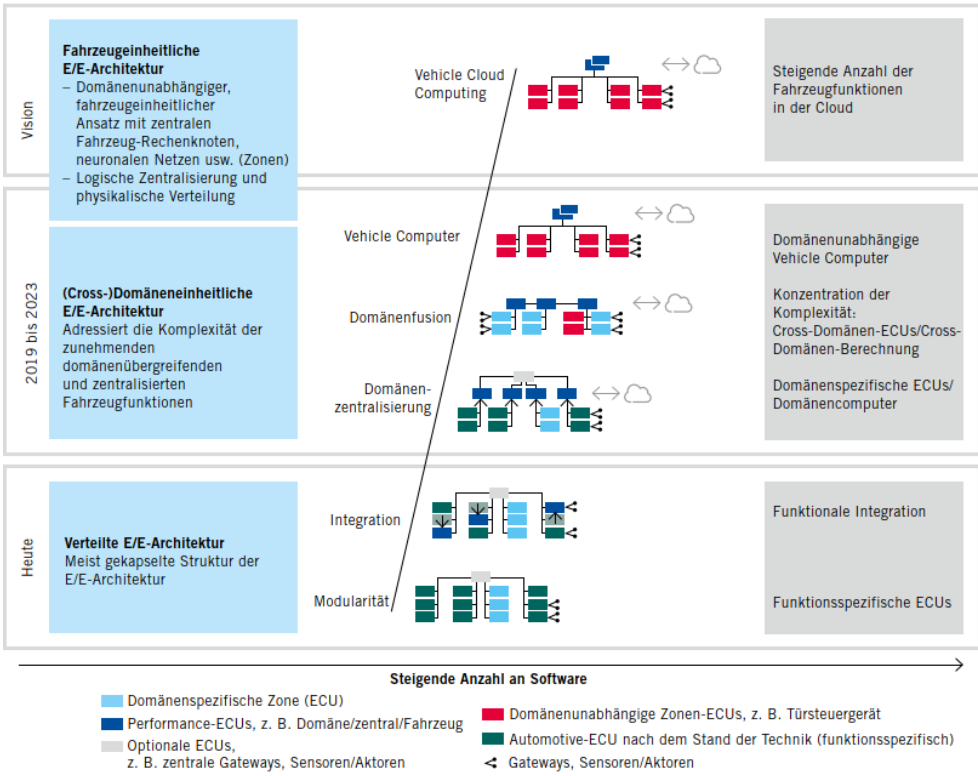



Abbildung 3.2.: Automobile E/E-Architekturen im Wandel (Quelle: [309]).

In [S5] wurden Konzepte für zukünftige E/E-Architekturen untersucht und gemeinsame Merkmale identifiziert. Tabelle 3.1 listet diese mit ergänzten Informationen auf und vergleicht sie anhand der **Topologie** (*Zentralarchitektur* (**1**), *Domänenarchitektur* (**2**) und *Zonenarchitektur* (**3**)), der **Kommunikation** (*Signalorientierung* (**I**) und *Serviceorientierung* (**II**)), den verwendeten Methoden zur Gewährleistung der **Betriebssicherheit** (*Redundanz* (**a**) und *Monitoring* (**b**)) sowie den Methoden zur Gewährleistung der **Informationssicherheit** (*Fernupdates* (**$\alpha$**), *Security-Hardware* (**$\beta$**) und *Virtualisierung* (**$\gamma$**)).

Die Zonenarchitektur kommt in den gefundenen Konzepten am häufigsten vor, was sich mit [309] deckt. Im Vergleich zu reinen Zentralarchitekturen, ohne Zonen-ECUs, wird nach [S5] die Auslastung des Kommunikationssystems reduziert, da eine lokale Aufbereitung und Filterung der Sensordaten in den Zonen stattfinden kann und nur relevante Daten an den Zentralrech-

ner weitergeleitet werden. Außerdem werden Fehler durch die lokale Verarbeitung schneller erkannt. Im Vergleich zur Domänenarchitektur, verbraucht die Zonenarchitektur weniger Platz und hat ein geringeres Gewicht. [S5]

Serviceorientierte Kommunikationskonzepte sind bei den gefundenen Architekturen weniger als signalorientierte verbreitet. Dabei bieten erstgenannte nach [S5] eine einfachere Erweiterung des Systems. Außerdem wird eine Kommunikation nur bei Bedarf initiiert, wodurch die Auslastung sinkt. Nachteil ist jedoch, dass serviceorientierte Systeme nicht zwingend deterministisch sind. Daher werden in Zukunft Mischungen zum Einsatz kommen, wie in [196] vorgeschlagen. [S5, 196]

Bezüglich der Betriebssicherheit werden in Zukunft sowohl Komponenten- und Informationsredundanz als auch Monitoring benötigt werden, da sich beide Konzepte ergänzen [S5]. Bei der Komponentenredundanz ist insbesondere die Kreuzschaltung nach [232] zu erwähnen, wobei Sensoren und Aktoren redundant in unterschiedlichen geometrischen Zonen ausgelegt sind. Jede Komponente ist dann mit der Zonen-ECU aus der eigenen Zone und der Zonen-ECU aus der Zone der redundanten Komponente verbunden. Für eine ausfallsichere Stromversorgung eignet sich außerdem eine Ringtopologie mit zwei Batterien für das Versorgungsnetz [232].

Hinsichtlich der Informationssicherheit werden in Zukunft drahtlose Softwareaktualisierungen (Fernupdates) verwendet werden. Damit lassen sich zur Nutzungsphase auftretende Sicherheitslücken dynamisch schließen [26]. Außerdem werden Hardwarevirtualisierungen und Hypervisor verwendet, um einzelne Partitionen oder Rechenkerne voneinander zu isolieren und damit den Zugriff einzuschränken. Ebenso werden entsprechende Hardwarelösungen, wie *Hardware Security Modules (HSMs)*, *Trusted Platform Modules (TPMs)* oder Security Gateways zur Realisierung von Authentifizierungen und Verschlüsselungen Einzug halten [273]. Mit letztgenannten sollen verschiedene Security-Ebenen, wie präventive Firewalls und proaktive IDS, für die Kommunikation innerhalb und außerhalb des Fahrzeugs realisiert werden [105, 4]. Alle drei Konzepte ergänzen sich.

Tabelle 3.1.: Vergleich bestehender Konzepte für zukünftige E/E-Architekturen auf Basis von [S5] mit zusätzlicher Betrachtung der Informationssicherheit.

| | Topologie | | | Kommunikation | | Betriebssicherheit | | Informationssicherheit | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Architekturkonzept | 1 | 2 | 3 | I | II | a | b | $\alpha$ | $\beta$ | $\gamma$ |
| Frischzellenkur [26] | + | | | + | | + | | + | | |
| Safe adaptation [301] | + | | | + | | + | | | | |
| Smart vehicle [232] | | | + | + | | + | | | | |
| Zentralarchitektur [98] | | | + | + | | | | + | + | |
| Zonenarchitektur: Ethernet [297] | | | + | + | | | + | | + | |
| Zonenarchitektur: Zentralcomputer [196] | | | + | + | | | + | | | |
| Das Rechenzentrum im Fahrzeug [273] | | + | | | + | + | + | | + | + |
| Autosar für HPCs [205] | | + | | | + | + | + | + | + | + |
| Die E/E Architektur der Zukunft [105] | | | | | | | | + | + | |

In Abbildung A.4 im Anhang ist ergänzend das in [S5] entstandene PREEvision-Modell einer zukünftigen E/E-Architektur zu finden.

### 3.1.4. Entwicklungsprozesse

MBSE-Prozesse haben eine ehemals dokumentenbasierte Entwicklung von E/E-Architekturen abgelöst [233, 111]. Das liegt daran, dass die Komplexität durch den Anstieg der Funktionen und der ECUs zugenommen hat. Ferner trägt der erhöhte Vernetzungsaufwand für die Kommunikation der E/E-Komponenten zur Komplexität bei. Des Weiteren benötigt die zunehmende Kollaboration in internationalen Teams eine gemeinsame Datengrundlage. Schließlich bieten Modelle den Vorteil, dass sie früh getestet werden können, vor allem im Vergleich zu reellen Prototypen.
Gerade komplexe Systeme bedürfen frühzeitig Methoden zur Absicherung und Optimierung, um eine hohe Zuverlässigkeit und eine kosteneffiziente Entwicklung zu garantieren [78]. Die Absicherung setzt sich in der Regel aus der Verifikation und der Validierung zusammen. Bei der *Verifikation* wird geprüft, ob das Produkt den intern festgelegten Anforderungen entspricht. Dagegen wird bei der *Validierung* geprüft, ob die Wünsche des Kunden durch das Produkt erfüllt werden.
MBSE-Prozesse sollen einerseits die beschriebene Komplexität beherrschbar machen und andererseits den Bedarf nach einer frühen Absicherung abdecken. Das *V-Modell* ist ein dafür geeigneter und in der Automobilindustrie weitverbreiteter Kernprozess [233]. Es werden sämtliche Entwicklungsschritte von der Auswahl der Anforderungen bis zum fertigen Produkt beschrieben. In Abbildung 3.3 ist die aktuelle Version für die Entwicklung von CPMS, nach dem Standard VDI/VDE2206 [295], abgebildet.

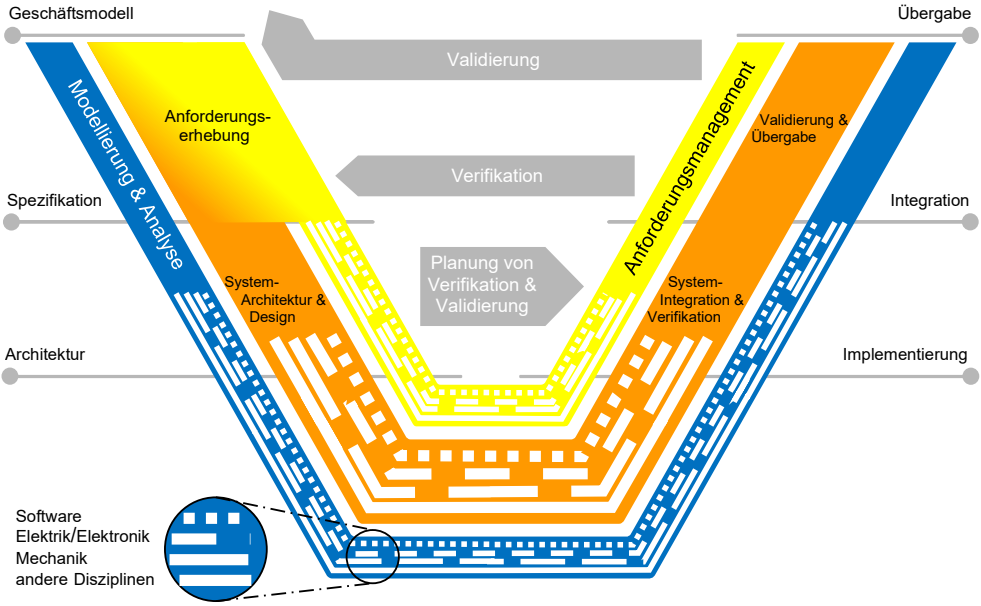



Abbildung 3.3.: V-Modell nach VDI/VDE2206 (Quelle: [295]).

Nach [295] handelt es sich bei dem V-Modell um ein „*Rahmenwerk, das die Vernetzung der Aufgaben in der Entwicklung cyber-physischer mechatronischer Systeme beschreibt*“[295]. Es besteht aus drei parallel ablaufenden Strängen. Der blaue Strang zeigt die durchgängige **Modellierung und Analyse** des Systems. Mögliche Subsysteme sind hier eingeschlossen. Der mittlere Strang bezieht sich auf das ursprüngliche V-Modell nach Bröhl [40] und damit auf die Kernaufgaben der Systementwicklung: die Auswahl der Anforderungen, das Systemdesign (*Grobentwurf*) und dessen Zerlegung in Komponenten (*Feinentwurf*), die Implementierung sowie Tests auf Komponenten-, Modul- und Systemebene. Im gelben Strang ist das **Anforderungsmanagement** dargestellt, welches die kontinuierliche Überwachung und Anpassung der Anforderungen beinhaltet. [295, 40]

Eine Entwicklung nach dem *Top-Down-Prinzip* mit dem V-Modell wird nach [219] vorwiegend bei Neuentwicklungen angewandt. *Bottom-Up-* und *kombinierte Entwurfsmethoden* sind jedoch ebenfalls typisch, da eine E/E-Architektur üblicherweise nicht für jede neue Modellreihe von Grund auf neu entwickelt wird. Es handelt sich eher um eine Evolution. Man geht von einer bereits bestehenden Architektur aus einer vorherigen Modellreihe aus, die an neue Anforderungen angepasst oder ergänzt wird. Neue Funktionen und Komponenten sind dabei, falls möglich, in die bestehende Architektur zu integrieren. Gegebenenfalls ist dabei eine Anpassung der Vernetzung und der Topologie erforderlich. [219]

Neben dem offiziellen VDI/VDE 2206 Standard, wurden weitere Abwandlungen des V-Modells entwickelt, die zudem eine modellbasierte Analyse oder Simulation mit in den Prozess einschließen. Dazu gehören das *Modellbasierte-Virtuelle-Produktentwicklungsmodell* nach Eigner [78], das *W-Modell* nach Nattermann [189] und das *MBSE-Prozessmodell* nach Fraunhofer IPK und TU Berlin [91]. Gerade hinsichtlich der Typenzulassung nach [199] ist die Integration von Simulationen im Prozess für die Entwicklung zukünftiger E/E-Architekturen unumgänglich.

#### Ergänzende Normen für eine sichere Produktentwicklung

Zur Gewährleistung der Produktsicherheit wurden in der Vergangenheit Normen eingeführt, welche die Entwicklungsprozesse ergänzen:

Bezüglich der Betriebssicherheit der sich im Fahrzeug befindenden Systeme wurde 2011 die Norm 26262 „*Functional safety*“ [134] von der *International Organization for Standardization (ISO)* eingeführt. Die Systeme sollen gegen Fehler abgesichert werden, die sie durch ihr eigenes Verhalten auslösen. Aufgrund des Einzugs von FAS wurde dieser Standard 2019 durch die ISO-Norm 21448 „*Safety Of The Intended Functionality (SOTIF)*“ [131] ergänzt. Hier werden zusätzlich Fehler über die Systemgrenzen hinaus adressiert, die in Abhängigkeit bestimmter Situationen bzw. Szenarien auftreten. Es wird davon ausgegangen, dass diese Fehler aufgrund einer Abweichung der vorgesehenen Funktionalität oder durch einen vorhersehbaren Missbrauch, begangen von Personen, entstehen.

In Bezug auf die Informationssicherheit wurde 2021 die ISO-Norm 21434 „*Cybersecurity engineering*“ [132] eingeführt. Ziel ist die Absicherung des Systems gegen bösartige Angriffe. Die Norm beschreibt dabei Anforderungen und Methoden zur Handhabung von Cybersecurity-Risiken. Ein zentraler Bestandteil davon ist die Bedrohungs- und Risikoanalyse (*Threat Analysis*

*and Risk Assessment (TARA)*). Anhand von Missbrauchsszenarien wird mithilfe einer Angriffspfadanalyse die technische Möglichkeit eines Angriffs bestimmt. Daraus werden schließlich passende Methoden abgeleitet, um die Informationssicherheit zu gewährleisten.

### 3.1.5. Sprachen und Werkzeuge

Generell können E/E-Architekturen dokumentenbasiert beschrieben werden. Aufgrund der in Unterabschnitt 3.1.1 dargestellten, stetigen Steigerung der Komplexität haben sich jedoch MBSE-Methoden im Automobilbereich durchgesetzt [233]. Sie ermöglichen Wiederverwendbarkeit, Nachverfolgbarkeit von Änderungen, Vollständigkeit, Konsistenz, einen besseren Austausch mit Kollegen oder ein besseres Verständnis [111].

Für den Einsatz von MBSE gibt es verschiedene Werkzeuge. Eine Liste ist unter [300] zu finden. Dazu gehören beispielsweise *IBM Rhapsody* [121] oder das kostenlose Werkzeug *Eclipse Capella* [74]. Die Werkzeuge implementieren entsprechende Modellierungs- bzw. Architekturbeschreibungssprachen (*Architecture Description Languages (ADLs)*). Nach ISO/IEC/IEEE 42010:2011 [129] handelt es sich bei ADLs um *„eine beliebige Ausdrucksform, um Architekturen zu beschreiben“*. Eine mögliche Ausdrucksform sind formale Sprachen. Vertreter davon und deren historische Entwicklung sind in Abbildung 3.4 abgebildet.

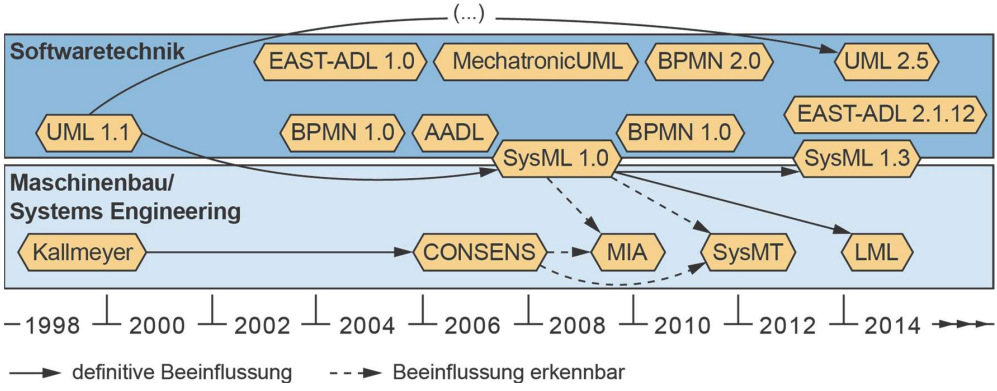



Abbildung 3.4.: Entstehung von MBSE-Modellierungssprachen (Quelle: [276]).

Es ist zu erkennen, dass die Sprachen ursprünglich für unterschiedliche Einsatzzwecke (**Softwaretechnik** oder **Maschinenbau/Systems Engineering**) und damit Anforderungen entwickelt wurden. Gleichzeitig bieten sie nach [276] oft hohe Freiheitsgrade, für eine möglichst breite Anwendung, wodurch allerdings Interpretationsspielräume entstehen. Diese erschweren wiederum sowohl das Verständnis als auch eine Rechnerinterpretation [148]. Daher haben sich Dialekte entwickelt, um die Sprachen und Methoden an die Bedürfnisse von spezifischen Anwendungen anzupassen [276]. Im Automobilbereich ist die Modellierungssprache *EEA-ADL* [95], die im Werkzeug PREEvision (vgl. Unterabschnitt 2.4.1) zum Einsatz kommt, ein Beispiel dafür. Insbesondere lassen sich nach [95] alle Aspekte aus den

unterschiedlichen E/E-Domänen in einem gemeinsamen Datenmodell beschreiben. Dazu zählen sowohl technische Aspekte, wie die Software- und Hardwarearchitektur, als auch prozessrelevante Aspekte, wie Anforderungen.

## 3.2. Modellbasierte Verifikation von E/E-Architekturen

E/E-Architekturmodelle lassen sich sowohl *formal*, d.h. automatisiert, als auch nicht formal, d.h. *manuell*, verifizieren. Primäres Ziel der Verifikation ist es bei der Entwicklung des Modells entstandene Fehler aufzudecken. Ein weiteres Ziel ist die Unterstützung von E/E-Architekten bei Designentscheidungen oder Optimierungen. Der Schwerpunkt dieser Arbeit und dieses Abschnitts liegt auf simulativ ausgeführten Tests und ihre Integration in den Entwicklungsprozess, da sie ein Hauptbestandteil dieser Arbeit sind.

### 3.2.1. Formale Verifikation

Formale Verifikation benutzt mathematische Logik bzw. mathematische Modelle, um zu zeigen oder zu beweisen, dass ein System oder Teile davon der Spezifikation entsprechen [229]. Als Grundlage dafür wird eine formale Systemspezifikation benötigt, die durch eine Modellierungssprache mit entsprechend präziser Grammatik und Syntax beschrieben ist [55]. Bei der formalen Verifikation gibt es nach [37, 85] zwei Hauptmethoden, die in den folgenden Unterabschnitten behandelt werden.

#### Model-Checking

*„Model-Checking ist eine Technik zur vollautomatischen Verifikation reaktiver Systeme mit endlichem Zustandsbaum“* [37]. Dabei soll für ein formal beschriebenes Transitionssystem T, das sich mit endlichen Zuständen beschreiben lässt, geprüft werden, ob es eine (temporal-) logisch definierte Spezifikation $\phi$ erfüllt [85]:

$$T \models \phi \tag{3.1}$$

Hiermit können z.B. Algorithmen oder Schaltungen der E/E-Architektur überprüft werden.

#### Theorem Proving

Theorem Proving findet nach [37] Anwendung, wenn auf Basis einer gegebenen Logik, z.B. Aussagenlogik, und einem dazu passenden Beweissystem, mit Axiomen und Regeln, Aussagen bewiesen werden sollen. Durch Anwendung des Beweissystems kann ein automatischer Theorem-Prover die zu beweisende Aussage so umformen oder vereinfachen, dass ihre Richtigkeit festgestellt oder widerlegt werden kann. Liegt eine Systemspezifikation in Form einer Logik vor, können mit einem automatischen *Theorem Prover* bestimmte Eigenschaften oder Fehler nachgewiesen werden. [37]

### 3.2.2. Manuelle Verifikation

Manuelle Verifikation beruht nach [37] auf der Ausführung von Tests. Es wird dabei zwischen *statischem* und *dynamischem* Testen unterschieden. [37]

#### Statisch

Bei statischen Tests bzw. Analysen wird nach [37] das zu verifizierende System oder Teilsystem nicht ausgeführt. Zu statischen Tests gehören z.B. Reviews von E/E-Architekturmodellen oder Code-Analysen von PMs. [37]

#### Dynamisch

Bei der dynamischen Verifikation werden nach [37] das System oder dessen Komponenten, entweder real oder virtuell (*simulativ*), über der Zeit ausgeführt. Basis dafür sind spezifizierte Tests und Testmethoden, die sich auch auf der rechten Seite des V-Modells befinden. [37]

**Simulative Verifikation** Zunächst werden die an einer E/E-Architektursimulation beteiligten Komponenten identifiziert. Für die Simulation von FAS und von Mechanismen zur Einhaltung der Informationssicherheit werden nach Kombination von [303] und [9] die folgenden Komponenten benötigt:

- Bestandteil der E/E-Architektur
  - Logik / Software
  - Hardwaresystem (ECUs, Netzwerk, Versorgung und sonstige HWCs)
- Bestandteil der Umgebung der E/E-Architektur
  - Ego-Fahrzeug mit Fahrdynamik und Fahrer
  - Teststrecke
  - Verkehr und weitere aktive/passive Akteure

Die Komponenten können in reeller oder virtueller Form vorliegen. In der Praxis und nach [303] wird deshalb, passend zum Entwicklungsprozess, zwischen den folgenden Testmethoden unterschieden: **Model in the Loop (MIL)**, **Software in the Loop (SIL)**, **Hardware in the Loop (HIL)** und **Vehicle in the Loop (VIL)**. Sie sind in Abbildung 3.5 dargestellt. Bei MIL-Tests sind alle beteiligten Komponenten virtuell. Dabei werden konkrete Aspekte des Hardwaresystems i.d.R. vernachlässigt. SIL-Tests erfordern das Vorliegen von kompiliertem Code, der auf einem realen oder virtuellen Hardwaresystem ausgeführt wird. Bei HIL-Tests sind verschiedene Ausprägungen zu unterscheiden. Dabei können sowohl nur die ECUs als auch das komplette Hardwaresystem real vorliegen, wobei die Umgebung, teilweise mit speziell dafür vorgesehener Hardware (HIL-Prüfstand), simuliert wird. Eine Sonderform bilden virtuelle HIL-Simulationen, wobei nur simulierte HWCs, einschließlich der ECUs, eingesetzt werden.

Bei VIL-Tests wird das komplette Fahrzeug, inkl. E/E-Architektur und Fahrdynamik, in seiner Zielumgebung getestet. Üblicherweise finden die Tests in der Realität statt. [303]

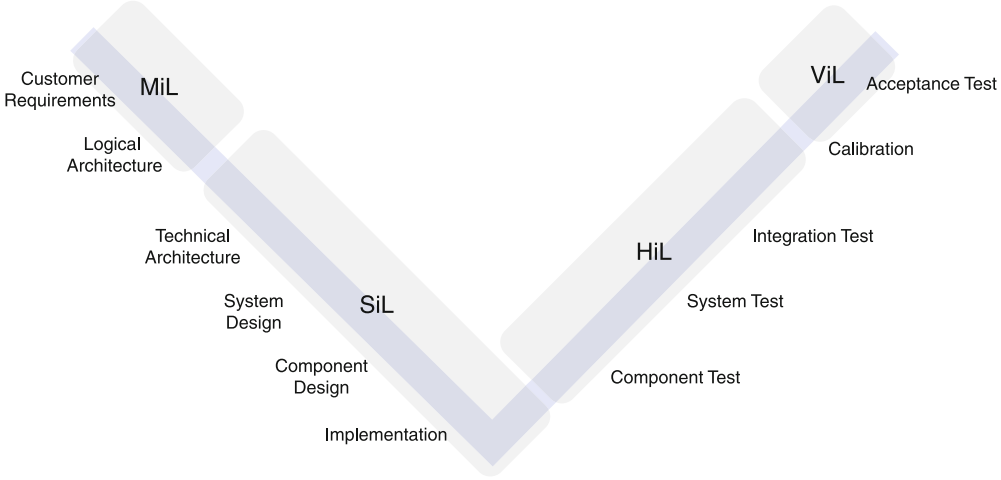



Abbildung 3.5.: In-the-Loop-Methoden (Quelle: [303]).

Eine Simulation kann theoretisch in allen Entwicklungsphasen stattfinden. Bei einer Simulation von FAS stehen die benötigten (exakten) Modelle des Zielhardwaresystems oder der Fahrzeugdynamik in frühen Entwicklungsphasen jedoch noch nicht zur Verfügung. Stattdessen können generische Modelle oder Komponenten aus vergangenen E/E-Architekturentwicklungen verwendet und im Laufe der Entwicklung angepasst werden. Letztgenannter Punkt ist auf eine evolutionären Entwicklung (vgl. [219]) beschränkt.

### 3.2.3. Diskussion

Die Diskussion in den folgenden Absätzen beruht auf Erkenntnissen aus [15, 211, 229, 37]:

Formale Methoden sollten nach [229] vor allem zur frühen und vollständigen Untersuchung des Systems bzw. seiner Teilsysteme eingesetzt werden. Außerdem benötigen formale Methoden keine Stimuli, wodurch sie schneller anwendbar sind. Sie können den Zustandsraum vollständig abdecken und finden Wurzelfehler ohne Propagation. [229]

Model-Checking ist nach [37] für E/E-Architekten einfach anzuwenden und zu verstehen. Außerdem kann Model-Checking bereits für Teilsysteme angewandt werden, wenn die Spezifikation noch nicht für das ganze System vorliegt. Allerdings ist Model-Checking primär zur Überprüfung von Kontrollflüssen geeignet [211] und führt bei hybriden Modellen, die schlecht partitioniert werden können, in die „State-Space-Explosion“ [15]. Hierbei steigt der benötigte Speicher stark mit der Modellgröße [37].

Theorem Proving durchsucht nach [37] den Zustandsraum unter Verwendung eines Beweissystems, sodass nicht alle Zustände für die Analyse benötigt werden, wodurch der Ansatz effizienter als Model-Checking sein kann. Allerdings können generierte Beweise lange sein (abhängig von den verwendeten Regeln) und sind i.d.R. schwer zu verstehen, weswegen der Anwender Erfahrung im Beweisen von logischen Aussagen benötigt. [37]

Durch formale Methoden wird grundsätzlich nur das Systemmodell verifiziert, ohne die Interaktionen der E/E-Architektur mit der Umgebung zu berücksichtigen. Hierfür werden manuelle Methoden bzw. Simulationen benötigt. Diese erlauben eine umfassendere Visualisierung und sind einfach zu verstehen. Allerdings ist dafür der Aufbau von Verhaltensmodellen und deren Validierung nötig.

## 3.3. Arten von E/E-Architektursimulationen

Die Verwendung von Simulation zur Verifikation von E/E-Systemen ist abhängig von dem jeweiligen Anwendungs- bzw. Testfall. Manche Testfälle erlauben durch Abstraktion oder Vernachlässigung die isolierte Betrachtung einer Domäne. Dafür werden üblicherweise keine hybriden Modelle benötigt. Andere erfordern die gleichzeitige Simulation mehrerer Domänen, was wiederum den Einsatz von hybriden Modellen erfordern kann. In den nächsten Unterabschnitten werden Beispiele für Anwendungsfälle beider Arten von Simulationen nach dem Stand der Technik zusammengetragen.

### 3.3.1. Simulationen mit Fokus auf einer Domäne

Im Folgenden werden Beispiele für Simulationen aus den einzelnen E/E-Domänen betrachtet. Simulationen können die ganze Domäne oder einzelnen Komponente daraus umfassen. Es werden i.d.R. keine hybriden Modelle gebildet.

#### Funktionale Simulation

Die funktionale Simulation dient zur Überprüfung der Logik oder von Algorithmen, unabhängig von der verwendeten Hardware oder Kommunikationsprotokollen. Ein Beispiel dafür ist der Einsatz von Simulation nach Tjonnaas [274] für die Auslegung eines Reglers zur Gierratenstabilisierung. Das Hardwaresystem wird hierbei vernachlässigt.

#### Hardware-Simulation

Bei der reinen Hardware- bzw. Performanz-Simulation liegt der Fokus auf der Beurteilung der Hardware. Die Verarbeitungsgeschwindigkeit von ECUs kann nach den Ansätzen von Li [161] und Shankar [236] mittels Simulation ohne implementierte Software beurteilt werden. Dabei wird die Software als eine Menge von auszuführenden Aufgaben mit zugewiesenen Rechenzeiten abstrahiert.

#### Netzwerk-Simulation

Mit einer isolierten Netzwerksimulation können z.B. Latenzzeiten, die bei der Kommunikation zwischen den HWCs der E/E-Architektur anfallen, ermittelt und optimiert werden. Ein Beispiel dafür zeigt Kawahara [142]. Hierbei geht es um die Evaluation und Auswahl eines Algorithmus zur Konvertierung von Nachrichten zwischen dem *Controller Area Network (CAN)* und Ethernet mittels Simulation. Zur Überprüfung der Latenz werden inhaltslose Nachrichten periodisch gesendet und konvertiert.

#### Leitungssatz-Simulation

Mit einer isolierten Leitungssatzsimulation können sehr unterschiedliche Fragestellungen beantwortet werden, wie die folgenden Beispiele zeigen.
Noguchi [193] untersucht die mechanische Beanspruchung des Leitungssatzes, ohne elektrische Ströme näher zu betrachten.
Diebig [69] nutzt Simulation zur Optimierung der Dimensionierung von Leitungssatzkomponenten. Es werden die elektrischen Ströme betrachtet, wobei für die Verbraucher statische Spitzenlasten angenommen werden.
Braun [35] nutzt ebenfalls Simulation zur Optimierung der Dimensionierung von Leitungssatzkomponenten. Verbraucher werden als ohmsche Lasten abstrahiert.
Ruf [225] beschäftigt sich ebenso mit der Optimierung der Dimensionierung von Leitungssatzkomponenten. Hierbei dienen ein realer Prüfstand und Szenarien als Basis für das Simulationsmodell.
Rueda [224] optimiert mittels Simulation Kabeldurchmesser. Dabei werden elektromagnetischer Felder berücksichtigt, die durch Kabelbündelung entstehen.

### 3.3.2. Multi-Domänen-Simulation

Bei der Multi-Domänen-Simulation von E/E-Architekturen werden mehrere Domänen zeitgleich simuliert, was sich entweder nach Hettig und Karner [115, 140] als *Co-Simulation* oder nach Bucher [43] *integriert*, d.h. in einem einzigen Simulator mit hybriden Modellen, realisieren lässt. Multi-Domänen-Simulationen können z.B. bei der Entwicklung von FAS [43] und elektrischen Fahrzeugen [115], bei kritischen Laufzeitanalysen [140] oder zur Prüfung der Netzwerkangriffssicherheit [86] eingesetzt werden.

## 3.4. Erstellen von Simulationsmodellen aus Architektur- und Strukturmodellen

Ein Nachteil der Simulation gegenüber formalen und statischen Verifikationsmethoden ist die nötige Erstellung von Simulationsmodellen. Außerdem muss sichergestellt sein, dass die Verhaltensmodelle validiert (vgl. Unterabschnitt 2.2.1) und insbesondere bei 3D-Simulationen oder im Verbund mit reellen Komponenten performant genug sind. Bestehende Ansätze zum

Erstellen von Simulationsmodellen, basierend auf Architektur- und Strukturmodellen, werden im Folgenden erläutert. Im Anschluss werden ergänzend Methoden zur Steigerung der Skalierbarkeit betrachtet.

### 3.4.1. AUTOSAR-Export und externe Verhaltensmodellierung

Im Automobilbereich ist der Export einer AUTOSAR-Architektur als *.ARXML*-Datei aus dem E/E-Architekturmodellierungswerkzeug üblich. Die exportierte Struktur kann anschließend, in Form leerer und über Schnittstellen verknüpfter Blöcke, in den Zielsimulator importiert werden (vgl. [261]). Dort werden die importieren Blöcke dann mit Verhaltensmodellen verfeinert, was die Generierung von C-Code und dessen Ausführung auf reeller und virtueller AUTOSAR-Hardware erlaubt. Beim Einsatz einer virtuellen, generischen Hardware zur Ausführung des generierten Codes werden geeignete Geräteabstraktionen verwendet [66, 180, 278]. Reimporte einer extern geänderten AUTOSAR-Struktur zurück zum E/E-Architekturmodellierungswerkzeug sind möglich.

### 3.4.2. Integrierte Modellsynthese und Modellgenerierung

Reimporte sind fehleranfällig in Bezug auf die Datenkonsistenz. Erfolgen Änderungen an einem Modellelement parallel im E/E-Architekturmodellierungswerkzeug und in einem externen Werkzeug, so können die Modelle bei einem Reimport ggf. nicht mehr konsistent verschmolzen werden. Es kann zu Fehlern oder Datenverlust kommen. Deshalb und aufgrund einer besseren Dokumentier- und Nachverfolgbarkeit, wählt Bucher [43] einen alternativen, integrierten Ansatz. Dieser bildet die Grundlage dieser Arbeit und wird daher im Folgenden genauer beschrieben. Abbildung 3.6 zeigt das dazugehörige Konzeptbild. Nach entsprechenden Abbildungsvorschriften wird das Simulationsmodell aus einer oder mehreren Ebenen eines PREEvision-Modells (**variantenreiches E/E-Architekturmodell**) synthetisiert. Dabei können unterschiedliche Varianten beachtet werden. Die Synthese ist ***LA**-zentriert*, d.h. die Struktur des Simulationsmodells und dessen Komponenten basieren auf dem LA-Modell. Zusätzlich können Aspekte aus weiteren Ebenen dekoriert werden. Zielsimulator ist der Multi-Domänen-Simulator Ptolemy II (vgl. Unterabschnitt 2.4.2). Um das Verhalten der LA-Blöcke in PREEvision zu modellieren und anschließend zu synthetisieren, werden zwei Hauptwege umgesetzt. Da in PREEvision standardmäßig schon Zustandsautomaten modelliert und den LA-Blöcken zugewiesen werden können, wurde für sie einerseits eine Abbildung hin zu Ptolemy-II-ModalModels (siehe Unterabschnitt 2.4.2) geschaffen. Zur erweiterten Verhaltensmodellierung wurde andererseits eine Hilfsebene, genannt *Behavioral Logical Architecture (BLA)*, eingeführt. Dort kann das Verhalten der LA-Blöcke, innerhalb von ihnen eindeutig zugewiesenen BBs, mit Stellvertreter-Blöcken aus einer importierten Ptolemy II-Bibliothek modelliert werden. Diese Stellvertreter-Blöcke liegen in Form von LA-Blöcken vor (**Ausführbare Funktionsbibliothek**). Über eine PM kann dann mittels der Struktur aus der LA, dem Verhalten aus der BLA und zusätzlich dekorierten Informationen (**Analyse-Dekorierer**), welche anhand einer durch die übrigen Ebenen erfasst werden, das Simulationsmodell generiert werden. Simulationsmodelle können für unterschiedliche **Varianten** synthetisiert und ihre **Simulationsergebnisse** manuell oder automatisiert verglichen werden. Somit können E/E-Architekten bei der

Bewertung ihres Designs unterstützt werden und auf Basis der Simulation entsprechende Modelländerungen (**Iterative Verbesserungen**) durchführen. [43]

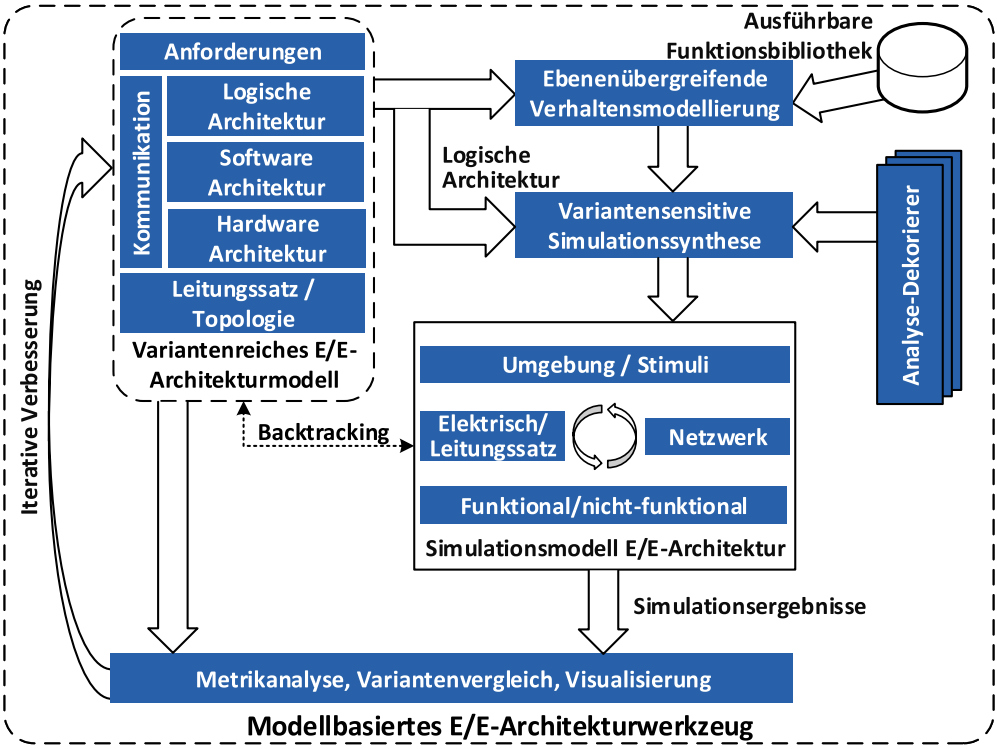



Abbildung 3.6.: Ansatz zur integrierten Simulationssynthese und dynamischen Bewertung modellbasierter E/E-Architekturen (Quelle: [43]).

Der vorgestellte Ansatz zur Modellsynthese reduziert nicht nur die Fehleranfälligkeit, sondern kann sich in Abhängigkeit der Abbildung auch auf die Laufzeit der Simulation auswirken. Daneben gibt es noch weitere Arbeiten, welche die Synthese bzw. Generierung von Simulationsmodellen thematisieren:

Zeigler [308] beschreibt die Generierung von Simulationsmodellen aus Struktur- und Verhaltensmodellen mit dem Werkzeug **MS4 ME** (siehe Abbildung 3.7). Hierbei wird die Struktur des Modells formal über eine *System Entity Structure* (**SES**) beschrieben, welche u.a. die Entitäten und Variablen eines Systems sowie die zwischen unterschiedlichen Systemen ausgetauschten Nachrichten enthält. Das Verhalten der Entitäten wird dabei formal mit Finite Deterministic DEVS (**FDDEVS**) beschrieben, welche Zustandsautomaten mit integrierter Zeitlogik sind. Als beispielhafte Anwendung werden aus einem Sequenzdiagramm sowohl die SES als auch FD-DEVS Formalismen generiert und daraus schließlich ein Simulationsmodell. [308]

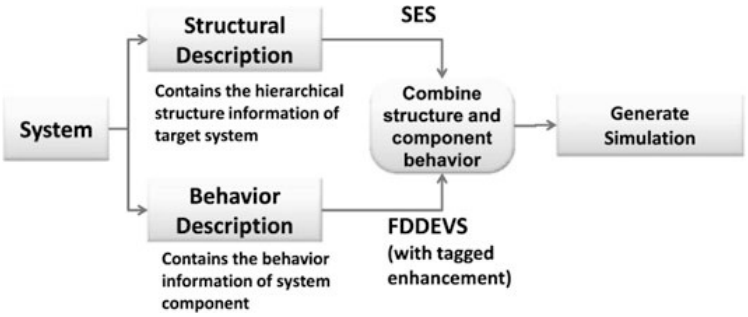



Abbildung 3.7.: Simulationsmodellgenerierung mit MS4 ME (Quelle: [308]).

Kahani [139] beschreibt, wie UML-konforme Zustandsautomaten formal aus einem Strukturmodell abgeleitet werden können. Dazu müssen zunächst Systemeinschränkungen definiert werden, welche Systemzustände, Nachrichten und dafür geltende Bedingungen enthalten. Den Ein- und Ausgängen müssen dann entsprechende Nachrichten zugewiesen werden. [139]

Vasilache [287] synthetisiert Zustandsautomaten aus UML-Sequenzdiagrammen, die zur Beschreibung von Szenarien verwendet werden. Da Szenarien jedoch nur eine bestimmte Form der Anwendung eines Zustandsautomaten beschreiben und eine Lösung deshalb nicht vollständig ist, werden sogenannte *Dependency-Diagramme* als Lösung vorgeschlagen. Diese beschreiben die Beziehungen aller möglichen Szenarien eines Systems und erlauben damit die Generierung einer vollständigen Lösung. [287]

### 3.4.3. Methoden zur Steigerung der Skalierbarkeit

Ergänzend zur Modellgenerierung lassen sich in der Literatur noch weitere Methoden zur Steigerung der Skalierbarkeit von Simulationen identifizieren. Diese werden in den folgenden Abschnitten beschrieben.

#### Template-basierte Modellgenerierung

Benjamin [23] beschreibt ein *Ontology driven Simulation Modeling Framework (OSMF)* zur schnellen Komposition von verteilten Simulationen. Dazu werden vom Anwender editierbare Template-Simulationsmodelle aus einer Bibliothek benutzt. Es wird primär das Ziel verfolgt eine schnelle und im Aufwand reduzierte Modellierung zu realisieren. [23]

Montecchi [185] setzt wiederverwendbare Teilmodelle als eine effiziente Möglichkeit ein, um großmaßstäbliche Modelle zu evaluieren und gleichzeitig die Wartbarkeit zu erhöhen. Die Arbeit beschreibt einen formalen, modellbasierten Ansatz, um die Teilmodelle anhand von *Kompositionspatterns* zu einem Gesamtmodell zu kombinieren. Die Vorgehensweise ist in Abbildung 3.8 dargestellt. Experten sollen ausgehend von der Architektur (**Architecture**) und den Anforderungen (**Requirements**) eine entsprechende Template-Bibliothek (**Model Templates**

**Library**) aufbauen (**1**). Templates sind formal in der entwickelten *Template Models Description Language* (**TMDL**) beschrieben. Ebenso werden zu analysierende Systemkonfigurationen in Form von Szenarien mit TMDL beschrieben (**2**). Im nächsten Schritt werden „Szenarien“ (**Scenario #...**) als Vorschrift verwendet, um die Templates in Abhängigkeit unterschiedlicher Konfigurationen (**Configuration #...**) zu einem Modell zusammenzusetzen (**3**). [185]

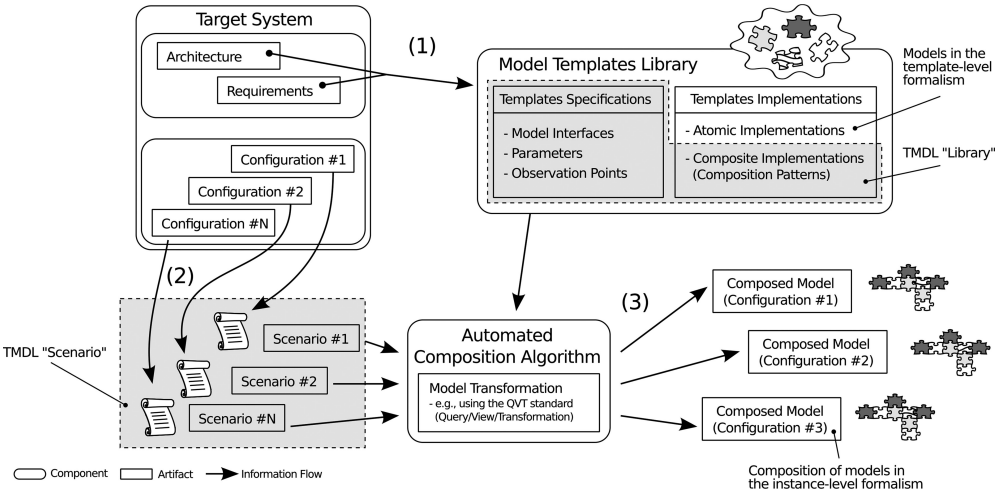



Abbildung 3.8.: Vorgehensweise zur automatischen Generierung von Modellen (Quelle: [185]).

Moradi [186] setzt vordefinierte und validierte Modellkomponenten, d.h. Templates, zu Simulationsmodellen zusammen, um die Kosten bei der Entwicklung von Simulationen zu senken und deren Benutzerfreundlichkeit zu erhöhen. Zur Definition der Templates verwendet er den Standard *Base Object Model (BOM)* (vgl. [238]). Das zu simulierende Modell wird dagegen in der Hochsprache *Simulation Reference Markup Language (SRML)* (vgl. [306]) definiert. Über ein sogenanntes *BOM-Matching* findet dann automatisiert, anhand der SRML-Beschreibung und mit in BOMs hinterlegten Metadaten, eine passende Auswahl und Verknüpfung der BOMs statt, wodurch das Simulationsmodell gebildet wird. [186]

**Diskussion** In der Literatur gibt es unterschiedliche Ansätze, um die Entwicklung von Simulationsmodellen mithilfe von Modellgenerierung zu vereinfachen und zu beschleunigen. Für ein Simulationsmodell wird dabei sowohl die Struktur als auch das Verhalten der Komponenten benötigt, wobei sich die Abhängigkeiten i.d.R. formal beschreiben lassen. Der zusätzliche Einsatz von Templates spielt gerade bei großmaßstäblichen Architekturen eine wichtige Rolle, da sowohl Verhaltensmodellierung als auch Validierung wegfallen oder zumindest vereinfacht werden. In den gefundenen Arbeiten wird allerdings nicht auf die Grenzen der Skalierbarkeit eingegangen, welche insbesondere von den emergenten Modelleigenschaften bestimmt werden, die bei einer Komposition mit Templates entstehen können. Ebenso wird in der betrach-

teten Literatur nicht untersucht, wie sich die Auswahl der für die Komposition verwendeten Templates und deren Abstraktionsgrad auf die Gesamtlaufzeit auswirken.

### Szenarien

Szenarien können eingesetzt werden, um den Zustandsraum einzuschränken, da vor allem bei großen Simulationsmodellen nicht alle Systemzustände bei der Verifikation abgedeckt werden können [229]. Außerdem lässt sich bei der Verifikation von FAS keine vollständige Testfallabdeckung ohne den Einsatz von Szenarien realisieren, da nicht alle Randfälle in der Testspezifikation berücksichtigt werden können [248].

Szenarien wirken sich also auf die Skalierbarkeit aus, weil sie die Simulationsmodellgröße einschränken, indem sie die Schnittstellen definieren, die für einen bestimmten Anwendungsfall benötigt werden. Im Folgenden werden bestehende Arbeiten betrachtet, in denen Szenarien für die Verifikation im automobilen Umfeld angewandt werden. Der Fokus liegt dabei auf der Art der Szenarienmodellierung. Im Anschluss werden entsprechende Modellierungstandards vorgestellt. Die Analyse schließt mit einem Fazit.

Hejase [113] nutzt Zustandsautomaten zur hierarchischen bzw. bedingten Modellierung von Szenarien, wie in Abbildung 3.9 dargestellt.

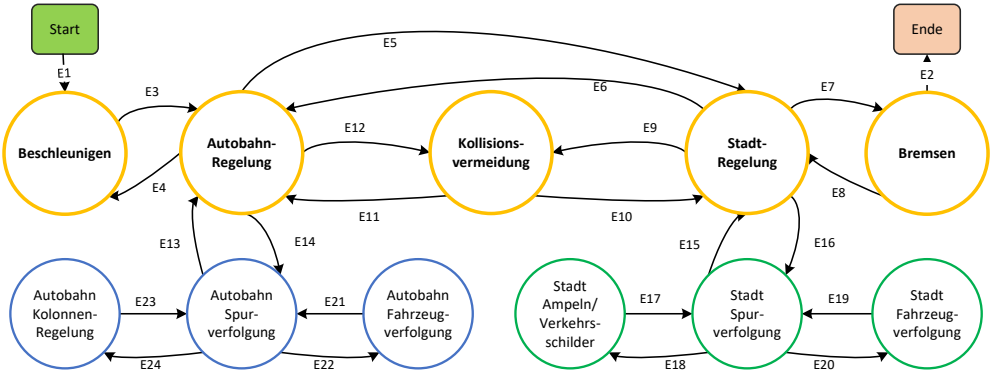



Abbildung 3.9.: Zustandsautomat eines autonomen Landfahrzeugs zur High-Level-Entscheidungsfindung (Quelle: eigene Darstellung nach [113]).

Der Zustandsautomat hat einen definierten **Start**- und einen definierten Endzustand (**Ende**). Die Übergänge werden durch Ereignisse (**E**) modelliert. Übergeordnete Zustände (gelb dargestellt) werden teilweise in kleinere Subszenarien (blau und grün dargestellt) unterteilt. Bei einer Simulation könnte ein dynamisches Nachladen von möglichen Folgezuständen, ausgehend von dem aktiven Zustand, für eine bessere Performanz genutzt werden. [113]

Bach [14] nutzt eine eigene Implementierung zur Modellierung von Szenarien, basierend auf einem *Regel- und Domain-Modell*. Das *Regelmodell* ähnelt einem Zustandsautomaten, wobei

Übergänge jedoch nicht über Ereignisse ausgelöst, sondern durch einschränkende Bedingungen zurückgehalten werden. Dadurch wird ein einziges Regelmodell zur Basis mehrerer Szenarioinstanzen, die dann als Zustandsautomat mit auslösenden Ereignissen modelliert werden. Für jede Szenarioinstanz sind mit einem übergeordneten *Domain-Modell* klare Beziehungen zu den Entitäten innerhalb der Simulation modelliert. Dies mindert die Fehleranfälligkeit bei der Szenarienmodellierung und gibt einen klaren Rahmen für die Modellierer vor. [14]

Im Projekt *PEGASUS* [201] wird der Standard *OpenSCENARIO* (vgl. [13]) zur Beschreibung von Szenarien verwendet. Dieser ist nachfolgend beschrieben.

Die Beschreibung von Szenarien im quelloffenen Fahrsimulator OpenDS [172] erfolgt mithilfe von .xml Dateien. Es wurden hierzu eigene XML-Schemen entwickelt, um die Konsistenz zu dem selbst implementierten Code, der die Dateien verarbeitet, zu gewährleisten (siehe Unterabschnitt 2.4.4).

**Modellierungsstandards** Im Folgenden werden Standards beschrieben, die in der Praxis zur Modellierung von Szenarien eingesetzt werden.

**UML-Verhaltensdiagramme** Grundsätzlich lassen sich Szenarien mit UML-Verhaltensdiagrammen modellieren. Dazu gehören Sequenzdiagramme [287], Aktivitätsdiagramme [188], Zustandsdiagramme [113], Anwendungsfalldiagramme [20], Kommunikationsdiagramme und Timing-Diagramme. Nach dem Standard bieten jedoch nur Sequenzdiagramme und Timing-Diagramme die Möglichkeit, Abfolgen unter Beachtung der Zeit zu modellieren. In Anwendungsfalldiagrammen sind außerdem Annotationen auf den Beziehungen erforderlich, um Bedingungen zu modellieren.

**OpenSCENARIO** OpenSCENARIO (vgl. [13]) definiert ein Datenmodell in Form eines XML-Derivats, um Szenarien zu beschreiben, die in Fahr- und Verkehrssimulatoren zum Testen von Fahrzeugen verwendet werden. Durch den Standard werden dynamische Entitäten beschrieben, wodurch der OpenDRIVE Standard (vgl. [12]), mit dem statische Elemente beschrieben werden, ergänzt wird. Durch Referenzen können beide Datenmodelle miteinander verknüpft werden. Abbildung 3.10 zeigt die Struktur eines *OpenSCENARIO-Storyboards*, welches zur Beschreibung eines kompletten Szenarios verwendet wird.
Nach [13] legt ein Initialisierungselement (**Init**) legt die Startzustände und Parameter der Entitäten fest. Eine Geschichte (**Story**) besteht aus Akten (**Act**), die über Bedingungen ausgelöst werden. Ein Akt besteht indessen aus einer Manövergruppe (**ManeuverGroup**), die wiederum aus Manövern (**Maneuver**) und den daran beteiligten Akteuren (**Actors**) besteht. Die Bezeichnungen decken sich teilweise mit denen, die in Abbildung 2.7 für Interaktionsszenarien verwendet werden. Manöver enthalten globale Ereignisse (**Event**), die von Akteuren beim Eintreten bestimmter Bedingungen (*Conditions*, wie **StartTrigger**) gesendet werden und im Anschluss Aktionen (**Actions**) auslösen. Über eine Referenz (**CatalogReference**) können außerdem bestehende Manöver als Wiederverwendung hinzugefügt werden. [13]

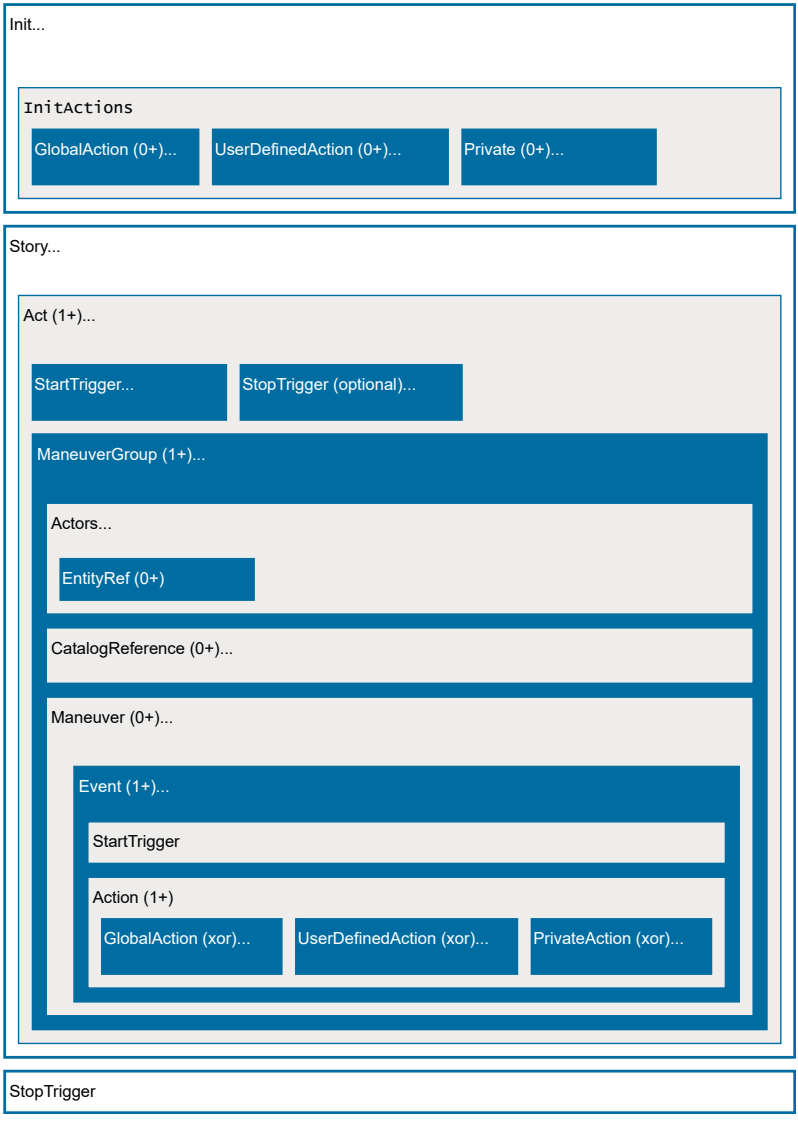



Abbildung 3.10.: Struktur eines OpenScenario Storyboards (Quelle: [13]).

**Diskussion** Szenarien können eingesetzt werden, um den Zustandsraum einzuschränken [229]. Es lassen sich sowohl grafische als auch textbasierte Formen zur formalen Modellierung von Szenarien finden. Entitäten und (bedingte) Interaktionen sind ein fester Bestandteil. Standards sind ferner notwendig, um die Freiheitsgrade einzuschränken und Schnittstel-

len vorzugeben, sodass die Verarbeitung des Szenarios in einem Simulator möglich ist. Der OpenSCENARIO-Standard gibt keine Modellierungsdiagramme vor, wobei die UML Möglichkeiten dafür bietet. Unterschiedliche UML-Diagrammarten ergänzen sich hierzu gegenseitig. UML-Sequenzdiagramme eignen sich, um Interaktionsszenarien abzubilden. Akteure, Entitäten oder Systeme (im Folgenden als Teilnehmer zusammengefasst) entsprechen den Lebenslinien. Interaktionen werden durch Nachrichten ausgedrückt. Bedingungen dafür können über Interaktionsoperatoren definiert und geprüft werden. So lassen sich Nachrichten abhängig vom Zustand des jeweiligen Teilnehmers versenden. Zur Modellierung dieser Zustände selbst benötigt man jedoch den Teilnehmern zugeordnete Zustandsdiagramme. In Zustandsdiagrammen fehlt allerdings Zeitlogik, wofür ergänzend UML-Timing-Diagramme verwendet werden können. Eine konkrete Auswirkung der Verwendung von Szenarien auf die Skalierbarkeit von Simulationen wird in keiner der in diesem Abschnitt vorgestellten Arbeiten untersucht.

### Weitere Methoden

In diesem Abschnitt werden weitere, zentrale Ansätze untersucht, die die Performanz von Simulationen verbessern sollen. Dabei findet eine Aufteilung in *automatische* und *manuelle* Methoden statt. Automatische Methoden lassen sich ohne den Modellierer durchführen, während manuelle Methoden aktiv vom Modellierer umgesetzt werden müssen.

**Automatische Methoden** Eine Methode ist die *verteilte Ausführung*. Zu deren Definition und Koordination lassen sich Standards verwenden. Dazu gehören das *Message Passing Interface (MPI)* (vgl. [169]) oder der *Microsoft-HPC-Pack*, mit denen sich Simulationen parallel ausführen lassen. In der Regel erfolgt die Anwendung automatisch durch entsprechende Simulatoren, welche die Standards implementieren.

Eine weitere Methode ist die *automatisierte Modellreduktion*, welche sich beispielsweise durch *orthogonale Dekomposition* [146] für kontinuierliche Simulationen oder dem *Backtracking-Algorithmus* für diskrete Simulationen [54] realisieren lässt.

Sagdeo beschreibt in [228] Beschleunigungstechniken für Verilog-Simulationen, welche sich aber generalisieren lassen. Dazu gehört einerseits die *Code-Generierung*, sodass das Simulationsmodell nicht interpretiert ausgeführt wird, sondern in Form von kompiliertem Code. Dieser kann zusätzlich durch Parallelisierung optimiert werden [220]. Außerdem beschreibt Sagdeo, dass die *Ausnutzung des CPU Cache für Simulationsdaten* eine positive Auswirkung auf die Laufzeit hat. Dieser Ansatz wird auch von Wang zur Laufzeitoptimierung von diskreten Simulationen [299] verwendet.

Eine weitere Methode ist die Erzeugung von *Ersatzmodellen* durch maschinelles Lernen [167, 107]. Hierbei wird das Verhalten des entsprechenden Modellteils oder der Entität durch ein stellvertretendes neuronales Netz ersetzt, das mit einer bekannten Ein- und Ausgangsmenge trainiert wird.

**Manuelle Methoden** Heegaard [112] beschreibt in seiner Analyse zur Laufzeitoptimierung von diskreten Simulationen zwei Klassen von manuellen Methoden. Sie zielen darauf ab die Anzahl der zu verarbeitenden Ereignisse einzuschränken. Dazu differenziert Heegaard zwischen *hybriden Techniken* und *statistischer Varianzreduktion*.
Hybride Techniken kombinieren diskrete Modelle mit mathematischen Gleichungen. Hervorzuheben ist dabei die hybride Technik *Dekomposition*, bei der das zu simulierende System in Subsysteme partitioniert wird. Dabei sollen möglichst viele dieser Subsysteme mathematisch modelliert werden, sodass nach deren einmaliger Ausführung die Ergebnisse gespeichert und für erneute Ausführungen wiederverwendet werden können.
Die statistische Varianzreduktion verfolgt hingegen durch Anwendung statistischer Methoden das Ziel die Stimuli einzuschränken. [112]

Motiviert durch die Aussage, dass die Performanz hybrider Simulationen mit diskreten und kontinuierlichen Anteilen oft nicht ausreicht, beschreibt Liu [165] einen Ansatz, um hybride Simulationen zu beschleunigen. Hierzu ersetzen Speicher und entsprechende Schnittstellen (**Shared Memory**) die sonst üblichen Wandler (**D/A**, **A/D**) zwischen Subsystemen mit unterschiedlichen Zeitausführungsformen. Dies zeigt Abbildung 3.11. Durch den Ansatz können die einzelnen Subsysteme unabhängig voneinander simuliert und auf unterschiedlichen Kernen ausgeführt werden. In der Evaluation konnte die Simulation dadurch um den Faktor 6,55 beschleunigt werden. [165]

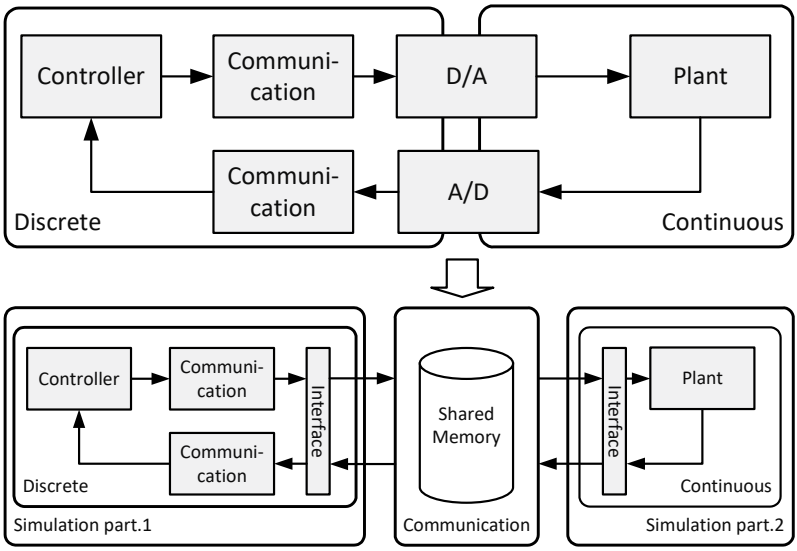



Abbildung 3.11.: Performanzsteigerung hybrider Simulationen durch Shared Memory (Quelle: Modifikation von [165]).

Choi [53] nutzt einen *Multi-Fidelity-Ansatz*, um die Effizienz von bestehenden Modellen zu steigern. Dabei kann der Ansatz sowohl für diskrete als auch für kontinuierliche Simulatio-

nen verwendet werden. Ausgehend von dem ursprünglichen Modell wird ein Ersatzmodell mit niedriger Wiedergabetreue entwickelt, d.h. mit höherer Abstraktion und entsprechend höherer Performanz. Ziel ist es herauszufinden, zu welchen Zeitpunkten in der Simulation eine hohe Wiedergabetreue benötigt wird. Zu diesen Zeitpunkten wird das ursprüngliche Modell ausgeführt. Zu den anderen Zeitpunkten wird dagegen das performantere, aber ungenauere Ersatzmodell ausgeführt. [53]

**Diskussion** Zentrale Ansätze automatisierter Methoden umfassen die Steigerung der Parallelisierbarkeit oder die automatisierte Erstellung performanter Ersatzmodelle. Sie sind allerdings nur in Abhängigkeit des verwendeten Simulators umsetzbar. Falls eine Methode nicht implementiert ist, können meist nur quelloffene Lösungen dahingehend erweitert werden. Für den Einsatz in dieser Arbeit kommen deshalb primär manuelle Methoden infrage. Diese erfordern hingegen die Erfahrung des Modellierers und sind i.d.R. nur abhängig von der Modellierungsform anwendbar.

Eine Dekomposition in Subsysteme lässt sich bei der Simulation von E/E-Architekturen anwenden, während Varianzreduktion nicht geeignet ist. Dies liegt daran, dass die Verifikation auf den in der Spezifikation festgelegten Anforderungen beruht, wodurch die Stimuli bereits festgelegt sind.

Der Ansatz von Liu konnte von Schwarz [S6] zum Aufbau einer reaktiven Testumgebung für E/E-Architekturen in angepasster Form umgesetzt werden. Dabei war eine Leistungssteigerung beim Verwenden von globalen Speichervariablen gegenüber Signalverbindungen möglich. Jedoch eignet sich die Methode nur für Simulatoren, welche die Modellierungsart Interaktion nutzen (vgl. Unterabschnitt 2.2.4). Für Simulatoren, die das anreichernde Prinzip verwenden, ist sie nicht anwendbar. Außerdem muss die zeitliche Synchronisation und Datenkonsistenz beim Verwenden des Speichers beachtet werden. Dies wurde von Liu vernachlässigt.

Für den Multi-Fidelity-Ansatz fehlt von Choi eine Aussage zur Einsetzbarkeit bei großmaßstäblichen, emergenten Systemen.

## 3.5. Einordnung und Abgrenzung dieser Arbeit

Diese Arbeit ist in die simulative Verifikation von E/E-Architekturmodellen einzuordnen. Dabei wird der Ansatz einer MIL-Simulation verfolgt. Diese beachtet die Aspekte Software, Hardware (inkl. generischer ECUs und dem Leitungssatz), Umgebung, Fahrdynamik und reaktive Testszenarien. Im Gegensatz zu *virtual Hardware in the Loop (vHIL)*-Simulationen werden keine emulierten ECUs verwendet, auf welchen kompilierter Code ausgeführt wird. Dadurch sollen Simulationen in frühen Entwicklungsphasen möglich sein, deren Anwendungsfälle, wie etwa die Verifikation von FAS, eine Multi-Domänensimulationen voraussetzen.

Unterabschnitt 3.2.3 zeigt, dass simulative Verfahren gegenüber formalen den Nachteil besitzen, dass zur Modellausführung ein Verhalten modelliert und validiert werden muss. Dies ist zeitlich aufwendig und fehleranfällig. Ziel dieser Arbeit ist es deshalb den Modellierungs- und Validierungsaufwand für Simulationsmodelle einzuschränken. Daher werden die Konzepte von Bucher [43] zur automatisierten **Generierung** von Simulationsmodellen aus E/E-

Architekturmodellen aufgenommen, angepasst und durch Methoden erweitert, die eine höhere Skalierbarkeit versprechen. Gerade die Skalierbarkeit wird nämlich in [229] im Vergleich zu formalen Methoden als Nachteil von Simulationen kritisiert. Es soll außerdem ein schnelles Feedback möglich sein.
Da bei Bucher [43] die Verhaltensmodellierung vorwiegend manuell durch den Entwickler erfolgt, wird eine Alternative benötigt, die eine höhere Automatisierung erlaubt. Dazu werden parametrisierbare, vorvalidierte **Modell-Templates** und eine HW-zentrierte Abbildung eingesetzt. Modell-Templates können die Skalierbarkeit erhöhen, wenn sie zuvor auf eine geringere Laufzeit optimiert wurden. Durch die Vorvalidierung wird der nötige Zeitaufwand der Validierung reduziert und die Genauigkeit der Ergebnisse im Vergleich zu einer manuellen Implementierung erhöht. Die HW-zentrierte Abbildung nutzt, im Vergleich zur LA-zentrierten Abbildung von Bucher [43], das physikalische E/E-Architekturmodell als Basis des Simulationsmodells. Wichtige Modellkomponenten sind hier Sensoren, Aktoren, ECUs und Leitungssatzkomponenten, wie Batterien, Kabel oder Spleiße. Aufgrund ihrer spezifischen Modellierung in PREEvision, lassen sich diese Komponenten mit sogenannten *Standard-Templates* verknüpfen, was den Modellierungsaufwand weiter einschränkt. Beispielsweise kann einer Leitung ohne die manuelle Modellierung der Verknüpfung ein parametrisierbares Leitungssimulationsmodell als Standard-Template zugewiesen sein. Parameter wie Länge, Durchmesser und Material lassen sich dann direkt aus dem E/E-Architekturmodell verwenden. Bei LA-Komponenten ist dies nur sehr eingeschränkt möglich, da sie generischer sind.
Ferner wird das Verhaltensmodell der E/E-Architektur in dieser Arbeit in eine szenarienbasierte und reaktive Testumgebung eingebettet. **Szenarien** helfen mit ihren Schnittstellen das SOI zu bestimmen und damit die Simulationsmodellgröße einzuschränken, was sich auf die Skalierbarkeit auswirkt. Durch die zusätzliche Beachtung von Umgebung und Fahrdynamik gegenüber Bucher [43] werden ferner alle notwendigen Komponenten zur Simulation von FAS nach [303] integriert. Dies geschieht in reaktiver Form, sodass Umgebung und E/E-Architektur in einer geschlossenen Schleife stehen und sich in ihrem Verhalten gegenseitig beeinflussen.

Tabelle 3.2 stellt zusammenfassend noch einmal die zuvor in Abschnitt 3.4 beschriebenen Arbeiten und ihre verwendeten Methoden dar:

Tabelle 3.2.: Abgrenzung der Arbeit.

| | **Generierung** | **Modell-Templates** | **Szenarien** |
|---|---|---|---|
| Bucher [43] | X | | |
| Zeigler [308] | X | | |
| Kahani [139] | X | | |
| Vasilache [287] | X | | |
| Bejamin [23] | | X | |
| Montecchi [185] | X | X | (X) |
| Moradi [186] | | X | |
| Hejase [113] | | | X |
| Bach [14] | | | X |
| **Neubauer** | **X** | **X** | **X** |

Es ist erkennbar, dass diese Arbeit (**Neubauer**) alle Methoden kombiniert. Bei **Montecchi** [185] dienen Szenarien zur Auswahl von Templates für eine Simulation. Sie sind damit eher als eine vom Anwendungsfall abhängige Modellkonfiguration zu sehen. Im Gegensatz dazu sind Szenarien in dieser Arbeit reaktive Komponenten und selbst Teil der Simulation. Sie beschreiben den Ablauf der Simulation, generieren davon abhängig Stimuli und verifizieren Zustände oder Signale der E/E-Architektur. Außerdem liefern sie die Schnittstellen zur Identifikation des SOI, das mit seiner Struktur und seinen Komponenten das Simulationsmodell bestimmt.

# 4. Forschungsfragen und Methodik

In diesem Kapitel werden zunächst die Herausforderungen für E/E-Architektursimulationen dargestellt, die sich aus der Beschaffenheit von den zugrundeliegenden E/E-Architekturmodellen ergeben. Daraus werden Forschungsfragen, Annahmen und Einschränkungen abgeleitet. Im Anschluss wird die Methodik zur Beantwortung der Forschungsfragen, in Anlehnung an den Stand der Technik, beschrieben.

## 4.1. Herausforderungen bei der Simulation von E/E-Architekturen

Die Herausforderungen, die bei der Simulation von E/E-Architekturmodellen entstehen, lassen sich u.a. aus den Eigenschaften solcher Modelle ableiten. Daher werden zunächst die Charakteristiken von PREEvision-E/E-Architekturmodellen und daraus generierten Simulationsmodellen herausgearbeitet. Im Anschluss werden daraus die konkreten wissenschaftlichen Herausforderungen abgeleitet.

### 4.1.1. Charakteristiken von PREEvision-E/E-Architekturmodellen

Ein PREEvision-E/E-Architekturmodell besteht i.d.R. aus dreidimensional verknüpften Komponenten. Abbildung 4.1 zeigt beispielhaft die Kernzusammenhänge und Merkmale, die es bei einer Überführung in ein Simulationsmodell zu beachten gibt. Innerhalb einer Ebene (erste Dimension) werden Komponenten und ihre Verbindungen in Form von Strukturdiagrammen modelliert. Verbindungen können einen Informations-, Stoff- oder Energiefluss darstellen. Beziehungen zwischen Komponenten aus unterschiedlichen Ebenen können mithilfe von Zuordnungen (**Mappings**) modelliert werden (zweite Dimension). Diese Beziehungen können unterschiedliche Sichten auf eine Komponente oder Zugehörigkeiten (Kompositionen) darstellen. In der dritten Dimension kann eine Komponente mit internen Details verfeinert werden, indem sie mit entsprechenden Diagrammen verknüpft wird. Die Schnittstellen dieser Diagramme entsprechen denen der Komponente oder einer Teilmenge davon. Darüber hinaus sind jeder Komponente Eigenschaften bzw. *Attribute* in Abhängigkeit ihres Typs zugewiesen.

Im oberen Teil von Abbildung 4.1 sind LA und SA (analog zu [246]) zusammengefasst, da sich die Prinzipien der beiden Schichten ähneln. Verbindungen auf diesen Ebenen sind logisch. Dabei wird der Informationsfluss gerichtet über logische Verbinder zwischen den Ein- und Ausgängen (►/<) modelliert. Logischen Verbindern können darüber hinaus Schnittstellen mit Datenelementen zugewiesen werden. Dadurch können Datentypen festgelegt und Wertebereiche eingeschränkt werden. Außerdem wird eine Signalpropagation ermöglicht. Einen Spezialfall stellen **Umgebungs-Funktionen** dar. Der zwischen Umgebungsanschlüssen (▷/□) modellier-

te Datenfluss ist kein Bestandteil der realen E/E-Architektur in Form von Signalen, sondern stellt Interaktionen mit der Umgebung dar. Umgebungs-Funktionen sind kein Bestandteil der E/E-Architektur selbst.

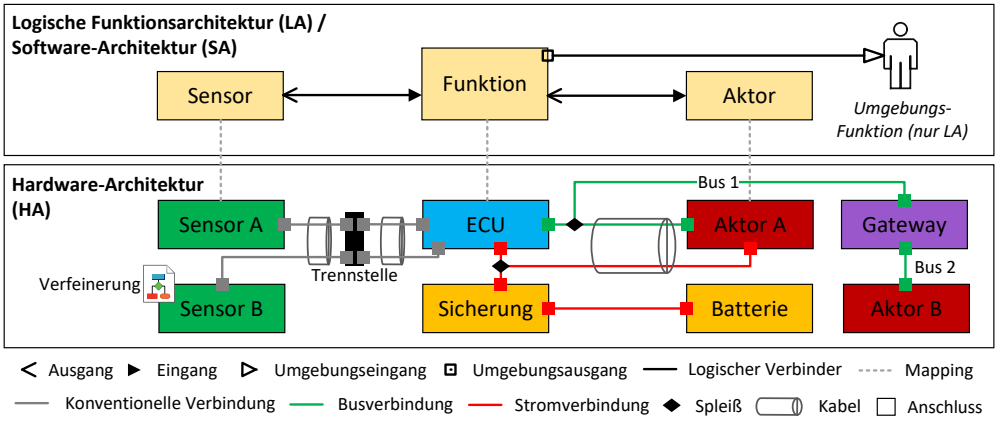



Abbildung 4.1.: Merkmale von PREEvision-E/E-Architekturen zur Beachtung bei der Überführung in ein Simulationsmodell (Quelle: Modifikation von [P3]).

Der untere Teil stellt ein Beispielmodell der physikalischen HA dar. Die Komponenten lassen sich auch in der Realität im Fahrzeug finden. Softwarekomponenten werden den **ECUs**, bzw. deren Recheneinheiten, zugewiesen. **Sensoren** und **Aktoren** aus der LA und SA werden über Mappings mit ihren entsprechenden HWCs verknüpft. Dies ermöglicht ferner ein automatisiertes Routing der Signalwege. Die Komponenten sind in dieser Ebene über **konventionelle Verbindungen**, **Stromverbindungen** oder **Busverbindungen** miteinander verbunden. Unter konventionellen Verbindungen sind Drahtleitungen zu verstehen, auf denen entweder analoge oder digitale Signale zwischen den angebundenen Komponenten versendet werden. Busverbindungen sind ebenfalls für die Kommunikation gedacht, aber rein digital. Sie sind in Abbildung 4.1 vereinfacht dargestellt und können in der Realität aus mehreren Leitungen bestehen. Übersetzungen zwischen unterschiedlichen Bussen oder Bustypen werden durch **Gateways** realisiert. Stromverbindungen sind rein analog und nur für die Versorgung der Verbraucher bestimmt. Leitungen können in Form von Adern zu **Kabeln** zusammengefasst werden. Kabel und einzelne Adern besitzen an ihren Enden Verbindungsstecker. Sobald eine Ader mit mehr als zwei Anschlüssen verbunden werden soll, werden **Spleiße** oder entsprechende Verteiler benötigt. Einen weiteren Aspekt bilden **Trennstellen**, die bei der Verortung und dem Routing der Kabel im Fahrzeug zu beachten sind.

### Größe und Verteilung

Eine in [219] bei *Daimler* modellierte PREEvision-Produktlinie umfasst insgesamt 1,5 Millionen Artefakte, wobei 52 % davon zur Softwarearchitektur und 22 % zum Leitungssatz gehö-

ren. Des Weiteren gehören 8 % zum Kommunikationsdesign und 6 % zu den Mappings. Es befinden sich hierbei ca. 1000 E/E-Komponenten, 6000 Pins, 4000 Kabel und 300 Diagramme im Modell. Das Modell basiert auf einem Serienstand und ist beschrieben im herstellereigenen Werkzeug XDIS (Zentrale Vernetzungsdatenbank). Die Anforderungen wurden in einem externen Tool modelliert und sind daher Bestandteil des Modells. Zusätzliche Zahlen lassen sich in anderweitigen Quellen finden. So sind aktuell bis zu 150 ECUs [65], 50 Sensoren [109] und über 100 Aktoren [158] in aktuellen Fahrzeugen verbaut. Bezüglich der Software wird sich laut [65] die Anzahl der Codezeilen bis zum Jahr 2030 verdreifachen, wobei der Speicherbedarf dafür in Zukunft um den Faktor 10000 auf bis zu 80 Gb ansteigen wird [309].

Insgesamt stellen diese Zahlen jedoch nur einen Ausschnitt bzw. eine Orientierung dar, da es keinen herstellerübergreifenden Designstandard für E/E-Architekturen gibt und Hersteller deshalb eine große Modellierungsfreiheit besitzen. Unter Berücksichtigung von Unterabschnitt 3.1.1 lassen sich aber Tendenzen festlegen. Seitens der Hardware ist aufgrund des Wechsels zur fahrzeugeinheitlichen Architektur [309] ein Anstieg der Anzahl an ECUs unwahrscheinlich. Die Anzahl der Sensoren und Aktuatoren wird jedoch zunehmen. Im Vergleich dazu wird die Software am stärksten wachsen und damit die größten Herausforderungen an eine skalierbare Simulation stellen.

#### Relevante Merkmale für die Generierung von Simulationsmodellen

- Das PREEvision-Modell stellt Strukturmodelle in verschiedenen Sichtweisen (Ebenen) zur Verfügung. Es handelt sich vorwiegend um Komponentendiagramme, die aus hierarchisch verfeinerbaren Blöcken bestehen, welche über spezifische Schnittstellen verbunden sind. Bestandteile der Modelle, *Artefakte* genannt, können über die Ebenen bzw. Sichtweisen hinweg miteinander verknüpft werden.
- Verhalten kann ab Version 9.5 explizit in Form von Zustandsdiagrammen auf den Prototypen der Blöcke oder in Form von Anwendungsfalldiagrammen beim Anforderungsmanagement modelliert werden. Implizit lassen sich auch Attribute der Artefakte nutzen, um beispielsweise ein zyklisches Sendeverhalten zu modellieren. Außerdem können Latenzen grafisch durch Zeitpfade modelliert werden.
- Auf der LA ist die E/E-Architektur implementierungsunabhängig und damit auf dem höchsten Abstraktionsgrad modelliert. Logische Blöcke können entweder in Hard- oder Software realisiert werden. In der Regel werden Funktionsblöcke entweder als Softwarekomponenten oder als Aktor- bzw. Sensorblöcke in Hardware realisiert. Signale sind rein logisch, also entsprechend der obersten Ebene des *Open-Systems-Interconnection-Schichtenmodells (OSI-Schichtenmodell)* [81], dargestellt. Des Weiteren stehen Umgebungsfunktionen zur Verfügung, die eine Modellierung der Interaktion zwischen der E/E-Architektur ihrer Umgebung ermöglichen. In einem logischen Diagramm ist zudem der Datenfluss zwischen den Komponenten modelliert.
- Auf der SA lassen sich SWCs und ihre Schnittstellen modellieren. Dies ist AUTOSAR-konform möglich, wobei entsprechende Artefakte zur Verfügung stehen. Das Scheduling lässt sich dabei über den Datenfluss, Ausführungszeiten, und entsprechende Trigger für den Kontrollfluss modellieren. Hierzu müssen die SWCs jeweils einer Ausführungs-

einheit auf der HA zugeordnet sein. Die SWCs können des Weiteren durch Klassen-, Anwendungsfall- oder Zustandsdiagramme verfeinert werden. Artefakte zur Modellierung von *serviceorientierten Architekturen (SOAs)* sind ebenfalls vorhanden.

- Auf der HA befinden sich die physikalischen Modellartefakte (HWCs) inklusive der Leitungen zur Kommunikation, der Leistungsversorgung und -verteilung. Durch eine Zuordnung der HWCs zu entsprechenden Bauräumen in einem geometrischen Modell des Fahrzeugs, lassen sich die Leitungswege verlegen und dadurch benötigte Trennstellen bestimmen.

### 4.1.2. Charakteristiken von E/E-Architektursimulationsmodellen

Der generelle Aufbau und die Komponenten eines Simulationsmodells werden in Abbildung 2.5 beschrieben. Aufgrund der Tatsache, dass es sich bei E/E-Architekturen immer mehr um mobile CPS handelt, können je nach Anwendungsfall der Simulation hybride Simulationsmodelle (vgl. Unterabschnitt 2.2.4) entstehen. Bei einer automatischen Generierung können diese insbesondere emergente Eigenschaften aufweisen und müssen validiert werden. Zur Identifikation der wissenschaftlichen Herausforderungen, werden im Folgenden die Charakteristiken mit Bezug auf die Performanz beschrieben.

#### Charakteristiken mit Einfluss auf die Performanz

Abbildung 4.2 zeigt ein Beispielmodell zur Darstellung der Eigenschaften von Simulationsmodellen mit Einfluss auf die Performanz.

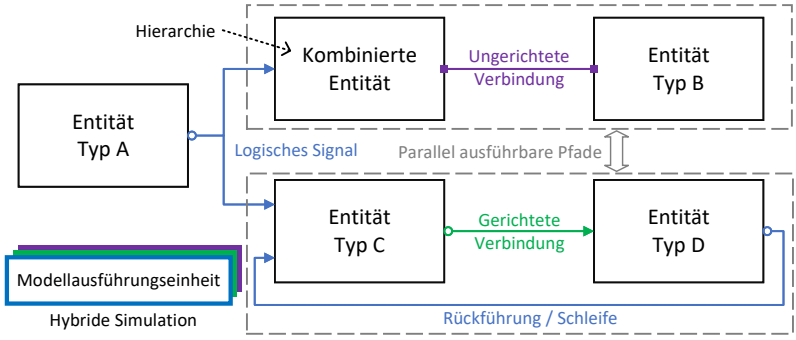



Abbildung 4.2.: Beispiel zur Darstellung der Eigenschaften von Simulationsmodellen mit Einfluss auf die Skalierbarkeit.

Damit lassen sich die Einflussparameter auf die Performanz dass folgendermaßen erläutern:

- Die **Modellstruktur** bestimmt zu welchem Anteil eine parallele/verteilte Ausführung der Simulation möglich ist. Dabei sind möglichst viele **parallel ausführbare Pfade** und geringe

Abhängigkeiten zwischen ihnen, aufgrund des zur Synchronisation entstehenden Rechenaufwands, erforderlich (vgl. Unterabschnitt 2.3.2). Des Weiteren sollte die Struktur möglichst wenige **Schleifen** aufweisen, da bei der Berechnung die Kausalität gewährleistet sein muss, was je nach Zeitausführungsform zu zusätzlichem Aufwand für das Scheduling der Entitäten führt (siehe Absatz Kausalität in Unterabschnitt 2.2.4).

- Jede Entität verursacht abhängig von ihrem Typ, d.h. dem implementierten Algorithmus entsprechend, anteilmäßig Kosten an der Laufzeit und dem Speicherverbrauch.
- Die **Modellgröße** sollte so klein wie möglich gewählt werden. Dies betrifft die Anzahl der nicht verfeinerbaren Entitäten, derer Parameter und Schnittstellen sowie der Signale. Der Berechnungsaufwand einer Entität hängt hierbei zwar primär von der Schwere des ihr zugewiesenen Algorithmus ab, aber jede weitere Komponente führt auch zu zusätzlichem Aufwand für das Scheduling und der Datensynchronisation. Außerdem wird der Ressourcenverbrauch beim Zwischenspeichern von Parametern und Zustandsvariablen erhöht.
- Die Anzahl der verwendeten **Hierarchiestufen**, in Form von kombinierten Entitäten, kann je nach Implementierung im Simulator zu Laufzeiteinbußen und Speicherverbrauchszuwachs führen. Dies ist beispielsweise der Fall, wenn die Hierarchie vor der Ausführung nicht aufgebrochen wird oder kombinierte Entitäten zu zusätzlichen Parametern oder Zustandsvariablen führen.
- Die **Modellausführungseinheit (MAE)** (siehe Unterabschnitt 2.2.4) beeinflusst je nach verwendeter Zeitausführungsform sowohl die Laufzeit und den Speicherverbrauch als auch die Genauigkeit der Ergebnisse. Auf Modellebene ist die Auswahl der MAE von den verwendeten Entitäten und deren Schnittstellen (**gerichtet**/**ungerichtet**) abhängig. Wann immer es die für eine Anwendung erforderliche Genauigkeit zulässt, sollten ereignisgesteuerte Modelle mit hoher Abstraktion verwendet werden. Bei kontinuierlichen MAEs sollte eine möglichst hohe bzw. variable Schrittweite gewählt werden. Ferner sollte die Ordnung des verwendeten Lösers so niedrig wie möglich gewählt werden. Die Fehlertoleranz sollte hingegen so hoch wie möglich konfiguriert sein.
- Werden mehrere MAEs innerhalb einer **hybriden Simulation** verwendet, entstehen Netzwerke, denen unterschiedliche MAEs zugeordnet sein können. Diese Netzwerke benötigen Wandler zum Informationsaustausch. Die Anzahl solcher Netzwerke und Wandler sollte minimal sein, um die Laufzeit und den Speicherbedarf zu reduzieren.
- Die **zeitliche Auflösung** sowie die betrachtete **physikalische Zeit** beeinflussen die Laufzeit und den Speicherverbrauch. Das betrachtete Zeitintervall in der Simulation sollte deshalb so niedrig wie möglich gewählt werden.

#### Weitere Einflussfaktoren

Neben den zuvor beschriebenen modellabhängigen Eigenschaften gibt es weitere Faktoren, die eine Skalierbarkeit beeinflussen:

- Die zur Ausführung der Simulation verwendete **Hardware** beschränkt die Modellgröße durch den verfügbaren *Random Access Memory (RAM)*, während der Prozessor in Abhängigkeit der Taktrate und der Anzahl an Recheneinheiten die Laufzeit beeinflusst.

- Die **Implementierung des Simulators** beeinflusst sowohl die Laufzeit als auch den Speicherverbrauch. Dazu zählen die eingesetzte Programmiersprache, die Softwarearchitektur und die verwendeten Algorithmen zur Implementierung der MAEs. Auch ob das Modell kompiliert oder interpretiert ausgeführt wird, spielt hierbei eine Rolle.

### 4.1.3. Zusammenhang zwischen Anwendungsfällen und Skalierbarkeit

Abbildung 4.3 stellt den Zusammenhang zwischen Anwendungsfällen und der Performanz von E/E-Architektursimulationen dar.

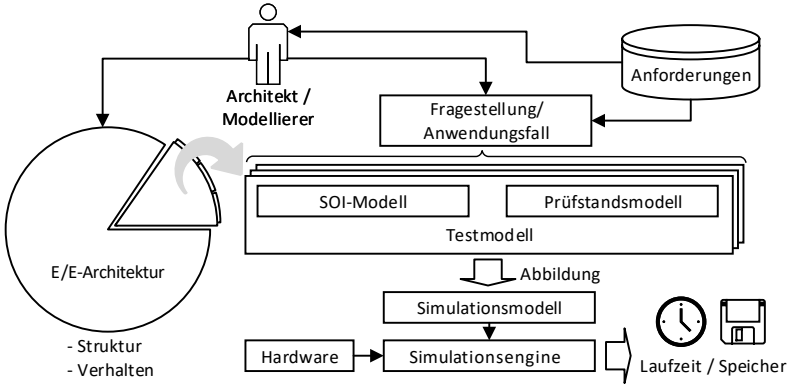



Abbildung 4.3.: Zusammenhang zwischen Anwendungsfall und Performanz.

Wie in Unterabschnitt 3.1.4 beschrieben, bilden **Anforderungen** die Grundlage des Designs von E/E-Systemen. E/E-**Architekten** setzten die Anforderungen durch Modellierung eines E/E-Architekturmodells um. Eine Simulation soll sie bei auftretenden Fragen und Entscheidungen unterstützen. Für die Entscheidungsfindung oder Beantwortung der Fragen bilden ebenfalls die Anforderungen die Grundlage. Aus einer oder mehrerer **Fragestellungen** lassen sich die **Anwendungsfälle** für eine E/E-Architektursimulation ableiten. Ein Anwendungsfall ist der Prototyp eines Testfalls oder mehrerer. Zur Ausführung eines Testfalls wird zum einen das **SOI** benötigt. Das ist der zur Beantwortung der Fragestellung relevante Teil des E/E-Architekturmodells. Dazu gehören die Artefakte mit ihren Attributen und ihrem Verhalten sowie die Struktur. Zum anderen wird ein **Prüfstandsmodell** benötigt, welches den Testablauf koordiniert und sowohl das SOI mit passenden Stimuli versorgt als auch Schnittstellen zur Auswertung bereitstellt. Beide Modelle bilden miteinander verknüpft das **Testmodell**, welches für die Ausführung in einem Simulator in ein **Simulationsmodell** abgebildet werden muss. Es wird ersichtlich, dass der Anwendungsfall maßgeblich das Testmodell und ebenso das Simulationsmodell sowie dessen Ausführungskonfiguration bestimmt. Dazu gehören die erforderlichen Entitäten, ihre Schnittstellen und Parameter sowie die Festlegung der Genauigkeit in Form des Abstraktionsgrades und der zeitlichen Auflösung. Dies stimmt mit dem in Abbildung 2.6 dargestellten Prozess überein, wonach eine Simulation immer für einen be-

stimmten Zweck bzw. eine Anwendung entwickelt wird. Des Weiteren beeinflusst die Erfahrung des Modellierers in der Verhaltensmodellierung die Performanz und den Nutzen der Simulation [311]. Zu den Aufgaben des Modellierers gehören nämlich die Auswahl und Verknüpfung der Entitäten, die Festlegung der Ausführungskonfiguration sowie die Validierung. Neben dem Anwendungsfall hängen Laufzeit und Speicherverbrauchbeeinflussen noch von der eingesetzten **Hardware** und der verwendeten **Simulationsengine** ab.

### 4.1.4. Resultierende wissenschaftliche Herausforderungen

Aus den vorausgegangenen Abschnitten resultieren die nachfolgend näher beschriebenen wissenschaftlichen Herausforderungen in Bezug auf die Skalierbarkeit von E/E-Architekturmodellsimulationen. Diese bilden die Grundlage der Forschungsfragen, die in dieser Arbeit beantwortet werden sollen.

#### Emergenz bei hybriden Simulationsmodellen

Bei modernen und zukünftigen E/E-Architekturen handelt es sich um mobile CPS. Da CPS mehrere Domänen mit logischen und physikalischen Anteilen kombinieren, entstehen hybride Simulationsmodelle. Bei der Simulation von PREEvision-E/E-Architekturen werden Kombinationen aus bis zu drei heterogenen Zeitausführungsmodellen benötigt:

1. **Zeitdiskret:** Logik und Software (Algorithmen).
2. **Zeitkontinuierlich:** Fahrdynamik, physikalische Aspekte, wie der Energiefluss zur Versorgung der Bordnetzkomponenten, oder Umgebungsaspekte.
3. **Wertdiskret:** Ausführung von in PREEvision modellierten Zustandsautomaten.

Eine Kombination der ersten beiden Zeitausführungsmodelle führt zu einem System mit variabler Struktur, das schwer zu implementieren und zu analysieren ist [312]. Des Weiteren muss der Datenfluss zwischen den unterschiedlichen Domänen und Komponenten in zeitgleich ablaufenden Prozessen synchronisiert sein, um Kausalität und Konsistenz zu gewährleisten [204]. Für die Modellierung hybrider Systeme gibt es jedoch keinen Standard, der den Datenfluss zwischen den einzelnen Domänen modelliert [79]. Insbesondere bei automatisch generierten Modellen kann dies bei einer Nichtbeachtung zu unvorhergesehenem Verhalten führen. Daher muss untersucht werden, inwiefern eine bei hybriden und automatisch generierten Modellen auftretende Emergenz den Einsatz einer E/E-Architektursimulation begrenzt.

#### Validierbarkeit

Aus den emergenten Eigenschaften folgt, dass sich hybride Modelle schwer validieren lassen. Es gibt nach [79] bisher keinen „inter-modularen“ Ansatz, bei dem hybride Modelle als Einheit evaluiert werden. Stattdessen werden „intra-modulare“ Ansätze verfolgt, bei denen die Teilmodelle nur isoliert validiert werden [79]. Aufgrund der Emergenz bei automatisch generierten Modellen ist dies jedoch nicht ausreichend. Dabei wirkt sich die Validierung entsprechend auf

die Wiedergabetreue bzw. Genauigkeit aus. Daher muss untersucht werden, inwiefern eine Validierung von E/E-Architekturmodellen möglich ist und welche Probleme dabei, u.a. aufgrund der Emergenz, auftreten.

#### Anwendbarkeit

Aufgrund der Größe und der Komplexität von E/E-Architekturen sowie fehlender Erfahrung der E/E-Architekten in der Modellierung von Simulationen [311] sind Methoden für eine erfolgreiche Anwendung der Simulation erforderlich. Ziel ist es eine zügige Modellierung von performanten und korrekten Testmodellen für Anwender ohne Expertenwissen zu ermöglichen. Außerdem sollen die Ergebnisse einfach verstanden werden können und bei der Entscheidungsfindung helfen. Es muss also untersucht werden, inwiefern die Anwendung einer E/E-Architektursimulation automatisiert werden kann und welche weiteren Methoden nötig sind, um diese Ziele zu erreichen.

#### Performanz

Zuvor wurden Charakteristiken mit Einfluss auf die Performanz einer E/E-Architektursimulation dargestellt. Die Performanz wird dabei maßgeblich vom jeweiligen Anwendungsfall der Simulation, der verwendeten Hardware und dem eingesetzten Simulator bestimmt. Die entstehenden Grenzen der Skalierbarkeit sind daher anhand von passend ausgewählten Anwendungsfällen zu untersuchen. Ferner wird ein Verfahren zur Leistungsbeurteilung von Simulatoren für E/E-Architektursimulationen benötigt.

## 4.2. Forschungsschwerpunkte

Auf Basis der Herausforderungen werden nun konkrete Forschungsfragen definiert. Anschließend werden dazu passende Einschränkungen festgelegt und das Vorgehen zur Beantwortung der Fragen innerhalb dieser Arbeit beschrieben.

### 4.2.1. Forschungsfragen

**Frage 1** Welcher Simulator ist für eine skalierbare E/E-Architektursimulation geeignet und welche Eigenschaften muss dieser aufweisen?

**Frage 2** Wie muss eine Abbildung gewählt werden und welche Methoden eignen sich für die Umsetzung einer skalierbaren E/E-Architektursimulation?

**Frage 3** Welche industriell relevanten Fragestellungen bzw. Anwendungsfälle für eine Simulation muss eine E/E-Architektursimulation abdecken?

**Frage 4** Wo liegen die Grenzen einer E/E-Architektursimulation in Bezug auf ihre Skalierbarkeit im industriellen Umfeld?

### 4.2.2. Annahmen und Anforderungen

Der Ansatz einer automatischen Modellgenerierung nach Bucher [43] soll verfolgt und unter Beachtung weiterer Randbedingungen ergänzt bzw. angepasst werden. Es lässt sich nämlich damit der Modellierungsaufwand reduzieren und die Anwendbarkeit erhöhen:

**Annahme 1** Ausgangspunkt und Referenz dieser Arbeit ist der bestehende Ansatz zur E/E-Architekturmodellgenerierng nach Bucher [43].

Da eine Abbildung spezifisch für den verwendeten Simulator ist und eine Implementierung für mehrere Simulatoren nicht das Ziel dieser Arbeit ist, wird dieser nach seiner Auswahl nicht mehr geändert:

**Annahme 2** Der verwendete Simulator bleibt konstant.

Da die Abbildung ebenso vom E/E-Architekturmodellierungswerkzeug abhängt, muss dieses ebenfalls als konstant angenommen werden. PREEvision, der de-facto Industrie-Standard, soll daher genutzt werden:

**Annahme 3** Zur E/E-Architekturmodellierung wird PREEvision eingesetzt.

Zu den modellunabhängigen Einflussparametern der Skalierbarkeit gehört die verwendete Hardware. Um die Freiheitsgrade der Untersuchung einzuschränken, soll die Hardware für die Untersuchungen in einem konstanten Leistungsbereich liegen:

**Annahme 4** Es wird ein Standardrechner verwendet, der die Mindestsystemanforderungen von PREEvision erfüllt. Die Untersuchung einer verteilten Ausführung der Simulation auf mehreren Rechnern ist nicht Gegenstand dieser Arbeit.

Außerdem werden folgende Anforderungen festgelegt:

**Anforderung 1** Der verwendete Simulator soll industriellen Anforderungen genügen.

**Anforderung 2** Die Simulation soll E/E-Architekten bei Designentscheidungen, beginnend in möglichst frühen Entwicklungsphasen, unterstützen.

**Anforderung 3** Bei der Modellbildung und Validierung soll beachtet werden, dass E/E-Architekten keine Simulationsexperten sind.

**Anforderung 4** Es sollen Methoden zur Steigerung der Performanz Beachtung finden.

## 4.3. Vorgehen und Methodik

Zur Beantwortung der Forschungsfragen sollen folgende Schritte unternommen werden, die auch den weiteren Aufbau dieser Arbeit definieren:

1. **Vergleich und Auswahl eines Simulators:** In [43] wird der Simulator Ptolemy II für eine integrierte Simulation von E/E-Architekturen verwendet. Der Simulator und die Implementierung sollen als Referenz dienen. Um die Eignung von Ptolemy II für den großmaßstäblichen Einsatz zu prüfen, sollen zunächst Alternativen gesucht werden, gegen die der Simulator verglichen werden kann. Bei deren Auswahl sollen Kriterien für den industriellen Einsatz (Anforderung 1) festgelegt werden und Hinweise auf die Skalierbarkeit aus der Literatur Beachtung finden. Sobald eine Alternative gefunden ist, sollen synthetische Benchmarks für beide Simulatoren konzipiert und ausgeführt werden. Anhand der Ergebnisse soll dann entschieden werden, ob Ptolemy II oder der alternative Simulator zur E/E-Architektursimulation in dieser Arbeit verwendet wird.
2. **Konzeption und Implementierung eines Prototyps:** Für den ausgewählten Simulator soll anhand des Stands der Technik und in Bezug auf die Anforderungen 1, 2, 3 und 4 ein Gesamtkonzept entworfen werden. Dies beinhaltet die Festlegung konkreter Abbildungsvorschriften von PREEvision zum Zielsimulator. Anschließend soll das Konzept prototypisch implementiert werden.
3. **Identifikation von Anwendungsfällen:** Da die Skalierbarkeit einer E/E-Architektursimulation von ihrem Anwendungsfall abhängt (vgl. Abbildung 4.3), werden relevante Anwendungsfälle für die Simulation als Grundlage für eine anschließende Evaluation ausgewählt.
4. **Evaluation:** Für die identifizierten Anwendungsfälle sollen konkrete E/E-Architektursimulationen durchgeführt werden. Hierbei sollen die Grenzen der Skalierbarkeit identifiziert und die eingesetzten Methoden bezüglich ihrer Eignung beurteilt werden. Es sollen insbesondere die Auswirkungen emergenter Modelleigenschaften sowie die Aspekte Performanz, Anwendbarkeit und Validierbarkeit berücksichtigt werden.

# 5. Vergleich und Auswahl eines Simulators

In diesem Kapitel geht es um die Auswahl eines Simulators für skalierbare E/E-Architektursimulationen, wobei der in [43] eingesetzte Simulator Ptolemy II als Referenz dient. Zunächst wird dazu eine Bewertungsmetrik mit passenden Auswahlkriterien entwickelt. Die Kriterien erhalten unterschiedliche Prioritäten und Gewichte. Der ausgewählte Simulator wird danach mit Ptolemy II anhand von synthetischen Benchmarks verglichen. Davon abhängig wird entschieden, welcher Simulator für die weiteren Betrachtungen in dieser Arbeit verwendet wird.

## 5.1. Vorauswahl mittels Online- und Literaturrecherche

Anhand einer Bewertungsmetrik und festgelegten Auswahlkriterien soll mithilfe einer Online- und Literaturrecherche ein alternativer Simulator zu Ptolemy II gefunden werden. Zunächst werden die Bewertungsmetrik und die Auswahlkriterien beschrieben. Danach erfolgt der Vergleich und die Auswahl anhand der zuvor festgelegten Kriterien.

### 5.1.1. Bewertungsmetrik

Für jedes Kriterium k werden bei den entsprechenden Simulatoren Punkte p vergeben, die mit einer Gewichtung g versehen sind. Am Ende werden die Punkte für alle gefundenen Kriterien mit der Gesamtanzahl n addiert und daraus der Mittelwert gebildet. Es ergibt sich damit für jeden $Simulator_i$ eine Maximalpunktezahl $P_{Simulator_i}$:

$$P_{Simulator_i} = \frac{1}{n} \sum_{k=1}^{n} p_k \cdot g_k \tag{5.1}$$

Der Punktebereich für jedes $p_k$ gliedert sich von -1 bis 1 Punkte:

- **-1 Punkt:** Das Kriterium ist nicht erfüllt.
- **0 Punkte:** Das Kriterium ist teilweise erfüllt.
- **1 Punkt:** Das Kriterium ist vollständig erfüllt.

In Unterabschnitt 5.1.2 sind die Kriterien und ihre jeweiligen Bedingungen zur Erfüllung beschrieben. Die Gewichtung für jedes Kriterium $g_k$ ist an die „Requirements Triage“ von Davis [62] angelehnt und teilt sich folgendermaßen auf:

- **g = 1:** Das Kriterium kann erfüllt sein.

- **g = 2:** Das Kriterium soll erfüllt sein.
- **g = 3:** Das Kriterium muss erfüllt sein.

Der $Simulator_i$ mit dem höchsten $P_{Simulator_i}$-Wert erfüllt die festgelegten Kriterien am besten.

### 5.1.2. Auswahlkriterien und Gewichtung

Die Auswahlkriterien werden im Folgenden nach Priorität bzw. Gewichtung geordnet beschrieben. Die Gewichtung steht in Klammern nach dem Namen des Kriteriums. Aufgrund von Anforderung 1 werden durch die Auswahlkriterien auch die Anforderungen bei einem industriellen Einsatz berücksichtigt. Die Bedingungen zur Erfüllung der Kriterien wurden u.a. anhand der Recherche zu den einzelnen Simulatoren bestimmt. Erst hierdurch konnten Referenzwerte für die Marktbeständigkeit oder die Anzahl der Mitarbeiter ermittelt werden.

- **Hybride Simulation (3):** Für CPS, wie E/E-Architekturen, müssen sowohl Stoff-, Energie- und Informationsfluss modellierbar und integriert ausführbar sein. Je nach Abstraktion setzt dies zeitkontinuierliche und zeitdiskrete MAEs voraus. Wegen Annahme 3 und der Möglichkeit in PREEvision Verhalten mit Zustandsdiagrammen zu modellieren, sollte auch deren Ausführung unterstützt werden. Das Kriterium ist erfüllt, wenn die gleichzeitige Ausführung von zeitkontinuierlichen, zeitdiskreten und wertdiskreten Modellen in Form von Zustandsautomaten möglich ist. Können Zustandsautomaten nicht ausgeführt werden, aber die beiden anderen Arten gleichzeitig, so handelt es sich um eine teilweise Erfüllung. Kann zu einem Zeitpunkt nur eine Zeitform ausgeführt werden, so ist das Kriterium nicht erfüllt.
- **Schnittstelle zu Java (3):** Da PREEvision (Annahme 3) auf der Eclipse *Rich Client Platform (RCP)* aufbaut und in Java implementiert ist, wird eine geeignete Schnittstelle bzw. API benötigt. Falls der Simulator quelloffen ist, kann diese selbst programmiert werden. Das Kriterium ist erfüllt, wenn es eine API gibt oder der Simulator in Java programmiert und quelloffen ist. Das Kriterium ist teilweise erfüllt, wenn der Simulator quelloffen, aber nicht in Java programmiert ist. Sonst ist das Kriterium nicht erfüllt.
- **Performanz (3):** Mit diesem Kriterium soll Anforderung 4 adressiert werden. Bei der Auswahl des Simulators sollen bereits durchgeführte Performanzuntersuchungen berücksichtigt werden. Auch der Einsatz des Simulators in großmaßstäblichen Referenzprojekten soll als Indikator für eine hohe Performanz herangezogen werden. Die Fähigkeit zu einer parallelen Ausführung der Simulation zählt ebenso dazu. Das Kriterium ist erfüllt, wenn es Belege gibt, die dem Simulator im direkten Vergleich mit anderen eine verhältnismäßig gute Performanz bescheinigen. Das Kriterium ist teilweise erfüllt, wenn es keine Hinweise zur Performanz gibt. Das Kriterium ist nicht erfüllt, wenn es Belege gibt, die dem Simulator im direkten Vergleich mit anderen eine verhältnismäßig schlechte Performanz bescheinigen.
- **Lizenzierung (3):** Je nach Lizenzmodell, kann eine Offenlegung des eigenen Quellcodes erforderlich sein, was bei einer Integration in kommerzielle Werkzeuge nicht möglich ist. Ferner sollen die Kosten Berücksichtigung finden. Das Kriterium ist erfüllt, wenn kein Quellcode offengelegt werden muss und der Simulator kostenlos ist. Das Kriterium ist teilweise erfüllt, wenn kein eigener Quellcode offengelegt werden muss, aber der Simulator kostenpflichtig ist. Das Kriterium ist nicht erfüllt, wenn der Quellcode offengelegt werden muss.

- **Wartbarkeit (2):** Sofern eine API existiert oder der Zugriff auf die Code-Basis des Simulators besteht, muss eine Wartung mit geringerem Aufwand möglich sein. Das Kriterium ist erfüllt, wenn die API standardisiert ist oder Kundendienst und Dokumentation vorhanden sind. Liegt der Quellcode offen, so müssen zusätzlich noch Tests vorhanden und Modularität erkennbar sein. Ist die API standardisiert und es gibt eine Dokumentation, aber keinen Kundendienst, dann ist das Kriterium nur teilweise erfüllt. Ansonsten ist das Kriterium nicht erfüllt.
- **Bibliothek (2):** Die Bibliothek des Simulators sollte die zur Modellierung von CPS benötigte Komponenten beinhalten. Dazu zählen elektrische und elektro-mechanische Bauelemente, wie Leitungen oder Motoren, sowie Komponenten zum Reglerentwurf. Das Kriterium ist erfüllt, wenn alle zur Modellierung von CPS benötigten Komponenten vorhanden sind. Sonst ist das Kriterium nicht erfüllt.
- **Erweiterbarkeit (2):** Die Bibliothek des Simulators soll durch das Hinzufügen von Paketen oder mittels Implementierung eigener Entitäten erweiterbar sein. Das Kriterium ist erfüllt, wenn es Erweiterungspakete und die Möglichkeit zum Implementieren von Entitäten gibt. Das Kriterium ist teilweise erfüllt, wenn es Erweiterungspakete oder die Möglichkeit zum Implementieren von Entitäten gibt. Sonst ist das Kriterium nicht erfüllt.
- **Robustheit (2):** Im industriellen Einsatz ist die Robustheit bzw. Zuverlässigkeit der Simulation wichtig. Häufige Abstürze oder Fehlermeldungen sowie eine gering getestete und fehleranfällige Code-Basis (bei quelloffener Software) sind ein Zeichen für eine geringe Robustheit. Die Robustheit kann sich ferner in der Anzahl der Anwender bzw. der Referenzen, der Anzahl der beteiligten Entwickler oder der Marktbeständigkeit widerspiegeln. Der gerundete Mittelwert aus der Bewertung der gesondert aufgeführten Punkte Marktbeständigkeit, Referenzen und Größe ergibt hier zusätzliche Punkte.
- **Co-Simulation (2):** Der Simulator sollte die Fähigkeit zur Co-Simulation aufweisen. Beispielsweise durch die Unterstützung von Standards wie FMI und HLA oder durch individuelle Lösungen. Dies ist ein weiterer Aspekt der Erweiterbarkeit und ebenso eine Möglichkeit die Performanz zu erhöhen. Ist die Durchführung einer Co-Simulation möglich, so ist das Kriterium erfüllt. Ansonsten nicht.
- **Aktualität (2):** Das Erscheinen von regelmäßigen Updates garantiert ein Fortbestehen auf dem Markt und eine robuste Software, wenn dadurch entsprechende Fehler behoben werden. Ein Fortbestehen auf dem Markt kann wiederum ein Fortbestehen der technischen Unterstützung bei Problemen garantieren, sofern es diese gibt. Das Kriterium ist erfüllt, wenn die zuletzt veröffentlichte Version nicht älter als ein Jahr ist. Das Kriterium ist teilweise erfüllt, wenn die letzte Version älter als ein Jahr ist, aber eine neue Version angekündigt ist. Das Kriterium ist nicht erfüllt, falls die letzte Version älter als ein Jahr ist und auch die Website seither nicht aktualisiert wurde.
- **Unterstützung (2):** Die Form der technischen Unterstützung kann ein Hinweis auf deren Qualität sowie die Schnelligkeit bei der Lösung eines Problems sein. Letztgenanntes beeinflusst wiederum die in einem Unternehmen entstehenden Kosten, falls es durch das Problem zur Blockierung in der Entwicklung kommt. Das Kriterium ist erfüllt, wenn es eine telefonische Unterstützung gibt. Das Kriterium ist teilweise erfüllt, wenn es andere Formen

der technischen Unterstützung gibt. Falls es keinen technischen Kundendienst gibt, ist das Kriterium nicht erfüllt.

- **Code-Generierung (1):** Auf einem vHIL-Prüfstand wird kompilierte Software auf einer emulierten Hardware ausgeführt. Dies kann gerade bei der Bestimmung von Signallaufzeiten genauere Ergebnisse liefern, als eine Simulation. Da ECU-Software zudem meist aus Simulationsmodellen generiert wird, bietet sich die Verwendung bereits bestehender Simulationsmodelle dafür an. Das Kriterium ist erfüllt, wenn es einen funktionierenden Code-Generator für Steuergerätesoftware gibt, sonst nicht.
- **Marktbeständigkeit (1):** Die Zeit, die der Simulator auf dem Markt verfügbar ist, kann ein Hinweis auf dessen Qualität und seine künftige Weiterentwicklung sein. Ist der Simulator länger als 15 Jahre verfügbar, so ist das Kriterium erfüllt. Sind es fünf bis 15 Jahre, dann ist das Kriterium teilweise erfüllt. Sind es weniger als 5 Jahre, so ist das Kriterium nicht erfüllt.
- **Referenzen (1):** Referenzen geben einen Hinweis auf die Verbreitung und Anwendung des Simulators. Eine hohe Anzahl an Referenzen bzw. Anwendern kann ein Hinweis auf eine hohe Robustheit und eine Garantie für das Fortbestehen am Markt sein. Zu den Referenzen zählen neben Firmenreferenzen auch die Anzahl veröffentlichter wissenschaftlicher Arbeiten auf *Scopus* [82] sowie die Anzahl der Follower auf der Plattform *LinkedIn* [164]. Der Industriezweig der Nutzer bzw. Follower kann insbesondere auch auf eine gute Skalierbarkeit hinweisen. Mehr als fünf Firmenreferenzen oder 10000 Anwender entsprechen einer Erfüllung des Kriteriums. Mehr als eine Firmenreferenz oder 2000 Anwender entsprechen einer teilweisen Erfüllung. Bei keinen Firmenreferenzen oder weniger als 2000 Anwendern ist das Kriterium nicht erfüllt.
- **Größe (1):** Die Anzahl der Entwickler oder der Beschäftigten einer Entwicklerfirma kann ein Indikator für die Softwarequalität oder für ein zukünftiges Fortbestehen des Simulators am Markt sein. Ab mehr als 100 Mitarbeitern oder Entwicklern ist das Kriterium erfüllt. Zwischen 50 und 100 Mitarbeitern oder Entwicklern ist das Kriterium teilweise erfüllt. Unter 50 Mitarbeitern oder Entwicklern ist das Kriterium nicht erfüllt.

### 5.1.3. Vergleich und Auswahl für eine weitere Analyse mit Benchmarks

Im Zuge dieser Arbeit wurden 94 Simulatoren und *Co-Simulations-Frameworks (CSFs)* betrachtet (siehe Abschnitt A.1). Davon sind 36 proprietär und 58 quelloffen. Nach der Vorauswahl wurden CSFs zur Realisierung von hybriden Simulationen ausgeschlossen, da diese das Kriterium Wartbarkeit nur schwer erfüllen können. Stattdessen soll eine hybride Simulation mit zumindest zeitdiskretem und zeitkontinuierlichem Anteil in einem Modell integriert möglich sein, da ein CPS simuliert werden soll. Außerdem werden Modelica-basierte Simulatoren wie *SimulationX*, *Dymola* oder *MapleSim* nicht betrachtet, da die gleichungsbasierte Sprache Modelica [184] primär für physikalische Systeme ausgelegt ist. Es gibt nach [230] zwar Zusatzbibliotheken für eine zeitdiskrete oder hybride Simulation (*DevSLib*, *SIMANLib* und *ARENALib*), aber diese verfolgen entweder einen prozessorientierten Ansatz oder einen parallelen DEVS-Formalismus. Durch die Integration in ein gleichungsbasiertes Simulations-Framework bleibt die Zeitachse immer kontinuierlich und nur die Wertachse wird diskretisiert [230, 48]. Daraus folgt, dass ein Speedup, der bei einer zeitdiskreten Ausführung durch das Überspringen von

Zeitschritten entsteht, nicht genutzt werden kann. Des Weiteren sollen reine DEVS Simulatoren, wie *CoSMOs* (vgl. [10]), bei der Auswahl nicht beachtet werden, da sie meist weniger genaue QSS-Verfahren (vgl. Unterabschnitt 2.2.4) für die kontinuierliche Simulation nutzen.

Bei der Vorauswahl ist aufgefallen, dass die meisten der quelloffenen Simulatoren für einen bestimmten Anwendungsfall innerhalb von wissenschaftlichen Projekten entwickelt wurden und damit i.d.R. nur eine MAE implementieren. Außerdem ist deren Bibliothek auf den Anwendungsfall beschränkt. Am Ende sind 9 Simulatoren übrig geblieben, welche die Vorbedingungen erfüllen. Diese wurden anhand der Kriterien und der Bewertungsmetrik aus den vorhergehenden Abschnitten analysiert. Das Ergebnis ist in Tabelle 5.1 dargestellt. An den leeren Stellen konnten keine ausreichenden Informationen für die Bewertung gefunden werden. Dementsprechend verkürzt sich bei diesen Kandidaten die Anzahl n an gefundenen Kriterien.

Tabelle 5.1.: Vergleich von Simulatoren.

| | Hybride Simulation | Schnittstelle zu Java | Performanz | Lizenzierung | Wartbarkeit | Bibliothek | Erweiterbarkeit | Robustheit | Co-Simulation | Aktualität | Unterstützung | Code-Generierung | Marktbeständigkeit | Referenzen | Größe | Punkte (gewichtet) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Ptolemy II [280]** | 1 | 1 | -1 | 1 | 0 | 0 | 1 | 0 | 1 | -1 | 0 | 0 | 1 | -1 | | **0,36** |
| **Triquetrum [76]** | 1 | 1 | -1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | -1 | -1 | | **0,5** |
| **Xcos/Scilab [72]** | 1 | 1 | -1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **1,4** |
| **Simulink**[1] [265] | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **1,67** |
| **20-Sim [56]** | 0 | 0 | | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | **0,86** |
| **Anylogic [258]** | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | **1,47** |
| **ExtendSim [125]** | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | **0,93** |
| **VisualSim [182]** | 1 | 1 | | 0 | 1 | 1 | 0 | 1 | 0 | | 1 | 0 | 1 | 0 | 0 | **1,15** |
| **MLDesigner [168]** | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | | 1 | 1 | 1 | | 0 | **1,46** |

Aus Tabelle 5.1 geht hervor, dass der in [43] für eine integrierte Simulation in PREEvision verwendete Simulator Ptolemy II die Kriterien für einen industriellen und skalierten Einsatz, im Vergleich zu den übrigen Simulatoren, nur schlecht erfüllt. Daher werden zunächst die Eigenschaften und Einschränkungen von Ptolemy II bzw. dem direkten Ableger Triquetrum beschrieben, die zu der vergleichsweise niedrigen Punktzahl geführt haben. Im Anschluss werden die Hauptmerkmale und Nachteile der Alternativen beschrieben, die dazu führen, dass Simulink[1] als Simulator gewählt wird.

[1] Inklusive SimEvents, Stateflow und Simscape.

### Ptolemy II und Triquetrum

Der quelloffene Simulator Ptolemy II [280] ist innerhalb des „Ptolemy Projects“ [281] an der Universität Berkeley, zwischen September 2002 und Juni 2018, entwickelt und stetig erweitert worden. Der Simulator ist in Java geschrieben und eine Neuentwicklung des zuvor, ebenfalls an der Universität Berkeley, in C++ geschriebenen Simulators *Ptolemy Classic* [279]. Auf diesem basieren die proprietären Simulatoren *Keysight Ptolemy* [145] und *MLDesigner* [168]. Keysight Ptolemy wird aufgrund seiner Fokussierung auf Mixed-Signal Anwendungen mit entsprechend eingeschränkter Bibliothek nicht weiter untersucht. *MLDesigner* ist Teil des Vergleichs und wird im Folgenden noch näher beschrieben. Es existiert auch eine kommerzielle Version von Ptolemy II mit dem Namen *VisualSim*. Sie wird im Zuge des Vergleichs ebenfalls näher betrachtet.

Ptolemy II unterstützt alle erforderlichen MAEs und ermöglicht deren zeitgleiche Ausführung durch hierarchische Kapselung und geeignete Wandlung (siehe Unterabschnitt 2.4.2).

Eine quelloffene Implementierung von Ptolemy II in Java ermöglicht die Implementierung einer eigenen Schnittstelle zur Kommunikation mit PREEvision und eine gute Erweiterbarkeit. Die verfügbaren Bibliotheken decken zwar ein großes Spektrum ab, aber elektrische Bauelemente fehlen. In [43] wurde hierzu ergänzend der Schaltkreissimulator *JSpice* [101] integriert. Allerdings ist dieser auf Gleichstromanalysen begrenzt, bietet keine grafische Modellierungsmöglichkeit und muss co-simuliert werden.

Eine zukünftige Erweiterung von Ptolemy II dagegen ist unwahrscheinlich, da das eigentliche Projekt beendet ist und in das Eclipse-RCP-Projekt *Triquetrum* [76] überführt wurde. Hierbei soll allerdings eine „*offene Plattform für wissenschaftliche Arbeitsabläufe*“ [76] geschaffen werden. Aufgrund von Lizenzrechten ist die ursprüngliche Bibliothek von Ptolemy II jedoch nicht standardmäßig im Projekt eingebunden. Nutzer müssen diese selbst lokal einbinden. Erweiterungen im E/E-Bereich zur physikalischen oder elektrischen Modellierung sind nicht zu erwarten.

Zur Beurteilung der Wartbarkeit und Robustheit von Ptolemy II wurde dessen Code analysiert, da die Wartung in Form einer Dienstleistung ausgeschlossen ist und es Unterstützung nur in Form einer Mailing-Liste sowie einer ausführlichen Dokumentation gibt. Von 5000 Java Dateien sind 520 als Test gekennzeichnet, also etwa 11,6 % des Umfangs. Für die Superklassen *Director* (MAE) und *Actor* (Entität) gibt es zwar Tests, jedoch i.d.R. nicht für deren Spezialisierungen, also den eigentlichen Bibliothekselementen. In den Kommentaren ist ferner ein Ampel-System für Reviews zu finden, welches den Reifegrad und damit die Freigabe der entsprechenden Code-Teile bestimmt. Des Weiteren wurde im Zuge dieser Arbeit die Programmstruktur mit dem Tool *iPlasma* [170] untersucht. Hierbei sind vor allem große Hierarchien in feinen Strukturen, also wenige Klassen pro Paket, und lange Methoden mit vielen Abhängigkeiten, aber einfacher Logik, aufgefallen (siehe Abbildung A.1 im Anhang). Zudem weist Ptolemy II eine Fehlerdichte von 2,31 in statischen Analysen mit dem Werkzeug *Coverity Scan* [250] auf. Die Struktur des Kernels ist gut dokumentiert [215]. Im Vergleich dazu sind bei Triquetrum 28 von 611 Java-Klassen als Test gekennzeichnet, was ca. 5 % entspricht. Da aber die für eine E/E-Architektursimulation relevanten Klassen manuell aus dem Ptolemy-II-Projekt übernommen werden müssten, können die Ergebnisse bzgl. der Wartbarkeit und Robustheit für Triquetrum von Ptolemy II übernommen werden. Bezüglich des Marktbestehens und der Aktualität unterscheiden sich Ptolemy II und Triquetrum hingegen, da Triquetrum Ptolemy II ursprünglich

fortführen sollte. Zu der Anzahl der Entwickler oder Unternehmensgröße ist keine Angabe möglich, da es sich in beiden Fällen um wissenschaftliche Projekte handelt. Als Referenzen wurden für Ptolemy II sechs Firmen und 104 wissenschaftliche Veröffentlichungen auf Scopus [82] gefunden.

Eine vorhandene C-Code-Generierung von Ptolemy II ist nach [247] sehr eingeschränkt, da bisher nur 23 Basis-Entitäten unterstützt werden. Globale Variablen oder Typenkonversionen werden nicht unterstützt. Außerdem werden nicht alle nötigen MAEs unterstützt. Eine Erweiterung ist hierbei zwar möglich, aber aufwendig. [247]

Zur Performanz von Ptolemy II wurden die zwei folgenden Analysen gefunden:

Brito zeigt in [39], dass durch die parallele Ausführung eines Simulationsmodells mittels HLA auf 8 Kernen eine Beschleunigung der Ausführungszeit (Speedup) um den Faktor 4 möglich ist. Dies setzt voraus, dass die Struktur des Simulationsmodells eine Parallelisierung zulässt. Eine weitere Erhöhung der Ressourcen führt aufgrund der zunehmenden Kosten der Synchronisierung zu einem Rückgang der Beschleunigung (siehe Unterabschnitt 2.3.2). Als Lösung wird eine Optimierung des Lookaheads vorgeschlagen [39] (siehe Unterabschnitt 2.2.5). Allerdings folgt aus einem Speedup von 4 und einer Erhöhung der Ressourcen um den Faktor 8 bei gleichbleibender Problemgröße, mit Gleichung 2.6, ein Scaleup von 0,5. Das lässt wegen $0{,}5 < 1$ auf einen sub-linearen Scaleup schließen (vgl. Unterabschnitt 2.3.1). Um dies sicher zu beantworten, muss man jedoch die genaue Modellstruktur mit ihren parallelen und seriellen Anteilen kennen (vgl. Gleichung 2.12).

Baumann untersucht in [19] allgemein die Performanz von Simulatoren am Beispiel des deutschen Toll-Systems zur Mauterhebung von Lastkraftwagen. Hierbei werden verschiedene Benchmarks mit einer Auswahl an Simulatoren in der zeitdiskreten Domäne ausgeführt. Abbildung 5.1 zeigt einen Kernel-Benchmark der Ereignisliste.

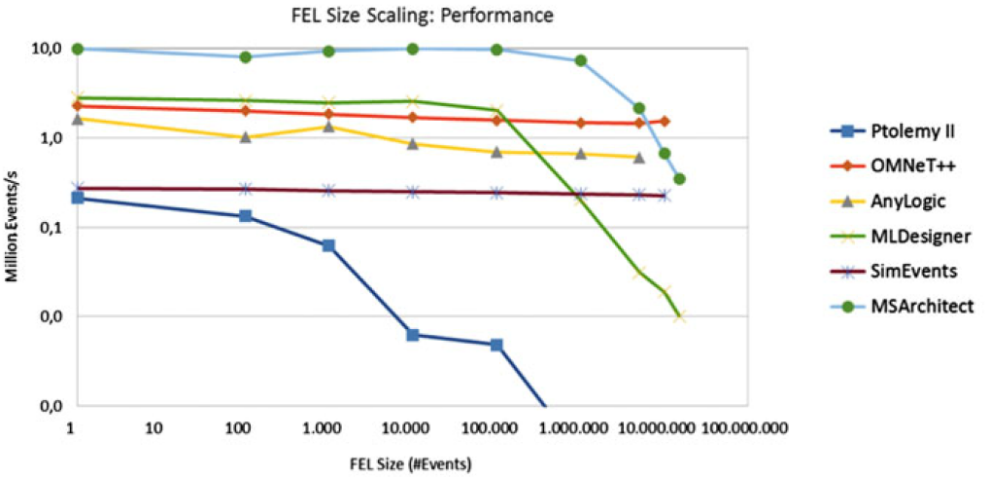



Abbildung 5.1.: Skalierung der Ereignislistengröße (Quelle: [19]).

Das Konzept dieses Benchmarks ist in Unterabschnitt 5.2.1 beschrieben. Die Performanz von Ptolemy II ist im Vergleich zu den Alternativen bei der Laufzeit deutlich geringer. Der Speicherverbrauch ist dagegen höher. Sobald hier 1000000 Ereignisse in der Liste sind, bricht die Performanz erheblich ein.

Insgesamt lässt sich aus [39] und [19] ableiten, dass Ptolemy II nicht für großmaßstäbliche Modelle konzipiert ist. Allerdings kann die Eignung für eine skalierbare E/E-Architektursimulation nicht ausgeschlossen werden, da die Skalierbarkeit neben der Performanz noch die Aspekte Anwendbarkeit und Validierbarkeit aufweist. Außerdem unterscheiden sich die Anwendungsfälle einer E/E-Architektursimulation zu denen in [39] und [19]. Daher folgen in Abschnitt 5.2 noch genauere Performanzanalysen, die im Gegensatz zu [19] auch zeitkontinuierliche und wertdiskrete MAEs einschließen.

In den folgenden Abschnitten werden die Besonderheiten und Schwachstellen der weiteren Vergleichskandidaten zum Zeitpunkt der Analyse im Jahr 2021 dargestellt.

#### 20-Sim

*20-Sim* [56] ist Teil der *Into-CPS Toolchain* [122], die speziell für die Modellierung und Simulation von CPS ausgelegt ist. Jedoch unterstützt 20-Sim keine Simulation von Zustandsautomaten. Eine API zum Skripting, also zur Fernsteuerung durch andere Tools, steht nur für Python, Matlab oder Octave zur Verfügung. Sie lässt sich nur über Umwege in Java nutzen. In Bezug auf die Wartbarkeit gibt es keine Wartungsoptionen. Insbesondere fehlen Schulungen oder eine Unterstützung bei Modellentwicklungen.

#### Anylogic

*Anylogic* [257] hat sich als drittbeste Alternative herausgestellt. Nach [34] ist er der einzige proprietäre Simulator, der hybride Modelle vollständig integriert ausführen kann. Bei der Performanz sind außerdem solide Werte zu erwarten (vgl. Abbildung 5.1). Des Weiteren bietet Anylogic die Möglichkeit zur Ausführung von Simulationen in einer eigenen Cloud. Modelle und Ergebnisse können ferner dort für eine einfachere Zusammenarbeit gespeichert werden. Ein Speedup ist bei mehrfacher, zeitgleicher Ausführung eines Modells mit unterschiedlichen Parametern zu erwarten [257]. Dies deckt sich mit den Informationen aus den Systemanforderungen. Hier wird eine erwartbare Leistungszunahme durch mehr Kerne bei Fußgängersimulationen beschrieben. Schwächen sind vor allem bei der Bibliothek zu finden, wobei elektrische Bauelemente fehlen. Eine C-Code-Generierung gibt es ebenfalls nicht.

#### ExtendSim

Eine Besonderheit von *ExtendSim* [125] ist die Fähigkeit zur Simulation hybrider „Discrete Rate“-Modelle [123, 59]. Dies entspricht einer ereignisorientierten kontinuierlichen Simulation, wobei nur Zeitschritte ausgeführt werden, bei denen sich der Zustand ändert. ExtendSim ist jedoch auf die Simulation von Prozessen spezialisiert. Eine API gibt es ferner nur für di-

verse Datenbanken, wie *Microsoft Excel*. Des Weiteren lässt sich aus den Systemanforderungen ablesen, dass es bei der Performanz der Ausführung eines einzelnen Simulationsmodells auf die Taktrate des Prozessors und nicht auf die Anzahl der Kerne ankommt [124]. Das spricht gegen eine gute Skalierbarkeit (Leistungssteigerung durch parallele Ausführung).

### VisualSim

*VisualSim* [182] ist eine kommerzialisierte Version von Ptolemy II. Daher ist es möglich auch in Ptolemy II erstellte Modelle im *Modeling Markup Language (MoML)*-Format (ein XML-Derivat) in VisualSim zu verwenden. Die in [43] entwickelte Synthese könnte also hier verwendet und angepasst werden. Über die Performanz lässt sich hingegen wenig herausfinden. Systemanforderungen stehen auf der Website nicht zur Verfügung und der Lizenzvertrag für Universitäten [181] verbietet explizit die Veröffentlichung von Benchmarks. Durch Kontaktaufnahme mit der Firma konnte zwar in Erfahrung gebracht werden, dass es Verbesserungen der Performanz gegenüber Ptolemy II gibt, wie umfangreich diese sind, jedoch nicht. Des Weiteren existieren online keine Informationen zur Versionierung und dem Datum der letzten Veröffentlichung.

### MLDesigner

*MLDesigner* [168] basiert auf der ersten Ptolemy Version *Ptolemy Classic* und ist im Gegensatz zu Ptolemy II in C++ geschrieben. Ein Skripting via API ist mit einer speziellen *Ptolemy Tool Command Language* möglich, für die es eine Java-Implementierung gibt. Das Dateiformat lautet *Model Markup Language (MML)*. Ein Öffnen von Ptolemy-II-MoML-Dateien ist nicht möglich. Nach [19] (siehe auch Abbildung 5.1) ist eine vergleichsweise gute Performanz zu erwarten. In der Bibliothek fehlen allerdings Elemente zur Modellierung elektrischer Schaltkreise. Außerdem wurde im Vergleich zu den anderen Simulatoren eine geringe Verbreitung festgestellt, da keine Referenzen und nur verhältnismäßig wenige wissenschaftliche Veröffentlichen bei Scopus [82] gefunden wurden.

### Scilab/Xcos

*Scilab/Xcos* [72] ist eine quelloffene und kostenlose Alternative zu Matlab/Simulink. Neben Simulink ist es zudem das einzige Werkzeug, welches sowohl den Import als auch Export von *Functional Mockup Units (FMUs)* [183] unterstützt. Scilab weist außerdem eine Fehlerdichte von 0,84 im Coverty Scan [251] auf, welche deutlich geringer ist, als die von Ptolemy II (2,31). Zur Performanzoptimierung werden darüber hinaus mehrere Möglichkeiten angeboten. Für parallele Berechnungen wird der MPI-Standard [169] und die Nutzung einer *parallelen virtuellen Maschine (PVM)* unterstützt. Außerdem steht eine auf *maschinellem Lernen (ML)* basierende Toolbox zur Modellreduktion nach der *Finite-Elemente-Methode (FEM)* zur Verfügung [71]. Dennoch ist Matlab bei verschiedenen Benchmarks deutlich performanter, wie Tabelle 5.2 zeigt. Betrachtet man ergänzend die Standardabweichung der Ergebnisse aus [6], so liefert Scilab auch ungenauere bzw. weniger stabile Ergebnisse als Matlab [6].

Tabelle 5.2.: Absoluter Laufzeitvergleich von Matlab, Scilab und Octave (Quelle: [18]).

| Benchmark Set | Matlab (MKL) [s] | Scilab (Atlas) [s] | Octave (OpenBLAS) [s] |
|---|---|---|---|
| pincon | 24,87 | 27,17 | 35,66 |
| poisson | 1,59 | 191,92 | 0,91 |
| nerunch | 3,64 | 19,49 | 14,98 |
| optim | 22,33 | 27,65 | 72 |
| **Total** | **52,43** | **266,23** | **123,55** |

#### Simulink

Im Vergleich zu den Alternativen hat sich Simulink [265], in Verbindung mit den Toolboxen Stateflow, SimEvents und Simscape, als besonders geeignet herausgestellt. Gegenüber den anderen Simulatoren weist Simulink folgende Besonderheiten auf:

Einerseits bietet Simulink die beste Erweiterbarkeit aufgrund der Anzahl an offiziellen Toolboxen und Blocksets (4257). Andererseits ist Simulink deutlich weiter verbreitet als Anylogic, der zweitbeste Simulator bei diesem Kriterium. Hier stehen 44184 wissenschaftliche Veröffentlichungen 555 gegenüber. Bei den Firmenreferenzen ist das Verhältnis 5687 zu 326. Insbesondere wurden für Simulink-Referenzen zur Simulation komplexer Systeme gefunden [259]. Des Weiteren führen umfangreiche Debug-Funktionen und Modelloptimierungsmöglichkeiten, wie regelbasierte Modellprüfungen oder eine C-Code-Kompilierung zur Laufzeitoptimierung, zur Auswahl des Marktführers Simulink [223] für einen Vergleich mit Ptolemy II.

## 5.2. Synthetische Benchmarks

In der Literatur wurden keine direkten Vergleiche zwischen Ptolemy II und Simulink anhand von Benchmarks gefunden. In diesem Abschnitt werden daher zunächst Konzepte unterschiedlicher synthetischer Benchmarks beschrieben, die teilweise in dieser Arbeit entstanden sind und teilweise der Literaturrecherche entstammen. Im Anschluss findet ein Vergleich von Ptolemy II und Simulink anhand von auf den Konzepten basierenden Benchmarks statt.

### 5.2.1. Konzeption

Wie in Unterabschnitt 4.1.2 beschrieben, spielt die Zeitausführungsform bei der Performanz von Simulationen eine Rolle. Die für CPS- und E/E-Architektursimulationen zu untersuchenden Formen sind zeitkontinuierlich und zeitdiskret. Aufgrund von Annahme 3 gehört auch die wertdiskrete Simulation von Zustandsautomaten dazu. Zudem führen Mischformen zu hybriden Simulationen. Die Verknüpfung der MAEs kann hierbei ebenfalls die Performanz beeinflussen.

Zur Analyse der MAEs sollen vorwiegend skalierbare, d.h. größenabhängig generierbare, synthetische Benchmarks verwendet werden. So kann in Abhängigkeit der Problem- bzw. Modellgröße und ohne expliziten Anwendungsfall die Performanz der MAEs beurteilt werden.

Neben der Modellgröße sollen auch die Simulationszeit und, falls zutreffend, die Anzahl der Hierarchiestufen variabel sein. Sofern möglich, soll auch die Auswirkung einer Berechnung von parallelen Pfaden auf die Performanz beurteilt werden. Für die MAEs sollen die Standardkonfigurationen der Simulatoren genutzt werden. Die Auswirkung von unterschiedlichen Entitäten auf die Laufzeit soll in anschließenden Anwendungsbenchmarks mit dem entsprechenden Zielsimulator untersucht werden. Bei der Implementierung sollen Skripte zur Erzeugung von unterschiedlich aufwendigen Benchmark-Modellen verwendet werden.

Im Folgenden werden die Konzepte der Benchmarks, nach MAE getrennt, aufgeführt.

#### Zeitkontinuierlich

Bei einer Literaturrecherche wurde deutlich, dass die gefundenen Benchmark-Modelle in der Regel für Anwendungsbenchmarks konzipiert sind [242, 103, 36]. Konzepte zu synthetischen, skalierbaren Benchmarks wurden für diese Zeitausführungsform nicht gefunden. Da die entsprechenden MAEs allerdings, wie in Unterabschnitt 2.2.4 beschrieben, auf dem Lösen von Gleichungssystemen basieren, lassen sich zusammen mit den Modellierungsrichtlinien für Simulink aus [269] und [263] geeignete, synthetische Benchmark-Modelle herleiten. Zu Ptolemy II oder VisualSim konnten keine vom Entwickler vorgegebenen Modellierungsrichtlinien gefunden werden. Die Beschreibung der Modellbildung einer Differenzialgleichung durch Integration und Rückführung in [215] (vgl. Abbildung 5.2) deckt sich aber damit, dass Integrator-Blöcke ($\int$) und Rückführungen einen maßgeblichen Einfluss auf die Performanz haben [269]. **f(x)** ist hierbei eine beliebige Funktion.

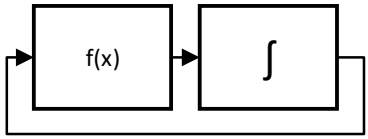



Abbildung 5.2.: Prinzip der Modellierung einer Differenzialgleichung (Quelle: eigene Darstellung nach [215]).

Die mathematische Beschreibung davon lautet nach [215]:

$$f(x) = x_0 + \int_{t_0}^{t} \dot{f}(\tau)\, d\tau \tag{5.2}$$

Da der Ausgang des Integrator-Blocks auf seinen Eingang zurückgeleitet wird, kann Gleichung 5.2 nach [215] auch als Differenzial ausgedrückt werden:

$$\dot{f}(x) = \frac{d}{dt} f(x) \tag{5.3}$$

**Parallele Sinusgeneratoren** Integratoren, Rückführungen und die verwendete Funktion f(x) bestimmen die Problemgröße. Um die Auswirkung der Anzahl der Integratoren und Rückführungen zu bestimmen, wird das in Abbildung 5.3 dargestellte Modell verwendet.

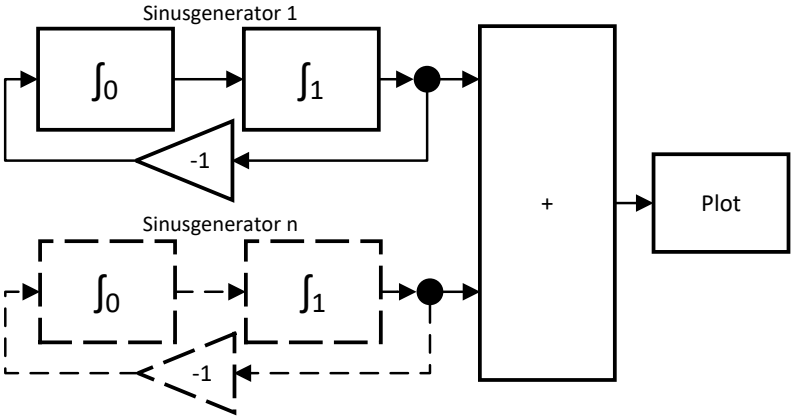



Abbildung 5.3.: Skalierbares, synthetisches Benchmark-Modell für zeitkontinuierliche MAEs: Parallele Sinusgeneratoren.

Das Benchmark-Modell leitet sich aus den vorausgegangenen Überlegungen ab und baut auf einem Ptolemy II-Beispielmodell für Sinusgeneratoren auf. $f(x)$ ist dabei als Verstärkung von **-1** gewählt. Die linken Integratoren ($\int_0$) eines Sinusgenerators weißen einen Startwert von 0 und die rechten ($\int_1$) von 1 auf, wodurch am Ausgang eine entsprechende Sinuswelle anliegt. Die Sinusgeneratoren werden parallel über einen Addierer-Block (**+**) verbunden, sodass eine Beurteilung der Skalierbarkeit in Abhängigkeit parallel ausgeführter Strukturen möglich ist.

**Integratorkette** Ein weiteres Benchmark-Modell ist in Abbildung 5.4 dargestellt.

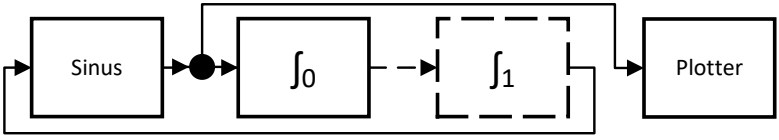



Abbildung 5.4.: Skalierbares, synthetisches Benchmark-Modell für zeitkontinuierliche MAEs: Integratorkette.

Es handelt sich um eine Kette aus Integratoren mit variabler Anzahl. Am Anfang befindet sich ein Sinusgenerator (**Sinus**). Alle Integratoren ($\int_0$), bis auf den letzten, erhalten den Startwert 0. Der letzte Integrator ($\int_1$) erhält den Startwert 1. Während der Ausführung werden die Integratoren nicht zurückgesetzt. Durch die Verkettung, nimmt der Eingangswert am Sinusgenerator

mit der Laufzeit immer stärker zu, was sich an einer immer höher werdenden Frequenz des Sinussignals am Plotter ablesen lässt. Mit zunehmender Laufzeit und/oder zunehmender Anzahl an Integratoren steigt also die Problemgröße.

### Zeitdiskret

Bei der zeitdiskreten Ausführung ist vor allem die Untersuchung der ereignisgesteuerten MAE relevant, da es dort keine Totzeit gibt und dadurch eine besonders schnelle Ausführung möglich ist. Bei der Analyse von Laufzeit und Speicherbedarf kommt es im Wesentlichen auf die Implementierung der Ereignisliste an. Hierzu eignet sich das in [19] beschriebene und in Abbildung 5.5 schematisch dargestellte Modell zur Untersuchung der Auswirkung der Ereignislistengröße auf die Laufzeit und den Speicherbedarf. Durch die Frequenz mit der die **Ereignisquelle** neue Ereignisse sendet und einer **Verzögerung**, die dazu führt, dass die Ereignisse in der Liste gehalten werden, lässt sich die Ereignisliste mit beliebig vielen Ereignissen füllen. Dadurch ist eine generische und skalierbare Untersuchung des MAE-Kernels möglich.

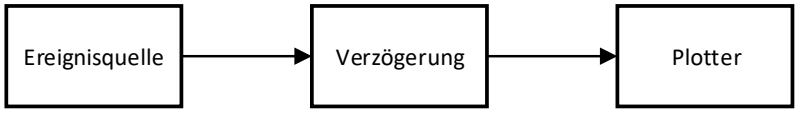



Abbildung 5.5.: Skalierbares, synthetisches Benchmark-Modell „Ereignisliste“ für zeitdiskrete MAEs (Quelle: eigene Darstellung nach [19]).

Zur Untersuchung der Parallelisierbarkeit werden mehrere, unabhängige Instanzen des Modells aus Abbildung 5.5 verwendet.

### Wertdiskret: Zustandsautomaten

Für den Benchmark der wertdiskreten MAEs werden weitere, skalierbare Benchmark-Modelle entwickelt. Durch Code-Analyse der entsprechenden MAE in Ptolemy II (FSM Director) und mithilfe der Simulink-Dokumentation [262] ist Abbildung 5.6 entstanden. Sie beschreibt den allgemeinen Algorithmus zur Ausführung von Zustandsautomaten, der bei beiden Simulatoren ähnlich ist. Ausgehend von einem **Anfangszustand** (oder auch History-Zustand), in dem sich der Automat zu Beginn der Ausführung befindet, werden die Übergänge entweder in einer festen Reihenfolge nach Priorität geordnet (Stateflow) oder zufällig (Ptolemy II) evaluiert. Bei der Evaluation der Übergänge wird geprüft, ob die Bedingung, an die der Zustandsübergang geknüpft ist, wahr ist. Wie in Abschnitt 2.4 für Stateflow beschrieben, so ist bei beiden Simulatoren der Zustandsautomat einem Diagramm zugeordnet. In Ptolemy II wird dieses als *ModalModel* und in Simulink als *Chart* bezeichnet. Ist ein Zustand selbst wiederum als ein solches Diagramm definiert, so müssen auch die internen Zustandsübergänge hierarchisch evaluiert werden, da sich die Zustandsübergänge aus verschiedenen Hierarchieebenen gegenseitig beeinflussen können. Sobald die Bedingung für einen Zustandsübergang erfüllt ist, wird der

Übergang zum entsprechenden Folgezustand ausgeführt. Sofern dieser nicht der finale **Endzustand** ist, beginnt hier erneut die Evaluation der Zustandsübergänge.

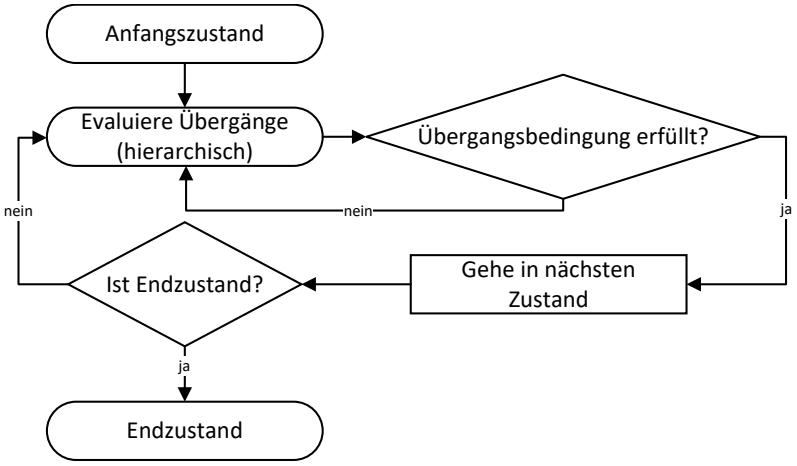



Abbildung 5.6.: Allgemeiner Ablauf der Ausführung eines Zustandsautomaten.

Aus Abbildung 5.6 lassen sich u.a. folgende Punkte mit Einfluss auf die Performanz herleiten:

1. **Anzahl und Art der Transitionen:** Die Anzahl der Transitionen korreliert mit der Anzahl der Ein- und Ausgänge des Charts bzw. ModalModels. Je höher die Anzahl der Ein- und Ausgänge, desto mehr Übergänge müssen evaluiert werden. Die Transitionsart beeinflusst hierbei den Aufwand der Evaluierung.
2. **Anzahl an Zuständen und lokalen Variablen:** Die Anzahl der Zustände beeinflusst indirekt die Transitionsanzahl. Lokale Variablen werden zur Speicherung von Zwischenergebnissen verwendet und haben damit Auswirkung auf den Speicherbedarf.
3. **Anzahl an Hierarchiebenen und deren Art:** Neben weiteren Zustandsdiagrammen können in Stateflow auch S-Fuktionen oder Subsysteme als Verfeinerungen für Zustände verwendet werden. Bei Ptolemy II können neben weiteren Zustandsdiagrammen auch Modelle mit beliebiger MAE als Verfeinerung für einen Zustand verwendet werden. Bei beiden Simulatoren bleiben die Ein- und Ausgänge der Diagramme für alle Hierarchieebenen konstant. Zusätzlich hat nicht nur die Anzahl der durch rekursive Verfeinerung entstehenden Hierarchieebenen Einfluss auf die Performanz, sondern auch die Anzahl an parallelen Zuständen innerhalb einer Verfeinerung .
4. **Anzahl und Art der Aktionen:** Dies betrifft vor allem zyklisch ausgeführte Aktionen mit hoher Frequenz.

Die aufgelisteten Punkte sollen mit skalierbaren Benchmarks untersucht werden.

**Transitions-, Zustands- und Variablenanzahl** Zur Performanzanalyse in Abhängigkeit von der Anzahl an Transitionen, Zuständen sowie Ein- und Ausgangsvariablen, soll das in Abbildung 5.7 dargestellte skalierbare Benchmark-Modell verwendet werden.

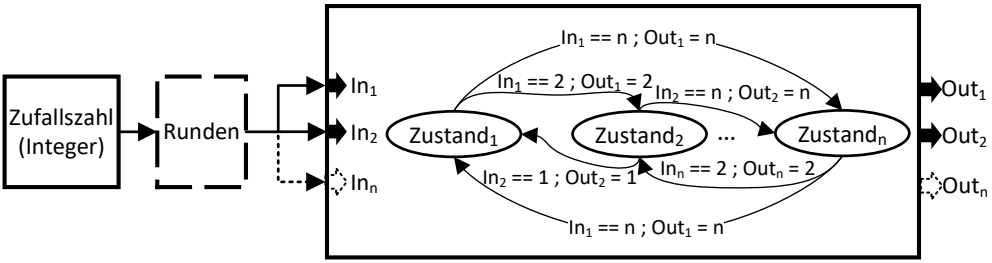



Abbildung 5.7.: Skalierbares, synthetisches Benchmark-Modell für wertdiskrete MAEs.

In Abhängigkeit der gewählten Benchmark-Modellgröße **n** werden immer gleich viele Zustände (**Zustand**$_i$), Ein- und Ausgänge (**In**$_i$ und **Out**$_i$) erzeugt. Von jedem Zustand aus gibt es Transitionen ($\rightarrow$) zu allen anderen Zuständen.Bei der Ausführung wird zunächst eine **Zufallszahl** $\in \mathbb{N}$ zwischen 1 und n generiert. Falls der Simulator dies nicht unterstützt, muss eine Zufallszahl $\in \mathbb{R}$ generiert und auf eine Zahl zwischen 1 und n gerundet werden (**Runden**). Die Zufallszahl dient gleichermaßen für alle In$_i$ als Eingabe und bestimmt welche Transition, ausgehend von dem aktuellen Zustand, als Nächstes genommen werden soll. Der Startzustand kann dadurch beliebig gewählt werden und hat keinen Einfluss auf das Ergebnis. Beim Übergang von Zustand$_j$ nach Zustand$_k$ wird der Eingangswert von In$_j$ an den Ausgang Out$_j$ übertragen.

**Transitionsarten** Bei den Transitionsarten wird generell zwischen **Ereignisbasiert**, **Zeitlogik** und **Booleschem Ausdruck** unterschieden. Abbildung 5.8 zeigt die Transitionsarten jeweils beispielhaft für einen Zustandsübergang ($\rightarrow$) nach **x** Sekunden.

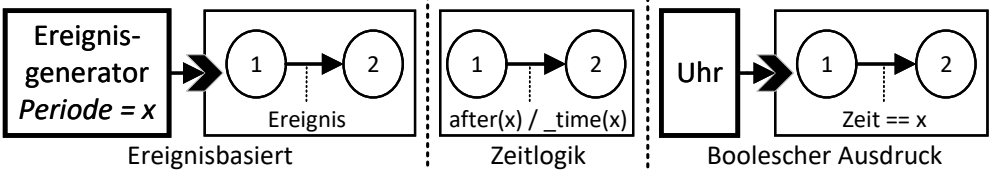



Abbildung 5.8.: Zustandsübergang nach x Sekunden von Zustand 1 nach Zustand 2 in Abhängigkeit unterschiedlicher Transitionsarten.

Beim ereignisbasierten Übergang wird an beliebiger Stelle im Zustandsautomat und nach Ablauf einer Zeit x ein **Ereignis** gesendet, welches den Übergang auslöst.
Bei Verwendung der Zeitlogik wird durch eine auf der Transition annotierte Syntax (z.B. **after(x)**) die Zeit x festgelegt, nach der ein Übergang erfolgen soll.

Bei der Evaluation mittels Booleschem Ausdruck wird die von einer **Uhr** bereitgestellte aktuelle Simulationszeit (**Zeit**) mit dem Zielwert der Zeit x verglichen. Sobald die Bedingung wahr ist, erfolgt der Übergang.

Mithilfe der dargestellten Prinzipien sollen die Evaluationsalgorithmen für die unterschiedlichen Transitionsarten in Bezug auf ihre Performanz analysiert werden. Dabei ist zu beachten, das Ptolemy II den ereignisbasierten Übergang an beliebiger Stelle im Zustandsautomat nicht unterstützt.

**Anzahl der Hierarchiestufen** Die Anzahl der Hierarchiestufen kann beeinflussen, wie oft Zustandsübergänge erneut evaluiert werden müssen, bevor sie tatsächlich ausgeführt werden können. Bei parallel ablaufenden Zustandsautomaten kann es ebenso sein, dass der Zustand eines Automaten den Zustand eines anderen, parallel ausgeführten, bedingt. Dies kann, je nach Implementierung, zu mehrfachen Evaluierungen der Übergangsbedingungen führen. Daher müssen die Auswirkungen von **Hierarchie** und **Parallelität** auf den Ressourcenverbrauch untersucht werden, wobei die in Abbildung 5.9 dargestellten skalierbaren Benchmark-Modelle verwendet werden sollen.

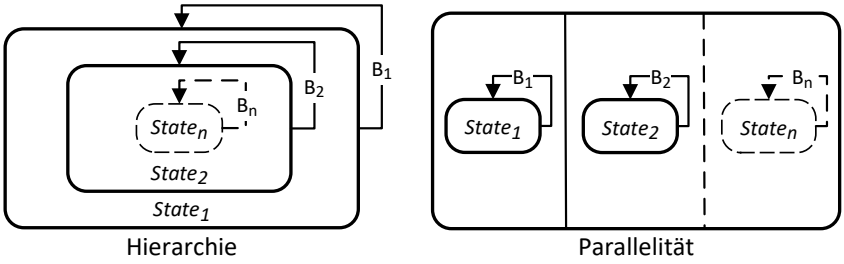



Abbildung 5.9.: Skalierbare, synthetische Benchmark-Modelle zur Untersuchung von Hierarchie und Parallelität.

Beim Modell Hierarchie befindet sich in jeder Hierarchiestufe ein Zustandsautomat mit einem Zustand (**State**$_i$), der eine Transition auf sich selbst besitzt. Dieser Transition ist eine Bedingung **B** zugeordnet. Jeder Zustand (State$_{i+1}$) ist dabei die Verfeinerung eines übergeordneten Zustandes (State$_i$).

Beim Modell Parallelität, befindet sich in jeder parallelen Stufe ein Zustand (State$_i$) mit einer Transition auf sich selbst. Dieser Transition ist ebenso eine Bedingung B zugeordnet.

**Art der Aktionen** Bei der Untersuchung des Ressourcenverbrauchs in Abhängigkeit der Aktionen muss bei Simulink zwischen *Set Actions* und *Send-event Actions* unterschieden werden. Letztgenannte sind in Ptolemy II nicht vorhanden, weswegen hier ein direkter Vergleich nicht möglich ist.

#### Hybrid

Zur Bildung von hybriden Modellen gibt es unterschiedliche HS-Modellierungsarten (siehe Unterabschnitt 2.2.4). In Ptolemy II wird hauptsächlich die Art *Interaktion* genutzt, bei der Submodelle in jeweils unterschiedlichen HS-Modellierungsarten modelliert werden können. Diese Submodelle werden hierarchisch unter einer Hauptmodellierungsart miteinander gekoppelt (siehe Abbildung 2.18). Bei Verfeinerungen von Zustandsautomaten mit beliebigen MAEs in aufeinanderfolgenden Zuständen, können hybride Modelle in *sequenzieller* Art modelliert werden. In Simulink wird dagegen eine Mischform der HS-Modellierungsarten *Anreichernd* und *Interaktion* verwendet (siehe Abbildung 2.19). So gibt es eine globale MAE (Solver), welche entweder kontinuierlich oder diskret ist. Bei Zustandsautomaten können, analog zu Ptolemy II, hybride Modelle in *sequenzieller* Art modelliert werden.

Zur Kopplung von Modellen mit unterschiedlichen MAEs werden bei beiden Simulatoren explizite oder implizite Wandler benötigt. Eine solche Wandlung kann, je nach Implementierung, den Ressourcenverbrauch erhöhen und hat daher Auswirkung auf die Performanz [165]. In Ptolemy II werden etwa für hybride Modelle mit zeitdiskreten und kontinuierlichen Anteilen Wandler (*Digital/Analog* (**D/A**) bzw. *Analog/Digital* (**A/D**)) benötigt, wie in Abbildung 5.10 beispielhaft dargestellt.

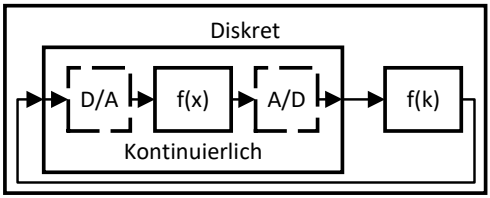



Abbildung 5.10.: Beispiel der Datenwandlung bei hybriden Modellen in Ptolemy II.

Der kontinuierliche (analoge) Teil **f(x)** ist hier über entsprechende Wandler (D/A und A/D) vom diskreten (digitalen) Teil **f(k)** getrennt. Bei Simulink werden dagegen hybride Systeme innerhalb eines kontinuierlichen, übergeordneten Solvers realisiert. Dazu werden entsprechende Schrittweiten festgelegt, wofür keine expliziten Wandler nötig sind. Explizite Wandler werden jedoch etwa bei Verwendung zusätzlicher Simulink-Toolboxen benötigt.

Aufgrund der unterschiedlichen HS-Modellierungsarten von Ptolemy II und Simulink ist ein adäquater Vergleich über synthetische Benchmarks nicht sinnvoll. Daher sollen anschließende Anwendungsbenchmarks Performanzprobleme aufdecken.

### 5.2.2. Ausführung und Ergebnisse

Alle Simulationen werden auf System A (siehe Tabelle 2.3) ausgeführt. Die Rohdaten der Ergebnisse sind ferner unter Abschnitt A.3 im Anhang zu finden.

Zur Ermittlung von Öffnungs-, Generierungs- und Laufzeiten sowie des Speicherbedarfs von Ptolemy II werden von den Entwicklern bereits implementierte Methoden verwendet, die wiederum auf Methoden der Java-Klasse *System* beruhen.

Zur Ermittlung der Generierungs- und Öffnungszeiten von Simulink-Modellen werden Matlab-Skripte und darin die Befehle `tic` und `toc` verwendet. Zur Bestimmung der Laufzeiten wird hingegen der integrierte Matlab-Profiler für Skripte verwendet, wobei die Simulation über den Befehl `sim('Modellname')` gestartet wird. Der Profiler erlaubt jedoch standardmäßig keine Aussagen über den Speicherbedarf. Mithilfe des undokumentierten Befehls `profile('-memory')` wird die Analyse des Speicherbedarfs innerhalb von Matlab-Skripten jedoch aktiviert. Somit lässt sich auch der Speicherbedarf ermitteln.

Es ist zu beachten, dass sich die Laufzeiten einer Simulation bei einer Wiederholung der Ausführung verbessern. Beispielsweise dadurch, dass Caches dann gefüllt und Klassen geladen sind oder Teile des Codes nicht mehr zur Laufzeit kompiliert werden müssen (Just-in-time-Kompilierung). Damit liefert die erste Ausführung die maximale Laufzeit. Da diese maximale Laufzeit für den Vergleich gesucht wird und die Laufzeiten teilweise so groß sind, dass eine Varianz keinen Einfluss auf die Aussage der Ergebnisse hat, wird bei den Ausführungen auf eine Wiederholung verzichtet.

Im Folgenden werden die Ergebnisse der synthetischen Benchmarks dargestellt.

### Zeitkontinuierlich

Für die Ausführung der Simulationen wurden weitestgehend die Standardkonfigurationen der MAEs verwendet, wobei Anpassungen im Folgenden beschrieben sind. Ptolemy II bietet als Gleichungslöser nur die expliziten Runge-Kutta Verfahren nach *Bogacki/Shampine* (ODE23) und *Dormand/Prince* (ODE45) an. Da der ODE23 bei Ptolemy II standardmäßig ausgewählt ist und Simulink automatisch einen geeigneten Gleichungslöser anhand des Problems auswählt, wird zur besseren Vergleichbarkeit für beide Simulatoren der ODE23-Löser ausgewählt. In Simulink ist dagegen eine relative Fehlertoleranz von $10^{-3}$ festgelegt, die auch in Ptolemy II übernommen wird (standardmäßig $10^{-4}$). Die maximale Schrittweite muss ferner explizit bei Ptolemy II angenommen werden. Sie wird auf 100 festgelegt, was innerhalb der betrachteten Zeiträume liegt.

**Parallele Sinusgeneratoren** In den Rohdaten sind bei Ptolemy II nur Ergebnisse für maximal 250 parallele Sinusgeneratoren eingetragen. Dies liegt daran, dass ein Modell weder mit 1000 noch mit 500 Generatoren ausgeführt werden konnte, da die Benutzeroberfläche beim Öffnen des Modells eingefroren ist. Aus diesem Grund wird im Folgenden nur eine maximale Anzahl von 250 Sinusgeneratoren betrachtet.

Der Vergleich der Generierungs- und Modellöffnungszeiten von Simulink und Ptolemy II ist in Tabelle 5.3 dargestellt. Bei den Generierungszeiten ist Ptolemy II im Durchschnitt um den Faktor 3,6 schneller, dafür aber um den Faktor 3,16 langsamer beim Öffnen der Modelle. Bei einer integrierten Simulation muss jedoch die Summe aus Generierungs- und Öffnungszeit betrachtet werden, also die benötigte Zeit, bis die Simulation beginnen kann. Diese liegt bei Simulink im Schnitt bei ca. 3,96 s und bei Ptolemy II bei ca. 7,66 s. Damit ist Simulink mit einem

Faktor von 1,93 fast doppelt so schnell, als Ptolemy II. Es ist außerdem zu beobachten, dass sowohl Generierungs- als auch Modellöffnungszeiten in beiden Fällen mit der Modellgröße (Anzahl der Sinusgeneratoren) zunehmen.

Tabelle 5.3.: Vergleich der Generierungs- und Modellöffnungszeiten beim Benchmark „Parallele Sinusgeneratoren“.

| Anzahl der Sinusgeneratoren | Zeiten Simulink [s] | | Zeiten Ptolemy [s] | |
|---|---|---|---|---|
| | Generierung | Öffnung | Generierung | Öffnung |
| 10 | 0,416222 | 0,377011 | 0,216 | 2,317 |
| 100 | 0,684888 | 0,952083 | 0,286 | 1,25 |
| 250 | 1,142672 | 1,635989 | 0,367 | 4,398 |
| 500 | 1,857296 | 2,998606 | 0,506 | 8,4 |
| 1000 | 4,348257 | 5,398868 | 0,969 | 19,567 |
| Durchschnitt | **1,689867** | **2,2725114** | **0,4688** | **7,1864** |

Abbildung 5.11 zeigt den Speicherbedarf der beiden Simulatoren mit einer logarithmischen Darstellung der y-Achse (**Speicherverbrauch**).

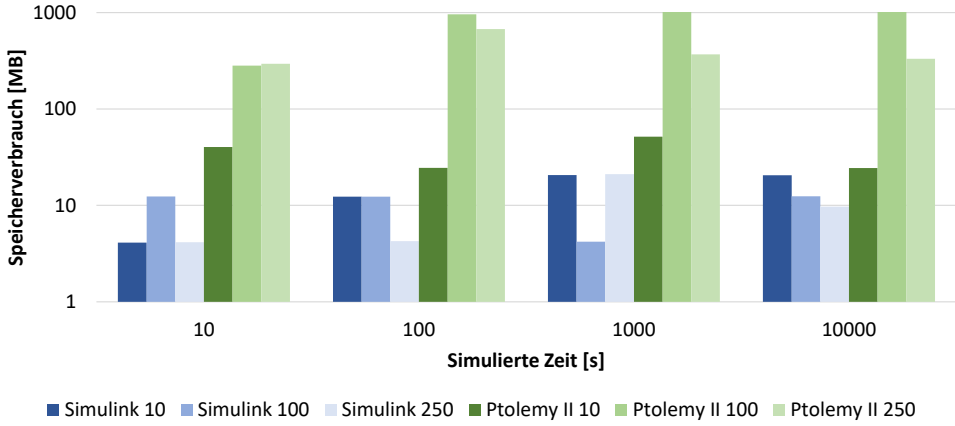
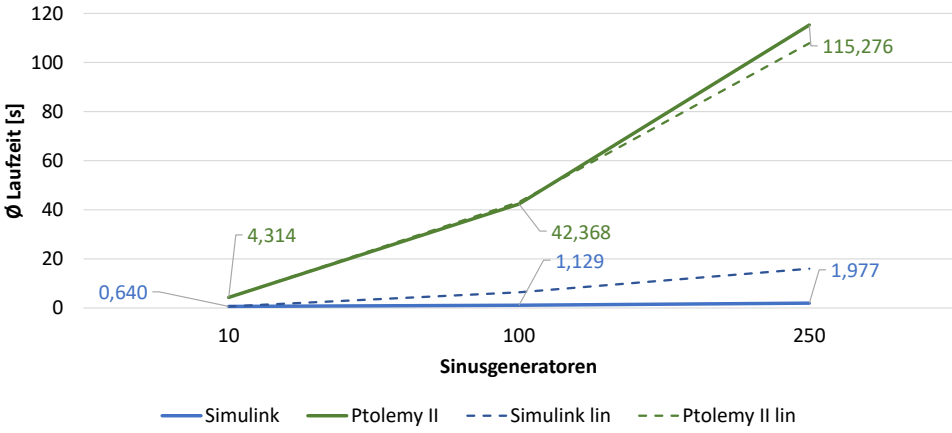


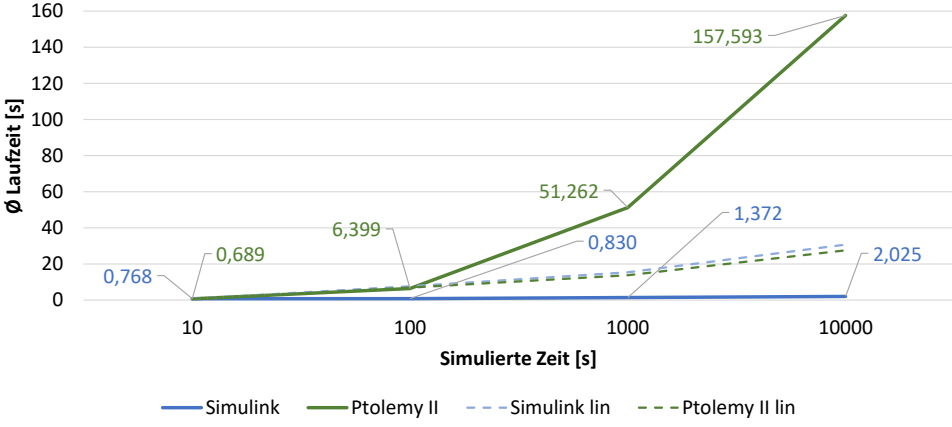

Abbildung 5.11.: Speicherbedarf beim Benchmark „Parallele Sinusgeneratoren“.

Die Messungen zeigen, dass der Speicherverbrauch bei Ptolemy II ca. 10 bis 100-fach gegenüber Simulink erhöht ist. Weiterhin ist zu erkennen, dass der Verbrauch weder mit der Modellgröße noch mit der Laufzeit korreliert.

Abbildung 5.12a zeigt den Durchschnitt der Laufzeiten für alle simulierten Zeiten in Abhängigkeit der Anzahl an Sinusgeneratoren. Abbildung 5.12b zeigt dagegen den Durchschnitt der Laufzeiten für 10, 100 und 250 Sinusgeneratoren in Abhängigkeit der simulierten Zeit.




(a) Vergleich anhand der simulierten Zeit.




(b) Vergleich anhand der Anzahl der Sinusgeneratoren.

Abbildung 5.12.: Durchschnittslaufzeiten beim Benchmark „Parallele Sinusgeneratoren“.

Damit soll die Laufzeit sowohl in Abhängigkeit der simulierten Zeit als auch der Anzahl der Entitäten überprüft werden. Es wird dabei deutlich, dass die Laufzeiten bei beiden Simulatoren sowohl mit der simulierten Zeit als auch mit der Anzahl der Elemente zunimmt. Bei einer simulierten Zeit von 10 s, ist zu erkennen, dass Ptolemy II schneller als Simulink ausführt. Dies liegt daran, dass bei Simulink die Kompilierzeit in die Laufzeit mit einfließt und die Modelle in Ptolemy II nicht kompiliert werden. Eine Verzehnfachung der simulierten Zeit von 10 s auf 100 s hat bei Simulink kaum Auswirkung auf die Laufzeit (Zunahme um Faktor 1,081), während eine Verzehnfachung der Sinusgeneratoren von 10 auf 100 einen Anstieg um den Faktor

1,74 bewirkt. Bei Ptolemy trifft dies ebenfalls zu, jedoch sind die Faktoren mit 9,28 bzw. 9,82 deutlich höher als bei Simulink. Die Kurven **Simulink lin** und **Ptolemy II lin** markieren den Verlauf eines linearen Verhaltens, ausgehend jeweils vom ersten Messwert Laufzeit. Simulink zeigt dabei durchweg ein sub-lineares Verhalten und skaliert deshalb sowohl mit der simulierten Zeit, als auch mit der Modellgröße. Anders ist es mit Ptolemy II, welcher sich ab ca. 114 Sinusgeneratoren respektive einer simulierten Zeit von 111,63 s super-linear verhält und damit nicht skaliert.

**Integratorkette** Aufgrund der im Vergleich zu den Modellen mit parallelen Sinusgeneratoren geringeren Modellgrößen mit maximal 20 Integratoren, wird an dieser Stelle nicht näher auf die Generierungs- und Öffnungszeiten eingegangen.

Tabelle 5.4 zeigt die Laufzeiten und den Speicherbedarf des Benchmark-Modells „Integratorkette“, in Abhängigkeit der **simul**ierten **Zeit** und der Anzahl der **Integratoen**.

Tabelle 5.4.: Laufzeiten und Speicherbedarf beim Benchmark „Integratorkette“.

| Integratoren ↘ | Simulink: Laufzeiten [s] | | | Ptolemy II: Laufzeiten [s] | | | Simulink: Speicher [MB] | | | Ptolemy II: Speicher [MB] | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ↓ Simul. Zeit [s] | 5 | 10 | 20 | 5 | 10 | 20 | 5 | 10 | 20 | 5 | 10 | 20 |
| 10 | 0,448 | 0,811 | 0,702 | 0,150 | 0,200 | 0,466 | 12,316 | 22,776 | 17,776 | 138,096 | 628,225 | 474,662 |
| 100 | 0,673 | 0,717 | 0,722 | 2,138 | 96,869 | NA | 25,928 | 12,316 | 12,316 | 304,113 | 302,823 | NA |
| 1000 | 0,851 | 1,023 | 0,815 | 27,134 | NA | NA | 3,736 | 16,396 | 5,348 | 281,973 | NA | NA |
| Ø | **0,657** | **0,850** | **0,746** | **9,807** | **48,535** | / | **13,993** | **17,163** | **11,813** | **241,394** | **465,524** | / |

In den Zellen, in denen nicht anwendbar (**NA**) eingetragen ist, konnten die Werte nicht ermittelt werden, da Ptolemy II die Simulation aufgrund von Speichermangel nach Laufzeiten von teilweise über 22 h abgebrochen hat. Eine Analyse mit dem Werkzeug VisualVM hat ergeben, dass der größte Anteil an der Laufzeit, mit ca. 53 %, der `fire()`-Methode der Integratoren zuzuordnen ist. Innerhalb dieser werden Tokens mit neuen Daten an den Ausgang gesetzt. Hiervon ist wiederum der Hauptteil dem Token-Management zuzuordnen. Lediglich 0,2 % der Gesamtlaufzeit werden dem Lösen der Integrale zugeordnet. Weitere 24 % sind dem Plotter mit der Methode `addPoint()` zuzuweisen.

Mit den Ergebnissen lässt sich erkennen, dass Simulink im Durchschnitt nicht nur performanter ist und einen niedrigeren Speicherbedarf aufweist, sondern auch alle Probleme lösen kann. In beiden Fällen ist ferner zu erkennen, dass die Simulationszeit einen größeren Einfluss auf die Laufzeit hat, als die Anzahl der Integratoren.

**Fazit und Deutung** Das Modell „Parallele Sinusgeneratoren“ entspricht einer Struktur, wie sie in der Realität beim Design von E/E-Architekturen in der Modellierung von Batteriezellen vorkommen kann. Der Berechnungsaufwand bei der üblichen Anwendung des *Electrical Equivalent Circuit Model (EECM)* ist sogar höher [S1, 310]. Bei einer Li-Ion-Batterie ergibt sich die Gesamtspannung aus der Summe der einzelnen Zellspannungen (Serienschaltung). Ausgehend von 3,7 V Zellspannung bei Li-Ion-Zellen, werden für typische 400 V also ca. 108 Zellen benötigt. Um die entsprechende Kapazität zu erreichen, müssen diese 108 Zellen durch Parallelschaltung vervielfacht werden, was bspw. bei einem Tesla Model S zu einer Gesamtzahl

von 7104 Zellen führt [80]. Die detaillierte Modellierung der Zellen kann zur Untersuchung von Alterserscheinungen und Erwärmung notwendig sein. Ptolemy II wäre für einen solchen Anwendungsfall nicht geeignet.

Das Modell „Integratorkette“ wird mit der Laufzeit aufgrund der zunehmenden Frequenz komplexer. In der Realität gibt es dazu keinen Vergleich. Dennoch werden durch diesen synthetischen Benchmark die Schwachstellen von Ptolemy II beim Datenmanagement offengelegt. Diese führen dazu, dass der Speicherbedarf immer weiter ansteigt und die Simulation nach einer Laufzeit von mehr als 22 h abbricht. Simulink löst die Probleme in durchschnittlich 0,75 Sekunden und benötigt bei den gemessenen Ergebnissen ca. 25-mal weniger Speicher. Stichprobenartig konnte bei den fehlgeschlagenen Simulationen von Ptolemy II ein Speicherverbrauch von mehr als 3,5 Gb beobachtet werden.

### Zeitdiskret

Für den Vergleich wird in Simulink die SimEvents-Toolbox verwendet, sodass eine ereignisgesteuerte Simulation möglich ist. Außerdem wird der Solver „discrete“ mit variabler Schrittweite ausgewählt. In Ptolemy II wird dagegen die MAE *Discrete Event (DE)* verwendet. Die MAE *Distributed Discrete Event (DDE)*, welche eine verteilte Berechnung erlauben soll, ist experimentell und nicht mit der benötigten Komponente *Server* kompatibel. Daher erfolgt im Anschluss eine gesonderte Betrachtung.

**Ereignisliste** Für diese Analyse wird das in Abbildung 5.5 dargestellte Modell verwendet. Die in den Simulatoren verwendeten Entitäten sind in Tabelle 5.5 dargestellt.

Tabelle 5.5.: Für zeitdiskrete Benchmarks verwendete Entitäten.

| **Element** | **Ptolemy II** | **Simulink** |
|---|---|---|
| Quelle | Discrete Clock | Entity Generator |
| Verzögerung | Server | Entity Server |
| Plot | Timed Plotter | Scope |
| Ereignis | Token | Entity |

Bei beiden Simulatoren ist die Kapazität der **Server** auf „unendlich“ gestellt. Für die Struktur der **Entity** wird die Option „anonym“ gewählt, um in beiden Simulatoren gleiche Bedingungen zu schaffen, da der Server in Ptolemy II die Datenstruktur der ankommenden **Token** weder ausliest noch verarbeitet. Die **Verzögerung** wird durch die „Service-Zeit“ der Server festgelegt. Sie ist in den Benchmarks konstant auf 10 s eingestellt. Variabel ist jedoch die in den **Quellen** eingestellte Sendeperiode, um die Anzahl der er Ereignisse in der Ereignisliste zu steuern. Beträgt die Sendeperiode zum Beispiel 0,1 s, so sind nach 10 s 100 Ereignisse in der Ereignisliste gespeichert. Für den Benchmark werden ferner zwei simulierte Zeiten (10 s und 20 s) und zwei Modelle (einfach und doppelt) betrachtet. Bei einer simulierten Zeit von 10 s wird ein Leeren der Ereignisliste nicht betrachtet, während nach 20 s die Ereignisse aus

der Liste abgearbeitet sind. Im einfachen Fall liegt das Modell nach Abbildung 5.5 einzeln vor, während im doppelten Fall zusätzlich eine unabhängige Kopie davon parallel simuliert wird.

Die Modelle wurden händisch erstellt, da es sich jeweils nur um drei bzw. sechs Entitäten handelt. Daher werden Generierung- und Öffnungszeiten vernachlässigt.

Abbildung 5.13 zeigt die benötigten Simulationslaufzeiten von Ptolemy II und Simulink in Abhängigkeit der Anzahl der Ereignisse in der Ereignisliste und bei verschiedenen Konfigurationen. Die **simulierte Zeit** steht in den runden Klammern in der Legende, während **x1** die Ausführung des einfachen und **x2** die Ausführung des doppelten Modells bedeutet. Es wird deutlich, dass Ptolemy II in allen Ausführungen schneller als Simulink ist. Im Durchschnitt ist Ptolemy II dabei um den Faktor 1,47 schneller. Außerdem wird in der Abbildung ersichtlich, dass die Auswirkungen auf die Laufzeit, hervorgerufen durch eine doppelte Ausführung oder Verlängerung der simulierten Zeit, bei beiden Simulatoren ähnlich sind. Eine Verdopplung des Problems, durch die Ausführung einer zusätzlichen Kopie des eigentlichen Modells, führt ferner in keinem Fall zur Verdopplung der Simulationslaufzeit. Beide Simulatoren skalieren demnach mit der Änderung der Problemgröße.

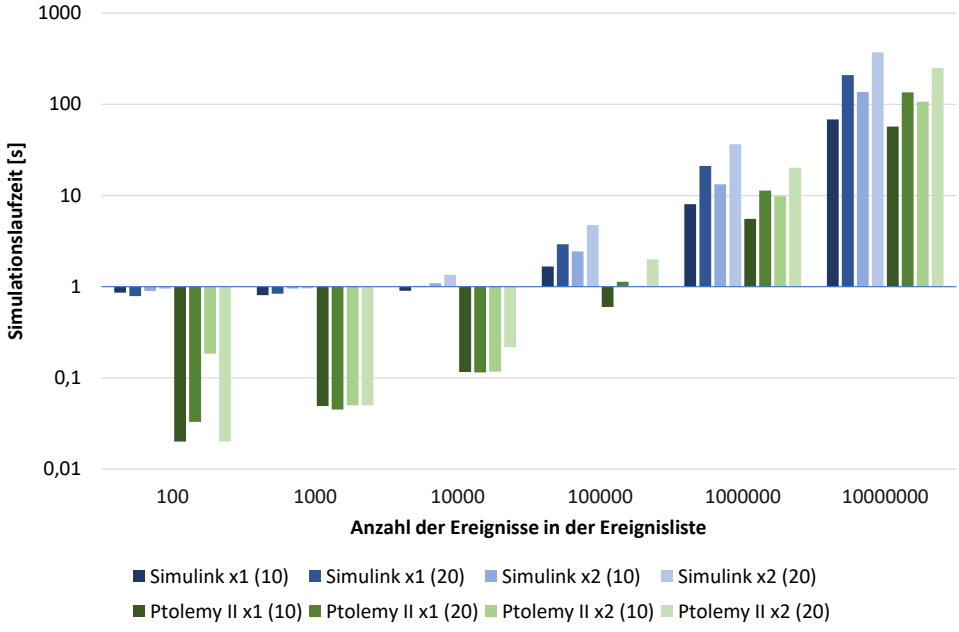
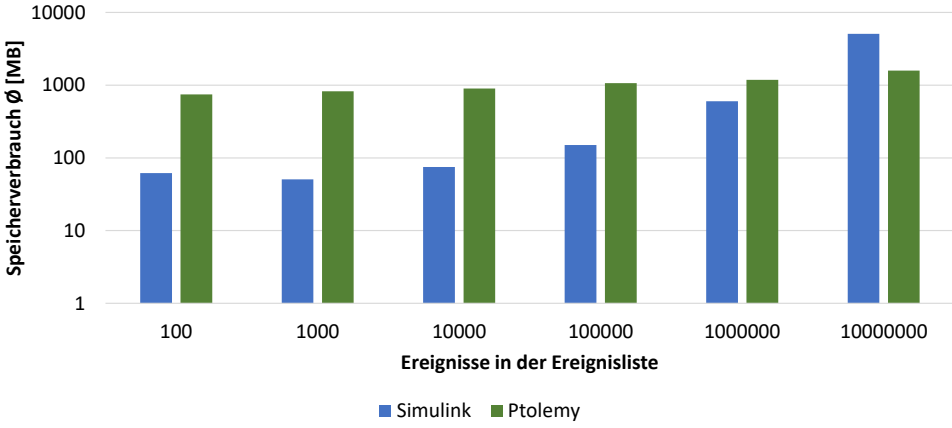


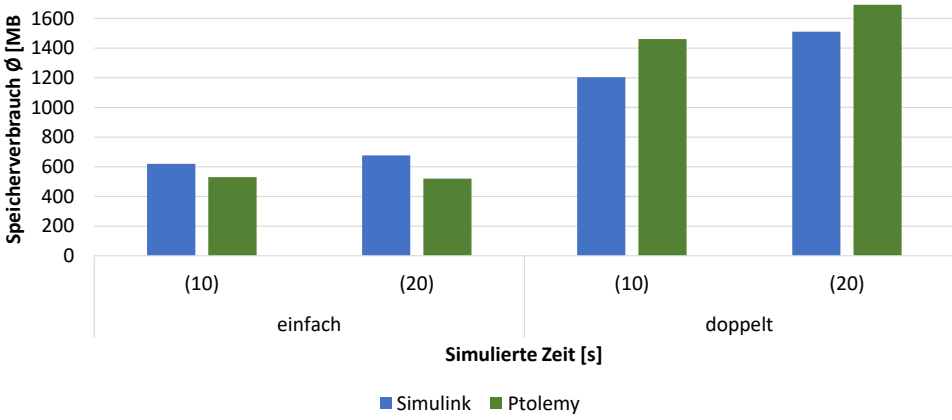

Abbildung 5.13.: Simulationslaufzeiten beim Benchmark „Ereignisliste“.

Der durchschnittliche Speicherverbrauch ist in Abbildung 5.14 dargestellt. Abbildung 5.14a zeigt den durchschnittlichen Speicherverbrauch für alle Konfigurationen in Abhängigkeit der Anzahl der **Ereignisse in der Ereignisliste**. Ptolemy II verbraucht bis zu einer Anzahl

von 10 Mio. Ereignissen mehr Speicher als Simulink. Danach übersteigt der durchschnittliche Speicherbedarf von Simulink den von Ptolemy II. Insgesamt beträgt der Mittelwert aller Durchschnittswerte bei Ptolemy II 1025,87 MB, mit einer Standardabweichung von 273,6. Bei Simulink beträgt der Mittelwert aller Durchschnittswerte hingegen 978,9 MB, mit einer Standardabweichung von 1789,5. Da der Speicherbedarf bei Simulink mit der Anzahl der Ereignisse zunimmt, lässt sich daraus schließen, dass Simulink über eine bessere Anpassung der Speichernutzung verfügt.




(a) Basierend auf der Ereignislistengröße.




(b) Basierend auf der Modellgröße und der simulierten Zeit.

Abbildung 5.14.: Durchschnittlicher Speicherverbrauch beim Benchmark „Ereignisliste“.

Abbildung 5.14b stellt den durchschnittlichen Speicherverbrauch in Abhängigkeit der **simulierten Zeit** und der Modellgröße (**einfach, doppelt**) dar. Das Leeren der Ereignisliste ab einer Ausführungszeit von 20 s führt hierbei nur zu einem verhältnismäßig geringfügigen Anstieg des Speicherbedarfs. Bei Ausführung des doppelten Modells ist ferner zu erkennen, dass bei beiden Simulatoren etwa der doppelte Speicher benötigt wird.

**Ereignisliste parallel** Da das vorherige Benchmark-Modell „Ereignisliste“ mit drei bzw. sechs Entitäten eine geringe Größe aufweist, soll ein größerer Maßstab durch Vervielfältigung des Ursprungsmodells untersucht werden. Dabei ist die Sendeperiode aller Quellen auf 0,001 s, die Service-Zeit auf 2,0 s und die simulierte Zeit auf 5 s festgelegt.

Die Generierungs- und Öffnungszeiten sind vergleichbar mit den in Tabelle 5.3 dargestellten Ergebnissen und in Unterabschnitt A.3.2 im Anhang zu finden.

Der Speicherverbrauch ist in Abbildung 5.15 in Abhängigkeit der Anzahl parallel ausgeführter Modelle „Ereignisliste“ dargestellt.

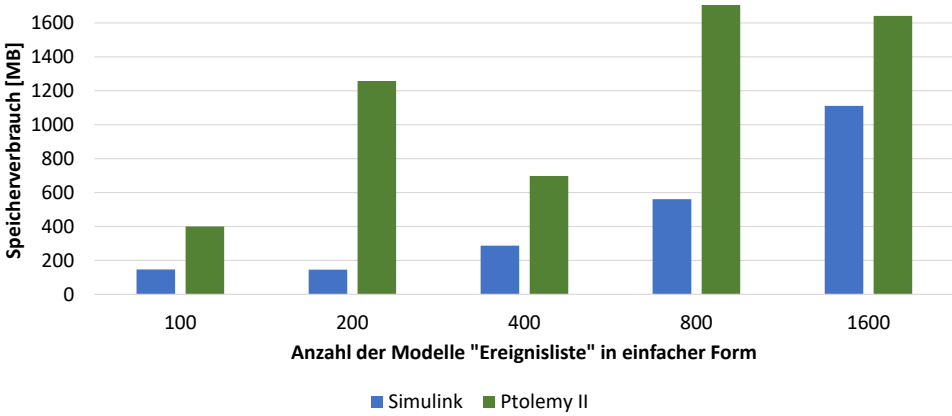



Abbildung 5.15.: Vergleich des Speicherbedarfs beim Benchmark „Ereignisliste parallel“.

Hierbei weist Ptolemy II, analog vorausgegangenen Benchmark „Ereignisliste“, einen höheren Speicherbedarf als Simulink auf, wobei sich Simulink mit steigender Problemgröße annähert. Im Mittel ist der Verbrauch bei Ptolemy II gegenüber Simulink ca. 2,54-mal höher.

Abbildung 5.16 stellt die Simulationslaufzeiten in Abhängigkeit der Anzahl von parallel ausgeführten Modellen „Ereignisliste“ dar. Es wird deutlich, dass beide Simulatoren bereits ab 200 parallel ausgeführten Modellen eine super-lineare Skalierbarkeit zeigen, also nicht skalieren. Dennoch skaliert Ptolemy II mit steigender Problemgröße schlechter als Simulink. Dies ist zum einen an den linearisierten Kurven (**lin.**), ausgehend von 100 Ursprungsmodellen, zu erkennen. Die Originalkurve **Simulink** liegt mit steigender Anzahl an parallel simulierten Modellen nur geringfügig über der linearisierten **Simulink (lin.)**. Bei Ptolemy II fällt der Abstand

zwischen der Originalkurve **Ptolemy II** und der linearisierten Kurve **Ptolemy II (lin.)** deutlich größer aus. Zum anderen wurden die Originalkurven exponentiell genähert (**exp.**). Je flacher die Steigung ist, desto höher ist die Skalierbarkeit. Die Steigung ist bei **Simulink (exp.)** etwa 1,6-mal geringer als bei **Ptolemy II (exp.)**.

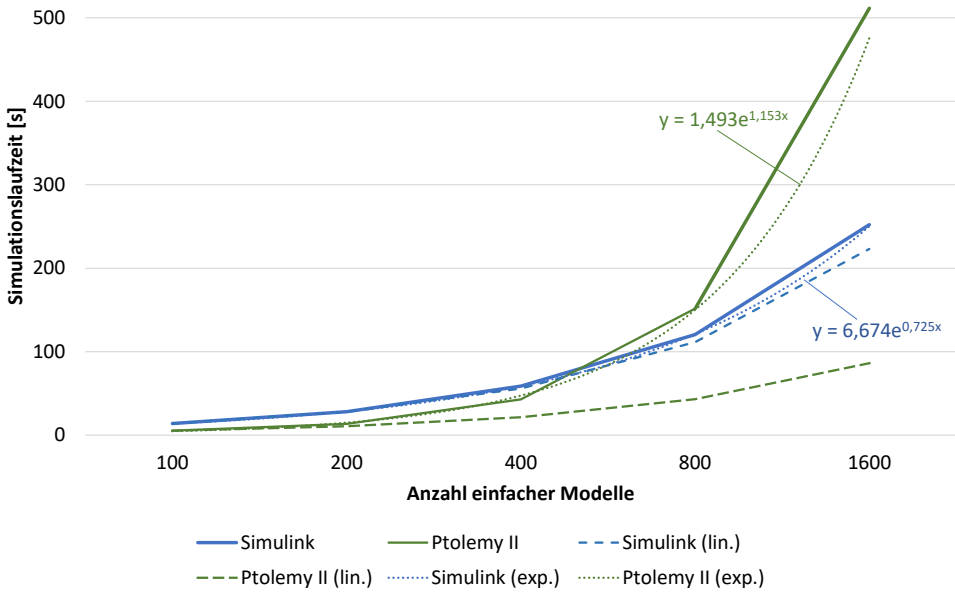



Abbildung 5.16.: Vergleich der Laufzeiten beim Benchmark „Ereignisliste parallel“.

**Vergleich der DE- und DDE-MAE von Ptolemy II** In der Arbeit [S8] wurden die Laufzeitunterschiede zwischen der DE- und der DDE-MAE untersucht. Mit der DDE-MAE sollen ereignisdiskrete Simulationen parallel berechnet werden können. Allerdings ist diese MAE als „experimentell“ gekennzeichnet. Für den Vergleich wurden **serielle** und **parallele** Strukturen mit typischen DE-Entitäten (**DE-typisch**) und Entitäten mathematischer Operationen (**Mathematisch**) miteinander verglichen. Es wurde dabei festgestellt, dass die DDE-MAE in den meisten Fällen langsamer als die DE-MAE ist. Der Vorteil in der Laufzeit liegt in wenigen Fällen (bei vorwiegend parallelen Strukturen) bei maximal 4-6 %. Abbildung 5.17 fasst die Ergebnisse dieser Arbeit zusammen. Der Geschwindigkeitsvorteil der DE-MAE gegenüber der DDE-MAE ist hierbei für vier unterschiedliche Modelle, mit parallelen und seriellen Strukturen, in Prozent aufgetragen. Negative Werte bedeuten, dass die DDE-MAE bei gleicher Problemstellung schneller als die DE-MAE ausführt. Das Ergebnis zeigt, dass die DDE-MAE vor allem bei seriellen Strukturen um bis zu 382 % langsamer als die DE-MAEs ist. Eine weitere, aus dem Diagramm hervorgehende Information ist, dass typische DE-Entitäten, wie Server

oder Weichen, schneller als mathematische Entitäten, wie + oder -, ausführen. Die Ursache davon konnte durch Profiling mit dem Tool VisualVM herausgefunden werden. Hierbei nimmt das Verschicken von Null-Nachrichten zur Zeitsynchronisation (siehe Unterabschnitt 2.2.5) einen Großteil der Laufzeit ein. Ursache für diesen „Lookahead-Creep“ ist die Verwendung eines CMB-Algorithmus in der Implementierung.

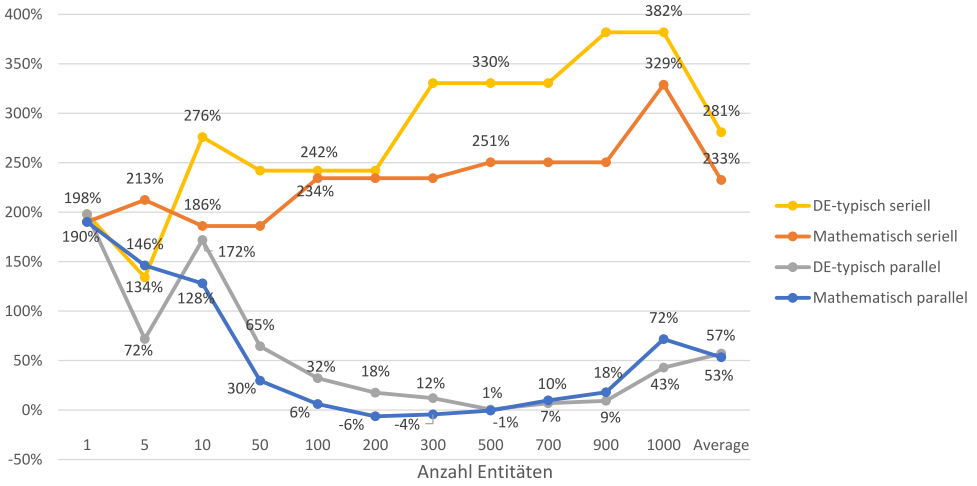



Abbildung 5.17.: Vergleich der DE- und DDE-MAEs von Ptolemy II (Quelle: [S8]).

**Fazit** Beim Benchmark „Ereignisliste“ hat Ptolemy II zunächst die bessere Performanz in der Laufzeit gezeigt. Bei größeren Modellen mit paralleler Struktur bricht seine Performanz jedoch ein. Hier skaliert Simulink deutlich besser, was der Benchmark „Ereignisliste parallel“ zeigt. Ferner steigt bei Ptolemy II der Speicherverbrauch nur geringfügig mit der Anzahl der Ereignisse in der Ereignisliste. Bei Simulink korreliert der Speicherverbrauch dagegen mit der Modellgröße und der Anzahl an Ereignissen in der Ereignisliste, was besonders bei kleineren Modellen von Vorteil ist.

### Wertdiskret

Basierend auf den in Unterabschnitt 5.2.1 vorgestellten Benchmark-Konzepten für wertdiskrete MAEs, werden im Folgenden die durchgeführten Benchmarks sowie deren Ergebnisse beschrieben und erläutert. Dabei ist zu anzumerken, dass sich Ptolemy II nur teilweise an der UML-Spezifikation für Zustandsautomaten orientiert. So lassen sich weder Ereignisse durch Aktionen verschicken, um damit Zustandsübergänge an beliebiger Stelle auszulösen, noch *During*, *Entry* oder *Exit* Aktionen definieren. Es gibt hier nur *set* und *output* Aktionen, die bei Stateflow zu *set data* Aktionen zusammengefasst sind. Unterschiedliche Arten von Aktionen

und die Verwendung von Ereignissen zum Auslösen von Zustandsübergängen werden daher bei den Benchmarks nicht näher betrachtet.

Für die Generierung der Benchmark-Modelle wurde entsprechender Java- bzw. Matlab-Code geschrieben. Bei beiden Simulatoren wird darüber hinaus die ereignisgesteuerte MAE als übergeordnete MAE verwendet. Des Weiteren sind bei der Ausführung die Animationen für Zustandsübergänge ausgeschaltet, um gleiche Bedingungen bei beiden Simulatoren zu schaffen.

Beim Messen der Modellöffnungszeiten der Simulink-Modelle konnte aufgrund der Verwendung der Stateflow-Toolbox nicht auf die Methoden `tic` und `toc` zurückgegriffen werden. In der `open_system()`-Methode fehlen dazu benötigte, Stateflow-spezifische Anteile und *P-Files* (nicht lesbarer Code). Daher wurde nur der Matlab-Profiler verwendet.

**Zustände und Übergänge** Mit diesem Benchmark wird die Auswirkung der Anzahl der Zustände und der Übergänge auf die Performanz untersucht. Als Basis dient das in Abbildung 5.7 dargestellte Modell. Durch unterschiedliche Sendeperioden des Zufallszahlgenerators wird die Anzahl der Übergänge pro Zeit und damit die Frequenz der Auswertung der Übergangsbedingungen beeinflusst. Des Weiteren werden zwei simulierte Zeiten betrachtet, um deren Auswirkung auf die Performanz zu analysieren.

Abbildung 5.18 stellt die Simulationslaufzeiten von Ptolemy II und Simulink in Abhängigkeit der Sendeperioden, der simulierten Zeit und der Anzahl der Zustände dar.

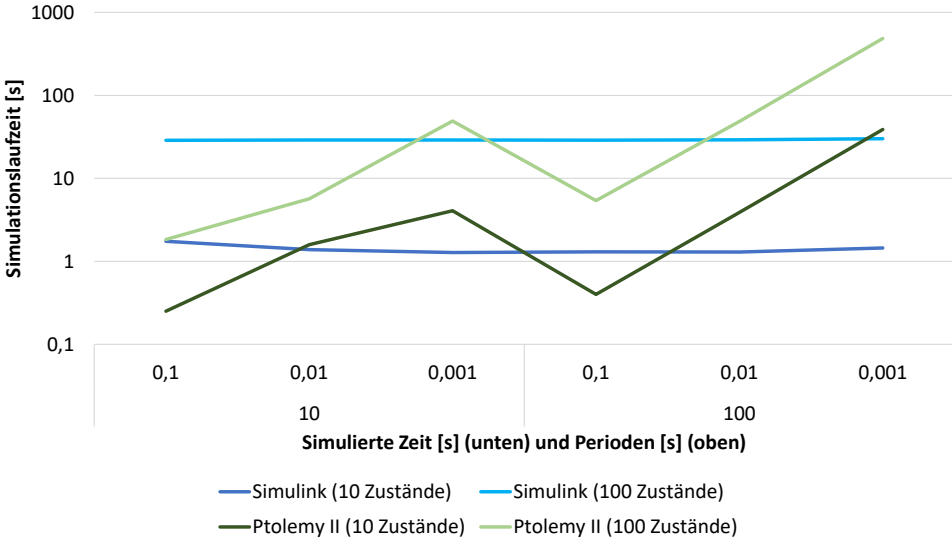



Abbildung 5.18.: Laufzeiten beim Benchmark „Zustände und Übergänge“.

Die Zeit ist logarithmisch aufgetragen. Es lässt sich erkennen, dass die Laufzeit bei Simulink kaum durch Erhöhung der simulierten Zeit oder Herabsetzen der Periodendauer beeinflusst

wird, da die blauen Kurven flach sind und eine nahezu konstante Laufzeit anzeigen. Allerdings beeinflusst die Anzahl der Zustände und die wiederum davon abhängige Anzahl der Ein-, Aus- und Übergänge die Laufzeit. Dies ist daran zu erkennen, dass die Kurve **Simulink (100 Zustände)** über der Kurve **Simulink (10 Zustände)** liegt. Eine stichprobenartige Analyse mit dem Simulink-Profiler ergab, dass zwischen 80 und 90 % der Simulationslaufzeit für das Kompilieren (Umwandlung in S-Funktionen) verbraucht wird und etwa nur 5 % für die eigentliche Simulation. Mit diesem Wissen würden sich bei Modellen mit 100 Zuständen ähnliche Werte für die reinen Simulationslaufzeiten ergeben, wie bei Modellen mit 10 Zuständen.
Bei Ptolemy II sieht das im Gegenzug anders aus. Hier steigt die Simulationslaufzeit sowohl mit kleiner werdender Periode als auch mit steigender Modellgröße, d.h. mit der Anzahl der Zustände. Dies ist an der quasi-linearen Steigung der grünen Kurven, jeweils innerhalb einer simulierten Zeit, zu erkennen. Ein Profiling der Ausführung mit VisualVM hat ergeben, dass der Großteil der Ausführungszeit dabei von der Funktion `generateParseTree()` verursacht wird. Für einen Zustand und dessen Verfeinerungen werden für jeden Zeitschritt in der Simulation *abstrakte Syntaxbäume (Abstract Syntax Trees (ASTs))* konstruiert, um die Bedingungen der Zustandsübergänge zu evaluieren. Ändert sich der Zustand in einer Verfeinerung, so werden die Zustandsübergänge der hierarchisch höher gelegenen Zustände erneut evaluiert. Dieses Vorgehen ermöglicht Modelländerungen zur Laufzeit auf Kosten der Ausführungszeit.
Bei kleineren Modellen und weniger Zustandsübergängen pro Zeit ist Ptolemy II schneller als Simulink. Simulink ist dagegen bei größeren Modellen und einer höheren Anzahl an Zustandsübergängen pro Zeit schneller als Ptolemy II.

Die in Tabelle 5.6 dargestellten Generierungs- und Öffnungszeiten unterscheiden sich deutlich zu denen in vorausgegangenen Benchmarks. Es fällt auf, dass die Generierungszeit bei Simulink von 10 auf 100 Zustände um mehr als das 1138-Fache ansteigt. Ein Profiling des Generatorskripts zeigt, dass bei der Generierung von Stateflow-Entitäten eine andere Syntax als bei den Simulink-Entitäten aus der Standardbibliothek verwendet wird. So erfordern Stateflow-Entitäten Zwischenspeicherungen in Workspace-Variablen, was zu höheren Laufzeiten führt.
In Bezug auf die Öffnungszeiten lässt sich ferner erkennen, dass Simulink im Vergleich zu den vorausgegangenen Benchmarks deutlich langsamer beim Öffnen von Modellen mit Stateflow-Anteilen ist. So war Simulink bei den zeitdiskrekten Benchmarks im Durchschnitt etwa dreimal schneller. Bei Ptolemy II haben sich hingegen die Öffnungszeiten kaum verändert.

Tabelle 5.6.: Generierungs- und Öffnungszeiten in Abhängigkeit der Zustände beim Benchmark „Zustände und Übergänge“.

| Simulator | Simulink | | Ptolemy II | |
|---|---|---|---|---|
| Anzahl der Zustände | 10 | 100 | 10 | 100 |
| Generierungszeit [s] | 3,555417 | 4045,979 | 0,326 | 3,283 |
| Öffnungszeit [s] | 2,047 | 15,131 | 0,511 | 16,617 |

Beim Speicherbedarf lassen sich keine konkreten Zusammenhänge zur Modellgröße oder der Periode erkennen. Tabelle 5.7 zeigt jedoch, dass Simulink viel weniger Speicher als Ptolemy II benötigt und einen etwas höheren Varianzkoeffizienten hat. Dies spricht für eine bessere Ausnutzung der zur Verfügung stehenden Ressourcen bzw. eine höhere Adaptabilität.

Tabelle 5.7.: Speicherbedarf beim Benchmark „Zustände und Übergänge“.

| | Simulink | Ptolemy II |
|---|---|---|
| Mittelwert [Mb] | 50,040 | 683,169 |
| Standardabweichung [Mb] | 25,603 | 284,496 |
| Varianzkoeffizient | 0,512 | 0,416 |

**Hierarchiestufen und Transitionsart** Mit diesem Benchmark wird die Auswirkung der Hierarchiestufen und der Transitionsarten auf die Performanz untersucht. Als Basis für das Modell dient das in Abbildung 5.9 dargestellte Modell „Hierarachie“. Es werden ferner zwei Transitionsarten bzw. Trigger unterschieden:

1. **Zeitlogik:** Für alle Übergangsbedingungen $B_n$ wird eine feste Zeitperiode verwendet.
2. **Zufallsgenerator:** Am Eingang des Zustandsautomaten wird ein Ereignisgenerator angeschlossen, der in einer festen Zeitperiode eine Zufallszahl $Z \in \mathbb{N}$ zwischen 0 und 1 erzeugt. Für alle Übergangsbedingungen $B_n$ wird der *Boolesche Ausdruck* $Z == 1$ gesetzt. Ist er wahr, so werden alle Übergänge gleichzeitig ausgelöst.

Abbildung 5.19 zeigt die Simulationslaufzeiten von Ptolemy II und Simulink in Abhängigkeit

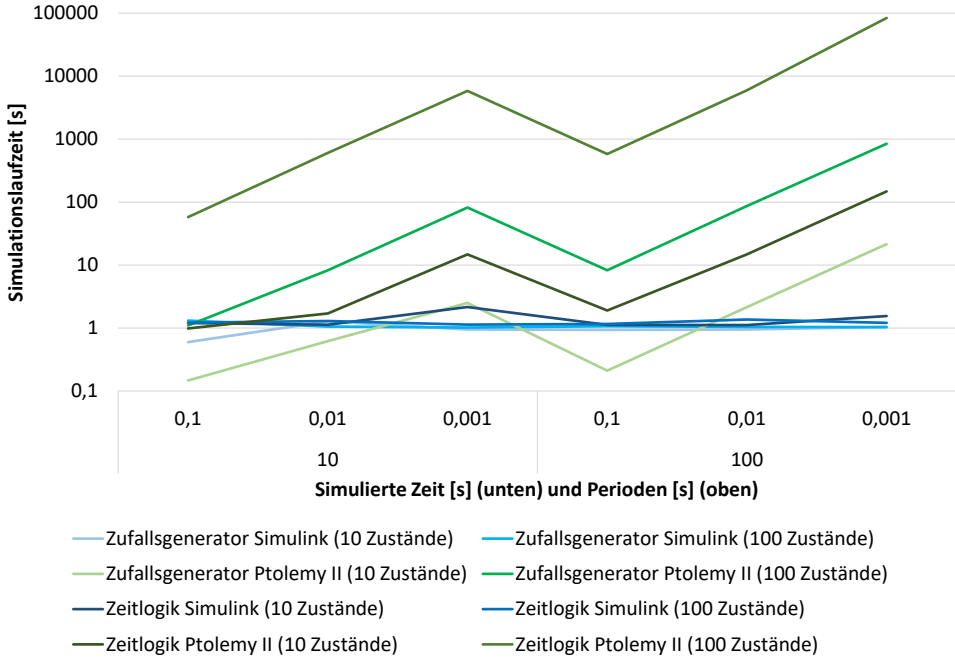



Abbildung 5.19.: Laufzeiten beim Benchmark „Hierarchiestufen und Transitionsart“.

der Hierarchiestufen (**Zustände**), des Triggers (**Zeitlogik/Zufallsgenerator**), der Transitionszeit (**Periode**) und der simulierten Zeit. Die Verläufe ähneln denen aus dem Benchmark „Zustände und Übergänge“. Es lässt sich erkennen, dass Periode und Anzahl der Hierarchiestufen die Laufzeit in Simulink wenig beeinflussen. Die Verwendung von Zeitlogik führt im Schnitt zu ca. 1,3-mal höheren Laufzeiten im Vergleich zum Zufallsgenerator. Bei Ptolemy II sind die Auswirkungen der Periode auf die Laufzeit an den linearen Anstiegen der grünen Kurven zu erkennen. Der vertikale Abstand zwischen den Kurven zeigt hingegen die Auswirkung von der Anzahl der Hierarchiestufen und der Triggerwahl. Hier führt die Verwendung von Zeitlogik zu 39-fach höheren Laufzeiten im Vergleich zur Verwendung des Zufallsgenerators.

Abbildung 5.20 zeigt den Speicherverbrauch von Ptolemy II und Simulink in Abhängigkeit der Hierarchiestufen (**Zustände**), des Triggers (**Zeitlogik/Zufallsgenerator**), der Transitionszeit (**Periode**) und der simulierten Zeit. Es lässt sich daraus erkennen, dass Zeitlogik den Speicherbedarf bei Simulink durchschnittlich um einen Faktor von ca. 1,65 und bei Ptolemy um einen Faktor von ca. 1,33 erhöht. Insgesamt ist der Speicherbedarf von Ptolemy II ca. 13,5-mal höher als der von Simulink. Bei beiden Simulatoren zeigt weder eine Erhöhung der Periode noch eine Erhöhung der Anzahl der Hierarchiestufen eine Auswirkung auf den Speicherverbrauch.

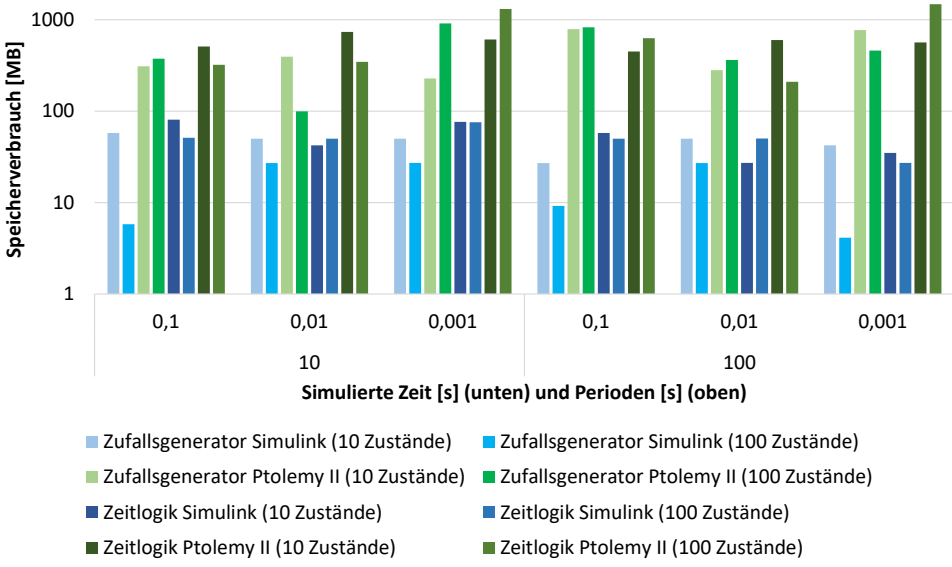



Abbildung 5.20.: Speicherbedarf beim Benchmark „Hierarchiestufen und Transitionsart“.

Die Generierungs- und Öffnungszeiten fallen beim Verwenden von Zeitlogik bei beiden Simulatoren geringer als bei einer Verwendung von Zufallsgeneratoren aus. Für Generierungs- und Öffnungszeiten zusammen beträgt der Mittelwert des Geschwindigkeitszuwachses bei Simulink ca. 19%, während es bei Ptolemy II etwa 171% sind.

**Parallele Zustände** Mit diesem Benchmark wird die Auswirkung der Anzahl von parallel ausgeführten Zuständen auf die Performanz untersucht. Als Basis für das Modell dient das in Abbildung 5.9 dargestellte Modell „Parallelität“. Für die Übergangsbedingungen $B_n$ wird Zeitlogik mit festen Zeitperioden verwendet. Die Zeitperioden sind für alle $B_n$ gleich, sodass alle Übergänge zeitgleich ausgelöst werden.

Abbildung 5.21 zeigt die Simulationslaufzeit von Ptolemy II und Simulink in Abhängigkeit der Anzahl von parallel ausgeführten **Zuständen**, der Transitionszeit (**Periode**) und der simulierten Zeit. Die Simulationslaufzeiten haben einen ähnlichen Kurvenverlauf, wie beim Benchmark „Hierarchiestufen und Transitionsart“. Bei Simulink sind die Kurvenverläufe (blau) erneut annähernd konstant, während die zu Ptolemy II gehörigen Kurven (grün) sowohl mit der Periode als auch mit der Anzahl parallel ausgeführter Zustände steigen. Der Laufzeitunterschied zwischen 10 und 100 parallel ausgeführten Zuständen ist am vertikalen Abstand der Kurven des jeweiligen Simulators zu erkennen. Bei Simulink fällt er im Vergleich zum Laufzeitunterschied zwischen 10 und 100 Hierarchiestufen beim Benchmark „Hierarchiestufen und Transitionsart“ größer aus. Das zeigt ein Vergleich der Kurve **Simulink (100 Zustände)** mit der Kurve **Zeitlogik Simulink (100 Zustände)** aus Abbildung 5.19. Für Simulink ist damit bei Modellen mit vielen Hierarchiestufen eine höhere Performanz zur erwarten, als bei Modellen mit vielen parallel ausgeführten Zuständen. Ptolemy II ist dagegen bei Modellen mit vielen parallel ausgeführten Zuständen um einige Größenordnungen schneller, als bei Modellen mit vielen Hierarchiestufen. Dies zeigt ein Vergleich der Kurve **Ptolemy II (100 Zustände)** aus Abbildung 5.21 mit der Kurve **Zeitlogik Ptolemy II (100 Zustände)** aus Abbildung 5.19.

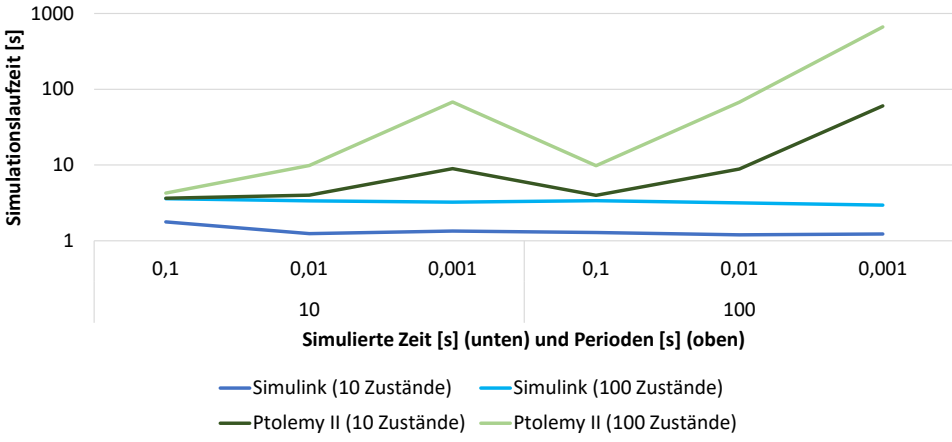



Abbildung 5.21.: Laufzeit beim Benchmark „Parallele Zustände“.

Abbildung 5.22 zeigt den Speicherverbrauch beider Simulatoren in Abhängigkeit der Anzahl von parallel ausgeführten **Zuständen**, der Transitionszeit (innerhalb der runden Klammern in der Legende) und der simulierten Zeit (**10 s/100 s** in der Legende). An den Kategorien **Simulink** und **Ptolemy II** ist zu erkennen, dass Ptolemy II allgemein einen größeren Speicherbedarf

hat. Bei Simulink kann eine Steigerung der Anzahl parallel ausgeführter Zustände bei längeren Transitionszeiten zur Reduktion des Speicherverbrauchs führen. Für Ptolemy II gilt dies unabhängig von der gewählten Transitionszeit.

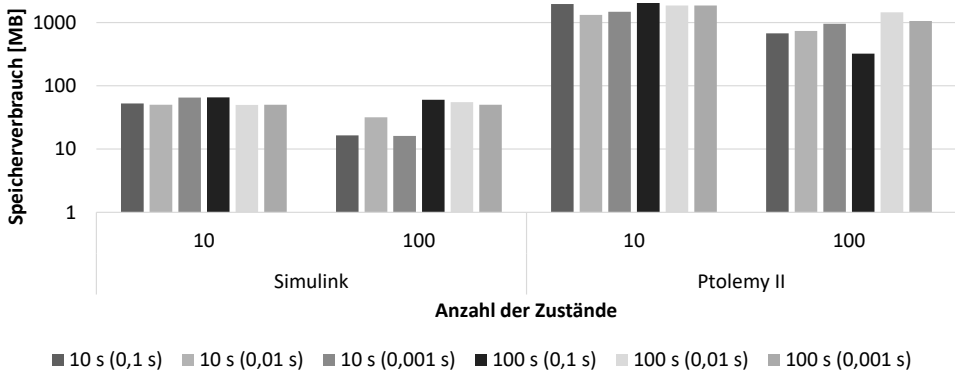



Abbildung 5.22.: Speicherverbrauch beim Benchmark „Parallele Zustände“.

Die Generierungszeit für 100 parallele Zustände liegt bei Simulink nun bei 9,3 s. Obwohl die Algorithmen in den Generierungsskripten ähnlich sind, ist diese Generierungszeit deutlich geringer als die für die Generierung von 100 hierarchischen Verfeinerungen benötigte Zeit von ca. 155 s im Benchmark „Hierarchiestufen und Transitionsart“. Der Unterschied liegt im Übergabeparameter des Befehls `Stateflow.State(parentChart)`, der zur Erzeugung eines neuen Charts verwendet wird. Bei der hierarchischen Verfeinerung wird die Variable `parentChart` zur Speicherung des Eltern-Charts in einer Schleife immer wieder durch ein zuvor neu erzeugtes Chart ersetzt. Im Gegenzug bleibt `parentChart` bei der Erzeugung der parallelen Charts immer gleich.

**Fazit und Deutungen** Die Benchmarks zeigen, dass Simulink annähernd konstante Simulationslaufzeiten in Abhängigkeit von der Anzahl der simulierten Zeit und der Anzahl an Zuständen aufweist. Dabei nimmt insbesondere die Kompilierzeit den größten Anteil an der Laufzeit ein. Außerdem wurde gezeigt, dass die Performanz von Stateflow bei der Generierung von Charts gegenüber Ptolemy II deutlich geringer ist. Ptolemy II weist dagegen einen Anstieg der Laufzeit in Abhängigkeit der Anzahl der simulierten Zeit und der Anzahl an Zuständen auf. Dies liegt einerseits an der Konstruktion der internen Struktur von **ModalModels** (siehe Abbildung 5.23) und andererseits an dem für das Evaluieren der Zustandsübergänge verwendeten Algorithmus. Für jede Verfeinerung (**Refinement**) eines Zustandes (**State**$_i$) mit einem Zustandsautomaten (**FSM Actor** $_{Ref}$) wird in Ptolemy II ein eigenes, für den Nutzer nicht sichtbares, **ModalModel** mit einem eigenen **FSM Director** für die interne Koordination der Ausführung erstellt. In Abbildung 5.23 ist **State**$_1$ beispielhaft als **ModalModel - Refinement: State**$_1$ verfeinert. Die gesamten Ein- und Ausgänge (▷/□) der Verfeinerungen (grün und blau gefüllt) sowie des Zustandsautomaten auf der obersten Ebene **FSM Actor** $_{Top}$ sind eine Ko-

pie der Ein- und Ausgänge auf der obersten Ebene (rot gefüllt). Dadurch vergrößert sich die interne Struktur des Modells mit jeder Verfeinerung und es entsteht eine Kommunikations- und Synchronisationslast, die zu einem höheren Ressourcenverbrauch führt. Beim Evaluieren der Übergänge wird dann für jeden aktiven Zustand ein AST gebaut, und rekursiv evaluiert. Für die Laufzeit von Ptolemy II ist damit die Anzahl der zu überprüfenden Bedingungen ausschlaggebend. Des Weiteren ist der Speicherverbrauch von Ptolemy II generell höher.

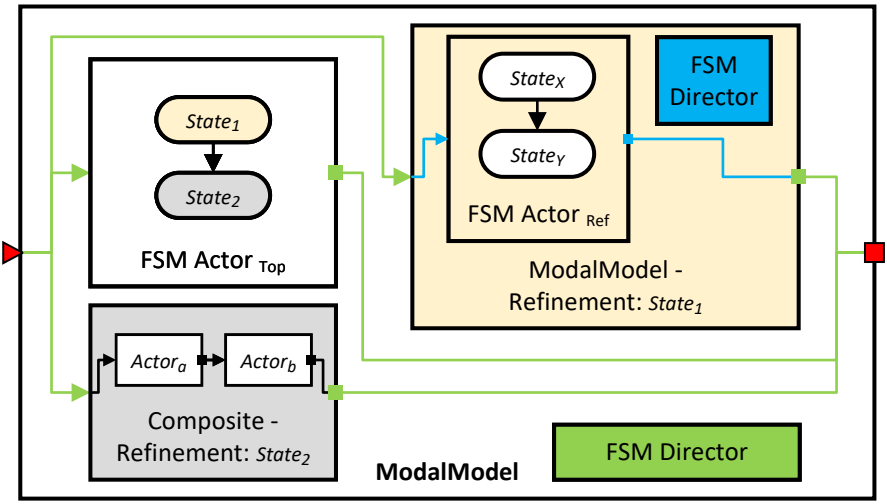



Abbildung 5.23.: Interner Aufbau eines ModalModels mit Verfeinerungen.

Aus den Beobachtungen lassen sich ferner folgende Regeln ableiten:

- Bei Simulink sollte eine häufige Generierung von Modellen durch Wiederverwendung mit Parametrisierung vermieden werden.
- Bei der Generierung mit Simulink sollte ein häufiges Schreiben und Lesen von Workspace--Variablen vermieden werden.
- Zeitlogik sollte bei beiden Simulatoren nur bedingt eingesetzt werden.
- Bei Ptolemy II sollte die Anzahl an zu evaluierenden Bedingungen pro Zeitschritt klein gehalten werden. Insbesondere sollte auf viele Verfeinerungen verzichtet werden.

## 5.3. Schlussfolgerungen

In Bezug auf Forschungsfrage 1 wurden u.a. die Fähigkeit zur hybriden Simulation, Performanz, Wartbarkeit, Erweiterbarkeit, Robustheit, Unterstützung und die Fähigkeit zur Code-Generierung als Auswahlkriterien festgelegt und zur Vorauswahl verwendet. Anhand von synthetischen Benchmarks wurde ferner gezeigt, dass Ptolemy II vor allem bei größeren Mo-

dellen und längeren Laufzeiten schlecht skaliert. Der größte Schwachpunkt dabei ist die zeitkontinuierliche MAE, welche bei E/E-Architektursimulationen vor allem für elektrische Verbindungen, aber auch für analoge Signale benötigt wird. Es wurde gezeigt, dass Simulink hier eine bessere Skalierbarkeit aufweist. Auch bei den Benchmarks der anderen MAEs wurde deutlich, dass Ptolemy II algorithmische Defizite bei der zeitdiskreten Berechnung und in der Evaluation von Zustandsübergängen hat. Ptolemy II erlaubt außerdem nur eine interpretierte Modellausführung, die zwar Flexibilität in Form von Modelländerungen zur Laufzeit erlaubt, aber keine automatische Optimierung eines kompilierten Codes. Im Folgenden soll daher eine geeignete Abbildung von PREEvision nach Simulink gefunden werden, die sowohl den Stand der Technik, als auch die Erkenntnisse aus den vorausgegangenen Benchmarks einschließt.

# 6. E/E-Architekturmodellbasierte Simulation

In diesem Kapitel werden zunächst Methoden nach dem Stand der Technik vorgestellt, welche die Skalierbarkeit von generierten Simulationsmodellen erhöhen. Dabei werden auch die Ergebnisse aus den in Unterabschnitt 5.2.2 durchgeführten Benchmarks aufgegriffen. Die Methoden werden bei der anschließenden Konzeption einer Abbildung von PREEvision-Architekturmodellen zu Simulink-Simulationsmodellen beachtet. Zum Schluss wird die konkrete Implementierung beschrieben.

## 6.1. Verwendete Methoden zur Steigerung der Skalierbarkeit

Im Folgenden werden Methoden mit Bezug auf die Skalierbarkeit der Simulation vorgestellt. Sie sollen bei der Generierung von Simulink-Simulationsmodellen aus PREEvision-Architekturmodellen beachtet werden. Es fließen insbesondere die in Kapitel 3 beschriebenen Methoden, entsprechend dem Stand der Technik, mit ein. Außerdem werden die Ergebnisse aus den in Unterabschnitt 5.2.2 durchgeführten Benchmarks aufgegriffen.

### 6.1.1. Templatebasierte Modellgenerierung

Mit den in Unterabschnitt 5.2.2 durchgeführten Benchmarks wurde passend zu den in Unterabschnitt 4.1.2 beschriebenen Einflussparametern deutlich, in welchem Umfang Simulationslaufzeiten und Generierungszeiten mit der Modellgröße steigen. Es lässt sich daraus schließen, dass die Generierung von größeren Modellen, sofern möglich, vermieden werden sollte. Dies lässt sich bspw. durch Wiederverwendung bereits generierter Modelle realisieren. Anstatt etwa für jede unterschiedliche Periodendauer ein eigenes Modell zu generieren, kann ein Modellparameter für die Periodendauer genutzt werden. Vor allem für die physikalischen HWCs in PREEvision bieten sich generische und parametrisierbare Simulationskomponenten an. Dies liegt an ihrem verhältnismäßig niedrigen Abstraktionsgrad. Mit parametrisierbaren Simulationskomponenten bzw. Modulen sinkt zudem der Modellierungsaufwand. Ferner kann mit ihnen die Genauigkeit und die Performanz der Simulation gesteigert werden, wenn sie vor ihrer Verwendung evaluiert bzw. optimiert wurden. Im Folgenden werden parametrisierbare, evaluierte und optimierte Simulationsmodellkomponenten, sowie deren Stellvertreter als *Templates* bezeichnet.

Wie in Bucher [43] gezeigt, bietet eine automatisierte Simulationsmodellgenerierung die Möglichkeit einer modellbasierten Evaluation von verschiedenen Varianten und erleichtert dabei die Entscheidungen von E/E-Architekten in frühen Entwicklungsphasen. Durch einen integrierten Ansatz fallen zudem fehleranfällige Im- und Exporte weg, wodurch die Datenkonsis-

tenz gewährleistet wird. Wenn sich das Ursprungsmodell unabhängig von einem exportierten Modell ändert, kann bei dem erneuten Import des zuvor exportierten Modells nämlich nicht immer eine konsistente Verschmelzung durchgeführt werden. Es könnten z.B. Daten verloren gehen oder doppelte Artefakte auftauchen. Eine Generierung erhöht ferner die Anwendbarkeit, da die Modellierungsfreiheit eingeschränkt wird. Die Architektur und die verfügbaren Modellartefakte geben dann vor, welche funktionalen Entitäten in der Simulation benötigt werden und wie diese miteinander verknüpft sind. In [43] wird das Verhalten der Modellartefakte entweder in Form von Zustandsautomaten auf der LA und der HA oder auf einer prototypischen BLA mit Ptolemy II-Stellvertreterblöcken modelliert. Die Verantwortung in der Verhaltensmodellierung liegt hier allerdings bei den E/E-Architekten. In der Regel verfügen diese aber nicht über die nötige Erfahrung in der Erstellung von Simulationsmodellen, wie sie Simulationsexperten haben [61, 104, 216, 311]. Für eine schnelle Entscheidungsfindung ist es außerdem wichtig relevante und anpassbare Verhaltensmodelle frühzeitig verfügbar zu haben. Aus diesem Grund, zur Einschränkung der Freiheitsgrade in der Verhaltensmodellierung und wegen der Validierbarkeit, sollen Templates für die Abbildung verwendet werden.

Ein Template soll den Artefakten auf der LA, der SA und der HA zuweisbar sein. Eine manuelle Verhaltensmodellierung soll nicht ausgeschlossen werden. Auf der HA lassen sich insbesondere Templates ohne eine explizite Zuweisung durch den E/E-Architekten anwenden, da die dort verwendeten Artefakte physikalisch und damit im Vergleich zur Software weniger abstrakt sind. Durch diese Beschaffenheit lassen sich gerade Artefakte zur Energieversorgung, wie Batterien und Kabel, oder auch Signalwandler, wie Sensoren und Aktoren, parametrisiert abbilden. Die Parametrisierung kann z.B. in Abhängigkeit der zugrundeliegenden Technologie oder der elektrischen Eigenschaften erfolgen. Bei Energiespeichern kann es sich z.B. um die Technologien Blei, NiCD, NiMH oder Li-Ionen handeln. Sie lassen sich und anhand von Energiedichte, Spannung oder spezifischer Energie parametrisieren [31]. Ebenso können Sensoren anhand ihres Energiewandlungsprinzips [221] und Motoren nach ihrer Bestromungsart [31] klassifiziert und parametrisiert werden. Diese Überlegungen führen neben weiteren Aspekten auch zu der in Abschnitt 6.3 beschriebenen HW-zentrierten Abbildung.

### 6.1.2. Reaktive Szenarien

Aus den Benchmarks in Unterabschnitt 5.2.2 wurde ebenso passend zu den in Kapitel 4 beschriebenen Einflussparametern deutlich, dass die Modellgröße generell gering gehalten werden sollte. Außerdem sollte eine möglichst parallele Modellstruktur für den maximalen Speedup (vgl. Unterabschnitt 2.3.2) vorliegen.

In Unterabschnitt 2.2.1 ist beschrieben, wie die Anzahl an Testfällen durch eine Zusammenfassung innerhalb von *Szenarien* eingeschränkt wird. Szenarien lassen sich ferner zur Reduktion der Modellgröße verwenden, wenn man aus ihren Schnittstellen das SOI ableitet. So wird beispielsweise in einem Szenario „Aufladen“ für *batterieelektrische Fahrzeuge (BEFs)* sowohl die Ladestation als auch die Batterie, das Batteriemanagement und ggf. die Kühlung für eine Simulation benötigt, mit der man den Ladestromverlauf untersuchen möchte. Da die restlichen Komponenten der E/E-Architektur beim Laden i.d.R. deaktiviert sind und damit für das Ergebnis nicht relevant sind, müssen sie nicht simuliert werden. Der Ausgang „Zündungsplus

(Klemme 15)“, als Schnittstelle des Szenarios, kann z.B. implizit die benötigten Artefakte der E/E-Architektur für die Simulation vorgeben.

PREEvision bietet zur Modellierung von Verhalten UML-konforme FSM-Diagramme an. Unter Beachtung der in Unterabschnitt 3.4.3 beschriebenen Einschränkungen von UML und der Arbeit von Hejase [113], ist eine Modellierung von Szenarien damit möglich. Vorteile bei der Verwendung von FSMs sind eine anschauliche Bildung und visuelle Nachverfolgbarkeit von Szenarien zur Fehleranalyse (vgl. Anforderung 3) sowie die Möglichkeit textuell vorliegende Prüfszenarien mit sequenziellem Ablauf in FSMs zu übersetzen [P2]. Außerdem ist die Realisierung reaktiver Szenarien möglich, die auf das Verhalten der E/E-Architektur reagieren und ausgehende Stimuli entsprechend anpassen. Da Anwendungsfälle jedoch die Modellierung zeitlicher Abhängigkeiten erfordern können, muss in PREEvision zusätzlich eine entsprechende Syntax für Zeitlogik definiert und bei der Abbildung beachtet werden.

### 6.1.3. Hardware-zentrierte Abbildung

Mit den Erkenntnissen aus Unterabschnitt 3.1.1 wird die Anzahl der ECUs aufgrund von Domänenfusion und dem Einsatz von zentralen HPCs mit Zonenrechnern sinken. Im Gegenzug wird der Umfang der Software stark steigen. Die Anzahl der SWCs, die auf den ECUs ausgeführt werden, steigt also. Bei einer Abbildung, der die Struktur der HA zugrunde liegt, würden die Modelle auf der obersten Hierarchieebene entsprechend übersichtlich sein. Eine solche Abbildung wird im Folgenden als *Hardware-zentriert* bezeichnet. Da durch Gleichung 2.12 ein Speedup maßgeblich durch Parallelisierung erzielbar ist, eignet sich diese Struktur auch zur Realisierung einer parallelen oder verteilten Simulation. Hierbei könnte jede ECU einem anderen Rechenknoten oder Kern zugeteilt sein. Ein HW-zentriertes Simulationsmodell erlaubt zudem die Kopplung der Simulation mit realen oder virtuellen ECUs, da die entsprechenden physikalischen und logischen Signale bereitstehen. Überdies eignet sich die HW-zentrierte Abbildung für die Anwendung von Bedrohungsanalysen (siehe Unterabschnitt 3.1.4), da die Angriffspfade Teil der Simulation sind und entsprechende Angriffs- oder Missbrauchsszenarien durchgeführt werden können [P4]. Zusammen mit den HA-Komponenten fest zugewiesenen Templates lässt sich ferner eine Leitungssatzsimulation realisieren, die ohne eine spezifische Verhaltensmodellierung der Komponenten auskommt. Dies ist möglich, weil HA-Komponenten spezifischer als SA- oder LA-Komponenten sind und sich damit besser parametrisieren lassen (vgl. Unterabschnitt 6.1.1).

Es gilt zu beachten, dass Simulationsmodelle im Vergleich zu E/E-Architekturmodellen nicht in Ebenen mit unterschiedlichen Sichten eingeteilt sind. Daher muss eine Integration der E/E-Architekturebenen im Simulationsmodell erfolgen. In [43] wurde dies mit einem aspektorientierten, dekorativen Ansatz und einer quasi-parallelen, datenflussbasierten Ausführung logischer Funktionen gelöst, wobei es sich um eine LA-zentrierte Abbildung handelt (siehe Unterabschnitt 3.4.2). Ein äquivalenter aspektorientierter Modellierungsansatz steht in Simulink nicht zur Verfügung, weshalb eine Orientierung am Urbild, also der physikalischen E/E-Architektur, stattfindet. Auf den ECUs findet entsprechend dem Urbild die Integration der HA mit der SA statt. Hierbei ist jede SWC einer ECU zugeordnet. Die Ausführungsreihenfolge der SWCs kann dann über ein Scheduling festgelegt werden. Dies ermöglicht eine

Entkopplung der Ausführungsreihenfolge vom Datenfluss. Szenarien aus der LA werden ähnlich zu Signalgeneratoren und Messinstrumenten in der Realität integriert.

## 6.2. Prinzip der Abbildung von PREEvision nach Simulink

In diesem Abschnitt werden das Prinzip und die notwendigen Elemente einer E/E-Architekturmodellsimulation mit Simulink erläutert. Darauf aufbauend werden sowohl das Gesamtkonzept als auch die Teilkonzepte zur Modellierung und Integration von Umgebung, Szenarien sowie mechatronischen Aspekten vorgestellt.

### 6.2.1. Kontext und erforderliche Elemente der Simulation

Abbildung 6.1 zeigt beispielhaft die notwendigen Simulationselemente für die Validierung von FAS innerhalb einer E/E-Architektursimulation.

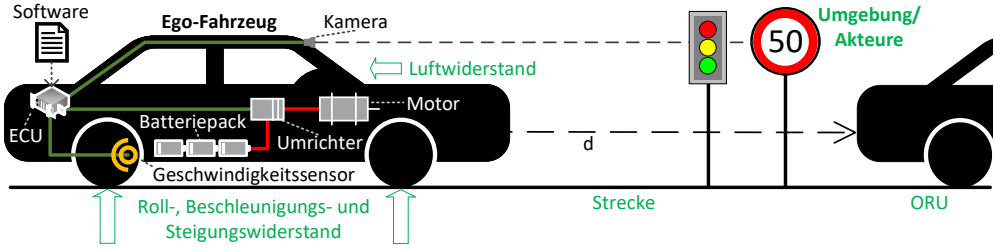



Abbildung 6.1.: Beispiel der notwendigen Elemente für eine Simulation von FAS.

Das **Ego-Fahrzeug** *(EF)* stellt den Container bzw. ein übergeordnetes, mechanisches System dar, in das die zu testende E/E-Architektur eingebettet ist. Auf dieses System wirken physikalische Kräfte ein, welche den zu überwindenden **Fahrwiderstand** (Luft-, Roll-, Beschleunigungs- und Steigungswiderstand) bestimmen. Neben dem EF sind in der Abbildung **Akteure** dargestellt. Sie sind, wie die physikalischen Kräfte, Bestandteil der **Umgebung** des EFs und Teil der Simulation bzw. des Szenarios. *Statische Akteure*, wie Schilder, ändern ihren Zustand innerhalb des Szenarios nicht. *Dynamische Akteure*, wie Ampeln oder weitere Verkehrsteilnehmer (*Other Road User* (***ORU***)), ändern hingegen ihren Zustand in Abhängigkeit des Testszenarios. Das EF und die dynamischen Akteure können sich gegenseitig beeinflussen. Zur Umgebung gehört des Weiteren die **Teststrecke** mit ihren spezifischen Eigenschaften, wie die Anzahl der Spuren, dem Streckenverlauf oder der Fahrbahnbeschaffenheit. Auf der Abbildung ist außerdem die vereinfachte E/E-Architektur eines BEFs zu sehen. Eine **Kamera** dient zur Objekterkennung, welche auf einer **ECU** mit entsprechender **Software** realisiert ist. Anhand der Kameradaten können etwa der Streckenverlauf für eine Querregelung wahrgenommen oder Verkehrsschilder erkannt werden. Letztgenannte können z.B. zur Festlegung der Sollgeschwindigkeit des EFs in einem Szenario verwendet werden. Zur Einregelung darauf steuert die ECU dann die

Schaltzeiten des **Umrichters** in Abhängigkeit des anliegenden Lastmoments, um die Zieldrehzahl (Sollwert) des **Motors** zu erreichen. Da für die Regelung ein Vergleichswert (Istwert) notwendig ist, wird die momentane Geschwindigkeit benötigt, die ein **Drehzahlsensor** liefert. Daraus kann schließlich der Istwert der Motordrehzahl bestimmt werden. Das Lastmoment ist direkt von den Fahrwiderständen (⇒) abhängig und nimmt mit steigender Geschwindigkeit zu. Somit sind neben der E/E-Architektur und im Einklang mit [303] noch zusätzlich die folgenden Elemente bei der Simulation zu berücksichtigen:

- Ego-Fahrzeug mit Fahrdynamik
- Testszenario
- Umgebung

### 6.2.2. Resultierendes Gesamtkonzept

Abbildung 6.2 zeigt das Gesamtkonzept der ebenenübergreifenden Abbildung von PREEvision zu Simulink nach [P4] auf dem höchsten Abstraktionsgrad.

Im oberen Teil der Abbildung sind schematisch die Komponenten und relevanten Ebenen als Minimalbeispiel eines PREEvision-E/E-Architekturmodells dargestellt. Der untere Teil zeigt schematisch ein entsprechend generiertes Simulink-Simulationsmodell auf der höchsten Hierarchieebene. Es fasst die Informationen aus dem PREEvision-Modell ebenenübergreifend zu einem integrierten Modell zusammen. Die Hauptstruktur basiert hierbei auf der **HA**. Zusätzlich werden Komponenten aus den anderen Ebenen hinzugefügt und integriert. Dadurch sind im Simulationsmodell sowohl ungerichtete physikalische Signale (**Konventionell** und **Versorgung**) als auch gerichtete logische Signale (**Umgebung** und **Bus**) vorhanden. Modellierte Bussysteme werden auf den Signalwegen zwischen den Entitäten integriert. [P4]

**SWCs** aus der **SA** werden als Teil der **ECU** gesehen, der sie zugeordnet sind [P4]. Die Integration von SA und HA in der ECU wird in Unterabschnitt 6.3.2 beschrieben.

Auf der **LA** befinden sich die zu ergänzenden **Umgebungskomponenten** (grau gefüllte Ellipsen), die für die Realisierung eines reaktiven Prüfstandes benötigt werden. Sie weisen spezifische logische Schnittstellen auf. Dabei ist zu berücksichtigen, dass die E/E-Architektur Komponenten enthält, die direkt oder indirekt über die Umgebung miteinander gekoppelt sind. Werden zeitlich änderbare Werte (**dynamische Attribute**) eines **Aktors** direkt von einem **Sensor** ausgelesen, so wird eine direkte Verbindung über eine Umgebungsschnittstelle modelliert. Dies ist z.B. bei der Position eines Stellmotors (Aktor) der Fall. In einer Simulation ist diese Position normalerweise ein dynamisches Attribut der Stellmotorkomponente. In der Realität wird zum Bestimmen der Position jedoch ein Sensor benötigt. Durch eine direkte Kopplung zwischen Aktor und Sensor mittels Umgebungssignal können dynamische Attribute des Aktors an einen stellvertretenden Sensor weitergegeben werden. Gibt es eine indirekte Kopplung zwischen einem Sensor und einem Aktor über die Fahrdynamik, dient die Umgebungskomponente **EF** als Schnittstelle zu den fahrdynamischen Kräften. So kann zum Beispiel das Lastmoment für einen Motor über die Umgebungskomponente EF und einen stellvertretenden Sensor bereitgestellt werden. Das EF kann außerdem seine Fahrzeugattribute und -zustände, wie z.B. seine aktuelle Geschwindigkeit, als Stimuli bereitstellen. [P4]

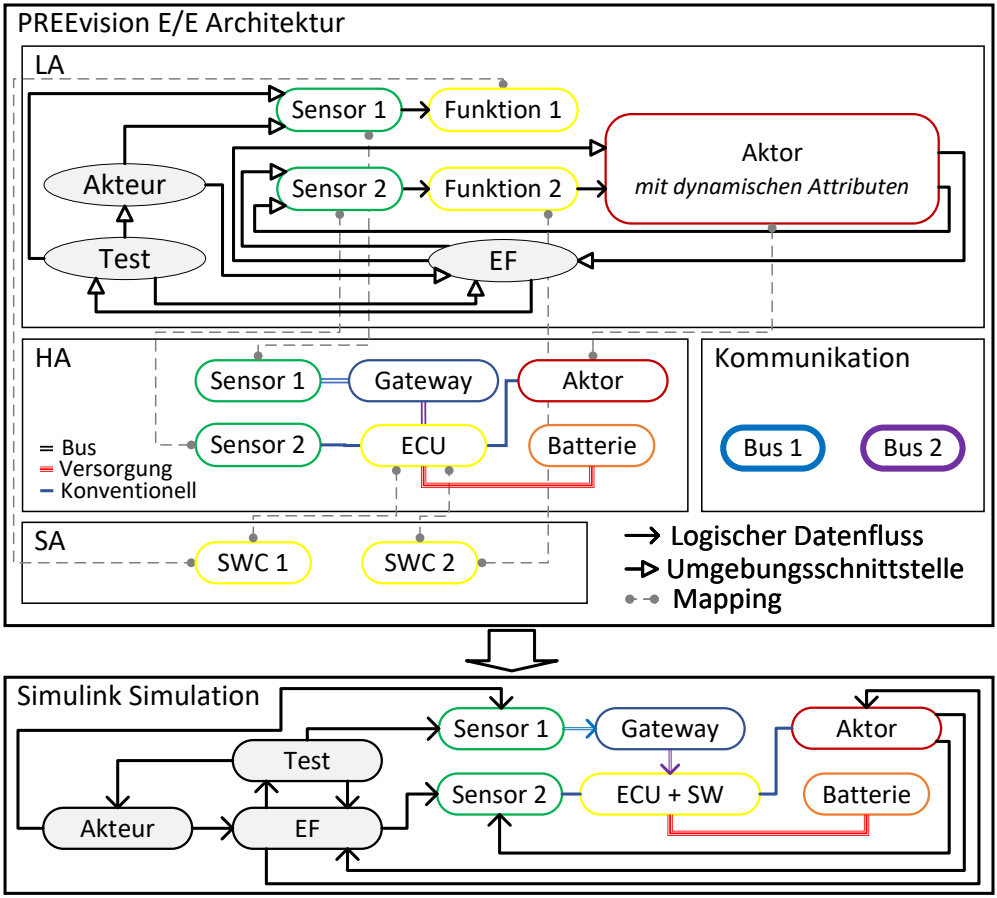



Abbildung 6.2.: Allgemeines Konzept der Abbildung von PREEvision nach Simulink (Quelle: Modifikation von [P4]).

Da das EF logische Steuersignale von den E/E-Komponenten entgegennehmen kann und ihnen im Gegenzug logische Sensordaten zur Verfügung stellt, entsteht eine geschlossene Schleife zwischen der E/E-Architektur und der physikalischen Umgebung. Sensoren und Aktoren sind dabei die Wandler zwischen logischen Umgebungssignalen und physikalischen E/E-Signalen. Insbesondere ist mit diesem Ansatz eine Simulation entsprechender Hardwarecharakteristiken und hardwaretypischer Fehler möglich. Zusätzlich kann das EF als Schnittstelle für eine Co-Simulation genutzt werden (vgl. Unterabschnitt 7.2.2). Der Aufbau des EF ist in Unterabschnitt 6.2.3 genauer beschrieben. [P4]

Stimuli können neben dem EF auch von **Akteuren** und dem Prüfstandmodell **Test** bereitgestellt werden. Akteure sind Teil der Umgebung, wie Ampeln oder ORUs. Von Akteuren aus-

gehende Stimuli können den E/E-Komponenten direkt oder über das EF zur Verfügung gestellt werden. Das Prüfstandmodell Test kapselt dagegen ein reaktives Testszenario, das Stimuli in Form logischer Signale an das EF oder an E/E-Komponenten liefert. Es können damit außerdem die für einen Test relevanten Zustände der E/E-Komponenten und deren Signale überwacht werden. Des Weiteren können damit Stimuli abhängig vom Zustand der E/E-Architektur angepasst werden, wodurch die besagte Reaktivität entsteht. Eine Modellierung als FSM ist in Unterabschnitt 6.2.4 beschrieben. [P4]

### 6.2.3. Das Konzept „Ego-Fahrzeug“ zur Kapselung fahrdynamischer Eigenschaften und als Schnittstelle zur Umgebung

Wie Abbildung 6.2 zeigt, dient das EF zur Kapselung der fahrdynamischen Eigenschaften. Es bildet die Schnittstelle zwischen dem E/E-System und der Umgebung bzw. dem Testszenario. Diese Schnittstelle sollte möglichst generisch und parametrisierbar sein. Daher werden die zwingend erforderlichen Ein- und Ausgänge sowie Parameter des EFs festgelegt. Hierzu werden die grundlegenden Fahraufgaben eines reellen Fahrers zur Quer- und Längsregelung des Fahrzeugs herangezogen: Beschleunigen, Bremsen, Lenken und Schalten. Sie lassen sich durch die Beschleunigung $\boldsymbol{a}$, den Lenkwinkel $\boldsymbol{\theta}$ und das Übersetzungsverhältnis $\boldsymbol{i}$ beschreiben. Da der reelle Fahrer einen Geschwindigkeits- und Richtungsregler darstellt, muss das EF seine Istgeschwindigkeit $\boldsymbol{v}$, seinen Ort $\boldsymbol{pos}$ und seine Orientierung $\boldsymbol{or}$ innerhalb des räumlichen Bezugssystems (z.B. Fahrstrecke) zur Verfügung stellen. Des Weiteren werden die Kräfte, welche auf die an der Simulation beteiligten elektromechanischen Komponenten wirken, benötigt.

Abbildung 6.3 zeigt die Standardlösung für einen EF-Antriebsstrang nach [154] mit einem zentralen Motor **M**, dessen Drehbewegung über ein Getriebe $\boldsymbol{G}$ übersetzt und mittels einem Differenzialgetriebe $\boldsymbol{DG}$ an die Antriebsräder $\boldsymbol{R}$ verteilt wird.

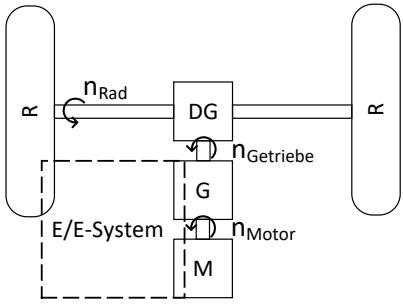



Abbildung 6.3.: Antriebsstrang des EF und Schnittstellen zum E/E-System (Quelle: eigene Darstellung nach [154]).

Je nach Motortyp (Verbrenner- oder Elektromotor), ist der Motor Teil des **E/E-Systems** oder innerhalb einer Regelschleife daran angebunden. Bei Verbrennungsmotoren können z.B. die Einspritzzeiten mit elektrischen Einspritzventilen elektronisch gesteuert sein, während bei bürs-

tenlosen Gleichstrommotoren die Schaltimpulse der Transistoren zur Steuerung genutzt werden. Das E/E-System kann des Weiteren über Getriebesensoren mit dem Getriebe in Kontakt stehen, um dessen Drehzahl für die Auswahl des Schaltzeitpunkts und des Ganges zu nutzen. Reifendruck- und Drehzahlsensoren sind wiederum an den Rädern bzw. Achsschenkeln zu finden und liefern dem E/E-System Informationen zur Berechnung der Geschwindigkeit.

Auf die Antriebswelle wirkt der Antriebskraft des Motors eine Kraft entgegen, die sich einerseits durch Verluste (Reibung) und andererseits aus den in Abbildung 6.1 gezeigten Fahrwiderständen ergibt. Die Summe davon $F_{FW}$ ist genau die Kraft, die das EF bei seiner Bewegung überwinden muss. Nach [33, S1] ergeben sich die Gleichungen 6.1 bis 6.6:

$$F_{FW} = F_L + F_R + F_S + F_B \tag{6.1}$$

$F_L$ ist der Luftwiderstand, der mit der Geschwindigkeit $v_{EF}$ $[\frac{m}{s}]$ des EFs steigt und von der Stirnfläche $A$ $[m^2]$ des EFs, der Luftdichte $\rho$ [Pa], dem Luftwiderstandsbeiwert $cw$ und der relativen Anströmgeschwindigkeit $v_A$ $[\frac{m}{s}]$ abhängt:

$$F_L(v) = \rho \cdot A \cdot cw \cdot \frac{v_A^2}{2} \tag{6.2}$$

$v_A$ beachtet neben $v_{EF}$ sowohl Gegenwind $v_G$ $[\frac{m}{s}]$ als auch Rückenwind $v_R$ $[\frac{m}{s}]$:

$$v_A = v_{EF} + v_G - v_R \tag{6.3}$$

$F_R$ bezeichnet den Rollwiderstand. Er setzt sich aus dem Rollwiderstandsbeiwert $f_r$, der Fahrzeugmasse $m_{EF}$ [kg], der Masse der Zuladung $m_{Zu}$ [kg], der Erdbeschleunigung $g$ mit 9,81 $\frac{m}{s^2}$ und dem Steigungswinkel der Fahrbahn $\alpha$ [rad] zusammen:

$$F_R = f_r \cdot (m_{EF} + m_{Zu}) \cdot g \cdot cos(\alpha) \tag{6.4}$$

$f_r$ hängt dabei vom Reifendruck, der Reifenkonstruktion und der Beschaffenheit der Straße ab. $F_S$ bezeichnet den Steigungswiderstand:

$$F_S = (m_{EF} + m_{Zu}) \cdot g \cdot sin(\alpha) \tag{6.5}$$

$F_B$ ist der Beschleunigungswiderstand. Er hängt vom Massenfaktor $e_i$ (>1), der die Trägheitsmomente des Antriebsstrangs beschreibt, und der Beschleunigung $a$ $[\frac{m}{s^2}]$ des EFs ab:

$$F_B = (e_i \cdot m_{EF} + m_{Zu}) \cdot a \tag{6.6}$$

Da die Istgeschwindigkeit des EFs von der Raddrehzahl $n_{Rad}$ $[min^{-1}]$ und diese wiederum von der Motordrehzahl $n_{Motor}$ $[min^{-1}]$ abhängt, muss die auf die Antriebswelle wirkende Kraft in Form des Drehmoments $M_{Motor}$ $[\frac{N}{m}]$ berücksichtigt werden. Aus der allgemeinen Formel $M = F \cdot \frac{d}{2}$ ergibt sich unter Berücksichtigung der Übersetzung:

$$M_{Motor} = \frac{F_{FW} \cdot r_{Rad}}{i_G \cdot i_{DG}} \tag{6.7}$$

$r_{Rad}$ ist der Radius der Reifen und $i_i$ beschreibt das Übersetzungsverhältnis am Getriebe G bzw. dem Differenzialgetriebe DG. Das Lastmoment $M_{Motor}$ wird in einer Simulation von der Motorkomponente benötigt. In Abhängigkeit von $M_{Motor}$ kann die entsprechende Motordrehzahl $n_{Motor}$ berechnet werden und daraus nach [120] letztlich $v_{EF}$:

$$v_{EF}[km/h] = \frac{2 \cdot \pi \cdot r_{Rad} \cdot n_{Rad} \cdot 60}{1000} = \frac{2 \cdot \pi \cdot r_{Rad} \cdot n_{Motor} \cdot 60}{1000 \cdot i_G \cdot i_{DG}} \tag{6.8}$$

Die resultierenden Ein- und Ausgänge sowie Parameter des EF-Modells als Ausgangspunkt für ein Template sind in Tabelle 6.1 dargestellt.

Tabelle 6.1.: Schnittstellen und Parameter des EFs.

| Eingänge | Konstante Fahrzeugparameter | Ausgänge |
|---|---|---|
| $a \mid n_{Motor} \mid i_G \mid \alpha$ | $m_{EF} \mid m_{zu} \mid \rho \mid r_{Rad} \mid A \mid cw \mid g \mid f_r \mid e_i \mid i_{DG}$ | $v_{EF} \mid M_{Motor} \mid pos \mid or$ |

Die Modellierung in PREEvision erfolgt als Umgebungsfunktion, die über die *Template And Layer Integration Architecture (TALIA)* mit dem Template verknüpft ist (siehe Unterabschnitt 6.3.3). Je nach Simulation und Schwerpunkt muss EF ggf. bezüglich seiner Schnittstellen angepasst oder erweitert werden.

### 6.2.4. Das Konzept „Testszenario“ zur Generierung von Stimuli und der Evaluation des E/E-Systems

Da das Testszenario kein Teil der E/E-Architektur ist und Teile der Umgebung simuliert, wird es in PREEvision auf der LA als Umgebungsfunktion mit speziellen *Umgebungsports* modelliert. Es entspricht der Komponente *Test* aus Unterabschnitt 6.2.2. In [P2] und [P4] wurde zur Modellierung von Testszenarien ein Konzept entwickelt und evaluiert. Hierbei kommt eine FSM zum Einsatz, wobei [212] und [113] als Inspiration dienten. Das Konzept ist in angepasster Form in Abbildung 6.4 dargestellt. Auf der rechten Seite ist die **Szenario-FSM** zu sehen, wodurch sequenziell ablaufende Szenarien modelliert werden können. Generell ist das Konzept nicht auf einen sequenziellen Verlauf beschränkt. Der Zustand **Startkonditionen** ist der Einstiegspunkt eines Testszenarios. Hier wird die Einhaltung vorgegebener Randbedingungen überprüft. Wenn zum Beispiel die Anfangsgeschwindigkeit $v_{t_0}$ des EFs 50 km/h betragen soll, kann dies mit einer Übergangsbedingung überprüft werden. $v_{t_0} = 50\,km/h$ wäre dann als **Evaluationsparameter** definiert, mit dem die Momentangeschwindigkeit $v_{ist}$ verglichen wird. Ist die Übergangsbedingung $v_{ist} >= v_{t_0}$ erfüllt, so erfolgt der Übergang zu **Testphase 1**. $v_{ist}$ kann vom EF als logisches Datum über die **Eingangsvariablen** des Zustandsautomaten bereitgestellt werden. Evaluationsparameter werden in Form **lokaler Variablen** gespeichert. Lokale Variablen speichern darüber hinaus auch Zwischenstände oder Werte, die als dyna-

mische Evaluationsparameter dienen. Über Bedingungen werden dann die einzelnen **Testphasen** durchlaufen. Bedingungen schließen Booleschen Vergleiche von Eingangsvariablen und Evaluationsparametern sowie Zeitlogik ein. So können nicht-funktionale Anforderungen überprüft oder Timeouts für Abbruchbedingungen bzw. Gegenbedingungen definiert werden. Werden alle Bedingungen zwischen den Testphasen erfüllt, so ist der Endzustand der Testmetrik **„Bestanden“**. Wird dagegen der Endzustand **„Fehlgeschlagen“** erreicht, so wurde eine Übergangsbedingung verletzt. Durch ein Monitoring der Ausgangs- und lokalen Variablen sowie einer Nachverfolgung des Übergangsverlaufs ist es möglich verletzte Bedingungen zu isolieren. **Ausgangsvariablen** werden in Abhängigkeit von der Testphase als Reaktion auf die Eingangsvariablen gesetzt. Zur Verfeinerung der Testphasenzustände bzw. deren Ausgaben können **Sub-Modelle** oder Aktionen verwendet werden. Ausgangsvariablen werden ferner zur Ausgabe von Stimuli in Form von Steuer- und Solldaten für das EF bzw. das E/E-System genutzt (siehe Unterabschnitt 6.2.3). **Steuerdaten** beziehen sich explizit auf die Steuerung des EFs durch einen virtuellen Testfahrer. Hierzu gehören bspw. der Lenkwinkel oder die Gaspedalstellung. Steuerdaten sind eine Teilmenge der **Solldaten**, welche den Sollzustand des EFs vorgeben. Das können z.B. die Sollgeschwindigkeit oder der Sollzustand einer Fahrfunktion sein. Weitere Sensordaten, die nicht vom Zustand des EFs und dessen Fahrdynamik abhängen, können dem E/E-System auch direkt als **logische Sensordaten** übergeben werden. Wie in Abbildung 6.2 dargestellt, soll dies über eine Umgebungsschnittstelle erfolgen. Das Sensormodell selbst bereitet diese logischen Daten dann je nach der gewählten Technologie auf, um entsprechende Hardwarecharakteristiken abzubilden. Über Umgebungsausgänge an den Aktoren können wiederum Werte für die Szenario-FSM bereitgestellt werden. So entsteht eine geschlossene Schleife zwischen Testszenario, EF und dem E/E-System. [P2, P4]

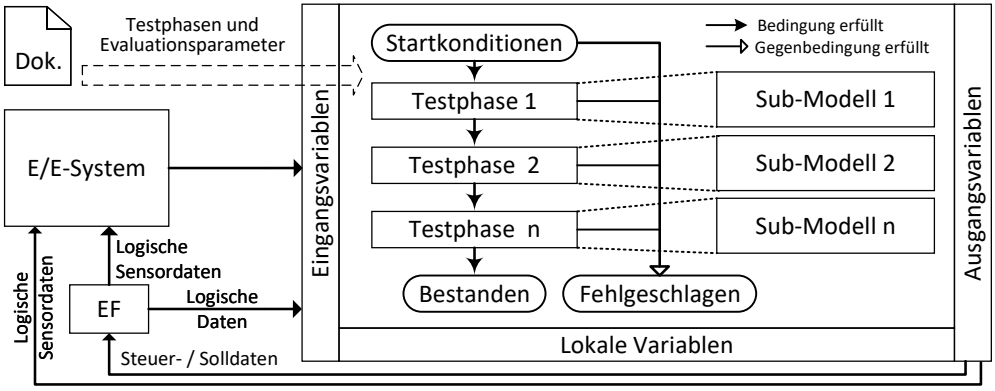



Abbildung 6.4.: Konzept des Szenarios (Quelle: Modifikation von [P2, P4]).

### 6.2.5. Schlussfolgerungen

In Bezug auf Forschungsfrage 2 wurden unterschiedliche Methoden im Konzept umgesetzt, um den Modellierungs- und Validierungsaufwand zu reduzieren sowie die Anwendbarkeit und Performanz der Simulation zu steigern. Zentrales Element ist die *HW-zentrierte Simulationsmodellgenerierung* mit pre-validierten und leistungsoptimierten *Templates*. Außerdem sollen *reaktive Szenarien* zur Stimuligenerierung, Evaluation und Modellreduktion verwendet werden. Ferner soll das *EF-Konzept* integriert werden. Es kapselt die zur Simulation nötige Fahrdynamik und ist eine Schnittstelle zur Co-Simulation.

## 6.3. Abbildung von PREEvision-Komponenten nach Simulink

In den folgenden Abschnitten sind die einzelnen Aspekte der HW-zentrierten Abbildung dargestellt. Dabei werden die Vorgehensweisen zur Generierung von Simulink-Simulationsmodellen aus PREEvision-E/E-Architekturmodellen präsentiert. Startpunkt bilden die bei der Generierung beachteten Leitungssatzelemente. Anschließend folgen Sensoren, Aktoren und ECUs, auf denen die Integration von SWCs aus der SA stattfindet. Zum Schluss wird die manuelle Definition von logischem Verhalten mittels Zustandsautomaten und der Zuweisung von Templates zu PREEvision-Artefakten thematisiert.

### 6.3.1. Leitungssatz

Abbildung 6.5 zeigt anhand eines Beispielmodells die physikalischen Elemente, die bei einer HW-zentrierten Abbildung auf der obersten Hierarchieebene beachtet werden sollen. Zur Leistungsversorgung werden neben den Leitungen auch **Batterien** und **Massestellen** benötigt. Um die E/E-Architektur vor zu hohen Strömen zu schützen dienen Sicherungen, die sich gesammelt in einem **Sicherungskasten** befinden. Daneben bilden **Sensoren**, **ECUs** und **Aktoren** die HW-seitigen Hauptkomponenten. Als Schnittstelle zwischen ihnen und den **Kabeln**, welche eine Bündelung aus **Adern** darstellen und die Komponenten verbinden, dienen Leitungssatz- und Komponentenseitige **Stecker**. Neben Adern gibt es auch **Einzelleitungen**, die nicht gebündelt sind. Abhängig von der Geometrie können auf den Verbindungswegen zwischen HWCs **Trennstellen** notwendig sein. Dies ist bei einem Übergang zwischen unterschiedlichen geometrischen Zonen (Dach, Türen, etc.) der Fall. **Spleiße** werden benötigt, wenn ein Kabel an mehr als zwei Enden verbunden werden soll, da es sich physikalisch um einen Draht bzw. ein Drahtgeflecht handelt.

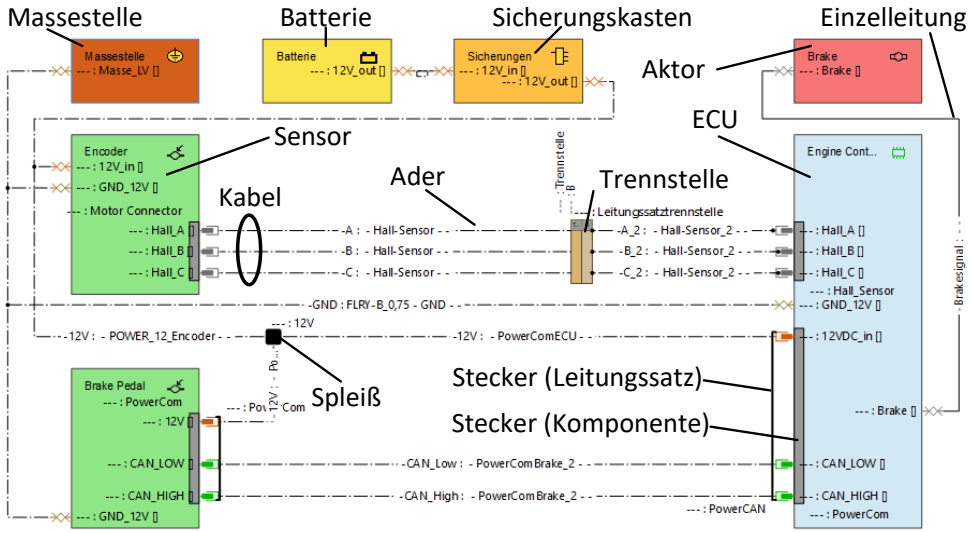



Abbildung 6.5.: Leitungssatzelemente zur Verbindung der HWCs.

### Grundprinzip

Das Grundprinzip der Leitungssatzabbildung nach Simulink ist in Abbildung 6.6 auf oberster Hierarchieebene nach [P3] dargestellt.

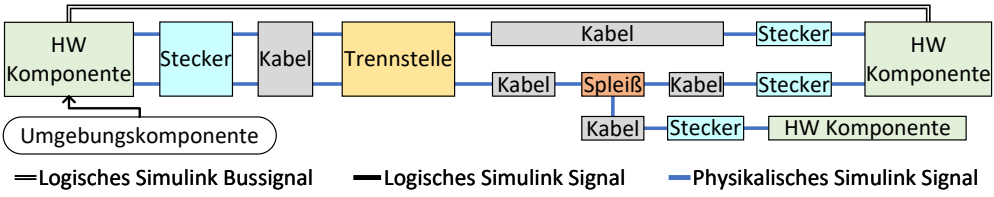


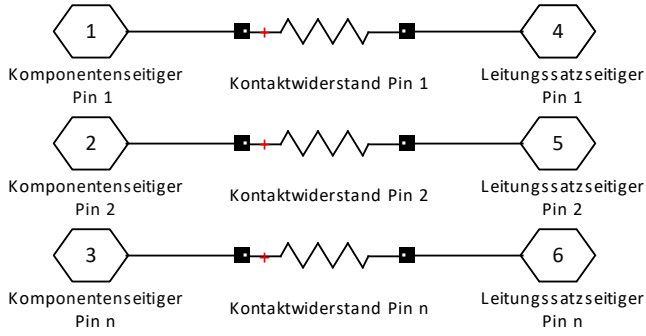

Abbildung 6.6.: Prinzip der Leitungssatzabbildung auf oberster Hierarchieebene (Quelle: Modifikation von [P3]).

Jede abgebildete Komponente entspricht einem Simulink-Subsystem. **HW-Komponenten** (HWCs), wie Sensoren oder Aktoren, sind hierbei über einen physikalisch bzw. elektrisch modellierten Leitungssatz miteinander verbunden. Ein elektrisches Simulink-Signal durchläuft, ausgehend von einer Quellkomponente, diverse Leitungssatzkomponenten wie **Stecker**, **Kabel** und **Trennstellen**, bis es an der Zielkomponente angekommen ist. Leitungssatzkomponenten haben richtungsunabhängige *Simscape Connection Ports* als Anschlüsse. Dadurch können intern Simscape- bzw. SESPS-Bibliotheksblöcke in Form fest zugewiesener Templates für die Abbildung verwendet werden. Diese Templates werden in den nachfolgenden

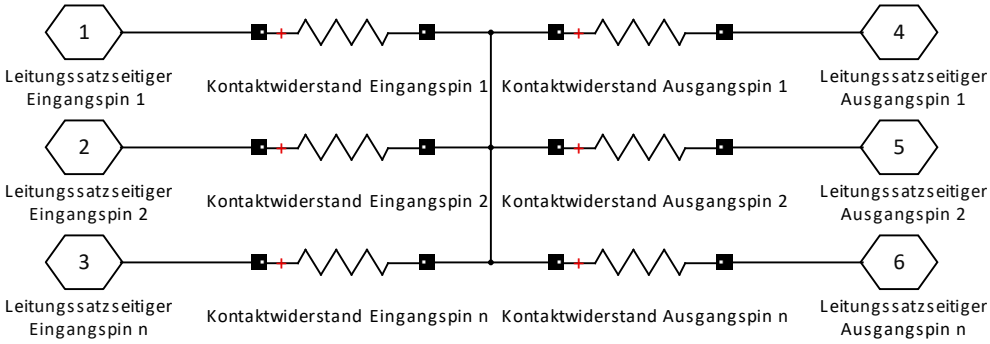
Abschnitten für konkrete Leitungssatzkomponenten beschrieben. Verbindungen zwischen HWCs und **Umgebungskomponenten** sowie Bussysteme werden logisch modelliert. Eine Wandlung zwischen logischen und elektrischen Signalen findet dabei in den HWCs statt. Sie wird in Unterabschnitt 6.3.2 beschrieben. [P3]

### Stecker, Trennstellen und Spleiße

Stecker, Trennstellen und Spleiße sind als serielle Kontaktwiderstände darstellbar. Die Umsetzung des Templates für **Stecker** ist in Abbildung 6.7a dargestellt. Dabei werden, je nach Typ des Steckers, Ersatzwiderstände (**Kontaktwiderstand**) zur Modellierung des Kontaktwiderstands verwendet. Das Konzept gilt auch für **Trennstellen**. Die Umsetzung des Templates für **Spleiße** ist in Abbildung 6.7b dargestellt. Da bei einem Spleiß weder Ein- noch Ausgänge definiert werden und damit jeder Anschluss sowohl Ein- als auch Ausgang ist, wurde eine Sternschaltung gewählt. Die Werte der Widerstände sind dabei abhängig von der Wahl bzw. dem Typ des Spleißes.




(a) Stecker




(b) Spleiß

Abbildung 6.7.: Template für Stecker und Spleiße mit SESPS-Komponenten.

### Kabel

Zur Modellierung von Kabeln mit SESPS-Komponenten steht der Block *Pi Section Line (PSL)* zur Verfügung. Die Anzahl der sich darin befindenden Adern ist über einen Parameter ($N$) definierbar. Abbildung 6.8 zeigt die beispielhafte Verwendung des PSL-Blocks (**Kabel**) als Template. Die **Adern** von **Quellkomponente** und **Zielkomponente** werden an das Kabel ange-

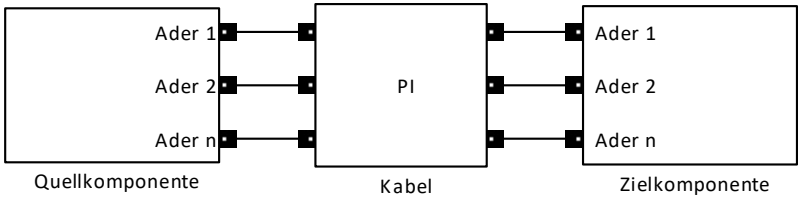



Abbildung 6.8.: Template für Kabel mit SESPS-Komponenten.

schlossen. Charakteristiken der Adern sowie deren gegenseitige kapazitive Beeinflussung werden durch $NxN$ Matrizen für Widerstände, Kapazitäten und Induktivitäten beschrieben. Diese Matrizen lassen sich in Simulink aus der geometrischen Position der Adern zueinander, deren Querschnitt, Material, Isolationsdicke und Länge mit dem *Power Line Parameters Tool (PLPT)* bestimmen. Aus dem PREEvision-Modell lassen sich zwar standardmäßig die Leitungslänge sowie der Querschnitt auslesen, aber eine Generierung der Matrizen ist nicht trivial umsetzbar. Daher wird die Parametrisierung des PSL-Blocks in der prototypischen Implementierung nach der Generierung manuell mithilfe des PLPT ausgeführt. Ferner fehlt bei Verwendung des Blocks PSL eine Schirmung. Sie wird aufgrund des Implementierungsumfangs vernachlässigt.

### Busverbindungen

Je nach Bustyp werden für Busverbindungen eine oder mehrere Adern benötigt. Im Falle von CAN sind es zwei Adern, die das Signal symmetrisch übertragen. Sie sind miteinander verdrillt, um eine geringe Störanfälligkeit zu gewährleisten. Beim *Local Interconnect Network (LIN)* ist es nur eine Leitung. Zusätzlich unterscheiden sich die Bussysteme in ihrem Zugangsverfahren. Die unterschiedlichen Busarten lassen sich prinzipiell physikalisch abbilden. Da Simulink aber per se kein Netzwerk- bzw. Bussimulator ist, soll die eigentliche Simulation von Bussystemen in einem externen Simulator, also innerhalb einer Co-Simulation, erfolgen. Außerdem liegt der Fokus in der Praxis meist auf der Analyse der Latenzen und Auslastungen von Bussystemen. Aus diesen Gründen wird eine logische Abbildung gewählt, wobei jede an der Buskommunikation beteiligte Komponenten einen Leseeingang (**RX**) und einen Schreibausgang (**TX**) für Nachrichten erhält. Das Prinzip ist in Abbildung 6.9 exemplarisch dargestellt. Es ist zu beachten, dass in Simulink nicht mehrere Ausgänge an einen Eingang angeschlossen werden können, weswegen ein *Merge*-Block (hier als **BusMerger** bezeichnet) verwendet wird. Die Co-Simulation einer CAN-Simulation kann bspw. mit dem Werkzeug **CANoe** [288] erfolgen. Hierzu werden **Message-In**- und **Message-Out**-Blöcke verwendet.

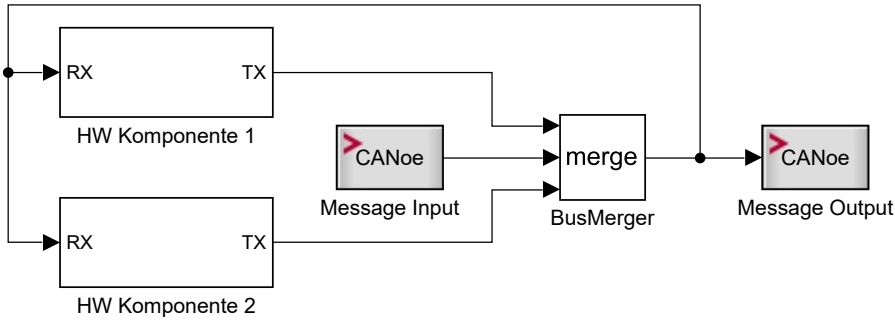



Abbildung 6.9.: Template für Bussysteme am Beispiel einer Co-Simulation mit CANoe.

#### Batterien und Massestellen

Für die Modellierung von Batterien bietet die SESPS-Bibliothek ein generisches Batteriemodell namens *Battery*. Dieses erlaubt durch Parametrisierung die Simulation von unterschiedlichen Batterietypen wie Blei oder Lithium-Ion, welche typischerweise im Fahrzeug vorkommen. Es wird als Template verwendet. Je nach Typ können dabei auch Alterungs- und Temperatureffekte simuliert werden. Außerdem besitzt das Modell Messausgänge für den aktuellen Ladezustand, die Batteriespannung oder den Entladestrom.

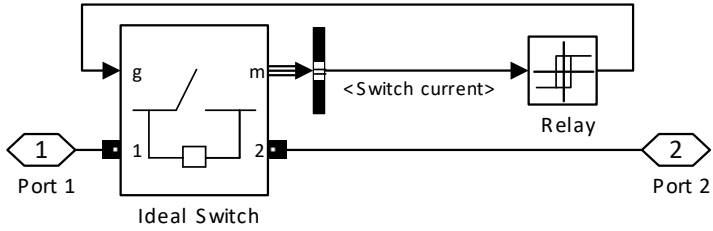
Als Template für Massestellen dient der SESPS-Block *Ground*.

#### Sicherungskasten

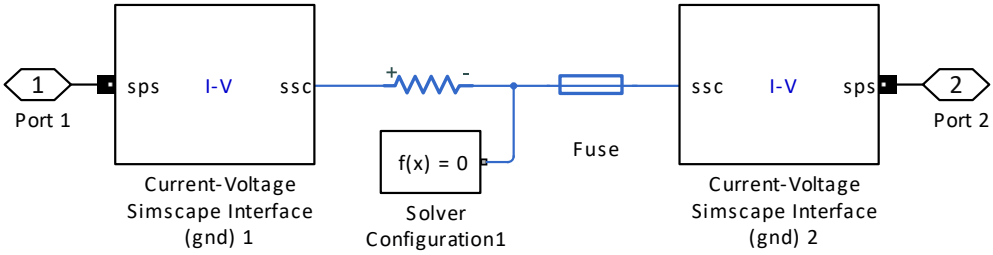
Sicherungskästen sind in PREEvision so gestaltet, dass sie in Komponentendiagrammen mit den internen Sicherungen verfeinert werden. Für die Sicherungen lassen sich die Parameter Auslösegeschwindigkeit, Auslösestrom sowie min. und max. Betriebsspannung in PREEvision festlegen. In der SESPS-Bibliothek gibt es für Sicherungen keine explizite Komponente. Daher wurde ein eigenes Modell umgesetzt, welches in Abbildung 6.10a dargestellt ist. Hierbei wird ein Schalter (**Ideal Switch**) genutzt, der bei einem Schwellwertstrom, d.h. dem Auslösestrom der in der Komponente **Relay** definiert ist, geöffnet wird. Auslösegeschwindigkeit sowie min. und max. Betriebsspannung werden hierbei vernachlässigt. Alternativ wurde deshalb eine zweite Realisierung untersucht (vgl. Abbildung 6.10b), bei der Komponenten aus der Simscape-Electric-Bibliothek (blau dargestellt) verwendet werden. Für die Sicherung wird die Komponente **Fuse** verwendet. Die Auslösegeschwindigkeit lässt sich hier zwar definieren, aber die min. und max. Betriebsspannung wird ebenso vernachlässigt. Hinzukommend müssen an den Anschlüssen Wandler (**Current-Voltage Simscape Interface**) zwischen den SESPS-Signalen und den Simscape-Electric-Signalen angebracht werden. Zusätzlich werden ein lokaler Gleichungslöser (**Solver Configuration**) und ein Serienwiderstand (*Resistor*) benötigt.

Zum Vergleich werden beide Modelle hinsichtlich ihrer Performanz geprüft, indem über 10 s Laufzeit ein um 10 $\frac{A}{s}$ ansteigender Strom angelegt wird. Beide Varianten der Sicherung werden

mit einem Auslösestrom von 50 A konfiguriert. Unter gleichen Testbedingungen ergibt sich auf System B (siehe Tabelle 2.4) für die Simscape-Electric-Variante eine Laufzeit von 75,507 s, während die SESPS-Variante 0,41 s benötigt. Da die SESPS-Variante damit ca. 184-mal schneller ist, wird sie als Template verwendet. Eine Erweiterung des Modells zur Beachtung der Auslösegeschwindigkeit sowie der min. und max. Betriebsspannung ist zwar möglich, aber wird in dieser Arbeit nicht realisiert.




(a) Realisierung mit SESPS.




(b) Realisierung mit Simscape.

Abbildung 6.10.: Realisierungsmöglichkeiten für Templates von Sicherungen.

### 6.3.2. Hard- und Software

Zur Simulation der übrigen HWCs, wie Sensoren, Aktoren und ECUs, werden Informationen aus mehreren Ebenen benötigt (vgl. Abbildung 6.2). Da es in Simulink keine Ebenen gibt, muss eine Integration der PREEvision-Ebenen in Simulink stattfinden. Mithilfe von ECUs werden SA und HA integriert. Sensoren und Aktoren integrieren dagegen LA und HA. Im Folgenden werden die Abbildungen dieser Komponenten erläutert.

#### Sensoren und Aktoren

Jeder Sensor oder Aktor auf der PREEvision-HA wird in ein Simulink-Subsystem abgebildet. Dabei sind Sensoren und Aktoren als Signal- und Energiewandler zwischen der logischen, elektrischen und mechanischen Domäne zu verstehen. Es muss daher definiert werden, in welcher Form diese Wandlung erfolgt. Da Sensoren anhand ihres Energiewandlungsprinzips [221]

und Aktoren, z.B. in der Form von Motoren, nach ihrer Bestromungsart klassifiziert werden können [31], lassen sich ihren Modellkomponenten parametrisierbare Templates zuweisen. Das Vorgehen der Template-Zuweisung mittels TALIA ist in Unterabschnitt 6.3.3 beschrieben. Zunächst werden die Prinzipien der Wandlung in Abbildung 6.11 nach [P3] für Sensoren und Aktoren beschrieben.

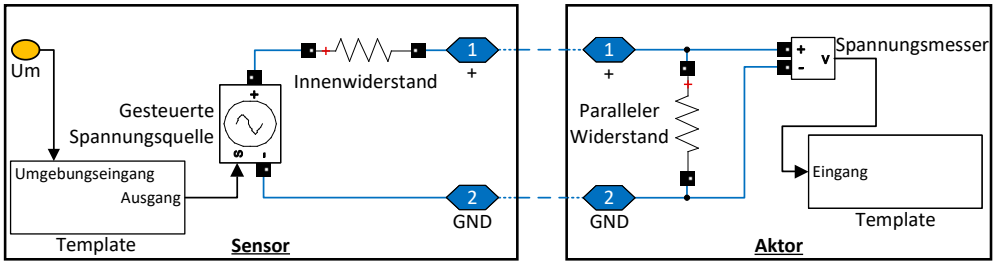



Abbildung 6.11.: Prinzip der Signalwandlung in Sensoren und Aktoren (Quelle: Modifikation von [P3]).

Ausgangspunkt ist das dem **Sensor** auf der linken Seite zugewiesene Template. Es ist i.d.R. logischer Natur und besitzt sowohl logische Umgebungseingänge als auch logische Ausgänge. Der Umgebungseingang **Um** (orange) des Sensors befindet sich in PREEvision auf der LA, während die elektrischen Anschlüsse **+** und **GND** (beide blau) auf der HA zu finden sind. Um nun von dem Template ausgehende logische Signale über den Leitungssatz zu schicken, ist eine Signalwandlung nötig. Diese wird über eine **gesteuerte Spannungsquelle** (*Controlled Voltage Source*) aus der SESPS-Bibliothek realisiert. [P3]

**Aktoren** sind analog zu den Sensoren aufgebaut. Im Gegensatz zu Sensoren erfolgt hier jedoch die Rückwandlung der elektrischen Signale in logische. Dazu wird die Komponente **Spannungsmesser** (*Voltage Measurement*) aus der SESPS-Bibliothek verwendet. [P3]

**Templates** können generell auch nur mit SESPS-Blöcken aufgebaut werden. In diesem Fall ist keine Wandlung nötig und die Ausgänge des Templates können direkt mit den Ausgängen des Subsystems (Sensor/Aktor) verbunden werden, da es sich in beiden Fällen um Simscape Connection Ports handelt. [P3]

**Diskussion** In der Theorie könnten für die Wandlung auch Simscape-Komponenten verwendet werden, die nicht Teil der SESPS-Bibliothek sind. Dazu müsste das logische Simulink-Signal jedoch zuerst in ein physikalisches Signal und anschließend noch in ein elektrisches Signal umgewandelt bzw. in umgekehrter Reihenfolge wieder zurückgewandelt werden. Außerdem würde für jedes entstehende elektrische Netz ein daran angeschlossener *lokaler* Gleichungslöser (Solver Configuration) benötigt werden. Bei Verwendung von SESPS-Blöcken ist dies nicht nötig, da ein *zentraler* Gleichungslöser auf der obersten Hierarchieebene namens *powergui* verwendet wird. Vor der Ausführung des Modells werden alle elektrische Netze und SESPS-Komponenten in einer globalen Ersatzschaltung zusammengefasst, welche in Form von kompiliertem C-Code ausgeführt wird. Dadurch führt ein mit SESPS-Komponenten auf-

gebautes Modell schneller aus, als ein mit elektrischen Simscape-Komponenten aufgebautes Vergleichsmodell. Da bei Bedarf jedoch elektrische Simscape-Komponenten über die Blöcke *Voltage-Current Simscape Interface* und *Current-Voltage Simscape Interface* integriert werden können, erlaubt die Abbildung auch die Verwendung von Templates, die mit elektrischen Simscape-Komponenten aufgebaut wurden.

**Limitierung** Eine explizite Leistungsaufnahme für Sensoren und Aktoren wurde nicht modelliert. Diese müsste bei Bedarf in den verwendeten Templates umgesetzt werden.

### ECUs und Software

Zunächst werden der allgemeine Aufbau einer ECU und das Prinzip der Ausführung von SWCs bei Verwendung von AUTOSAR betrachtet. Daraus erfolgt die Herleitung des resultierenden Gesamtkonzepts, welches anschließend beschrieben wird.

**Herleitung der Abbildung** Abbildung 6.12 zeigt den allgemeinen und vereinfachten internen Aufbau einer ECU.

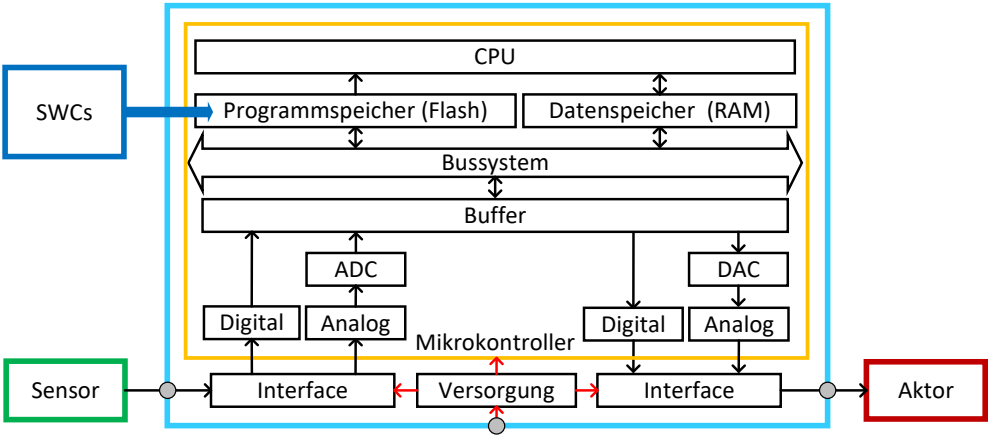



Abbildung 6.12.: Abstrakter interner Aufbau einer ECU (Quelle: eigene Darstellung nach [213]).

Sensordaten durchlaufen beim Einlesen zunächst eine Filterung, eine Pegelanpassung und diverse Schutzschaltungen. Diese Schritte sind im Block **Interface** zusammengefasst. **Digitale Signale** können im Anschluss direkt in einen **Buffer** geschrieben werden, während **analoge Signale** noch davor mit einem A/D-Wandler (**ADC**) gewandelt werden müssen. Vom Buffer aus werden die eingelesenen Daten dann über einen oder mehrere **Busse** in den **Datenspeicher** geschrieben. Ein Speichern in den **Programmspeicher** ist durch Konfiguration bei man-

chen **Mikrocontrollern** ebenfalls möglich. Der Programmspeicher ist jedoch primär für die Software bzw. die **SWCs** reserviert. SWCs werden auf der *Central Processing Unit* (**CPU**) ausgeführt, welche die zuvor eingelesenen Daten für die Verarbeitung aus den entsprechenden Speichern lädt und die Ergebnisse der Operation in diese zurückschreibt. Von dort aus stehen die Daten zur Ausgabe über das Bussystem bereit. Sie werden dann in den Buffer geschrieben und anschließend über ein Interface auf den benötigten Ausgangspegel gebracht. Handelt es sich um ein analoges Ausgangssignal, so werden die Daten vor der Ausgabe noch von einem D/A-Wandler (**DAC**) gewandelt. Eine Anbindung an externe Bussysteme, wie CAN, ist in der Abbildung nicht direkt dargestellt. Über entsprechende (Bus-) Interfaces können Daten von externen Bussystemen aber ebenfalls gelesen und geschrieben werden. Zur Durchführung aller beschriebenen Schritte wird letztendlich noch eine entsprechende **Energieversorgung** benötigt. Sie versorgt neben dem Mikrocontroller auch die Interfaces und gegebenenfalls weitere Peripheriekomponenten der ECU.

Anhand von Abbildung 6.12 sind die Verarbeitungsschritte sowie die Integration von SWCs und ECU zu erkennen. SWCs liegen im Programmspeicher und werden auf der CPU ausgeführt. Sie verbrauchen dadurch Ressourcen wie Zeit, Speicher und Energie. Die Integration der SWCs soll nun genauer betrachtet werden. PREEvision stellt mit der SA eine Möglichkeit bereit AUTOSAR-SWCs und ihre Schnittstellen zu beschreiben. Abbildung 6.13 zeigt dies beispielhaft in Zusammenhang mit dem Autosar-Standard. Die Ausführung der SWCs kann periodisch ($t_1$, $t_2$, $t_3$) oder durch eine andere SWCs angestoßen (**Trigger**) modelliert werden. Ferner sind die SWCs in PREEvision jeweils einer ECU zugewiesen, auf der sie ausgeführt werden. In AUTOSAR wird die reelle Hardware abstrahiert. Abbildung 6.13 zeigt das Konzept exemplarisch für ECU 1. Die **Basis SW** stellt dabei abstrakte Schnittstellen zur reellen Hardware bereit, welche die einzelnen SWCs stellvertretend über die *Laufzeitumgebung* (*Runtime Environment* (**RTE**)) nutzen. Die Schnittstellen bieten u.a. den Zugriff auf die Ein- und Ausgänge (**I/O Driver**) sowie auf den Festspeicher (*Non-Volatile RAM* (**NVR**)) der HW. Über das RTE werden dann die Eingangsdaten bereitgestellt, auf welche die SWCs zugreifen (**Lesen**). Betrachtet man z.B. den Datenfluss zwischen **SWC 2** und **SWC 4**, so liest **SWC 4** die Daten am Ausgang von **SWC 2** . Damit dies über die ECUs hinweg möglich ist, werden alle Ausgangsdaten der SWCs in dem RTE persistiert (**Schreiben**). Das RTE ist außerdem für das Triggern der Ausführung der SWCs bzw. derer **Runnables/Tasks** ($\mathbf{T}_i$) verantwortlich. Die Zeitpunkte dafür werden in **Schedule-Tables** festgelegt, die Teil vom AUTOSAR Betriebssystem (**OS**) sind. Daneben werden auch Tasks der Basis SW selbst ($\mathbf{T}_{BS}$) vom **BS Scheduler** eingeplant. Die einzelnen Schedule-Tables$_i$ werden zyklisch und in fester Reihenfolge ausgeführt.

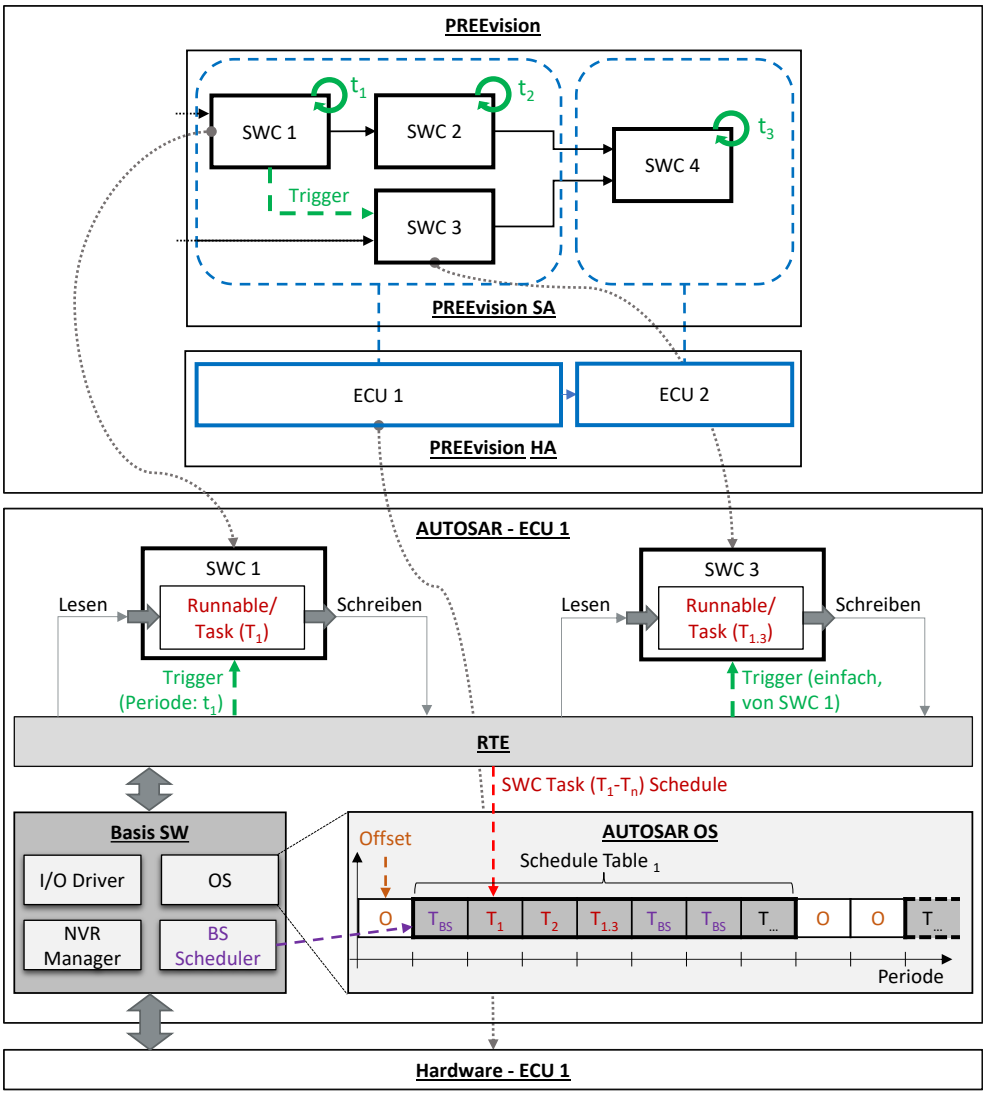



Abbildung 6.13.: Scheduling von AUTOSAR-SWCs (Quelle: eigene Darstellung unter Verwendung von [313]).

**Resultierendes Gesamtkonzept** Aus den Überlegungen zum internen Aufbau der ECU und der Funktionsweise des AUTOSAR-Schedulings lässt sich nach [P3] das in Abbildung 6.14 dargestellte Konzept ableiten. Es zeigt in abstrahierter Form ein generisches AUTOSAR-ECU-Modell, welches als Simulink-Subsystem realisiert wird.

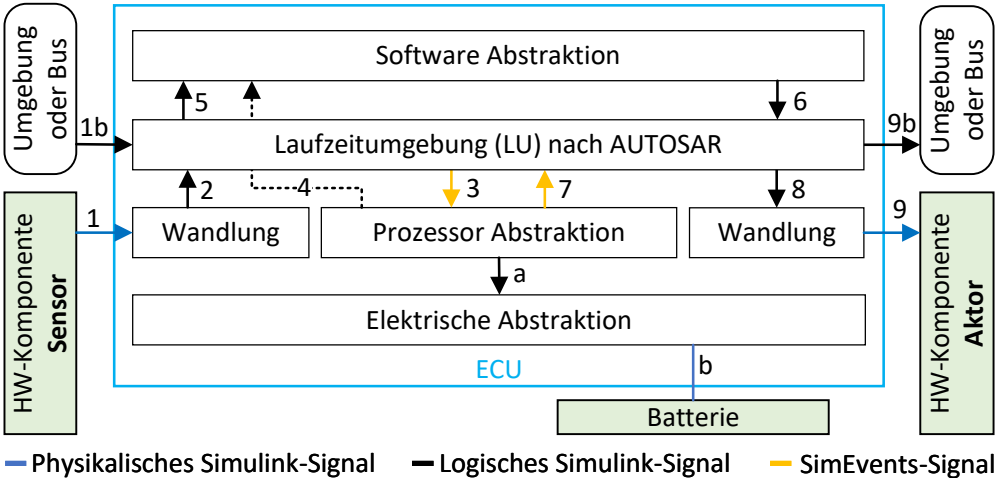



Abbildung 6.14.: Konzept einer generischen AUTOSAR-ECU in Simulink (Quelle: Modifikation von [P3]).

Im Folgenden wird zunächst der Zusammenhang der abstrakten Komponenten nach [P3] beschrieben. Danach folgt die Beschreibung der einzelnen Elemente im Detail.

Der erste Schritt ist das Einlesen von Sensordaten (**1**). Wie in den vorherigen Abschnitten beschrieben, handelt es sich dabei um SESPS-Signale. Diese werden in logische Signale gewandelt (**Wandlung**) und dann in einen Speicher, der Teil der **Laufzeitumgebung (LU) nach AUTOSAR** ist, geschrieben (**2**). Eingehende logische Signale von Umgebungskomponenten oder Bussignalen (**1b**) bedürfen keiner Wandlung und werden daher direkt in die LU geschrieben. Da die LU sich an AUTOSAR orientiert, wird dort auch das Scheduling der SWCs ausgeführt. SWCs liegen in Simulink in Form von *Function Call Subsystems (FCSs)* mit jeweils einem zugewiesenen Template in der **Software Abstraktion** vor. Die Ausführung eines solchen FCS wird durch Aufruf einer ihm zugewiesenen Simulink *Trigger-Function* ausgelöst. Um die Funktionsaufrufe zu realisieren, werden von der LU ausgehende Events (SimEvents-Entity) (**3**) an die Komponente **Prozessor Abstraktion** gesendet. Dort werden die Events zunächst in eine Warteschlange zur Prozessierung eingereiht. Das Prinzip hierzu ist aus [161] abgeleitet. Bei der Verarbeitung einer Entity wird die Trigger-Function für das FCS der ihr zugewiesenen SWC aufgerufen (**4**). Das FCS wird dann einmalig ausgeführt, wobei die seinen Eingängen zugewiesenen Daten aus der LU eingelesen werden (**5**). Anschließend werden die Ergebnisse zurück in den Speicher der LU geschrieben (**6**). Sobald die Prozessierung einer Entity abgeschlossen ist, wird ein Event (**7**) an die LU geschickt, sodass die zuvor geschriebenen Speicherwerte aus der LU zuerst gewandelt (**8**) und dann als elektrische Signale an den Ausgang gelegt werden

(**9**). Logische Signale von Umgebungskomponenten oder Bussignalen benötigen keine **Wandlung** und werden direkt ausgegeben **(9b**). Mithilfe der Komponente **Elektrische Abstraktion** wird die Auslastung der CPU bei der Prozessierung (**a**) in einen Ersatzwiderstandswert (**b**) gewandelt. So wird der Energieverbrauch der ECU genähert. [P3]

**Laufzeitumgebung (LU) nach AUTOSAR** Die LU bildet durch die Koordination der Task-Ausführung und Speicherverwaltung das Hauptelement der generischen AUTOSAR-ECU. Abbildung 6.15 zeigt die Implementierung der LU in Simulink.

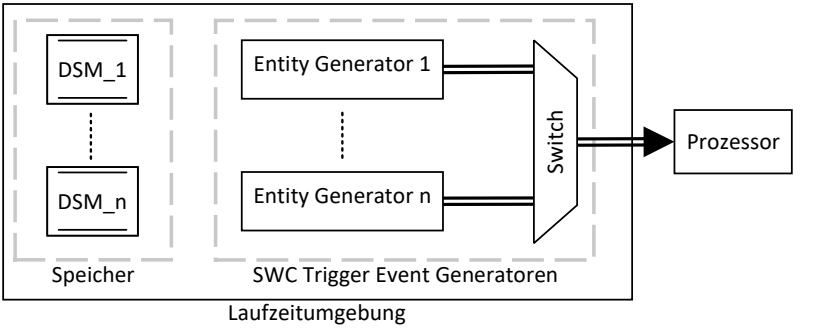



Abbildung 6.15.: Implementierung der Laufzeitumgebung in Simulink (Quelle: eigene Darstellung nach [S4]).

Der **Speicher** ist nach [S4] in Simulink mit Blöcken vom Typ *Data Store Memory (DSM)* realisiert. Jedem Ein- und jedem Ausgang der SWCs, ist jeweils ein eigener DSM-Block zugeordnet. Die Umsetzung des Schedulings kann mit Simulink auf verschiedene Arten erfolgen. In [S4] wurden dabei drei Standardmethoden verglichen und beschrieben:

1. **Statecharts und Rate-Based Models**: Ein *Chart* wird für das Scheduling verwendet, wobei jeder Zustand einen Task repräsentiert und einen Ausgangsport besitzt. Beim Eintritt in den Zustand findet über seinen Ausgangsport ein Funktionsaufruf statt, der die Ausführung des daran angeschlossene Subsystems auslöst (*Rate-Based Model*).
2. **Schedule Editor**: Mit dem Schedule-Editor lässt sich manuell und automatisiert aus dem Datenfluss ein Schedule erzeugen. Dabei wird jeder Task als Subsystem modelliert, das über der Aufruf einer Trigger-Function im Schedule-Editor aktiviert wird.
3. **Simevents**: Die Verwendung von Simevents für ein Scheduling wurde in [161] vorgestellt und mit einem entsprechenden Prozessormodell kombiniert. Hier entsprechen die Tasks ebenfalls Subsystemen, deren Ausführung über eine Trigger-Function aktiviert wird. Jedem Subsystem ist eine Entity (Ereignis) zugewiesen, die in einer Warteschlange eingereiht und in einem Entity-Server verarbeitet wird. Hier werden die Trigger-Functions aufgerufen.

Da nach [S4] in den ersten beiden Methoden das Prozessormodell fehlt, wodurch keine Zuweisung von einem Task zu einem spezifischen Rechenkern oder einer Priorität möglich ist, wird

der dritte Ansatz gewählt. Zudem ist damit eine Abstraktion der Leistungsaufnahme über die Auslastung der Rechenkerne möglich. [S4]

Um die Ausführung von SWCs auszulösen, werden Entities mit Blöcken vom Typ Entity-Generator erzeugt und an den Prozessor geschickt (vgl. Abbildung 6.15). Eine Entity hat dabei die in Tabelle 6.2 nach [S4] beschriebenen Attribute.

Tabelle 6.2.: Attribute von Entities (Quelle: eigene Darstellung nach [S4]).

| **Attribut** | **Beschreibung** |
|---|---|
| TaskID | Die Verknüpfung des SWC-Subsystems mit seinem individuellen Task erfolgt über diesen *Identifikator (ID)*. |
| TaskPriority | Tasks bekommen eine Priorität damit kritische Funktionen schneller auf dem Prozessor ausgeführt werden können. |
| TaskRuntime | Abstrakte Laufzeit, die der Prozessor zur Verarbeitung der Komponente benötigen soll. |
| CoreID | Ermöglicht die explizite Zuweisung des Tasks zu einem bestimmten Rechenkern. |
| PreemptionPriority | Priorität, mit der wartende Tasks verarbeitet werden, wenn ihre Bearbeitung aufgrund eines Tasks mit einer höheren TaskPriority unterbrochen wurde. |
| NextTaskID | Falls eine SWC eine andere triggert, so erfolgt hier die Referenz auf die nachfolgende SWC. |
| Complete | Gibt den Zustand der Verarbeitung in Integer-Werten an. 0 = „Entity erzeugt“ , 1 = „löst weitere SWC aus“ und 2 = „Bearbeitung abgeschlossen“. |

**Software-Abstraktion** Auf der linken Seite in Abbildung 6.16 sind Teile der **PREEvision-SA** sowie der **PREEvision-HA** beispielhaft nach [P3] dargestellt. Jede PREEvision-SWC wird in ein Simulink-FCS mit selbem Namen überführt. Die FCSs sind rechts innerhalb der **Software Abstraktion** mit entsprechenden Trigger-Eingängen (**'Task'+'TaskID'+'()'**) abgebildet. Jedem FCS ist ein **Template** zugewiesen. Der Wert für TaskID kommt aus dem PREEvision-Modell. Im Falle von SWC 4 ist TaskID = 4. Über den Aufruf einer Trigger-Funktion, die dem Namen des Trigger-Eingangs entspricht, wird ein FCS einmalig ausgeführt. Der Aufruf findet innerhalb eines Matlab-Skripts (**M-Skript**) statt und ist Bestandteil der **Prozessor Abstraktion**. Der **Speicher** wird in der **Laufzeitumgebung** mit **DSM**-Blöcken gebildet. Mithilfe von Blöcken des Typs *Data Store Write* (**DSW**) kann in die DSM-Blöcke geschrieben werden, während mit Blöcken des Typs *Data Store Read* (**DSR**) daraus gelesen werden kann. Jede **Sender-Receiver-Schnittstelle** (logische Verbindung in der PREEvision-SA) bekommt einen solchen DSM-Block im Speicher der LU zugewiesen. An den Eingängen der FCSs sind dann die zugehörigen DSR-Blöcke und an den Ausgängen die entsprechenden DSW-Blöcke angeschlossen. Durch diesen Ansatz können SWCs zeitlich unabhängig von dem in der Struktur der PREEvision-SA vor-

gegeben Datenfluss ausgeführt werden, d.h. es können unterschiedliche Scheduling-Arten in der Simulation untersucht werden. Neben den Sender-Receiver-Schnittstellen gibt es in der PREEvision-SA auch **Trigger**-Schnittstellen (grün dargestellt). Für das Scheduling muss einer SWC entweder eine solche Trigger-Schnittstelle zugewiesen sein oder ein zyklisches Zeitverhalten. Letztgenanntes wird über die **Portkommunikationsanforderung** für jeden betreffenden Ausgang definiert. Für jede SWC, die einen Ausgang mit einem zyklischen Sendeverhalten besitzt, wird dann ein Entity-Generator (**Entity Gen.**) mit der entsprechenden Periode (**Period**) bei der Generierung angelegt. Der Wert **Periode** muss dazu in der Portkommunikationsanforderung angegeben werden. Weisen die Ausgänge einer SWC unterschiedliche Perioden auf, so wird die niedrigste Periode von allen verwendet. Mit der Periode erzeugen die Entity-Generatoren dann zyklisch Entities, welche die in Tabelle 6.2 beschriebenen Attribute aufweisen. **TaskRuntime** entspricht der mittleren Latenzzeit der SWC (**Latency**). Für **CoreID**, **Priority** und **TaskID** gibt es keine direkten SWC-Attribute. Es lassen sich dafür jedoch entsprechende *generische Attribute* in PREEvision anlegen. Einem **Mikroprozessor** zugewiesene Kerne weisen zwar eine **CoreID** auf, jedoch können SWCs in PREEvision keinem expliziten Kern, sondern nur einem Mikroprozessor zugewiesen werden. Daher kann dieses CoreID-Attribut nicht verwendet werden. Für getriggerte SWCs wird ferner kein Entity-Generator erzeugt, sondern das Attribut **NextTaskID** genutzt. Dabei erhalten triggernde Entities dieses Attribut als Referenz auf die getriggerte Entity. Nach der Verarbeitung einer triggernden Entity im **Server**, werden ihre Attribute durch die der getriggerten Entity ersetzt. Das Attribut **Complete** wird dabei für den Status der Verarbeitung verwendet. Die Verwendung des Attributs **Preemption-Priority** ist optional. Es wird ebenfalls als generisches Attribut in PREEvision angelegt. [P3]

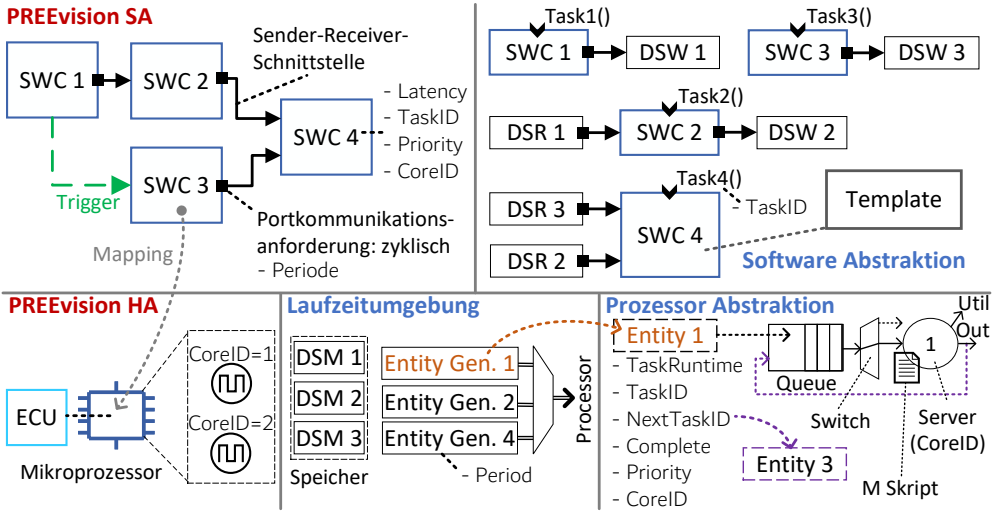



Abbildung 6.16.: Implementierung der Software-Abstraktion in Simulink (Quelle: Modifikation von [P3]).

**Prozessor Abstraktion** Das Konzept des Prozessors ist anhand eines Beispiels für einen Mikroprozessor mit zwei Kernen in Abbildung 6.17 nach [S4] dargestellt.

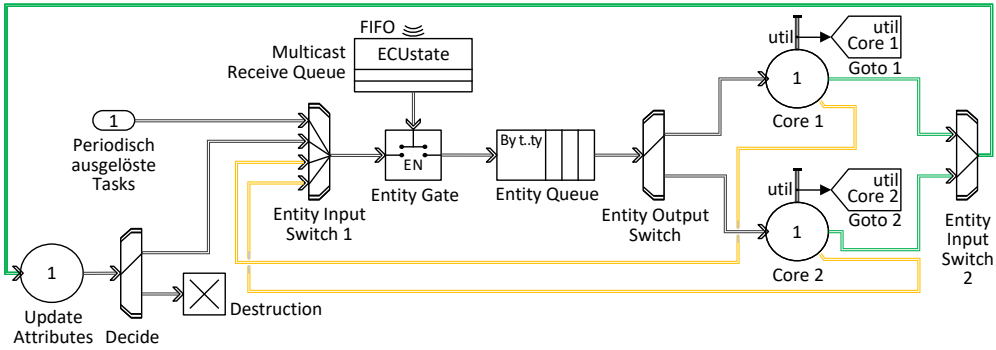



Abbildung 6.17.: Implementierung des Prozessormodells in Simulink (Quelle: eigene Darstellung nach [S4]).

Die Entities aus der LU kommen nach [S4] am Eingang **Periodisch ausgelöste Tasks** an und werden zunächst über den Block **Entity Input Switch 1** zusammen mit weiteren Entities über ein **Entity Gate** an die prioritätenbasierte Warteschlange **Entity Queue** weitergeleitet. Das Entity Gate lässt nur Entities durch, wenn sich die ECU im Zustand *RUN* befindet, d.h. aktiv ist. Der ECU-Zustand wird dabei von einer **Multicast Receive Queue** empfangen. Durch die prioritätenbasierte Entity Queue können Entities mit höherer Priorität zuerst verarbeitet werden. An ihrem Ausgang werden die Entities dann über einen **Entity Output Switch** an die jeweiligen Kerne verteilt. Dies kann auf mehrere Arten erfolgen. Beispielweise mit dem Algorithmus *Round Robin* oder anhand der den Entities zugewiesenen CoreIDs (vgl. Tabelle 6.2). In der Standardimplementierung wird festgelegt, dass immer der Kern verwendet werden soll, der gerade frei ist. Somit kann eine gleichmäßige Auslastung der Kerne erreicht werden. Die Anzahl der Kerne ist variabel. In den dargestellten Kernen **Core 1** und **Core 2** vom Typ *Entity Server* findet die Verarbeitung der Entities innerhalb einer *Entry Event Action* statt. Diese Action löst die Ausführung eines Matlab-Skripts aus. Bei dessen Verarbeitung wird das TaskID-Attribut der Entity ausgelesen und davon abhängig die Trigger-Funktion desjenigen FCSs aufgerufen, welches der SWC entspricht, die mit der Entity verknüpft ist. Hierzu wird im Skript eine Fallunterscheidung generiert. Als *Service Time* der Entity Server wird das Attribut *TaskRuntime* aus der Entity ausgelesen und gesetzt. Die Service Time entspricht der Zeit, in der sich die Entity zur Verarbeitung im Entity Server befindet. Kommt während der Verarbeitung eine Entity mit höherer Priorität an, so wird die sich in Bearbeitung befindende Entity, abhängig vom gewählten Scheduling-Verfahren, wieder zurück in die Warteschlange geführt. Dazu werden Preemption-Verbindungen (orange dargestellt) generiert. Standardmäßig sind die Prioritäten für alle Entities gleich. Nach einer erfolgreichen Verarbeitung (grün dargestellt) wird der Status der Entity in Form des Complete-Attributs entweder auf den Wert 1 gesetzt, wenn die SWC noch eine nachfolgende SWC triggert, oder auf den Wert 2, wenn die Verarbeitung abgeschlossen ist. Anschließend wird die Entity über den **Entity Input Switch 2** zu dem

Entity Server **Update Attributes** geführt. Hier sind die Attribute der Entities in einem generierten M-Skript gespeichert, deren SWCs von einer anderen getriggert werden. Die Ausführung des M-Skripts wird hier ebenfalls durch eine Entry Event Action ausgelöst. Dabei wird das Complete-Attribut der angekommenen Entity ausgewertet. Ist das Complete-Attribut = 1, so wird das Attribut NextTaskID ausgelesen. Anhand dessen werden sämtliche Attribute der angekommenen Entity mit den Attributen der nachfolgenden Entity überschrieben. Das Attribut *Priorität* wird dabei auf den Maximalwert erhöht, wodurch die Verarbeitung der getriggerten Entity direkt im Anschluss erfolgt. Dazu wird die aktualisierte Entity als neuer Task über den oberen Ausgang des Blocks **Decide** und den Block **Entity Input Switch1** wieder zurück in die Warteschlange geführt. Ist das Complete-Attribut vor dem Eintritt in **Update Attributes** hingegen auf den Wert 2 gesetzt, so wird es dort nur durchgeschleust und im Block **Decide** am unteren Ausgang ausgegeben. Die Entität wird dadurch anschließend im Block **Destruction** zerstört. Die Auslastung beider Kerne wird über die **Goto** Label **util Core 1** und **util Core 2** zur elektrischen Abstraktion geschickt. Dies wird in einem eigenen Abschnitt beschrieben. [S4]

**Wandlung** Abbildung 6.18 zeigt die Wandlung an den Eingängen einer ECU nach [S4], jedoch mit SESPS-Komponenten. Sie entspricht dem in Abbildung 6.11 dargestellten Prinzip.

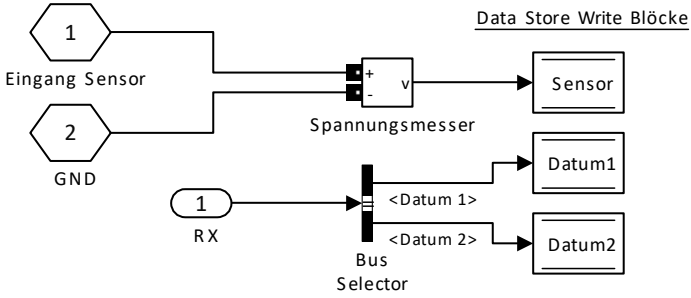



Abbildung 6.18.: Wandlung an den Eingängen einer ECU (Quelle: eigene Darstellung unter Verwendung von [S4]).

Nach der Konvertierung der elektrischen SESPS-Signale (vgl. **Eingang Sensor** und **GND**) in logische Simulink-Signale über den Block **Spannungsmesser**, erfolgt die Speicherung der Werte nach [S4] in den ihnen zugewiesenen DSM-Block innerhalb der LU. Für den Schreibzugriff werden DSW-Blöcke verwendet (vgl. **Sensor**). Bussignale (vgl. **RX**) müssen hingegen nicht gewandelt werden, da sie bereits in Form logischer Simulink-Signale vorliegen. Stattdessen werden die einzelnen Busdaten (vgl. **Datum 1** und **Datum 2**) über den Block **Bus Selector** extrahiert und über einen DSW-Block in den ihnen zugewiesenen DSM-Block in der LU gespeichert. [S4]

Abbildung 6.19 zeigt die Wandlung von logischen Daten aus dem DSM-Speicher innerhalb der LU in elektrische SESPS-Signale nach [S4], jedoch mit SESPS-Komponenten. Sie entspricht ebenfalls dem in Abbildung 6.11 dargestellten Prinzip. Über DSR-Blöcke (vgl. **Aktor**) werden logische Simulink-Singale über den Block **Gesteuerte Spannungsquelle** in elektrische SESPS-Signale gewandelt. Nachrichten, die einem Bus zugeordnet sind (vgl. **Datum 1** und **Datum 2**),

werden dagegen über den Block **Bus Creator** nach [S4] zu einem Bussignal kombiniert und entsprechend ausgegeben (vgl. **TX**).

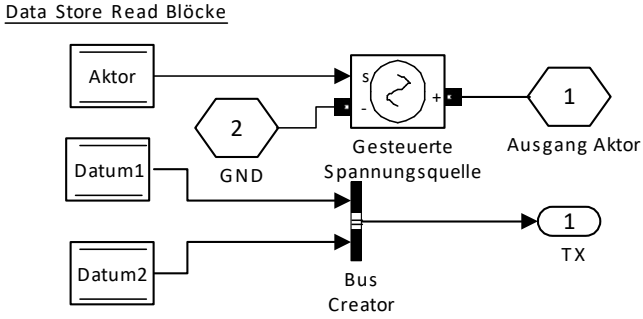



Abbildung 6.19.: Wandlung an den Ausgängen einer ECU (Quelle: eigene Darstellung unter Verwendung von [S4]).

**Elektrische Abstraktion** Abbildung 6.20 zeigt das Konzept und die Abhängigkeiten bei der Leistungsberechnung innerhalb einer ECU nach [S4] mit Anpassungen.

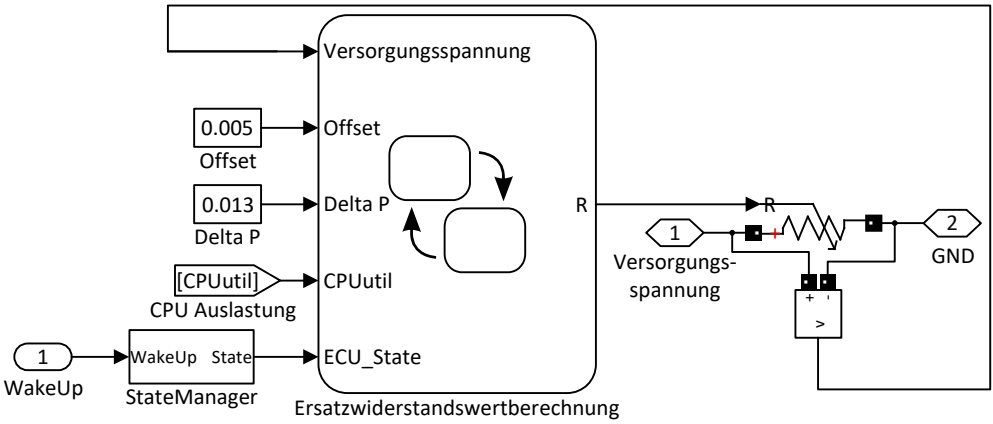



Abbildung 6.20.: Modellierung der Leistungsaufnahme einer ECU (Quelle: eigene Darstellung unter Verwendung von [S4]).

So werden in der Anpassung SESPS-Komponenten und ein dynamischer Versorgungsspannungswert verwendet. Anhand der Überlegungen aus [43] und der AUTOSAR-Spezifikation [2] wurde in [S4] zunächst ein an AUTOSAR angelehnter Zustandsautomat namens **StateManager** entwickelt. Er hat die Zustände *STARTUP*, *RUN*, *WAKEUP*, *SHUTDOWN*, *SLEEP* und

*OFF*. Dieser StateManager ist Teil der ECU. Die Zustandsübergänge werden entweder extern von einem Szenario oder von einem vergleichbaren logischen Signal über den Eingang **WakeUp** ausgelöst. Der vom StateManager ausgegebene Zustand (**State**) dient neben weiteren Werten zur Berechnung eines Ersatzwiderstandswertes **R** im Zustandsautomaten **Ersatzwiderstandswertberechnung**. Dazu zählt der Wert **CPU_Util** der über das Label **CPU Auslastung** von den Kernen des Prozessors empfangen wird. Auch die **Versorgungsspannung** des Microcontrollers, das **Offset** (die Leistungsaufnahme des Microcontrollers im aktiven Zustand, ohne Auslastung) und **Delta P** (Steigung der Leistungsaufnahmekurve) werden benötigt. Über einen am Ausgang R angeschlossenen variablen Widerstand R wird dann die Leistungsaufnahme abstrahiert. Die Berechnung des Ersatzwiderstandswerts ist abhängig von dem gewählten Microcontroller und muss entsprechend seinem Datenblatt parametrisiert werden. Da in diesem i.d.R. keine AUTOSAR-Zustände zu finden sind, werden auf dem Datenblatt basierend eigene ECU-Zustände eingeführt und den AUTOSAR-Zuständen zugewiesen (vgl. Tabelle 6.3). Mit diesen Zuständen können nach konstante Werte für die Leistungsaufnahme der ECU angenommen oder detailliert berechnet werden. [S4]

Tabelle 6.3.: ECU-Zustände und zugehörige Leistungsbereiche (Quelle: eigene Darstellung unter Verwendung von [S4]).

| **Datenblatt** | **ECU-Zustand** | **AUTOSAR-Zustände** | **$P_{Kern}$(Auslastung)** |
|---|---|---|---|
| Active Mode | RUN | *STARTUP*, *WAKEUP*, *RUN*, *SHUTDOWN* | $P_{aktiv}$ - $P_{inaktiv}$ |
| Idle Mode | IDLE | SLEEP | $P_{inaktiv}$ |
| Power-Down Mode | OFF | OFF | $P_{aus}$ |

Der Verlauf der Leistungsaufnahme eines Kerns, in Abhängigkeit der Auslastung und des Betriebszustands der ECU $P_{Kern}(Auslastung)$, ist in Abbildung 6.21 skizziert.

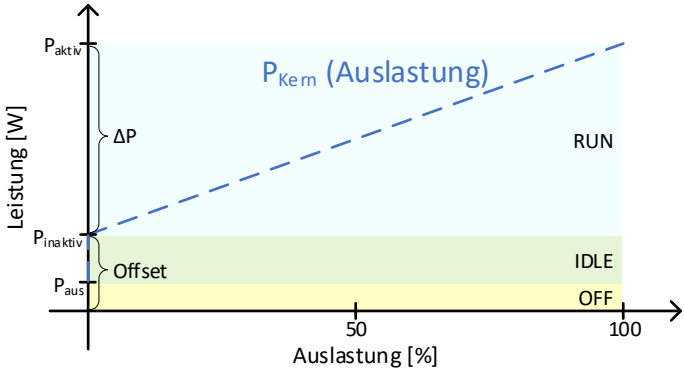



Abbildung 6.21.: Leistungsaufnahme einer ECU in Abhängigkeit ihres Betriebszustandes (Quelle: Modifikation von [P3]).

Für den Zustand RUN lässt sich die Leistungsaufnahme $\boldsymbol{P_{Kern}(Auslastung)}$ nach [S4, P3, 141] zwischen $\boldsymbol{P_{inaktiv}}$ und $\boldsymbol{P_{aktiv}}$ linearisieren. Dazu werden zunächst die Steigung $\boldsymbol{\Delta P}$ in Gleichung 6.9 sowie das Offset in Gleichung 6.10 nach [S4, P3] definiert.

$$\Delta P = P_{aktiv} - P_{inaktiv} \tag{6.9}$$

$$Offset = P_{inaktiv} \tag{6.10}$$

Die Leistungsaufnahme eines Rechenkerns $P_{Kern}$ im Zustand RUN ergibt sich nach [S4, P3] zu:

$$P_{Kern}(Auslastung) = \Delta P \cdot Auslastung + Offset \tag{6.11}$$

Der gesuchte Lastwiderstandswert R lässt sich schließlich nach [S4, P3] für alle Zustände in Abhängigkeit der Versorgungsspannung des Mikrocontrollers $U_{Mikro}$ bestimmen:

$$R = \frac{U_{Mikro}^2}{P_{ECU}(Auslastung)} \tag{6.12}$$

Hierbei ergibt sich nach [S4, P3] die Gesamtleistungsaufnahme der ECU $P_{ECU}(Auslastung)$ aus der Summe der Leistungsaufnahmen aller Rechenkerne mit der Anzahl $n$:

$$P_{ECU}(Auslastung) = \sum_{i=1}^{n} P_{Kern}(Auslastung) \tag{6.13}$$

**Limitierungen** Bei der Implementierung wird die Taktrate des Prozessors nicht explizit beachtet. Stattdessen müssen SWCs eine Laufzeit zugewiesen bekommen, die entweder auf Erfahrungswerten oder auf einer Zielhardware gemessenen Zeiten aufbauen. Eine einfache Untersuchung der Auswirkung einer Änderung der Taktrate ist daher nicht möglich. Bei der Generierung könnte der entsprechende Taktwert für die Standardlaufzeiten eingegeben und daraus ein entsprechender Faktor errechnet werden. Dies wird jedoch aufgrund der erforderlichen Kalibrierung für genaue Ergebnisse vernachlässigt.

Des Weiteren findet an den Eingängen keine Modellierung der Leistungsaufnahme statt. Da die Leistungsaufnahme zur Wandlung an den Eingängen gering ist, wird sie vernachlässigt. Durch einen Zustandsautomaten mit den Zuständen *Einlesen* und *Wandeln*, entsprechender Zeitlogik für die Übergänge und konstanten Werte für die Leistungsaufnahme, könnte der Mittelwert des Energieverbrauchs über der Zeit bestimmt werden.

Darüber hinaus findet an den Eingängen kein Buffering statt, sodass neue Werte die alten direkt überschreiben. Zur Umsetzung eines Bufferings könnten an den entsprechenden Eingängen der DSW-Blöcke Warteschlangen mit definierter Kapazität verwendet werden.

Bei der Modellierung der Leistungsaufnahme wurden auch die Ausgänge vernachlässigt. Da die Spannungsgeneratoren an den Ausgängen als eigene Quellen zu sehen sind, die vom

eigentlichen Versorgungsnetz getrennt sind, müssten die erzeugen Ströme, analog zur Leistungsaufnahme des Prozessors, mit einem Ersatzwiderstand abstrahiert werden.

Ferner hängen die Wertebereiche bei der Interpretation der ankommenden Daten und der an den Ausgängen erzeugten Spannungen bisher nicht von den logischen Datentypen ab.

Außerdem wurden bisher keine speziellen Ein- und Ausgänge wie Komparatoren, Spannungsreferenzen, Taktkristalle oder *Pulsweitenmodulation (PWM)* implementiert.

### 6.3.3. Logisches Verhalten

Verhalten lässt sich in der Version PREEvision 9.5 standardmäßig in Form von Zustandsautomaten auf entsprechenden Artefakten modellieren. Eine Abbildung von in PREEvision modellierten Zustandsautomaten nach Stateflow ist dazu im Folgenden beschrieben. Da diese jedoch Grenzen aufweist, erfolgt die Einführung einer weiteren Möglichkeit zur Verhaltensdefinition mittels Templates und der in [P3] entwickelten Template And Layer Integration Architecture (TALIA).

#### Zustandsautomaten

Zur Verhaltensmodellierung bietet PREEvision standardmäßig die Möglichkeit Zustandsautomaten zu modellieren. In einem ersten Schritt wurde die PREEvision-Modellpalette zur Erstellung von Zustandsdiagrammen dahingehend analysiert, welche Artefakte auch in Stateflow zur Verfügung stehen. Einen Überblick dazu bietet Abbildung 6.22. Da beide Werkzeuge sich bei der Implementierung der FSMs an dem UML-Standard [195] orientieren, lassen sich die meisten Elemente nahezu direkt ineinander überführen. Ein **orthogonaler Zustand** lässt sich in Stateflow als ein **Subchart** darstellen, in welchem weitere Subcharts parallel ausgeführt werden. Explizite **Endzustände** und **Verzweigungszustände** gibt es in Stateflow nicht. Da es sich bei beiden vorwiegend um ein grafisches Mittel zur visuellen Unterscheidung handelt, können diese Zustände funktional als **States** abgebildet werden. **Entry-**, **Do-** und **Exit-Aktivitäten** lassen sich in die entsprechend äquivalente **Actions** der Typen **entry:**, **during:** und **exit:** abbilden. **Transitions** -**Aktivitäten**, -**Trigger** und -**Bedingungen** werden innerhalb von Stateflow in einer Annotation zusammengefasst, bestehend aus **Actions**, **Events/Messages** und **Conditions**. Abbildung 6.23 verdeutlicht dies an einem Zustandsübergang. Stateflow-Anteile sind blau und die entsprechenden PREEvision-Anteile rot dargestellt. Es ist zu beachten, dass die Modellierung der Transitionsarten in PREEvision und Stateflow unterschiedlich ist. In Stateflow stehen für *Eingangs-* (**Input**), *Ausgangs-* (**Output**) und *lokale Daten bzw. Ereignisse* (**Local**) jeweils eigene Datentypen zur Verfügung (siehe Tabelle 6.4), wobei jede angelegte Instanz von einem Typ auch genau einem Port des Diagramms zugeordnet ist. Im Gegensatz dazu werden **Daten- und Eventinstanzen** in PREEvision entsprechenden Ports und/oder *Daten- bzw. Event Provider* zugewiesen. Diese schließen *Datenelemente*, *Signal Transmissionen* und weitere Datentypen ein. Daten- und Eventinstanzen sind lokal, wenn sie keinem Port zugewiesen sind. Die Zuweisung der Daten- und Eventinstanzen zu einem Eingangsport entspricht dann in Stateflow dem Typ **Input Data/Event**. Die Zuweisung zu einem Ausgangsport dagegen dem Typ **Output Data/Event**. Die Vorgehensweise über Dateninstanzen in PREEvision erlaubt hierbei

eine größere Flexibilität bei der Wiederverwendung eines Zustandsautomaten, verbunden mit einem höheren Modellierungsaufwand.

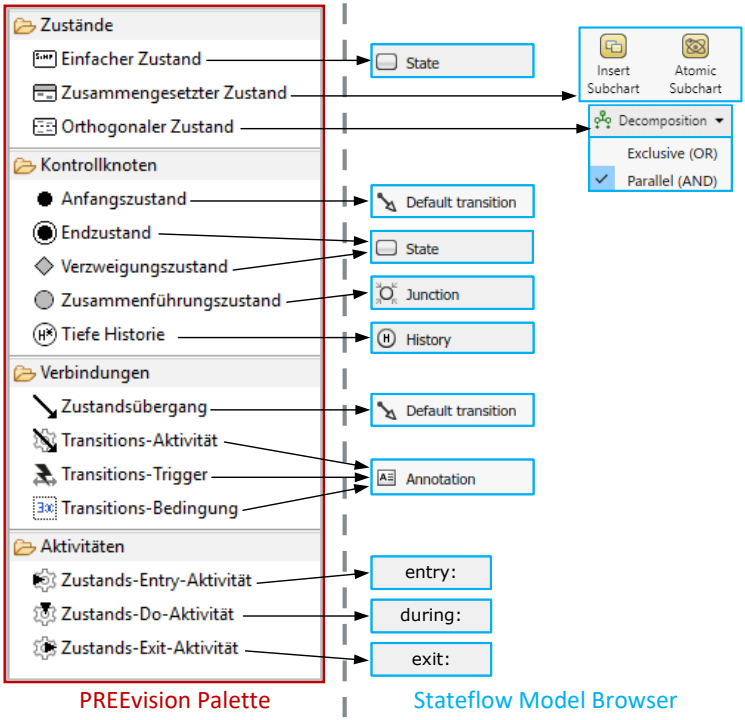



Abbildung 6.22.: Beziehungen zwischen den Artefakten aus der PREEvision-Modellpalette und den Artefakten aus dem Stateflow Model Browser.

Ein *Transitions-Trigger* wird in PREEvision mit Eventinstanzen modelliert. In Stateflow werden dagegen Event-Daten im Workspace definiert. Sie werden durch die Annotation `send(Eventname)` gesendet und lösen den Übergang aus, an dem nur `Eventname` annotiert ist. Alternativ können in Stateflow Zeitlogikausdrücke als Event verwendet werden.

Zur Modellierung von *Transitions-Bedingungen* (*Conditions*) wird in PREEvision ferner ein „Bedingungseditor“ verwendet. Hierzu werden Boolesche Ausdrücke mit Datenelementen geformt, die einem Port über ein Interface (z.b. Sender-Receiver-Schnittstelle) zugewiesen sind und zusätzlich mit einer Dateninstanz verknüpft sein müssen. D.h., dass Bedingungen nur abhängig von einem Ein- oder Ausgang modelliert werden können, also keine lokalen Daten dazu verwendet werden können.

Eine *Transitions-Aktivität* besteht aus Aktionen (*Actions*) und kann sowohl direkt auf Zuständen als auch auf Übergängen definiert werden. In PREEvision wird zwischen Aktionen vom Typ *Fire Event*, *Set Data* und *Opaque* unterschieden. Opaque-Aktionen sind generisch und auch

im UML-Standard als solche spezifiziert [195]. Einer Fire-Event-Aktion ist eine Eventinstanz zugewiesen, die entweder bei einem Zustandsübergang, Eintritt oder Austritt in einen Zustand oder während dessen Gültigkeit ausgelöst wird. Bei Set-Data-Aktionen werden an den gleichen Stellen Datenzuweisungsoperationen ausgeführt. Zur genaueren Definition muss eine Dateninstanz als Bereitsteller (*Provider*) y und eine zu setzende Dateninstanz x definiert werden. Dadurch sind Datenzuweisungsoperationen in der Form x = y möglich.

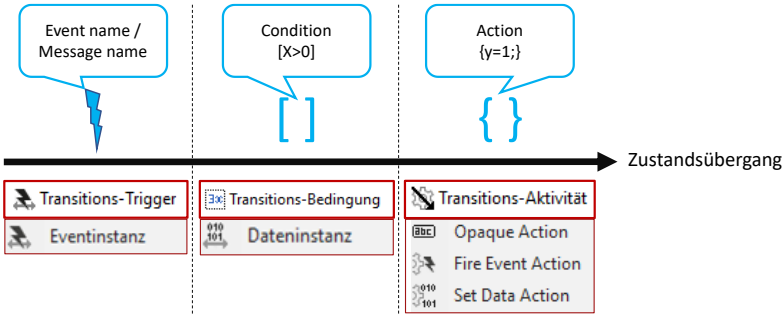



Abbildung 6.23.: Gegenüberstellung der Transitionsarten in PREEvision und Stateflow.

Tabelle 6.4.: Vergleich der Datentypen von PREEvision und Stateflow zur Modellierung von Zustandsübergängen und Aktivitäten.

| PREEvision | Stateflow |
|---|---|
| Dateninstanzen | Input, Local, Output, Constant, Parameter Data + Data Store Memory |
| Eventinstanzen | Input, Local, Output Event |
| - | Input, Local, Output Message |

**Limitierungen** *Messages* aus der UML-Spezifikation [195] sind in PREEvision im Gegensatz zu Stateflow nicht verfügbar. Sie werden daher bei der Abbildung nicht betrachtet.

Des Weiteren ist bei der Modellierung von Events mit den PREEvision-Transitions-Triggers keine Verwendung von Zeitlogik möglich. Als Lösung für diese Arbeit wird Zeitlogik in PREEvision String-basiert als Name von Eventinstanzen modelliert. Dies liegt daran, dass die Syntax der Zeitlogik nicht standardisiert ist und damit nicht für mögliche andere Simulatoren einsetzbar. Eine weitere Lösung wäre der Einsatz von UML-Timing-Diagrammen.

Da die Modellierung von Bedingungen in PREEvision ferner an Ports gebunden ist, wird eine Lösung für lokale Variablen benötigt, da diesen kein Port zugewiesen ist. Eine Zwischenlösung wäre es für jede lokale Variable sowohl einen Ausgang als auch einen Eingang anzulegen und diese über eine Rückführungsschleife zu verbinden. Neben dem erhöhten Modellierungsaufwand und der Fehleranfälligkeit, bliebe durch diesen Ansatz jedoch das Problem bestehen, dass die Bedingungen nur in Boolescher Form, ohne mathematische Ausdrücke der Form

`(a+b)<c`, modellierbar sind. Gleiches gilt auch bei den Set-Data-Aktionen für die Modellierung mathematischer Operationen. Hier kommt noch hinzu, dass immer nur eine einzige bereitstellende Dateninstanz definiert werden kann. Eine Erweiterung des Bedingungseditors und die Entwicklung eines „Operationseditors“ ist jedoch nicht Fokus dieser Arbeit. Daher werden auch zur Beschreibung von Bedingungen und mathematischen Ausdrücken Strings verwendet. Sie werden in Form der Namen von Bedingungen und Set-Data-Aktionen hinterlegt.

Außerdem muss beachtet werden, dass einem PREEvision-Artefakt mehrere Zustandsautomaten zugewiesen sein können. In der prototypischen Generierung (vgl. Unterabschnitt 6.4.1) wird dies durch das Erstellen eines Subsystems für jedes Zustandsautomatendiagramms gelöst. Zeitgleiche Zugriffe auf ggf. gemeinsame Ein- und Ausgänge der Komponenten werden dabei nicht unterbunden und können zu Fehlern führen.

### Templates

Templates sind in PREEvision Stellvertretermodelle von generischen und parametrisierbaren *Simulink Masked Subsystems*, die eine geschlossene Funktionseinheit bilden. Auf diese Weise kann eine Verhaltensmodellierung mit einer Bibliothek aus geprüften und bzgl. ihrer Laufzeit optimierten Teilmodellen realisiert werden, um den E/E-Architekten den eigentlichen Modellierungs- und Validierungsaufwand abzunehmen. Dies soll Zeit sparen und eine schnelle Rückmeldung mittels Simulation für Designentscheidungen ermöglichen. Außerdem soll dadurch sowohl die Genauigkeit erhöht werden als auch die Skalierbarkeit ermöglicht werden (vgl. Anforderungen 3 und 4).

**Realisierung** Eine Verknüpfung zwischen PREEvision-Artefakten und Simulink-Masked-Subsystems wird mittels einer in [P3] im Zuge dieser Arbeit eingeführten Template And Layer Integration Architecture (**TALIA**) realisiert. Das Prinzip der TALIA ist beispielhaft nach [P3] in Abbildung 6.24 für einen Aktor auf der HA dargestellt.

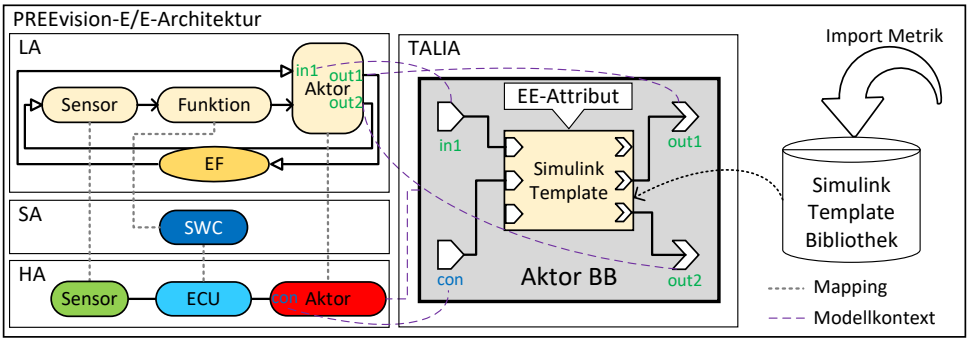



Abbildung 6.24.: Template And Layer Integration Architecture (TALIA) zur Verknüpfung von PREEvision-Artefakten mit Simulink-Masked-Subsystems (Quelle: Modifikation von [P3]).

Voraussetzung für den Einsatz der TALIA sind in PREEvision vorliegende Stellvertreter von Simulink-Templates. Diese Stellvertreter werden nach [P3] mit einer entwickelten PREEvision-Metrik (PM) erzeugt. Nach Angabe des Dateipfades wird damit eine im *.slx* Format vorliegende Simulink-Bibliothek eingelesen. Die sich darin befindenden Entitäten werden dann in logische PREEvision-Funktionsblöcke (angelehnt an [43]) übersetzt und in einem logischen Funktionspaket (**Simulink Template Bibliothek**) gespeichert. Tabelle 6.5 gibt einen Überblick zur Abbildung der Komponenten beim Einlesen.

Tabelle 6.5.: Import von Simulink-Masked-Subsystems in PREEvision als Templates (Quelle: [P3]).

| **Simulink** | **PREEvision** |
|---|---|
| Subsystem (Template) | Logische Funktion |
| Parameter (Mask) | EE-Attribut |
| Inport / L Connection Port | Logischer Required Port |
| Outport / R Connection Port | Logischer Provided Port |

Mittels **EE-Attributen** werden die Standardwerte der **Parameter** eines Simulink-Masked-Subsystems im PREEvision-Modell gespeichert. Diese können in PREEvision geändert werden, sodass bei einer von PREEvision ausgehenden Generierung die entsprechenden Parameter in Simulink übernommen werden. Zu den Parametern zählt auch die *Bibliotheksreferenz*, welche bei der Generierung benötigt wird. Diese Referenz gibt den Pfad des Templates (Simulink-Masked-Subsystem) innerhalb der verwendeten Simulink-Bibliothek an. Ein- und Ausgänge aus Simulink (**Inports** und **Outports**) werden in **Logische Required Ports** und **Logische Provided Ports** übersetzt. Da es auf der LA keine richtungsunabhängigen Ports gibt, gilt diese Abbildungsvorschrift auch für **Connection Ports**. Nachdem die Template-Bibliothek erstellt ist, sind die Verknüpfungen der PREEvision-Artefakte mit den darin enthaltenen Templates zu realisieren. HWCs (außer die in Unterabschnitt 6.3.1 beschriebenen Leitungssatzkomponenten), SWCs und LA-Blöcke benötigen zur Modellierung dieser Verknüpfung ein entsprechendes Diagramm. Da ein solches standardmäßig nicht zur Verfügung steht, wird die TALIA als Zwischenebene eingeführt. Dabei werden Building Blocks (BBs) als Diagramme verwendet. Sie sollen die Informationen aus mehreren PREEvision-Ebenen integriert darstellen und die Verknüpfung mit den Templates ermöglichen. Um die benötigten Informationen aus den Ebenen in den TALIA-BBs zusammenzufassen, wurde eine weitere PM entwickelt. Hierbei werden zunächst die mit einem Template zu verknüpfenden PREEvision-Artefakte eingelesen. Im abgebildeten Beispiel ist dies der **Aktor** auf der HA. Nach dem Einlesen wird ein BB auf einer zusätzlichen Produktlinie erzeugt und über einen **Modellkontext** mit dem eingelesenen Ausgangsartefakt (im Beispiel der Aktor auf der HA) verbunden. Der BB erhält dabei die Ports des Ausgangsartefakts mit entsprechender Richtung und in logischer Form. Im Beispiel ist dies der Eingangsport **con**. Die so erzeugten BB-Ports sind ebenso per *Modellkontext* mit ihren ursprünglichen Ports verknüpft. Sind die Ausgangsartefakte ferner per **Mapping** mit einer logischen Komponente auf der LA verbunden (im Beispiel ist dies der Aktor auf der LA), so werden dem BB zusätzlich die Umgebungsports der logischen Komponenten hinzugefügt. Im Beispiel sind das die Ports **in1**, **out1** und **out2**. In den anschließend auf der TALIA-Produktlinie vorhandenen BB-Hüllen (im Beispiel **Aktor BB**) können nun die in der

Bibliothek als logische Funktionsblöcke vorliegenden **Simulink-Templates** eingefügt und mit den intern vorhandenen Ports verknüpft werden. Für SWCs als Ausgangsartefakte ist das Vorgehen analog. Allerdings werden den erzeugten BBs keine Umgebungsports aus der LA hinzugefügt. Bei Blöcken der LA als Ausgangsartefakte werden nur die Umgebungsports bei der Erzeugung der BBs beachtet. [P3]

**Limitierungen** Durch den Einsatz von Templates kann es zu Konvertierungsfehlern und Dateninkonsistenzen an den Ports kommen. Besonders besteht bei der prototypischen Umsetzung die Gefahr, dass auf der TALIA ein normaler Simulink-Port und ein *Connection Port* miteinander verbunden werden, da sie beide in PREEvision als *Logischer Port* repräsentiert sind. Dies würde im Simulationsmodell aber nicht funktionieren und bei der Generierung zu einem Fehler führen. In der Praxis müssten daher entsprechende Portunterscheidungen und zulässige Datentypen in PREEvision modelliert werden können.

Anders als bei Leitungssatzkomponenten mit einer festen Zuordnung (z.B. Stecker), ist bei der Generierung mit Templates eine erweiterte Modellpropagation über mehrere Modellkontexte notwendig. Da die Verknüpfungen der PREEvision-Artefakte mit Templates über die TALIA realisiert werden und nicht starr in einem Metamodell definiert sind, können Inkonsistenzen auftreten, wenn Modellteile verschoben oder kopiert werden.

Außerdem ist zu beachten, dass bei der template-basierten Generierung des Verhaltens von SWCs mit verknüpften Umgebungsports zusätzliche Ein- und Ausgänge bei der ECU-Komponente, der sie zugewiesen sind, vorhanden sein müssten. Eine SWC ist jedoch in der Realität nicht direkt mit der Umwelt verbunden, sondern nimmt diese über Sensoren wahr. Daher müssen Sensoren oder Pseudo-Sensoren die Umgebungssignale zunächst in logische Signale wandeln. Die Abbildung sieht damit keine Dekoration von Umgebungsports an TALIA-BBs vor, deren Basis eine SWC ist.

Eine weitere Limitierung ist, dass SESPS-Komponenten teilweise nur einen Busausgang (*m*) für mehrere Messsignale besitzen. Beim Batteriemodell werden z.B. der Ladezustand, der aktuelle Entladestrom und die Spannung in einem solchen m-Ausgang zusammengefasst. Die einzelnen Signale des m-Ausgangs können jedoch nicht direkt innerhalb der Templates verwendet werden, weil eine entsprechende Auswahl der relevanten Signale auf der TALIA nicht realisiert werden kann. Entsprechend müssen Templates so angepasst sein, dass sie alle relevanten Ausgangssignale einzeln, also ohne Zusammenfassung in einem m-Ausgang, bereitstellen.

## 6.4. Implementierung

PREEvision nutzt als Dateiformat *.eaa* und Simulink das *.mdl* bzw. seit Version R2012a das *.slx* Format. Eine direkte Modell-zu-Modell-Transformation gestaltet sich aufgrund der proprietären und von Mathworks entwickelten Datenformate *.mdl* und .slx schwierig (siehe Unterabschnitt 6.4.2). Daher wird zur Generierung von .slx-Modellen eine Generierungs-PM nach dem Grundprinzip von [43] entwickelt. Die einzelnen Schritte, von der Auswahl einer geeigneten Schnittstelle bis zur Implementierung, sind in den folgenden Unterabschnitten dargestellt.

### 6.4.1. Simulationsmodellgenerierung mittels PREEvision-Metrik

Abbildung 6.25 zeigt das Prinzip der PM zum Erzeugen von Simulink-Modellen aus PREEvision. Zunächst werden die an der Generierung beteiligten HWCs einem **Model Context Block** hinzugefügt. Dieser übergibt die Auswahl bei der PM-Ausführung an einen **Custom Metric Block**, der den Generierungsalgorithmus in Form von Java-Code implementiert. Hier wird über die Auswahl der HWCs iteriert und das Simulationsmodell rekursiv, von der niedrigsten bis zur obersten Hierarchieebene, aufgebaut. Sobald jeweils die Blöcke einer Ebene erzeugt wurden, werden sie miteinander vernetzt. Über die Propagation von Mappings und von Verknüpfungen zwischen den Artefakten per Modellkontext können weitere, benötigte Artefakte hinzugefügt werden. Die Details werden in den folgenden Abschnitten beschrieben.

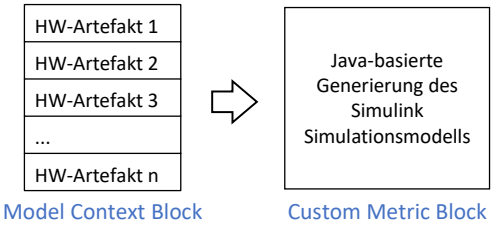



Abbildung 6.25.: Prinzip der PREEvision-Metrik zur Generierung von Simulink-Modellen.

### 6.4.2. Definition einer Schnittstelle zwischen PREEvision und Simulink

Um Simulink-Modelle automatisiert aus den PREEvision-Artefakten zu generieren, gibt es mehrere Wege, die im Folgenden dargestellt werden.

Eine direkte Modell-zu-Modell-Transformation von .eea zu .mdl/.slx ist für eine prototypische Implementierung aus mehreren Gründen nicht sinnvoll. Das Format .mdl ist älter und erlaubt kein inkrementelles Laden des Modells oder lokale Modelländerungen. Es muss also immer die komplette Datei geladen und geschrieben werden, was sich negativ auf die Performanz auswirkt. Das .slx-Format dagegen basiert auf dem Containerformat *Open Packaging Conventions (OPC)* [130], welches eine oder mehrere XML-Dateien enthält. Modelle von Simulink, Stateflow, SimEvents, etc. sowie die Ausführungskonfiguration werden dabei in getrennten Dateien, innerhalb des Containers, beschrieben. Zum Einlesen bzw. Parsen der Formate in Java gibt es bereits die *ConQAT Simulink Library* [100]. Allerdings können Modelle hier nur gelesen und als *Plain Old Java Object (POJO)* verpackt werden. Einen Export mit zugehöriger Bibliothek gibt es nicht. Da beide Formate proprietär und nicht ausreichend dokumentiert sind, müsste ein entsprechender Exporter über Reverse-Engineering implementiert werden. Das verursacht einen hohen Aufwand und ist darüber hinaus schlecht wartbar, da Änderungen der Formate mit dem Erscheinen neuer Versionen von Matlab/Simulink eingepflegt werden müssen. Als Alternative bietet Mathworks mit der ME-API-J eine eigene Lösung an, um Modelle programmatisch in Java zu erstellen, einzulesen oder zu ändern. Hierzu werden Matlab-Kommandos aus Java-Programmen heraus im String-Format an Matlab gesendet.

Ferner gibt es das *Massif-Framework* zur modellbasierten Integration von Simulink in Eclipse [102]. Hierbei steht ein E-Core-Metamodell von Simulink im *Eclipse Modeling Framework (EMF)* zur Verfügung. Mit dessen Hilfe ist die grafische Modellierung und Generierung von Simulink-Stellvertretern möglich. Bei der Generierung werden die EMF-Modellinformationen verwendet und in entsprechende Matlab-Kommando-Strings übersetzt, die dann über die ME-API-J an Matlab gesendet werden. Um also eine Transformation des PREEvision-Modells zu erreichen, müsste dieses zunächst in ein entsprechendes EMF-Modell transformiert werden. Dabei ist jedoch zu beachten, dass das Massif-Framework generisch ist und keine Bibliothek mit Simulink-Stellvertreterblöcken sowie ihren zugehörigen Parametern vorhanden ist. Außerdem müsste es für die Unterstützung von Blocksets, wie Simscape, manuell erweitert und gepflegt werden.

Eine weitere Alternative stellt das *Epsilon-Framework* [75] dar, welches als Eclipse Plugin zur Verfügung steht. Basis davon ist die für das Framework entwickelte Sprache *Epsilon Object Language (EOL)*. Dabei handelt es sich um eine einheitliche Skriptsprache für unterschiedliche Modellbeschreibungssprachen, wie EMF, XML oder Simulink. Quelltext 6.1 zeigt ein beispielhaftes EOL-Skript, um zwei Simulink-Blöcke zu erstellen, ihnen eine Position im Diagramm zuzuweisen und sie miteinander zu verbinden.

```
1  // Erstelle zwei Blöcke (sineWave und gain):
2  var sineWave = new 'simulink/Sources/Sine Wave';
3  var gain = new 'simulink/Math Operations/Gain';
4  // Lege die Position der Blöcke im Diagramm fest:
5  sineWave.position = "[100 100 130 130]";
6  gain.position = "[200 100 230 130]";
7  // Verbinde beide Blöcke:
8  sineWave.link(gain);
```

Quelltext 6.1: Beispiel zum Aufbau von Simulink-Modellen mit EOL.

Es wird ersichtlich, dass für das Anlegen der Blöcke die Bibliothekspfade übergeben werden müssen (z.B. **'simulink/Sources/Sine Wave'**). Die Generierung des Simulink-Modells erfolgt auch hier über die ME-API-J, wobei entsprechende Kommandos erzeugt werden. Verwendet man die EOL als Mittelsprache müsste jedoch ein weiterer Generator geschrieben werden, der das PREEvision-Modell zunächst in die EOL überführt. Dafür wird wiederum eine Bibliothek benötigt. Ferner wird bei der Implementierung des Modellgenerators eine Abhängigkeit zur EOL geschaffen, die kein offizieller Standard ist.

Ein Vergleich aller dargestellten Möglichkeiten zeigt, dass die kommandobasierte Verwendung der ME-API-J der zentrale Bestandteil aller Lösungen ist. Das Massif-Framework bietet eine grafische Modellierungsmöglichkeit für Simulink-Modelle, während beim Epsilon-Framework Skripte zur Modellerstellung verwendet werden. In beiden Fällen werden bei der Implementierung eines Generators jedoch Abhängigkeiten eingegangen. Außerdem wäre der Aufbau einer eigenen Bibliothek nötig. Daher soll eine eigene Lösung auf Basis der Verwendung der ME-API-J geschaffen werden. Diese wird in Unterabschnitt 6.4.3 beschrieben.

### 6.4.3. Generierung von Java-Stellvertretern für Simulink-Komponenten

In diesem Abschnitt erfolgt zunächst eine Analyse der Struktur und der Zusammenhänge zwischen Simulink und den verwendeten Blocksets Stateflow, Simscape und SimEvents. Daraus wird ein Klassendiagramm für Simulink-Modelle mit Pseudo-Funktionen abgeleitet, um anschließend damit geeignete Klassendiagramme für Java-Stellvertretermodelle erstellen zu können. Im Anschluss wird die Code-Generierung beschrieben.

#### Analyse der Struktur von Simulink und den verwendeten Blocksets

Abbildung 6.26 zeigt ein Klassendiagramm, dass auf Basis des Massif-EMF-Modells [102] (vgl.

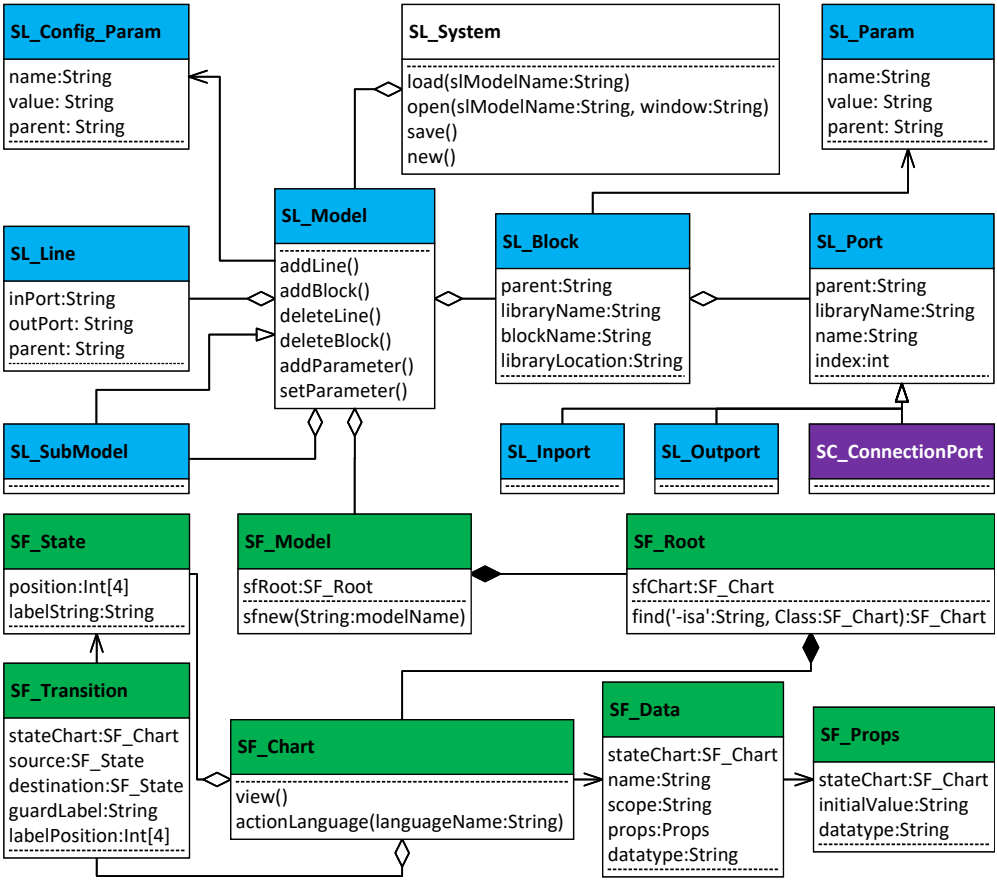



Abbildung 6.26.: Elemente und Zusammenhänge von Simulink, Stateflow und Simscape.

Unterabschnitt 6.4.2) und einer Analyse der Simulink-Dokumentation zum programmatischen Aufbau von Modellen [267] entstanden ist. Es stellt die Hauptelemente eines Simulink-Modells (**SL**, blau) mit den Erweiterungen Stateflow (**SF**, grün) und Simscape (**SC**, lila) dar. Außerdem zeigt es die Beziehungen der Elemente untereinander, ihre grundlegenden Methoden und Attribute. Für SimEvents sind keine zusätzlichen Klassen nötig. Das Hauptelement bildet zunächst das Simulink-System (**SL_System**), welchem genau ein Modell (**SL_Model**) zugeordnet ist. Ein Modell besteht wiederum aus Blöcken (**SL_Block**) mit Ein- und Ausgängen (**SL_Inport** und **SL_Outport**), die miteinander über Leitungen (**SL_Line**) verbunden sind. Eine Sonderform von (**SL_Port**) ist der *Simscape Connection Port* (**SC_ConnectionPort**) für physikalische Modelle, der Ein- und Ausgang zugleich ist. Er lässt sich nur mit Ports der gleichen Art verbinden. Zur Realisierung einer Hierarchie können einem Modell ferner Submodelle (**SL_Submodel**) zugeordnet sein. Ferner besitzen das Modell und die Blöcke Parameter (**SL_Config_Param** bzw. **SL_Param**). Diese sind durch einen Namen, einen Modellpfad und einen Wert definiert. Beim SL_Model wird z.B. der Solver über solche Parameter festgelegt.

Neben Blöcken können dem SL_Model auch Stateflow Anteile zugeordnet werden. Dafür wird analog zum SL_Model ein Stateflow-Modell (**SF_Model**) benötigt. Ein SF_Model braucht dann wiederum ein Hauptdiagramm (**SF_Root**) in dem die Zustandsautomaten (**SF_Chart**) modelliert werden. SF_Charts bestehen aus Zuständen (**SF_State**) und Zustandsübergängen (**SF_Transition**). Ihnen sind ferner Konfigurationsdaten (**SF_Data**) zugeordnet, welche mittels Eigenschaften (**Props**) definiert werden. So können die erlaubten Datentypen eingeschränkt oder Wertebereiche dafür festgelegt werden.

#### Ableitung nötiger Java Klassen zum Aufbau von Simulink-Stellvertretermodellen

Da ein programmatischer Aufbau von Simulink-Modellen anhand von String-Befehlen (vgl. Unterabschnitt 2.4.3) nicht objektorientiert, fehleranfällig und schwer wartbar ist, wurde das in Abbildung 6.27 dargestellte Klassendiagramm zum Aufbau von Simulink-Stellvertretermodellen mit Java-Klassen aus Abbildung 6.26 abgeleitet. Die Hauptklasse ist der Singleton **SimulinkSystem**. Analog zu einem Simulink-System (vgl. *SL_System* in Abbildung 6.26) werden hier die Java-Stellvertreter der Simulink-Blöcke (**SimulinkBlock** und **SimulinkSubSystem**) einer Liste (**blockList**) hinzugefügt. Bei Subsystemen (**SimulinkSubSystem**) werden Ein- und Ausgänge (**SimulinkSubSystemPort**) ebenfalls wie Blöcke betrachtet und der blockList des SimulinkSubSystems hinzugefügt. Die Verbindungen zwischen den Blöcken (**SimulinkRelation**) werden in der Liste **relationList** abgelegt. Einer SimulinkRelation sind immer ein Eingangs- und ein Ausgangsport zugewiesen. Diese sind jeweils vom Typ **SimulinkPort** bzw. **SimulinkSubSystemPort**. Komplexere Verbindungen für Kabel und Bussysteme werden in weiteren Listen (**connectorList** und **conductorList**) gespeichert. Für diese Verbindungen sind Propagationen im PREEvision-Modell nötig, um eine entsprechende SimulinkRelation zu erstellen. Des Weiteren werden Parameter (**SimulinkParameter<T>**) zur Konfiguration der Simulink-Modellausführung in der Liste **paramters** abgelegt.

Die Modellgenerierung wird durch den Aufruf der Methode `generateModel()` in der Klasse SimulinkSystem gestartet. Zunächst werden die Blöcke erstellt. Dazu wird über die sich in der blockList befindenden Blöcke iteriert und rekursiv deren `generateModel()`-Methode aufgerufen. Anschließend werden die Verbindungen generiert, indem in allen Blöcken über die

relationList iteriert wird. Analog wird bei den Parametern vorgegangen. Mithilfe der Klasse **CommandUtil** werden die benötigten Matlab-Befehle (Tabelle 2.2) erzeugt und über die **MatlabEngine** (siehe Unterabschnitt 2.4.3) an Matlab gesendet.

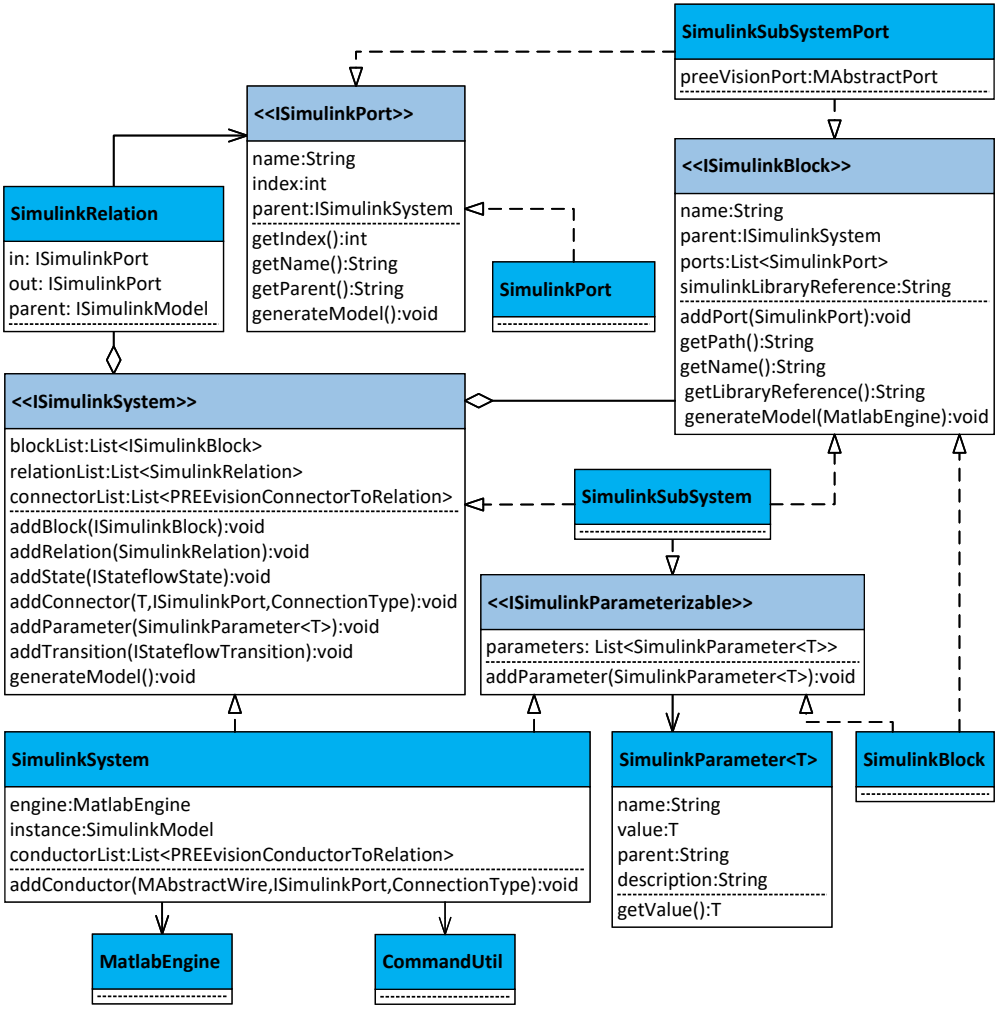



Abbildung 6.27.: Klassendiagramm von Java-Stellvertretern für Simulink.

### Ableitung nötiger Java Klassen zum Aufbau von Stateflow-Stellvertretermodellen

Abbildung 6.28 zeigt das in dieser Arbeit entstandene Klassendiagramm zum Aufbau von Stateflow-Stellvertretermodellen.

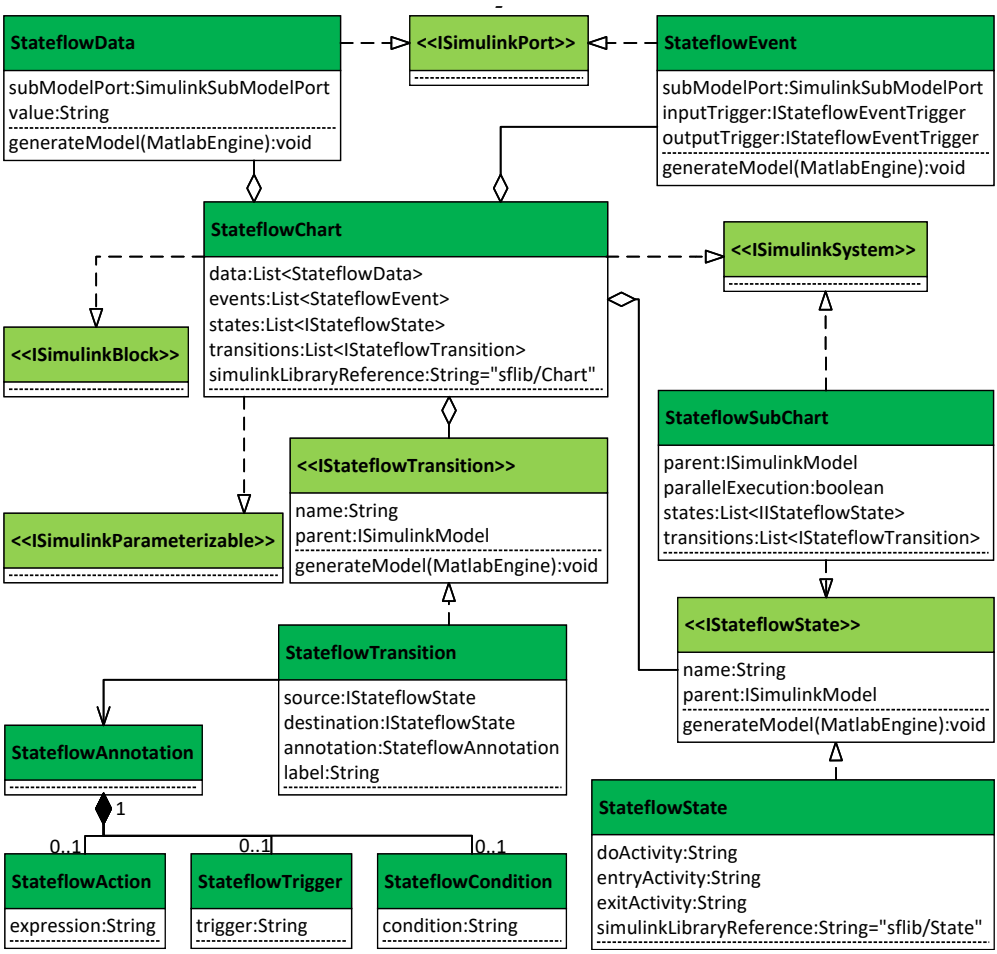



Abbildung 6.28.: Klassendiagramm von Java-Stellvertretern für Stateflow.

Stateflow-Zustandsautomaten (**StateflowChart**) haben mit **«ISimulinkBlock»** zunächst dieselbe Schnittstelle, wie ein *«ISimulinkSubSystem»* (vgl. Abbildung 6.27). Sie werden analog über die `generateModel()`-Methode generiert. Da sich ein *Simulink-Chart* im Aufbau jedoch grundlegend von einem *Simulink-Subsystem* unterscheidet, sind zusätzliche Java-Klassen und Attribute nötig. Wie in Unterabschnitt 2.4.3 beschrieben, bestehen Charts aus Zuständen und

Übergängen. Für diese werden entsprechende Stellvertreter (**StateflowState** und **StateflowTransition**) benötigt. StateflowStates und StateflowTransitions sind einem StateflowChart innerhalb der Listen **states** und **transitions** zugeordnet. Ein- und Ausgänge von StateflowCharts werden als Datum (**StateflowData**), Ereignis (**StateflowEvent**) oder Nachricht (*StateflowMessage* - nicht abgebildet) beschrieben. Sie sind dem StateflowChart ebenfalls in Form von Listen (**data** und **events**) zugeordnet. Da alle einem StateflowChart zugeordneten StateflowStates und deren Verfeinerungen (**StateflowSubCharts**) dieselben Ein- und Ausgänge haben, sind ihnen selbst weder StateflowData noch StateflowEvents zugeordnet. Eine parallele Ausführung von StateflowSubCharts wird über das Boolesches Flag **parallelExecution** gesteuert. Den StateflowStates können im Vergleich zu den StateflowSubCharts Aktivitäten (**doActivity**, **entryActivity** und **exitActivity**) zugeordnet sein.

Nach dem Aufruf der `generateModel()`-Methode des StateflowCharts werden zunächst die **«IStateflowStates»** durch Aufruf ihrer `generateModel()`-Methode in rekursiver Weise erstellt. Nach dem Erstellen aller «IStateflowStates» in einer Hierarchiebene, werden sie über StateflowTranstions miteinander verknüpft. Dafür werden die Attribute **source** für den Quell-«IStateflowState» und **target** für den Ziel-«IStateflowState» verwendet. Jeder StateflowTranstion ist ferner eine **StateflowAnnotation** zugewiesen. Diese besteht aus einer **StateflowAction**, einem **StateflowTrigger** und einer **StateflowCondition**.

#### Codegenerator zum Aufbau einer Simulink-Stellvertreterbibliothek in Java

Mithilfe der in Abbildung 6.27 dargestellten Schnittstelle «ISimulinkBlock» lassen sich konkrete Java-Stellvertreter der Blöcke aus einer bestehenden Simulink-Bibliothek automatisiert, mittels Code-Generator, implementieren. Das Ziel davon ist der Aufbau einer Java-basierten Skriptsprache zur automatisierten Erstellung von Simulink-Modellen aus Java-Programmen heraus. Simulink-Blöcke unterscheiden sich im Wesentlichen in:

1. der Anzahl und Art ihrer Ein- und Ausgänge (*SimulinkPorts*)
2. der Anzahl und Art ihrer Parameter (*SimulinkParameter*)
3. ihrem Bibliothekspfad (*simulinkLibraryReference*)

Die Informationen dazu lassen sich aus Simulink-Bibliotheksdateien im .slx Format gewinnen. Hierbei liegen die Blöcke entweder nach Kategorien sortiert, wodurch eine Baumstruktur entsteht, oder zusammenhangslos in der Bibliotheksdatei vor.

Das Vorgehen zur Code-Generierung für eine Simulink-Stellvertreterbibliothek ist wie folgt. Mithilfe der ME-API-J wird die Simulink-Bibliothek zunächst innerhalb eines Java-Programms (Stellvertreterbibliotheksgenerator) geöffnet. Anschließend wird über ihren Inhalt iteriert. Hierbei werden Parameter, Ports sowie der Bibliothekspfad ausgelesen und entsprechend der Dateistruktur (z.B. Baum) in einem Objekt namens `SimulinkLibraryTree` zwischengespeichert. Die Elemente haben dabei Referenzen auf ihre Eltern und Kinder, wodurch sich der Baum traversieren lässt. Dadurch ist eine Filterung zum Entfernen von Duplikaten möglich. In einem weiteren Schritt werden aus den Elementen des SimulinkLibraryTree dann die Java-Bibliotheksklassen als Kinder der Klasse **SimulinkBlock** (siehe Abbildung 6.27) erzeugt. Hierzu werden die AST-Werkzeuge aus dem Standard-Eclipse-Paket

`org.eclipse.jdt.core.dom` verwendet. Der prinzipielle Aufbau einer so erzeugten Klassen ist in Quelltext 6.2 dargestellt.

```
1   public class Blockname extends SimulinkBlock {
2
3     private SimulinkInPort in_1 = new SimulinkInPort("InPort", 1, this);
4     private SimulinkOutPort out_1 = new SimulinkOutPort("OutPort", 1, this);
5     public final String PARAMETER_NAME = "ParameterName";
6     public final String LIBRARY_REFERENCE = "library/.../Blockname";
7
8     public SimulinkInPort in_1() {
9       return in_1;
10    }
11
12    public SimulinkOutPort out_1() {
13      return out_1;
14    }
15  }
```

Quelltext 6.2: Beispiel einer generierten Java-Klasse als Simulink-Block-Stellvertreter.

**Blockname** ist hierbei stellvertretend für den Klassennamen des Simulink-Blocks, wie bspw. "Gain". Für jeden Ein- und Ausgang wird dann eine globale Instanz des entsprechenden Ports angelegt und ihr ein Name sowie ein Index zugewiesen (Zeile 3-4). Dazu bekommt auch jeder Port eine Rückgabemethode für den öffentlichen Zugriff zugewiesen (Zeilen 8-14). Außerdem wird für jeden Parameter entsprechend seinem Namen ein String-Literal angelegt (Zeile 5). Über entsprechende `get()`- und `set()`-Methoden kann dann auf die in der Elternklasse als Liste vorliegenden Parameter zugegriffen werden. Des Weiteren wird der Bibliothekspfad als String-Literal **LIBRARY_REFERENCE** (Zeile 6) angelegt und der Elternklasse über den Konstruktor (nicht dargestellt) übergeben.

### Java-basierter Aufbau von Simulink-Modellen

Quelltext 6.3 zeigt beispielhaft die Anwendung und die Syntax zum Java-basierten Aufbau von Simulink-Modellen mit der zuvor erstellten Stellvertreterbibliothek.

```
1  //Anlegen eines neuen Modells
2  String currentDir = System.getProperty("user.dir");
3  String systemName = "test";
4  MatlabEngine engine = MatlabEngine.startMatlab();
5  SimulinkSystem simulinkSystem = SimulinkSystem.getInstance(engine, systemName);
6  simulinkSystem.createNewSystem(currentDir);
7  //Aufbau von Modellen mit Simulink-Blöcken
8  Constant constant = new Constant("Constant", simulinkSystem);
9  constant.setParameterValue(constant.VALUE, 12.0);
10 Scope scope = new Scope("Scope", simulinkSystem);
11 new SimulinkRelation(constant.out_1(), scope.in(), simulinkSystem);
12 //Verwendung von Sub-Systemen
13 SimulinkSubSystem subSystem = new SimulinkSubSystem("SubSystem", simulinkSystem);
```

```
14  new SimulinkSubSystemInPort("In", subSystem);
15  new SimulinkSubSystemOutPort("Out", subSystem);
16  //Aufbau von Stateflow Modellen
17  StateflowChart sfParentChart = new StateflowChart("ch", simulinkSystem);
18  StateflowState sfState = new StateflowState("st", sfParentChart);
19  StateflowState sfState2 = new StateflowState("st2", sfParentChart);
20  StateflowTransition transition = new StateflowTransition("transitionName",
        sfParentChart, sfState, sfState2);
21  transition.setLabel("u==2");
22  //Generierung des Modells und Speicherung
23  simulinkSystem.generateModel();
24  simulinkSystem.saveSystem();
25  engine.close();
```

Quelltext 6.3: Java-basierter Aufbau von Simulink-Modellen.

Um ein neues Modell zu erstellen, müssen zunächst der Speicherpfad (**currentDir**) sowie der Name (**systemName**) des Modells definiert werden (Zeile 2-3). Auch eine Instanz der ME-API-J (**engine**) wird benötigt (Zeile 4), um ein **SimulinkSystem** zu instanziieren und vorzubereiten (Zeile 5). Danach können dem SimulinkSystem Blöcke hinzugefügt werden, indem das SimulinkSystem im Konstruktor des Blocks als Container übergeben wird (vgl. Zeile 8 und Zeile 10).

Zeile 9 zeigt beispielhaft, wie der Wert des Parameters **VALUE** vom Block **constant** auf **12.0** gesetzt wird. Der Zugriff auf die Parameter ist möglich, da sie in den jeweiligen Block-Klassen als `public` deklariert sind. Jeder generierte Block besitzt außerdem eine `setParameterValue()`-Methode, welche die benötigten Matlab-Befehle zum Setzen von Parametern generiert.

Verbindungen zwischen zwei Ports werden durch das Anlegen einer **SimulinkRelation** realisiert (Zeile 11). Hierbei werden dem Konstruktor Aus- und Eingangsport der zu verbindenden Blöcke sowie der Container (simulinkSystem) übergeben.

Zeile 13-15 zeigen, wie Subsysteme sowie deren Ein- und Ausgänge angelegt werden. Die Indizierung erfolgt in der Erstellungsreihenfolge, kann aber angepasst werden.

Der grundlegende Aufbau von Stateflow-Modellen ist in den Zeilen 17-20 beispielhaft dargestellt. Das Anlegen der Komponenten erfolgt nach dem gleichen Prinzip, wie bei den Simulink-Blöcken. Statt SimulinkRelations werden jedoch **StateflowTransitions** zur Modellierung der Beziehungen zwischen den **StateflowStates** benötigt. Diese bekommen neben einem Namen (**"transitionName"**) und einem Container (**sfParentChart**) sowohl den Quellzustand (**sfState**) als auch den Zielzustand (**sfState2**) im Konstruktor übergeben. Ferner wird in Zeile 21 über die Methode `setLabel()` ein Label als String zur Definition der Übergangsbedingung gesetzt. Alternativ kann dies über das Anlegen und Zuweisen einer *StateflowAnnotation* erfolgen (nicht dargestellt, siehe Abbildung 6.28).

In Zeile 23 wird die `generateModel()`-Methode ausgeführt (siehe Unterabschnitt 6.4.3), wobei die entsprechenden Matlab-Befehle für den rekursiven Aufbau des Simulink-Modells ausgeführt werden. Zum Schluss wird das Simulink-Modell als .slx gespeichert (Zeile 24) und die Verbindung zu Matlab getrennt (Zeile 25).

# 7. Anwendungsfallbasierte Evaluation der Konzepte und Simulatoren

In diesem Kapitel werden zunächst geeignete Anwendungsfälle für die Analyse der Skalierbarkeit der E/E-Architektursimulation ausgewählt. Dazu werden Auswahlkriterien und Einschränkungen definiert, die einerseits die Suche erleichtern sollen und andererseits die Beantwortung der Forschungsfragen ermöglichen. Im Anschluss werden Simulationen aus den Anwendungsfällen abgeleitet und mit Ptolemy II sowie Simulink ausgeführt. Anhand der Ergebnisse erfolgt die Bewertung der Skalierbarkeit hinsichtlich Performanz, Nutzen, Anwendbarkeit und Validierbarkeit.

## 7.1. Identifikation von Anwendungsfällen

Das aus einem E/E-Architekturmodell abgeleitete Simulationsmodell wird maßgeblich durch den Anwendungsfall bestimmt (vgl. Abbildung 7.1).

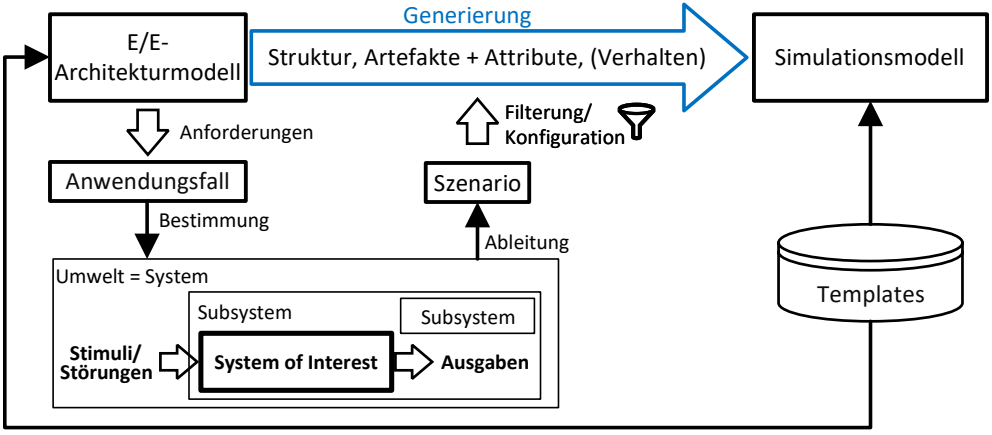



Abbildung 7.1.: Einfluss des Anwendungsfalls auf das Simulationsmodell.

Zunächst ergeben sich die **Anwendungsfälle** für eine Simulation aus den Designfragen von E/E-Architekten. Solche Fragen kommen bei der Umsetzung der Anforderungen auf (vgl. auch Abbildung 4.3). Die Anwendungsfälle geben dann das zu betrachtende Teilsystem (**SOI**),

dessen Schnittstellen zur Umgebung, die Messgrößen (**Ausgaben**), die **Stimuli** und ggf. **Störungen** vor. Aus einem Anwendungsfall lässt sich anschließend ein **Szenario** ableiten. Dieses definiert zusätzlich die Systemzustände (**Attribute**) sowie die Werte der Stimuli und Störungen in Abhängigkeit der Zeit. Außerdem legt es die Bewertungskriterien fest. Durch die Schnittstellen, die das Szenario für das SOI vorgibt, kann es bei der Generierung des Simulationsmodells als **Filter** verwendet werden. Dadurch ist es möglich nur die zur Ausführung des Szenarios notwendigen **Artefakte**, deren Attribute und die zugehörige **Struktur** zu bestimmen. Bei der Generierung werden die bei den Artefakten hinterlegten Referenzen zu den **Templates** genutzt.

Der Anwendungsfall bestimmt damit nicht nur maßgeblich die Laufzeit der Simulation, sondern auch den Nutzen für die E/E-Architekten. Daher sind eine umfassende Analyse und die Auswahl geeigneter Anwendungsfälle für einen Anwendungsfallbenchmark notwendig. Um passende Anwendungsfälle für die E/E-Architektursimulation und deren Benchmarks festzulegen, werden folgende Schritte umgesetzt:

1. Festlegung der Auswahlkriterien für Anwendungsfälle.
2. Untersuchung von technischen Trends im Automobilsektor.
3. Untersuchung der klassischen E/E-Domänen.
4. Umfrage bei PREEvision-Produktmanagern.
5. Analyse verwandter Industriebereiche.
6. Analyse der Modellpaletten von PREEvision.

Die Ergebnisse der jeweiligen Schritte werden im Folgenden beschrieben.

### 7.1.1. Festlegung der Auswahlkriterien für Anwendungsfälle

In den folgenden Abschnitten werden Kriterien zur Auswahl von Anwendungsfällen für eine E/E-Architektursimulation beschrieben und festgelegt.

#### Ziele einer E/E-Architekturmodellierung

Zunächst müssen die grundlegenden Ziele einer E/E-Architektursimulation und deren Nutzen bestimmt werden. Diese können je nach Interessengruppe oder der Spezialisierung des E/E-Architekten unterschiedlich sein. In [219] werden z.B. die generellen Ziele der E/E-Architekturmodellierung aus der Sicht eines Fahrzeugherstellers (*Original Equipment Manufacturer (OEM)*) beschrieben. Diese lauten nach [219]:

- Frühzeitiger Entwurf, Bewertung, Dokumentation und Absicherung.
- Unterstützung beim Technologieentscheid.
- Bewertung von Auswirkungen bei E/E-Architekturänderungen.
- Frühzeitige Optimierung der E/E-Architektur.

Auf eine Simulation übertragen ergibt sich daraus das erste Kriterium:

**Kriterium 1** Ein Anwendungsfall soll eine Bewertung der E/E-Architektur, eine frühzeitige Optimierung oder eine Unterstützung beim Technologieentscheid ermöglichen.

#### Komplexität

In Unterabschnitt 4.1.2 wurde dargestellt, dass der Anwendungsfall maßgeblich darüber entscheidet, wie das resultierende Simulationsmodell aussieht und damit sowohl die Laufzeit als auch den Speicherverbrauch der Simulation bestimmt. Ziel ist es, möglichst komplexe Anwendungsfälle zu identifizieren, die bestenfalls auch die Ableitung eines strukturellen Benchmark-Modells ermöglichen. Die Komplexität wird anhand der Anzahl der benötigten Entitäten und der Verbindungen definiert. Dabei sollen die Daten aus möglichst viele Ebenen der E/E-Architektur verwendet werden.

**Kriterium 2** Ein Anwendungsfall soll möglichst komplex sein.

#### Evaluierbarkeit

Je nach Anwendungsfall und Entwicklungsphase sind unterschiedlich genaue Ergebnisse gefordert. Um die Genauigkeit zu überprüfen, muss die Simulation evaluierbar sein. Es soll eine Beurteilung der Wiedergabetreue und des Nutzens im Hinblick auf Designentscheidungen bei der E/E-Architekturmodellierung möglich sein. Möglichkeiten um Simulationsmodelle zu evaluieren sind in Unterabschnitt 2.2.1 dargestellt. Eine Methode ist der Abgleich des zu evaluierenden Modells mit einem bereits evaluierten. Dies wird durch die Verwendung von Templates teilweise erfüllt (vgl. Unterabschnitt 6.1.1). Teilweise, weil es bei einer Komposition verschiedener Templates zu emergenten Eigenschaften des Gesamtmodells kommen kann. Genau diese Eigenschaften sind für die Beantwortung der Forschungsfragen jedoch relevant. Deshalb ergibt sich die folgende Anforderung:

**Kriterium 3** Das für den Anwendungsfall eingesetzte Simulationsmodell soll durch eine Komposition von Templates gebildet werden können.

#### Fahrzeugdomäne

Mechanische Modelle von Verbrennungsmotoren, Chassis, etc. sollen nicht oder nur stark abstrahiert betrachtet werden. Schwerpunkte der Simulation liegen im E/E-Bereich.

**Kriterium 4** Der Anwendungsfall betrifft primär den E/E-Bereich.

#### Relevanz

Der Anwendungsfall soll aktuell und auch in Zukunft industriell relevant sein. Bereits länger bestehende Funktionen haben während ihrer Evolution viele Iterationen und damit Verbesserungen durchlebt. Der Aufwand für eine Simulation wäre dabei unverhältnismäßig hoch.

Deshalb soll der Fokus bei der Auswahl von Anwendungsfällen auf aktuellen Problemen in der Automobilindustrie liegen, wobei der Aufwand der Simulation im Verhältnis zum Nutzen stehen soll.

**Kriterium 5** Der Anwendungsfall soll aktuell relevante Probleme adressieren, während der Aufwand zur Umsetzung der Simulation im Verhältnis zum Nutzen steht.

#### Umsetzbarkeit

Umsetzbar heißt, dass der Anwendungsfall mit dem E/E-Modellierungswerkzeug beschreibbar ist und das dafür nötige Modell mit allen relevanten Artefakten und Attributen zum zeitlich geforderten Einsatz der Simulation vorhanden ist. Der Einsatzzeitpunkt der Simulation muss also zum Entwicklungsprozess passen (vgl. Unterabschnitt 3.1.4).

**Kriterium 6** Der Anwendungsfall ist in PREEvision beschreibbar.

### 7.1.2. Technische Trends in der E/E-Architekturentwicklung

Aufgrund von Kriterium 5 soll der Fokus beim Einsatz der Simulation auf Technologien liegen, die noch verhältnismäßig jung sind oder deren Einsatz erst in Zukunft geplant ist. Die im Anwendungsfall betrachteten Technologien sollen damit ein entsprechendes Innovationspotential bieten. In Unterabschnitt 3.1.2 werden aktuelle technische Trends in der Automobilindustrie beschrieben. Dazu passende Anwendungsfälle für Simulationen werden in den folgenden Abschnitten erörtert.

#### Elektrifizierung

**Isolationsüberwachungssysteme** Der Einzug von Hochvoltsystemen führt zu parallel existenten, unterschiedlichen Spannungsnetzen im Fahrzeug. Um Personen und das System zu schützen, ist eine galvanische Trennung unterschiedlicher Spannungsnetze erforderlich. Die Spannungstrennung ist zu jeder Zeit einzuhalten [177]. Sie muss darüber hinaus auch beim Anschluss an Ladestationen beachtet werden [307]. Zur Überprüfung der Einhaltung sind Systeme erforderlich, welche die Isolationsspannung überwachen und im Falle einer Grenzwertüberschreitung, z.B. bei einem Kurzschluss, den entsprechenden Schaltungsteil vom restlichen Netz trennen. Ein solches System wird als *Isolation Monitoring System (IMS)* bezeichnet.

Bei einer Simulation ist die Einhaltung der galvanischen Trennung zu untersuchen. Daran beteiligte Komponenten (*SOI*) sind neben dem IMS die Versorgungsnetze, Trennstrecken und Isolatoren, wie Trenntransformatoren oder Optokoppler. Bei einem IMS kommt es vor allem auf eine effiziente Verortung (*Zustand*) und ausreichend schnelle Reaktionszeiten (*Ausgabe*) an, welche durch eine Simulation überprüft werden können.

**Prüfung der Versorgungsspannungsstabilität** Durch den Einsatz von Elektromotoren und Batterien im Hochvoltbereich steigen die thermischen Anforderungen. Ebenfalls tragen HPCs (siehe. Unterabschnitt 3.1.1) durch dynamische Lastspitzen zu einer Erwärmung der Leitungen bei und können bei schnellen Schaltvorgängen induktiv wirken, was zum abrupten Einbruch der Versorgungsspannung führen kann.

Bei der Modellierung müssen daher Leitungen ausreichend kurz und dick dimensioniert sein, um die Erwärmung durch hohe Ströme zu kompensieren. Außerdem sind entsprechende Ladungsspeicher zur Verhinderung eines Versorgungsspannungseinbruches einzuplanen. Darüber hinaus muss die Ausführung sicherheitskritischer Funktionen selbst bei einem Einbruch der Spannung gewährleistet sein. Dies ist z.B. durch Redundanz oder die Einschränkung von Komfortfunktionen möglich. Mit einer Simulation kann die Versorgungsspannungsstabilität (*Ausgabe*) des Bordnetzes (*SOI*) in Abhängigkeit induktiver Lasten (*Störgröße*) geprüft werden.

**Energiebedarfsoptimierung** Batterien haben nach aktuellem Stand der Technik noch Optimierungsbedarf, was vor allem Energiedichte, Kosten und Haltbarkeit betrifft [84]. Eine geringe Reichweite und hohe Kosten führen heute bei den OEMs dazu, dass viele E/E-Systeme hinsichtlich ihres Energieverbrauchs optimiert werden.

Mögliche Anwendungsfälle für eine Simulation sind hier der Technologieentscheid oder die Beurteilung eines dynamischen, energieoptimierten Mappings von abschaltbaren Funktionen auf Steuergeräte. Die Alterung einer Batterie wirkt sich hierbei auf ihre Kapazität aus und erfordert eine Spannungsüberwachung zur Gewährleistung der Stabilität. Sie ist sowohl von der Beanspruchung als auch vom Design der E/E-Architektur (realisiertes Batterie- und Lademanagement) abhängig. Die Spannungen (*Ausgabe*) in Abhängigkeit der Batteriealterung oder des Mappings von Funktionen bzw. SWCs auf ECUs (*SOI*, *Zustand*) können simuliert werden.

### Autonomes Fahren

**Funktionale Absicherung** Das Testen von FAS ist aufgrund ihrer Komplexität herausfordernd. 13,2 Mrd. Testkilometer sind nach [304] zur Absicherung eines Autobahnpiloten nötig. Ein Testaufwand, der auf realen Strecken nicht zu erfüllen ist. Mit einher geht auch die Frage nach der Straßenzulassung solcher Systeme. Die OICA schlägt deshalb einen „Drei-Säulen-Ansatz“ vor, bei dem Simulation schon zu Beginn der Entwicklung eines Fahrzeugs eingesetzt werden soll.

Regelungen der *United Nations Economic Commission for Europe (UNECE)* können, wie in [P2] beschrieben, zur Modellierung von Szenarien verwendet werden, mit denen die Zulassung von FAS-Funktion simulativ geprüft werden kann.

**Untersuchung der Hardwareperformanz** Steigender Softwareumfang und erhöhtes Datenaufkommen durch den Einsatz einer Vielzahl von Sensoren zur Umfelderkennung erfordern eine performante Hardware zur Datenverarbeitung in Echtzeit. Hierbei ist eine optimale Lastverteilung zwischen Zonen-ECUs und HPCs (siehe Unterabschnitt 3.1.1) nötig. Sensordaten für FAS werden normalerweise zur Reduktion des Kommunikationsaufkommens in den Zonen-ECUs vorverarbeitet und dann zur Weiterverarbeitung an die HPCs gesendet.

Das Mapping der Funktionen auf die Zonen-ECUs (*Zustand*) und die Auswahl des Kommunikationssystems (*SOI*) wirken sich auf die Kommunikationslast und die Auslastung der HWCs (*Ausgaben*) aus. Diese Auswirkung kann simulativ geprüft werden.

**Ausfallsicherheit** Zur elektronischen Steuerung in automatisierten Fahrzeugen werden mechanische Komponenten durch elektrische ersetzt (X-By-Wire [118]). Bei einem Ausfall dieser und anderer Teilsysteme muss ein automatisiertes oder autonom fahrendes Fahrzeug jedoch weiterhin funktionieren, sodass eine Gefährdung der Verkehrsteilnehmer ausgeschlossen ist. Neben redundant ausgelegter Systeme eignen sich hierzu eine kontinuierliche Überwachung des E/E-Systems (Monitoring) und die Anwendung einer „Graceful Degradation" [302]. Hierbei werden stufenweise stärkere Maßnahmen zum Erhalt der Gesamtfunktionalität durchgeführt und anhand ihrer Effektivität bewertet.

Bei der E/E-Architekturmodellierung können diese dynamischen Änderungen (*Zustand*) abgebildet und simuliert werden. Die verwendeten Maßnahmen zum Erhalt der Gesamtfunktionalität (*Ausgabe*) können damit in Abhängigkeit der Topologie und der verwendeten Komponenten (*SOI*) beurteilt werden.

### Vernetzung

**Berechnung KI-basierter Funktionen in der Cloud** *Künstliche Intelligenz (KI)* im autonomen Fahrzeug basiert häufig auf einem hybriden Modell [11]. Dabei müssen Algorithmen für die Fahrzeugführung ohne Vernetzung auskommen, während ergänzende Komfortfunktionen in der Cloud berechnet werden dürfen. Generell wären einige dieser Komfortfunktionen zum jetzigen Stand nicht wirtschaftlich, da Berechnungen ohne die Cloud eine leistungsstärkere Hardware voraussetzen würden [277].

Bei der Modellierung der E/E-Architektur ist die Partitionierung (*Zustand*) zu beachten. Es stellt sich also die Frage, welche Berechnungen in der Cloud ausgeführt werden und welche lokal. Für die Bewertung sind sowohl die Kosten der benötigten Rechner (*SOI*) als auch die Ende-zu-Ende-Laufzeiten der Signale (*Ausgaben*) vom Fahrzeug zum Backend-Server und zurück relevant. Sie können mittels Simulation untersucht werden, wobei neben der E/E-Architektur auch die reaktive Umgebung (Backend-Server) zu beachten ist.

**Angriffssicherheit** Schnittstellen jeglicher Art zur E/E-Architektur machen Fahrzeuge angreifbar. Das System „Keyless-Go" hat gezeigt, dass Angreifer vorhandene Schwachstellen gezielt ausnutzen können [5]. Vor allem Funkanbindungen oder Diagnoseschnittstellen sind mögliche Eintrittstüren für Angreifer. Um Fahrzeugdiebstahl oder böswillige Angriffe zu verhindern, wird unter anderem das Paradigma „Security by Design" [152] verfolgt. Hierbei ist die Applikation entsprechender Schutzmaßnahmen, wie sie in [283] vorgeschlagen werden, oder die Einhaltung des ISO 21434 Standards [133] entscheidend (siehe auch Unterabschnitt 3.1.4).

Bei der E/E-Modellierung sollten die Schutzmaßnahmen zur Einhaltung der Informationssicherheit bereits zu frühen Entwicklungszeitpunkten überprüft werden können. In [P4] wurde

deshalb Simulation bei der modellbasierten Entwicklung eines IDS eingesetzt. Dabei wurde deutlich, dass gerade die HW-Topologie bei einer dynamischen Bedrohungsanalyse zu beachten ist. Außerdem können mit einer Simulation dynamische Angriffsszenarios, wie bei der „Denial-of-Service-Attacke“ [285], ausgeführt werden.

**Hypervisor und virtuelle ECUs** Unterschiedliche Betriebssysteme im Automobil und die Anbindung an eine Cloud erfordern Mittelstellen. Hypervisor werden zur Verwaltung, Überwachung und Zuweisung der Ressourcen eines HPCs eingesetzt. Sie bieten damit die Möglichkeit mehrere voneinander trennbare Betriebssysteme oder virtuelle ECUs auf einem HPC auszuführen. Da ein Hypervisor jedoch oft mit hohen Rechten ausgestattet ist, wird hier eine Angriffsfläche für böswillige Attacken geboten [174].

Mit einer Simulation kann die Zuweisung der Ressourcen (*Zustand*) eines HPCs (*SOI*) zu dem jeweiligen Betriebssystem beurteilt werden, um so Leistungseinbußen (*Ausgabe*) zu vermeiden. Ferner kann eine simulative Rechteoptimierung helfen Sicherheitslücken einzuschränken.

### Shared Mobility

**Erweitertes Varianten-Management** Aktuell entstehen neue Geschäftsmodelle, weil OEMs zum Dienstleister werden. So können beispielsweise Kundenfunktionen bei Bedarf nachgerüstet oder zeitlich begrenzt gemietet werden („Functions on Demand“) [73]. Dies setzt ein erweitertes Varianten-Management voraus. Je nach Konfiguration des Kunden müssen nämlich unterschiedliche Abhängigkeiten bei der Hardwareausstattung berücksichtigt werden. So setzt z.B. die Nachrüstung eins automatischen Fernlichts installierte LED-Scheinwerfer und entsprechende Sensoren zur Umfelderkennung voraus. Neben der Nachrüstung der Funktion selbst müssen womöglich auch vom Kunden nicht wahrnehmbare Funktionen nachgerüstet und bestehende Funktionen angepasst werden.

Auswirkungen einer Funktionsnachrüstung (*Zustand*) auf die Laufzeit anderer Funktionen oder den Energieverbrauch (*Ausgaben*) können in Abhängigkeit der Hardwareausstattung (*SOI*) frühzeitig durch Simulation beurteilt werden.

**Predictive Maintenance** Als geteilte Ressource wird das Automobil zu einem Nutzfahrzeug, wobei Ausfälle zu hohen Kosten der Betreiber führen. „Predictive Maintenance“, also eine vorausschauende Wartung, soll Ausfälle und unnötige Werkstattbesuche vermeiden [255]. Voraussetzung dafür sind Überwachungssysteme und Fehlervorhersagen, die auf Datenanalysen basieren. Schnittstellen zu Überwachungssystemen und typische Fehlermuster müssen daher bei der Entwicklung eingeplant werden. Ungünstige Kombinationen von Fehlermustern unterschiedlicher Bauteile müssen ebenso identifiziert werden, um die Dringlichkeit einer Wartung zu bestimmen.

Simulation kann dabei helfen die Wartungszyklen (*Ausgabe*) schon bei der Planung abzuschätzen, indem man einzelne E/E-Komponenten (*SOI*) altern lässt (*Zustand*) und die Auswirkungen davon analysiert. Dies kann auch beim Technologieentscheid helfen.

#### Updates

**Interne Softwareentwicklung und OTA Updates** Eine große Anzahl an Softwarekomponenten von unterschiedlichen Zulieferern entwickeln zu lassen kann zu einem hohen Integrationsaufwand führen. Außerdem sind zentrale OTA-Updates dadurch schwer zu realisieren. Bei vielen Herstellern wird deshalb eine interne Softwareentwicklung allmählich zur Realität. Bis 2025 sollen bei der *Volkswagen AG* 60% der Software in einem eigenen Entwicklungszentrum entstehen [60]. Die *Daimler AG* möchte bis 2024 ein eigenes Betriebssystem entwickeln [99].

Bei der Entwicklung der E/E-Architektur müssen in Zukunft auch die verwendeten Betriebssysteme in der SA Beachtung finden. Schnittstellen, Algorithmen und deren Änderungen durch Updates sollen nachverfolgt werden können. Simulation kann helfen die entsprechenden Laufzeitverzögerungen bei der Kommunikation zwischen herstellerspezifischen Betriebssystemen und anderen Frameworks, wie AUTOSAR, zu untersuchen.

### 7.1.3. Klassische E/E-Domänen

Obwohl klassische E/E-Domänen in Zukunft aufgrund der Domänenfusion keine Rolle mehr spielen (siehe Unterabschnitt 3.1.1), helfen sie den Suchraum nach Anwendungsfällen zu strukturieren und einzuschränken. Die Domänen *FAS* (autonomes Fahren), *Antriebsstrang* (Elektrifizierung) und *Connectivity* (Vernetzung) wurden bereits durch den vorherigen Abschnitt abgedeckt. Sie werden daher nicht weiter betrachtet.

#### Infotainment

Charakteristisch für diese Domäne sind hohe Datenraten, die beispielsweise beim Streaming anfallen. Dabei müssen die Daten nicht unbedingt in Echtzeit verarbeitet werden. Im Vergleich zur FAS-Domäne sind die Anwendungen in der Infotainment-Domäne nämlich nicht kritisch. Typischerweise bildet heutzutage ein integriertes System die Hauptkomponente (*Head Unit*). Es weist interne (CAN, Ethernet) und externe (Bluetooth, WiFi) Schnittstellen auf. In Zukunft soll sich dieses integrierte System jedoch zu einem verteilten wandeln [90]. Bei der Bedienung der Systeme soll der Fahrer außerdem möglichst wenig abgelenkt sein. Trends gehen daher in Richtung ubiquitäres Computing. Des Weiteren soll zukünftig das Fahrerverhalten vorhergesagt werden können, was eine Erstellung von Fahrerprofilen voraussetzt.

Simulation wird in dieser Domäne zum Beispiel zur Optimierung der Schallausbreitung im Innenraum [144] oder für ressourcensparende Robustheitstests [119] eingesetzt. Bei letztgenannten fällt durch den Einsatz von Simulation eine Aufheizung des Fahrzeugs im Klimakanal bei zeitgleicher Abkühlung des Innenraums weg. Konkrete Anwendungsfälle unter Berücksichtigung der zuvor definierten Anforderungen ergeben sich jedoch nicht.

#### Safety & Chassis

Die Fahrgestell-Domäne (Chassis) umfasst die Systeme, die mit der Straße und den Fahrwerkskomponenten interagieren. Zu diesen gehören z.B. das Antiblockiersystem oder auch das *elektronische Stabilitätsprogramm (ESP)* [237]. Bei der Entwicklung dieser Systeme steht vor allem der Reglerentwurf im Vordergrund. Charakteristisch hierfür ist die notwendige Beachtung von unterschiedlichen Samplezeiten und harten Echtzeitanforderungen. Ein Trend in dieser Domäne ist die Elektrifizierung von ursprünglich mechanischen Systemen („X-by-wire“) [118].

Simuliert werden überwiegend physikalische Systeme, aber auch Regelungssysteme [209] oder Unfalltests nach dem *New Car Assessment Programme (NCAP)* [25]. Unter Berücksichtigung der Auswahlkriterien ergeben sich hier keine weiteren Anwendungsfälle.

#### Body & Comfort

Diese Domäne umfasst Systeme wie die Schließanlage, das Beleuchtungssystem, das Diagnosesystem oder die Klimatisierung. Charakteristisch ist für diese Systeme, dass sie i.d.R. weniger komplex sind und keine harten Echtzeitanforderungen haben. Sie werden zudem häufig bei Zulieferern entwickelt, wodurch Designfehler erst im Integrationstest beim OEM auffallen. Trends betreffen hier vor allem die Optimierung des Fahrerkomforts und die Entwicklung von Beleuchtungssystemen für den Innenbereich (Ambientenbeleuchtung) sowie für den Außenbereich (Matrix-LED-Scheinwerfer) [45].

Häufig wird in der Literatur der thermische Fahrerkomfort [1, 227, 226] sowie auch allgemein der Fahrkomfort [70, 156] simuliert. Außerdem wird in [206] die Fahrerzustandsüberwachung mittels *Elektrokardiogramm (EKG)* simuliert. Bei diesen Simulationen handelt es sich um sogenannte „Human-in-the-Loop“-Simulationen, die biologische Körpermodelle voraussetzen. Entsprechend setzen Beleuchtungssimulationen 3D-Modelle der Umgebung voraus. Daher eignet sich auch diese Domäne nicht zur Ableitung von Anwendungsfällen für eine E/E-Architektursimulation.

### 7.1.4. Umfrage bei PREEvision-Produktmanagern

Es wurde ein Fragebogen erstellt (siehe Abbildung A.2), um Expertenwissen beim Finden von Anwendungsfällen einfließen zu lassen. Dabei hat sich vor allem der Bedarf einer Leitungssatzsimulation herausgestellt. Diese umfasst die folgenden Teilbereiche:

- **Optimierung der Topologie:** Sowohl die Anzahl der Spleiße als auch die Kabellängen sollen mithilfe von Simulation reduziert werden.
- **Dimensionierung:** Mittels Simulation sollen Leitungsquerschnitte reduziert und Sicherungen, Pufferkondensatoren oder Schutzschaltungen an ECUs ausgelegt werden.
- **Reduktion von Verlusten:** Störungen der elektromagnetischen Verträglichkeit führen zu Verlusten und können die ordnungsgemäße Funktion des Systems gefährden. Sie entstehen z.B. durch die Bündelung von Adern in Kabeln oder bei hochfrequenten Digitalschaltun-

gen. Verluste und ihre Auswirkung auf die ordnungsgemäße Funktion des Systems sollen simulativ überprüft werden.

- **Codegenerierung:** Je nach verwendetem Simulator lässt sich aus einem Simulationsmodell bereits ein lauffähiger C-Code für ECUs generieren.
- **Fahrszenarien:** Es besteht das Interesse an der Simulation von Fahrszenarien zur Energieverbrauchsabschätzung von BEFs.

### 7.1.5. Analyse verwandter Industriebereiche

In Unterabschnitt 7.1.2 wurden bereits Trends beschrieben, die aus anderen Industriebereichen stammen. „Predictive Maintance“ entstammt der Fabrikation innerhalb des Industrie-4.0-Umfelds. Hardwarezustände werden hier überwacht und mit Mustern abgeglichen, um die Austauschzeitpunkte von Verschleißteilen zu bestimmen. „X-by-wire“ kommt aus der Luft- und Raumfahrt („Fly-by-wire“). Prozesse für die Zertifizierung von Fahrzeugen werden zum Teil auch aus der Luft- und Raumfahrt abgeleitet.

Die *International Standard Industrial Classification (ISIC)*, welche die *United Nations Organization (UNO)* festgelegt hat, beschreibt 33 Industriezweige. In Europa existiert hinzukommend eine weitere Klassifikation der Industriezweige nach der *Nomenclature statistique des activités économiques dans la Communauté européenne (NACE)*. In den meisten dieser Industriezweige wird Simulation hauptsächlich für Produktions- und Verarbeitungsprozesse angewandt. Bei der Herstellung von Konsumgütern, Metallen, Holzartikeln und Chemieprodukten werden z.B. vorwiegend Prozesssimulationen eingesetzt. Dies erklärt auch die Fülle an Prozesssimulatoren, die im Zuge der Recherche für diese Arbeit gefunden wurden (siehe Abschnitt A.1). Im Bereich der Energie- und Wasserversorgung sowie in der Verkehrstechnik werden hauptsächlich Netzwerksimulationen eingesetzt. Im Hinblick auf eine E/E-Architektursimulation, sind vor allem die Bereiche Kommunikationstechnik, Elektrogeräte, Maschinen und Verteidigung relevant. Neben Netzwerk- und Schaltungssimulationen finden hier auch hybride Simulationen mit Echtzeitanforderungen Anwendung. Konkrete Anwendungsfälle für eine E/E-Architektursimulation konnten daraus jedoch nicht abgeleitet werden. Aus dem Bereich Verteidigung bieten sich allerdings „Was-Wäre-Wenn“-Simulationen für den Variantenvergleich an. Dabei werden verschiedene Lösungsoptionen (Varianten) simuliert und ihre Ergebnisse miteinander verglichen. Zum Schluss wird die beste Lösung ausgewählt.

### 7.1.6. Analyse der Modellpaletten von PREEvision

Es wurden die Modellpaletten und Artefakte von PREEvision untersucht. Dadurch konnten weitere Anwendungsfälle für eine E/E-Architektursimulation identifiziert werden.

#### Prozesssimulation

In PREEvision lassen sich neben E/E-Architekturen auch Arbeitsprozesse modellieren. Dazu gehören die Ressourcenverwaltung und das Projektmanagement. Eine Simulation kann hier

helfen etwaige Risiken bei der Planung zu erkennen und zu minimieren. [256] zeigt die Möglichkeiten und den Einsatz einer solchen Simulation in der Praxis. Hierbei wird das Was-Wäre-Wenn-Prinzip aus Unterabschnitt 7.1.5 aufgegriffen, um die gegenseitige Auswirkung parallel laufender Projekte auf die Gesamtlaufzeit zu untersuchen.

#### SOA und TTE

Serviceorientierte Architekturen (SOAs) und *Time Triggered Ethernet (TTE)* lösen signalorientierte Architekturen ab, um in Zukunft das Hinzufügen neuer Funktionen oder Updates während des Betriebs zu ermöglichen (vgl. Unterabschnitt 3.1.1). Außerdem ergeben sich eine höhere Flexibilität und geringere Kosten in der Wartung, da HWCs z.B. einfacher ausgetauscht werden können. Jedoch kann es bei SOAs vorkommen, dass bei der verteilten Ausführung einer Funktion auf unterschiedlichen HWCs unnötig viele Nachrichten ausgetauscht werden. Des Weiteren werden Schnittstellen zu den signalorientierten Teilen der E/E-Architektur benötigt, um den Determinismus von sicherheitskritischen Anwendungen zu garantieren. Ferner müssen entsprechende Mechanismen zur Gewährleistung der Informationssicherheit entwickelt und umgesetzt werden [254].

Durch Simulation könnte die Anzahl an ausgetauschten Nachrichten (*Ausgabe*) bei der verteilten Ausführung (*Zustand*) von Funktionen auf mehreren HWCs (*SOI*) überprüft werden. Außerdem könnten Latenzen (*Ausgabe*), die durch die Schnittstellen zwischen den service- und signalorientierten Bereichen (*SOI*) entstehen, analysiert werden.

#### Diagnoseschnittstellen

Durch die Zunahme des Funktionsumfangs und der steigenden Anzahl an SWCs wächst auch die Zahl an OEM-spezifischen Diagnoseschnittstellen. Um sie zu testen werden in der Praxis spezielle Geräte verwendet. Diese simulieren das eigentliche Diagnosegerät, führen dabei automatisiert Diagnosen aus und protokollieren die Ergebnisse. Allerdings erfolgt dies erst bei der Integration und damit sehr spät.

Simulationen, die ein modelliertes Verhalten der SWCs verwenden, könnten in früheren Phasen der Entwicklung helfen Designfehler bei den Diagnoseschnittstellen aufzudecken.

### 7.1.7. Festlegung der Anwendungsfälle und Schlussfolgerungen

Tabelle 7.1 zeigt eine Gegenüberstellung der gefundenen Anwendungsfälle. Hierbei werden die final entscheidenden Kriterien 2, 3, 5 und 6 betrachtet. Die übrigen Kriterien konnten bereits bei der Auswahl der Anwendungsfälle berücksichtigt werden. Punkte reichen von -1 (nicht erfüllt) über 0 (teilweise erfüllt) zu 1 (erfüllt). Je höher die Gesamtpunktzahl, desto geeigneter ist ein Anwendungsfall. Anwendungsfälle mit einer Punktzahl über zwei sollen im weiteren Verlauf der Arbeit berücksichtigt werden und sind daher grün hervorgehoben. Der Anwendungsfall „Codegenerierung“ wurde nicht in der Tabelle aufgenommen, da es sich nicht um einen direkten Anwendungsfall der Simulation handelt.

Tabelle 7.1.: Auswahl der Anwendungsfälle für eine Evaluation.

| | IMS | Versorgungsstabilität | Energiebedarfsoptimierung | Funktionale Absicherung | Hardwareperformanz | Ausfallsicherheit | KI-Funktionen in der Cloud | Angriffssicherheit | Hypervisor | Functions on Demand | Predictive Maintanance | OTA | Topologieoptimierung | Dimensionierung | EMV | Fahrszenarien | SOA Netzwerksimulation | Diagnoseschnittstellen |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2) Komplexität | 1 | 1 | 1 | 1 | 1 | -1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3) Evaluierte Templates verfügbar | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 5) Aufwand / Nutzen | 0 | 0 | 1 | 1 | 1 | 0 | -1 | 1 | -1 | -1 | -1 | 0 | 0 | 1 | -1 | 1 | -1 | -1 |
| 6) Beschreibbar in PREEvision | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| Punktzahl | **2** | **2** | **3** | **4** | **3** | **0** | **0** | **3** | **2** | **1** | **0** | **2** | **2** | **4** | **0** | **3** | **1** | **1** |

Ziel der Evaluation ist es die Skalierbarkeit der E/E-Architektursimulation in Abhängigkeit der Anwendungsfälle zu bewerten. Daher sollen die konkreten Anwendungsfälle möglichst komplex sein. Als Maß für die Komplexität von Funktionen soll das SAE-Level [240] von FAS dienen. Aktuell hat das FAS „Autobahnpilot“, mit einem SAE-Level von 3, das höchste SAE-Level [58]. In einer einfachen Realisierung setzt sich das FAS aus einem „Abstandsregeltempomat“ und einem „Spurhalteassistenten “zusammen. Die Anwendungsfälle **Funktionale Absicherung** und **Angriffssicherheit** sollen bei der Simulation von FAS mit möglichst hohem SAE-Level umgesetzt werden. Ferner sollen **Fahrszenarien** Bestandteil der Simulationen sein.

Die Auswirkung der HW-zentrierten Abbildung (siehe Abschnitt 6.3) auf die Skalierbarkeit soll ebenfalls bewertet werden. Im Bereich Leitungssatz und Elektrik ist etwa die Simulation von permanenterregten Gleichstrommotoren aufgrund vieler Rückkopplungsschleifen und einem nicht-linearen Charakter besonders komplex [52]. Solche Motoren werden in BEFs eingesetzt. Die integrierte Simulation eines batterieelektrischen Antriebs inkl. der benötigten Leitungen, der Motoransteuerungsalgorithmen, der Fahrdynamik und der Umgebung erlaubt die Umsetzung verschiedener Anwendungsfälle. Unter Verwendung von **Fahrszenarien** soll damit der **Energiebedarf**, die **Hardwareperformanz** und die **Dimensionierung** simulativ untersucht werden. Ferner soll dabei die HW-zentrierte Modellgenerierung mit Standard-Templates angewandt und beurteilt werden.

## 7.2. Simulationen mit Ptolemy II

Parallel zur Entwicklung des Simulink-Simulationsframeworks wurden einzelne Konzepte aus dieser Arbeit mit Ptolemy II und anhand von Anwendungsfällen evaluiert. Dies liegt daran, dass die Code-Basis zur Interpretation von PREEvision-Modellen und der Generierung von Ptolemy-II-Modellen bereits aufgrund von [43] vorhanden war. Der Quelltext konnte entsprechend verwendet und angepasst werden. Basierend auf den in Kapitel 5 gewonnenen Erkenntnissen, lässt sich jedoch erwarten, dass eine Implementierung mit Simulink mindestens genauso gut skaliert. Insbesondere, da Simulink auch die Grundlage des 3D-Fahrsimulators *DYNA4* [289] bildet.

### 7.2.1. Fahrerassistenzsysteme

Innerhalb von verschiedenen studentischen Arbeiten sollte die Simulation eines Autobahnpiloten mit Ptolemy II entwickelt werden, der mit einem SAE-Level von 3 den bisher komplexesten, verfügbaren FAS zuzuordnen ist. Es handelt sich dabei um eine Kombination aus unterschiedlichen FAS: Spurwechselassistent, Abstandsregeltempomat, Spurhalteassistent, Notbremsassistent, Nothalteassistent, Verkehrszeichenassistent, Aufmerksamkeitsassistent und intelligente Geschwindigkeitsassistenz. In den Arbeiten [S7, S3, S6, 190, S1] sollten Eigenschaften, Verhalten und Schnittstellen des EFs unter verschiedenen Randbedingungen konzeptioniert und evaluiert werden.

#### Evaluation eines Spurhalteassistenten mittels 3D-Co-Simulation

In [S7] wurde eine Co-Simulation von Ptolemy II und dem 3D- und Fahrphysiksimulator OpenDS realisiert. Zur Kopplung beider Simulatoren wird der Standard *Remote Method Invocation (RMI)* verwendet. Ptolemy II wird dabei zum Master und kontrolliert die Zeitausführung von OpenDS. OpenDS wird dafür um eine RMI-Server-Implementierung erweitert. Über die Implementierung eines RMI-Clients im als Actor umgesetzten EF in Ptolemy II können dann Kontrollbefehle an OpenDS (Starten, Pausieren, etc.) und an das OpenDS-EF (Lenkwinkel, Gaspedalstellung, etc.) geschickt werden. Außerdem kann Ptolemy II damit auf Daten aus OpenDS zugreifen. Dazu gehören das Kamerabild oder auch die Geschwindigkeit des EFs. In der Arbeit wurde ferner, unter Verwendung von *OpenCV*, ein Kantenerkennungsalgorithmus zur Querregelung des Fahrzeugs implementiert. So konnte, zusätzlich zu einem bestehenden Abstandsregeltempomat aus [43] und einer Verkehrszeichenerkennung für Geschwindigkeiten aus [S8], ein Spurhalteassistent realisiert werden. Durch die Anbindung der 3D-Simulation an die E/E-Architektursimulation konnte der Kantenerkennungsalgorithmus evaluiert und optimiert werden. Dabei konnten Einflüsse auf die Regelgüte, wie der Befestigungswinkel der Kamera, untersucht werden. Darüber hinaus wurde verglichen, wie sich der Kantenerkennungsalgorithmus in der Simulation gegenüber real aufgezeichneten Videos aus der *KITTI*-Datenbank [96] verhält. Hierfür wurden mit *OpenstreetMap* [198] und *OSM2World* [147] zu den Videos passende 3D-Modelle erstellt. Mittels *blender* [29] konnte dann das entstandene 3D-Modell in das von OpenDS lesbare *.ogre*-Format gewandelt werden. Ausschnitte

der unterschiedlichen Vergleiche zwischen den Videos und den 3D-Modellen sind in Abbildung 7.2 dargestellt. Die Ergebnisse der Evaluation sind in Tabelle 7.2 aufgelistet. [S7]

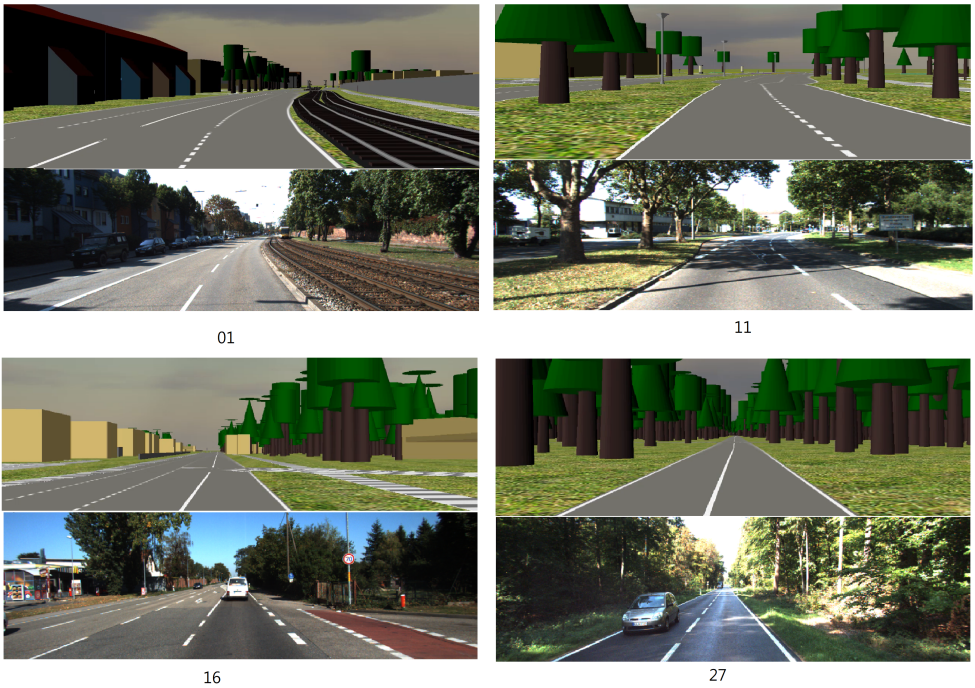



Abbildung 7.2.: Abgleich realer Teststrecken-Videos mit der 3D-Simulation (Quelle: [S7]).

Tabelle 7.2.: Evaluation der Kantenerkennung mit realen und simulierten Daten (Quelle: [S7]).

| Datensatz | Id | Genauigkeit | Trefferquote | F-Maß [%] | Kategorie |
|---|---|---|---|---|---|
| OpenDS | 01 | 0.5746 | 0.5 | 53,47 | Stadt |
| KITTI | 01 | 0.5609 | 0.4907 | 52,35 | Stadt |
| OpenDS | 11 | 0.4563 | 0.3602 | 40,61 | Stadt |
| KITTI | 11 | 0.6025 | 0.4491 | 51,46 | Stadt |
| OpenDS | 16 | 0.6925 | 0.6240 | 65,65 | Landstraße |
| KITTI | 16 | 0.2811 | 0.2187 | 24,60 | Landstraße |
| OpenDS | 27 | 0.8481 | 0.7912 | 81,87 | Landstraße |
| KITTI | 27 | 0.5139 | 0.3505 | 41,68 | Landstraße |

Die Genauigkeit G gibt an, wie oft eine Kante richtig ($Richtig_{positiv}$) erkannt wird, im Verhältnis zu allen richtig ($Richtig_{positiv}$) und falsch ($Falsch_{positiv}$) erkannten Kanten:

$$G = \frac{Richtig_{positiv}}{Richtig_{positiv} + Falsch_{positiv}} \tag{7.1}$$

Die Trefferquote T beschreibt, wie oft eine Kante richtig erkannt wird, im Verhältnis zur Anzahl aller richtig erkannten Kanten und den nicht erkannten Kanten ($Falsch_{negativ}$):

$$T = \frac{Richtig_{positiv}}{Richtig_{positiv} + Falsch_{negativ}} \tag{7.2}$$

Das F-Maß F ist das gewichtete harmonische Mittel aus G und T:

$$F = 2 * \frac{G * T}{G + T} \tag{7.3}$$

Da das F-Maß Genauigkeit und Trefferquote vereint, wird es in [S7] für eine abschließende Beurteilung genutzt. Betrachtet man die Datensätze für die **Stadt**, so ist die Abweichung der Kantenerkennung in der Simulation von den realen Daten mit 1,12 % bzw. 10,85 % relativ gering. Die höhere Abweichung beim Datensatz mit der **Id 11** ist vor allem auf Schatten zurückzuführen, die in der Simulation nicht beachtet wurden. Auf der **Landstraße** fällt die Abweichung mit 41,05 % bzw. 40,19 % deutlich höher aus. Auch hier sind vor allem Schatten und Unterschiede in der Art der Fahrbahnmarkierung für die Abweichung verantwortlich. [S7]

Des Weiteren wurde in [S7] die Laufzeit der Implementierung untersucht. Der zeitabhängige Scaleup der 3D-Simulation lag bei $S_{t\_3D} = 1,258$. Da $S_{t\_3D} > 1$ gilt, skaliert die Simulation für diesen Anwendungsfall. Die 3D-Simulation wurde außerdem mit einer Implementierung verglichen, die auf Wegpunkten basiert. Hier ist die Ausführung mit $S_{t\_Weg} = 1,686$ zwar um ca. 20 % schneller, aber die Abstraktion im Vergleich zur Realität noch größer, da z.B. Lichteinflüsse komplett vernachlässigt werden. Außerdem ist der Nutzen geringer, weil der Einfluss der Bilderkennungsqualität auf den Regelungsalgorithmus ohne 3D-Simulation nicht beurteilt werden kann. [S7]

**Fazit** Die Untersuchung hat gezeigt, dass das Konzept des EFs zur Co-Simulation verwendet werden kann. Durch die Kopplung der E/E-Architektursimulation mit einer 3D-Simulation sind darüber hinaus realitätsnahe Ergebnisse möglich (vgl. Datensätze mit Id=01 in Tabelle 7.2). Dadurch lässt sich die Qualität von Algorithmen verbessern. Allerdings ist eine Kalibrierung mit Realdaten nötig, da die Genauigkeit der Simulation durch ungewollte Vernachlässigung von relevanten Aspekten bei der Modellbildung abnimmt.

#### Evaluation eines Abstandsregeltempomaten mittels Co-Simulation von Verkehr

In [S3] wurde die Evaluation eines Abstandsregeltempomaten aus [43] um eine reaktive Co-Simulation mit dem Verkehrssimulator *Simulation of Urban MObility (SUMO)* [77] erweitert.

Ziel der Untersuchung war es die zuvor durch eine Sinuswelle erzeugten Stimuli durch realitätsnähere Daten zu ersetzen. Sie stammen aus einer externen Verkehrssimulation mit SUMO. Für jeden Akteur aus der SUMO-Verkehrssimulation gibt es in der E/E-Architektursimulation mit Ptolemy II einen entsprechenden Stellvertreter (Actor), der Stimuli für die restlichen E/E-Architekturkomponenten bereitstellt. Eine zentrale und in [S3] implementierte Co-Simulationsschnittstelle synchronisiert die Stellvertreter in Ptolemy II mit den SUMO-Akteuren. Die Zeitsynchronisation zwischen den beiden Simulatoren wird ebenfalls mit dieser Schnittstelle realisiert. Sie wurde mittels der *Traas*-Bibliothek, einer Java-API für SUMO, als Ptolemy II Actor implementiert. [S3]

Abbildung 7.3 zeigt die Ergebnisse der zeitdiskreten Simulation eines Abstandsregeltempomaten mit Ptolemy II (DE Director) aus [S3]. Dabei sind die Geschwindigkeiten des EFs $V_{Ego-Fahrzeug}$ und eines vorausfahrenden Fahrzeugs ($V_{ORU}$) sowie deren Abstand zueinander bei unterschiedlichen Zeitschrittweiten über der Zeit aufgetragen. In **Phase II** nähert sich das EF dem vorausfahrenden Fahrzeug (ORU) an. **Phase III** zeigt die Einregelung der Geschwindigkeit, die sich in **Phase IV** auf den Zielwert stabilisiert hat. Anhand der Amplituden in Phase III ist zu erkennen, dass die Zeitschrittweite einen Einfluss auf die Regelgüte hat, was auf unterschiedliche Totzeiten in der Regelungsschleife zurückzuführen ist. Gleichzeitig wirkt sich die Zeitschrittweite auf den zeitabhängigen Scaleup $\mathbf{S}_t$ aus, wie in Tabelle 7.3 für unterschiedliche Schrittweiten aufgelistet. Eine zehnfach höhere Auflösung führt dabei zu einem ca. 5,3-fachen Anstieg der Laufzeit. [S3]

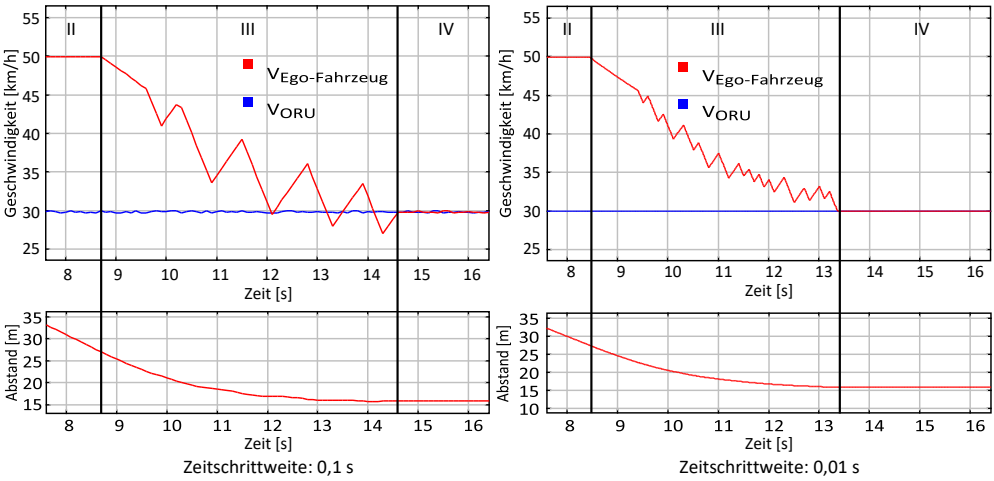



Abbildung 7.3.: Regelgüte eines Abstandsregeltempomaten in Abhängigkeit der zeitlichen Auflösung der Simulation (Quelle: Modifikation von [S3]).

Tabelle 7.3.: Zeitabhängiger Scaleup $S_t$ in Abhängigkeit unterschiedlicher Zeitschrittweiten (Quelle: eigene Darstellung nach [S3]).

| $S_t(0,1)$ | $S_t(0,05)$ | $S_t(0,01)$ |
|---|---|---|
| 7,04 | 4,14 | 1,33 |

**Fazit** Die Zeitschrittweite beeinflusst sowohl die Genauigkeit der Simulation als auch deren Laufzeit. Es ist notwendig eine anwendungsgerechte Wahl der Zeitschrittweite zu treffen, um sowohl eine ausreichende Genauigkeit als auch eine entsprechende Performanz zu erreichen. Da die Zeitschrittweite jedoch in der kontinuierlichen Realität nicht vorkommt, kann die Auslegung des Reglers auf dieser Basis nicht stattfinden. Es müsste entweder ein kontinuierliches Berechnungsmodell oder eine entsprechend andere Abstraktion gewählt werden, welche die Zeitschrittweite rechtfertigt. Die Schrittweite könnte so z.B. auf Basis der Sample-Frequenz der A/D-Wandlung eines Sensorsignals festgelegt werden. Zusätzlich sind die Änderungsraten der Signale in den verschiedenen Phasen zu beachten. Während in Phase III eine möglichst kurze Zeitschrittweite nötig ist, reicht in Phase IV auch eine längere. Damit ist ein Simulator mit variabler Schrittweitenanpassung zu empfehlen.

#### Verifikation eines Notbremsassistenten anhand von UNECE-Richtlinien

Die Konzepte des EFs wurden in [P2] erweitert und unter Beachtung von [S6] um die Modellierung und Integration von Szenarien ergänzt, sodass FAS auf der Grundlage von Prüfvorschriften evaluiert werden können. UNECE-Richtlinien beschreiben textuell u.a. Prozesse, Prüfvorschriften, Grenzwerte und Testszenarien zur Zulassung von Fahrzeugsystemen. Zunehmend werden UNECE-Richtlinien für die Zulassung von FAS erweitert. So existiert für das FAS *Notbremsassistent* eine eigene Richtlinie [284]. Das darin textuell beschriebene Szenario kann in ein Modell übersetzt und zur simulativen Evaluation einer Notbremsassistenzfunktion in die E/E-Architektursimulation integriert werden. Das resultierende Konzept aus [P2] ist in Abbildung 7.4 dargestellt. **UNECE**-Richtlinien bilden nach [P2] die Grundlage des **Testszenarios** (wie in Unterabschnitt 6.2.4 beschrieben), da sie **Evaluationsparameter** und **Testphasen** vorgeben. Des Weiteren dient die Beschreibung der **Teststrecke**, der **Akteure** (statisch und dynamisch) und weiterer **Randbedingungen** als Grundlage für die **Konfigurationsdateien** des 3D- und Physiksimulators **OpenDS** (siehe Unterabschnitt 2.4.4). Die Teststrecke wird dabei in **SUMO** modelliert, um einen Export im **OpenDrive**-Format zu ermöglichen. Die E/E-Architektur wird in **PREEvision** modelliert, woraus anschließend nach [43] ein **Ptolemy II** Modell für die **FAS-Funktion** generiert wird. Das Modell wird dann manuell um die Komponenten **Testszenario FSM** und **Ego-Fahrzeug** (EF) erweitert. Das EF ist ein Stellvertreter des in OpenDS co-simulierten EFs und stellt einerseits eine Schnittstelle zu **Sensor**- und Fahrzeugdaten als Stimuli für die Ptolemy II Simulation bereit. Andererseits werden Daten zur **Kontrolle** des EFs, wie *Beschleunigen* oder *Bremsen*, an das OpenDS-EF weitergeleitet. Das OpenDS-EF wird also von der Ptolemy II E/E-Architektursimulation „ferngesteuert“. [P2]

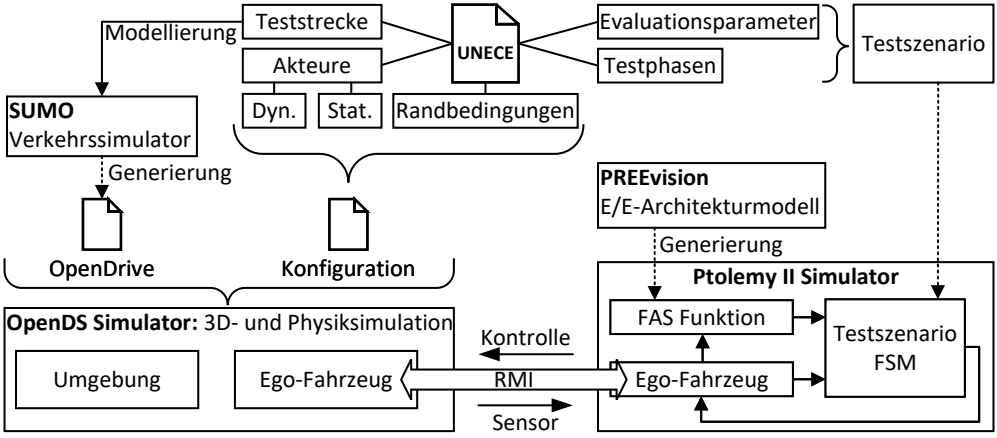



Abbildung 7.4.: Konzept: Simulative Zulassungsevaluation von FAS anhand von UNECE-Richtlinien mit einer Co-Simulation von Ptolemy II und OpenDS (Quelle: Modifikation von [P2]).

Die UNECE-Richtlinie 131 [284] definiert das FAS *Advanced Emergency Braking System (AEBS)* (Notbremsassistent) als *„ein System, das automatisch eine mögliche Frontkollision erkennen und das Fahrzeugbremssystem aktivieren kann, sodass das Fahrzeug abgebremst wird, um eine Kollision zu vermeiden oder abzuschwächen.“*.

Für die simulative Zulassungsevaluation des AEBS wird in [P2] eine AEBS-Funktion implementiert, deren Algorithmus die Gleichungen 7.4 bis 7.6 verwendet.

Die maximale Verzögerung $a_{max}$ berechnet sich nach [P2] aus der Bremskraft des EFs $F_{Bremse}$ und dessen Masse $m_{EF}$:

$$a_{max} = \frac{F_{Bremse}}{m_{EF}} \tag{7.4}$$

Nach [P2] beschreibt $\hat{a}$ die nötige Verzögerung, um das EF bei einer Geschwindigkeit von $V_{EF}$ innerhalb einer Strecke von $\Delta d$ abzubremsen:

$$\hat{a} = \frac{V_{EF}^2}{2 * \Delta d} \tag{7.5}$$

$\Delta d$ entspricht dabei gerade dem Abstand zwischen dem EF und einem Hindernis (vgl. Abbildung 7.5), dessen Wert über ein Radar ermittelt wird. Leitet das EF bei $\hat{a} = a_{max}$ eine Notbremsung ein, würde es also in einem Abstand $d$ von 0 m an einem Ziel halten. Durch einen Faktor C kann $d$ nach [P2] angepasst werden:

$$\hat{a} >= C * a_{max} \tag{7.6}$$

Abbildung 7.5 zeigt nach [P2] die einzelnen Phasen (0-5) des für die Evaluation gewählten Szenarios aus der UNECE-Richtlinie 131 [284].

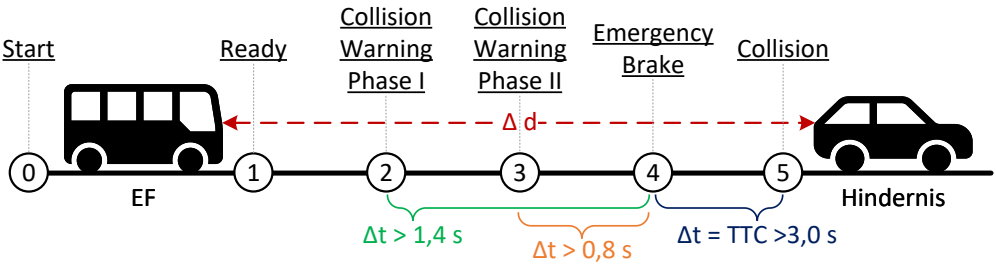



Abbildung 7.5.: Szenario, abgeleitet aus der UNECE-Richtlinie 131 [284], zur Zulassungsevaluation eines AEBS (Quelle: eigene Darstellung nach [P2]).

Der daraus abgeleitete Zustandsautomat (Testszenario FSM) ist in Abbildung 7.6 dargestellt.

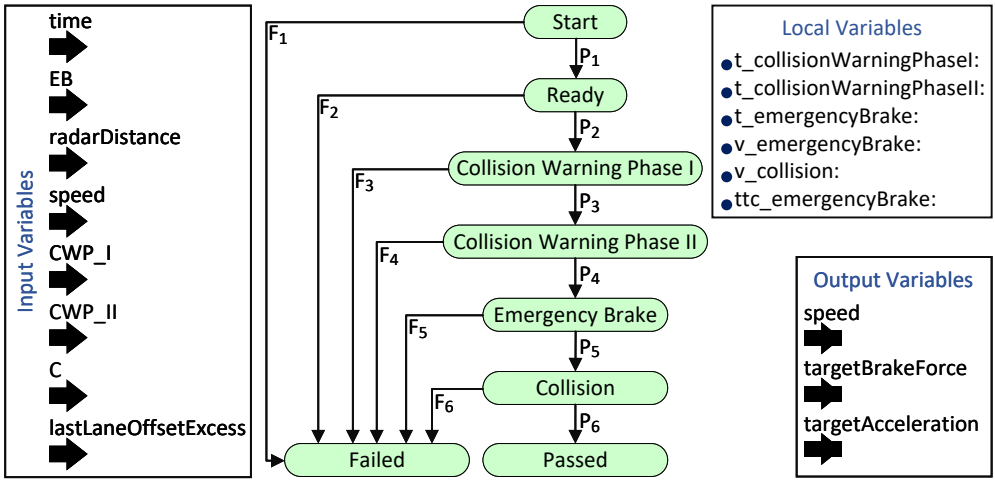



Abbildung 7.6.: Zustandsautomat, abgeleitet aus der UNECE-Richtlinie 131 [284], zur Zulassungsevaluation eines AEBS (Quelle: Modifikation von [P2]).

In der Phase **Start** wird das EF (Fahrzeugklasse M3 mit mehr als acht Sitzen und einem Gewicht von 5 t) auf eine Zielgeschwindigkeit von 80 km/h beschleunigt. [P2]

Für den Übergang in die Phase **Ready** muss das EF nach [P2] beim Erreichen der Zielgeschwindigkeit noch mindestens 120 m von einem auf derselben Fahrbahn stehenden **Hindernis** (Fahrzeugklasse M1, PKW oder Soft-Target) entfernt sein und darf nicht mehr als 0,5 m von der Fahrspurmitte abweichen. [P2]

Kollisionswarnphase I (**Collision Warning Phase I**) soll nach [P2] nicht später als **1,4 s** vor der Notbremsphase (*Emergency Brake*) aktiviert werden. Es muss dabei mindestens ein Warnhinweis in akustischer oder haptischer Form erfolgen. Eine Geschwindigkeitsreduzierung soll in dieser Phase 15 km/h oder 30 % der gesamten Geschwindigkeitsreduktion nicht übersteigen. Sie wird eingeleitet, wenn $\hat{a} >= a_{warn1} = C_1 * a_{max}$ erfüllt ist. [P2]

Kollisionswarnphase II (**Collision Warning Phase II**) folgt nach [P2] auf Kollisionswarnphase I und soll nicht später als **0,8 s** vor der Notbremsphase aktiviert werden. Es müssen mindestens zwei Warnhinweise in akustischer, haptischer oder optischer Form erfolgen. Eine Geschwindigkeitsreduzierung soll in dieser Phase 15 km/h oder 30 % der gesamten Geschwindigkeitsreduktion nicht übersteigen. Sie wird eingeleitet, wenn $\hat{a} >= a_{warn2} = C_2 * a_{max}$ gilt. [P2]

Die Notbremsphase darf nach [P2] frühestens **3 s** vor einer in der Richtlinie geforderten Kollision (**Collision**) eingeleitet werden. Die Geschwindigkeit beim Aufprall auf das Zielobjekt soll nicht weniger als 10 km/h betragen. Sie wird eingeleitet, wenn $\hat{a} >= C_3 * a_{max}$ gilt. [P2]

Der Test ist nach [P2] bestanden (**Passed**), wenn alle auf den Zustandsübergängen definierten Positivbedingungen ($\boldsymbol{P_i}$) erfüllt sind. Sobald eine Negativbedingung ($\boldsymbol{F_i}$) wahr wird, ist der Test nicht bestanden (**Failed**). Die Bedingungen werden mit Booleschen Ausdrücken definiert, wobei die Eingänge (**Input Variables**) die Istzustände des AEBS und des OpenDS-EF dafür bereitstellen. Lokale Variablen (**Local Variables**) werden zum Speichern von Zwischenergebnissen verwendet. Über die Ausgänge (**Output Variables**) werden die vom Szenario vorgegebenen Sollwerte als Steuersignale für das OpenDS-EF ausgegeben. [P2]

Für einen ersten Test wurden nach [P2] die Parameterwerte $C_i$ aus Tabelle 7.4 gewählt:

Tabelle 7.4.: Parameterwerte für Test 1 (Quelle: [P2]).

| $C_1$ | $C_2$ | $C_3$ |
|---|---|---|
| 0,4 | 0,6 | 1 |

$C_1$ und $C_2$ wurden nach [P2] empirisch per Simulation ermittelt. $C_3$ entspricht der Annahme, dass das EF direkt vor dem Hindernis zum Stehen kommt.

Abbildung 7.7 zeigt im linken Diagramm den Verlauf der Zustände des Testszenario-Zustandsautomaten. Das rechte Diagramm zeigt den Abstand $\boldsymbol{\Delta d}$ zwischen dem EF und dem Hindernis sowie die Geschwindigkeit des EFs $\boldsymbol{V_{EF}}$ über der Zeit. Es ist zu erkennen, dass der Test nicht bestanden wurde (**Failed**), da keine Kollision stattgefunden hat. $\Delta d$ zeigt über den konstanten Verlauf nach ca. 27 s, dass das EF mit $d$ = 32 m vor dem Hindernis zum Stehen gekommen ist. Eine Analyse ergab, dass entsprechende Fahrwiderstände, die z.B. durch den Rollwiderstand des Reifens auf der Straße verursacht werden, vernachlässigt wurden. Diese führen jedoch zu einer größeren Verzögerung, wodurch das EF früher zum Stehen kommt und eine Kollision ausbleibt. [P2]

Abbildung 7.8 zeigt nach [P2] die Verläufe der Verzögerungen $a_i$ des EFs im linken Diagramm und im rechten Diagramm die Aktivierung bzw. Deaktivierung der Signale der AEBS-Funktion über der Zeit. Ein Signal ist aktiv, wenn es einen Pegel > 0 aufweist. Es handelt sich um die Signale *Collision Warning Phase I* (**CWP_I**), *Collision Warning Phase II* (**CWP_II**), *Emergency Brake* (**EB**) sowie *Collision* (**C**). Sie sind in Abbildung 7.6 als **Input Variables** zu

finden und werden für die Evaluation genutzt. Anhand der linken Grafik ist zu erkennen, dass der Algorithmus funktioniert und eine Verzögerung eingeleitet wird, sobald $\hat{a} = a_{max}$ gilt (Schnittpunkt von roter und blauer Kurve). Die rechte Grafik zeigt, dass die Bedingungen *CWP_I > 1,4 s vor EB* und *CWP_II > 0,8 s vor EB* erfüllt sind. Allerdings findet keine Kollision statt, weswegen **C** inaktiv bleibt. Daraus lässt sich schließen, dass $C_1$ und $C_2$ korrekt gewählt wurden, aber eine Anpassung von $C_3$ notwendig ist. Mit den gewonnenen Werten und Erkenntnissen aus der Simulation lässt sich $C_3$ bestimmen. [P2]

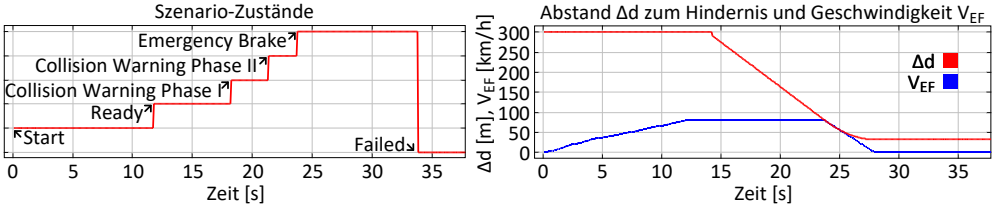



Abbildung 7.7.: Test 1: Testszenariozustände, Abstand $\Delta d$ zwischen EF und Hindernis sowie Geschwindigkeit des EFs $V_{EF}$ über der Zeit (Quelle: Modifikation von [P2]).

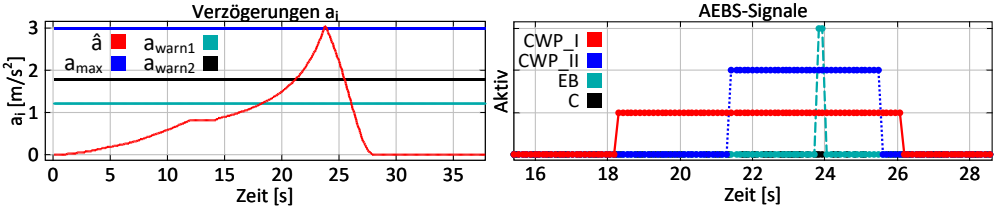



Abbildung 7.8.: Test 1: Verzögerungen $a_i$ des EFs und Aktivierung bzw. Deaktivierung der AEBS-Signale über der Zeit (Quelle: Modifikation von [P2]).

Der Idealbremsweg $d_{ideal}$ des EFs ergibt sich nach [P2] bei 80 km/h mit $a_{max} = 3\frac{m}{s^2}$ zu:

$$d_{ideal} = \frac{V_{EF}^2}{2 * \hat{a}} = \frac{(80\,km/h)^2}{2 * 3\,\frac{m}{s^2}} = 82,3\,m \tag{7.7}$$

Der reale Bremsweg $d_{real}$ lässt sich nach [P2] anhand der Differenz zwischen $d_{ideal}$ und der restlichen Distanz von $d = 32$ m zum Hindernis berechnen:

$$d_{real} = d_{ideal} - d = 82,3\,m - 32\,m = 50,3\,m \tag{7.8}$$

Daraus lässt sich nach [P2] die reale, maximale Verzögerung des EFs berechnen:

$$a_{max_{real}} = \frac{V_{EF}^2}{2 * d_{real}} \approx 4,9\,\frac{m}{s^2} \tag{7.9}$$

$C_3$ lässt sich nach [P2] dann bestimmen zu:

$$C_3 = \frac{a_{max_{real}}}{a_{max}} = \lceil 1,63 \rceil \approx 1,7 \tag{7.10}$$

In einem zweiten Test in [P2] wird $C_3 = 1,7$ gesetzt. Die Ergebnisse sind analog zum ersten Test in Abbildung 7.9 und Abbildung 7.10 dargestellt. Mit Abbildung 7.9 lässt sich im linken Diagramm erkennen, dass der Test erfolgreich abgeschlossen wurde (**Passed**). Im rechten Diagramm wird $\Delta d$ bei ca. 28,6 s kurzzeitig negativ und vergrößert sich durch den Impuls nach dem Aufprall auf das Hindernis wieder. Bei dem Hindernis handelt es sich um einen Personenkraftwagen mit gelöster Feststellbremse. Des Weiteren wird durch den Aufprall die Verzögerung des EFs sprunghaft erhöht, wie in der vergrößerten Ansicht von $V_{EF}$ für den Zeitbereich von 28,55 s bis 28,65 s zu sehen ist. [P2]

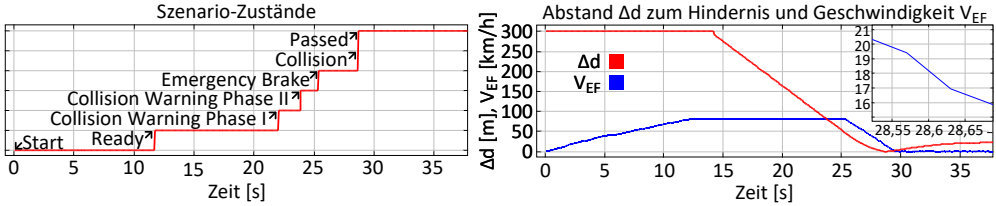



Abbildung 7.9.: Test 2: Testszenariozustände, Abstand $\Delta d$ zwischen EF und Hindernis sowie Geschwindigkeit des EFs $V_{EF}$ über der Zeit (Quelle: Modifikation von [P2]).

Abbildung 7.10 zeigt nach [P2] im linken Diagramm, dass $\hat{a}$ an der Stelle $1,7 * a_{max}$ weiter ansteigt, anstatt wie in Abbildung 7.8 zu sinken. Der Punkt, an dem eine Notbremsung eine Kollision noch vermeiden würde, ist also überschritten. Die AEBS-Signale im rechten Diagramm bestätigen ferner, dass die geforderten Zeiten für die Übergänge zwischen den Phasen eingehalten werden. [P2]

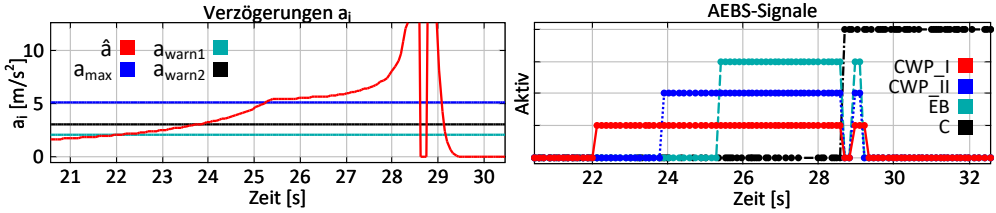



Abbildung 7.10.: Test 2: Verzögerungen $a_i$ des EFs und Aktivierung bzw. Deaktivierung der AEBS-Signale über der Zeit (Quelle: Modifikation von [P2]).

**Fazit** Die Untersuchung aus [P2] hat gezeigt, dass die Integration von ausführbaren Szenarien einen Mehrwert für E/E-Architekten bei Entwurfsentscheidungen bietet (Bestimmung des

Faktors $C_3$). Durch entsprechende Schnittstellen, die sich aus den Übergangsbedingungen im Zustandsautomaten ergeben, werden außerdem die für den Test notwendigen Signale und damit die benötigten Komponenten der E/E-Architektur vorgegeben. Die mittels der Komponente EF realisierte Co-Simulation einer 3D- und Physikumgebung hat durch die Visualisierung beim Verständnis geholfen. Des Weiteren wurde damit aufgezeigt, dass die Fahrdynamik bei der Untersuchung von FAS nicht vernachlässigt werden sollte, wenn die implementierten FAS-Algorithmen auf Sensordaten zugreifen, die von der Fahrdynamik des EFs abhängen (AEBS-Regelungsalgorithmus). [P2]

Trotz einer 3D-Co-Simulation und dem damit verbundenen Visualisierungsaufwand, wurde die Simulation auf System A mit ca. 30 Bildern pro Sekunde ausgeführt. Der Anwendungsfall konnte daher mit Ptolemy II vollständig ausgeführt werden.

### 7.2.2. Angriffserkennungssystem für Spurwechselassistenten

In [P4] wurde der Einsatz einer E/E-Architektursimulation zur Evaluation eines Angriffserkennungssystems (IDS) untersucht. Durch die in [9] beschriebenen Anforderungen an Cybersecurity-Simulationen und die vorausgegangenen Erkenntnisse wird deutlich, dass neben den vernetzten E/E-Komponenten auch die Umgebung und die Fahrzeugdynamik simuliert werden müssen. Um Angriffe auf definierten Pfaden innerhalb von Missbrauchsszenarien simulieren zu können (vgl. Unterabschnitt 3.1.4), wurde in [P4] das in Abbildung 6.2 beschriebene Konzept einer HW-zentrierten Simulation umgesetzt. Dafür wurden die Schnittstellen der Co-Simulation im Vergleich zu [P2] angepasst, wie Abbildung 7.11 zeigt.

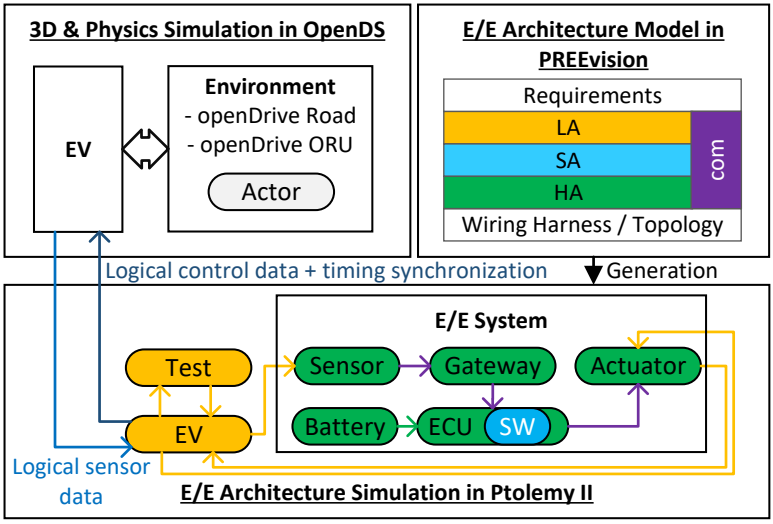



Abbildung 7.11.: Erweiterte Co-Simulation mit HW-zentrierter Abbildung (Quelle: Modifikation von [P4]).

Nach [P4] bleibt Ptolemy II der Master in einer Co-Simulation mit OpenDS, analog zur Darstellung in Abbildung 7.4. Allerdings basiert das E/E-System nun auf der Struktur und den Komponenten der **HA**. Sensoren (**Sensor**) wandeln vom EF (***Ego Vehicle (EV)***) kommende logische Daten (gelb dargestellt) in physikalische Signale oder Bussignale (beide violett dargestellt). Aktoren (**Actuator**) führen eine Wandlung in die entgegengesetzte Richtung durch. So können Hardware-Charakteristiken, wie z.B. Rauschen oder Verzögerungen, oder das Versorgungsnetz (**Battery**) mitsimuliert werden. [P4]

Zum Test des IDS für einen Spurwechselassistenten wird nach [P4] das in Abbildung 7.12 beschriebene und an [44] angelehnte Szenario verwendet.

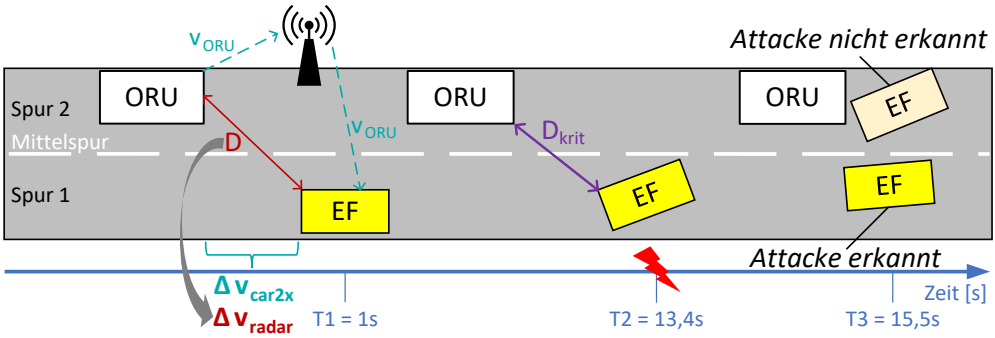



Abbildung 7.12.: Konzept des Szenarios einer Angriffserkennung (IDS) für Spurwechselassistenten (Quelle: Modifikation von [P4]).

Zum Zeitpunkt **T1** beschleunigt das **EF** auf **Spur 1** von 0 auf 80 km/h, während der **ORU** auf **Spur 2** von 0 auf 120 km/h beschleunigt. Beide haben zu Beginn einen Abstand ***D*** von 300 m. Bei **T2** erreichen beide ihre konstanten Zielgeschwindigkeiten und einen kritischen Abstand $\boldsymbol{D_{krit}}$ von 60 m. Ein Abstand unterhalb $D_{krit}$ würde bei einem Spurwechsel zur Kollision führen. An dieser Stelle werden zwei unterschiedliche Missbrauchsszenarien ausgeführt. [P4]

In einem ersten Testszenario wird nach [P4] der Abstand $D$, der über das Radar des EFs ermittelt werden soll, von einem Angreifer manipuliert, sodass immer $D > D_{krit}$ gilt. $D$ bildet jedoch innerhalb des Spurwechselassistentenalgorithmus die Entscheidungsgrundlage für einen Spurwechsel, sodass dieser nur bei $D > D_{krit}$ durchgeführt wird. Das IDS soll zur Erkennung der Manipulation ein redundantes Car-2-X-Signal verwenden. Die Funktionsweise basiert auf physikalischen Merkmalen, Informationsredundanz und Überwachung. Gleichungen 7.11 bis 7.14 beschreiben die Funktionsweise. [P4]

Der mit dem Radar ermittelte Abstand $D$ wird nach [P4] abgeleitet, um die Differenzgeschwindigkeit zwischen dem EF und dem ORU zu ermitteln:

$$\Delta v_{radar} = D' \tag{7.11}$$

Mit dem Car-2-X-Signal $v_{ORU}$ (Geschwindigkeit des ORUs) und der Geschwindigkeit des EFs $v_{EF}$, wird die Differenzgeschwindigkeit nach [P4] redundant berechnet:

$$\Delta v_{car2x} = v_{ORU} - v_{EF} \tag{7.12}$$

Die Differenz der beiden Differenzgeschwindigkeiten $\Delta v_{radar} - \Delta v_{car2x}$ sollte nach [P2] also 0 betragen, wenn sie übereinstimmen. Falls nicht, liegt eine Anomalie vor und es ist ein Angriff zu vermuten [P4]. Um Rauschen herauszufiltern, wird nach [P4] zunächst über der Differenz noch ein gleitender Mittelwert über M (20) Stichproben gebildet:

$$ma = \frac{1}{M} \sum_{i=1}^{M} (\Delta v_{radar} - \Delta v_{car2x})_i \tag{7.13}$$

Der gleitende Mittelwert ma multipliziert mit einem konstanten Faktor c bildet nach [P4] den Schwellwert $th_{IDS}$. Ein Angriff wird nach [P4] dann angenommen, wenn $\Delta v_{radar} - \Delta v_{car2x}$ eine höhere Änderungsrate als $th_{IDS}$ aufweist:

$$\Delta v_{radar} - \Delta v_{car2x} \geq ma \cdot c = th_{IDS} \tag{7.14}$$

In einem zweiten Testszenario wird in [P4] das redundante Car-2-X-Signal $v_{ORU}$ noch zusätzlich um 2 s verzögert, um in der Realität auftretende Kommunikations- und Berechnungslatenzen nachzubilden.

Das für [P4] erzeugte Ptolemy-II-Modell ist in Abbildung A.3 im Anhang zu finden. In Übereinstimmung mit dem in Abbildung 6.2 dargestellten Konzept dient der Block **EV** (EF) als Schnittstelle zu OpenDS. Der Block **Test** beinhaltet das jeweilige Testszenario in Form eines Zustandsautomaten nach Abbildung 6.4. Das **E/E-System** besteht aus **Sensoren** (grün), **Aktoren** (rot), **ECUs** (blau) und einem **Gateway** (lila), auf welchem der Angriff erfolgt. Hier wird das Radarsignal, das dem Abstand D zum ORU entspricht, vom Angreifer manipuliert. Logische Signale sind gelb, Bussignale grün, physikalische Signale grau und das manipulierte Bussignal rot dargestellt. Der **FAS-HPC** führt neben dem IDS und dem Spurwechselassistenten noch die Funktionen *Spurhalteassistent* und *Abstandsregeltempomat* aus.[P4]

Die Ergebnisse beider Testszenarien aus [P4] sind in Abbildung 7.13 dargestellt.

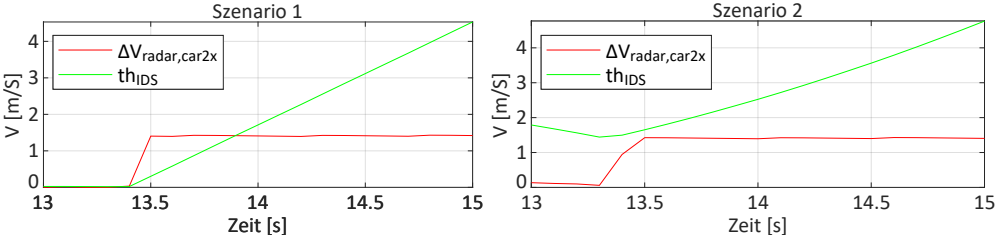



Abbildung 7.13.: Angriffserkennungssimulation: Szenario 1 zeigt eine erfolgreiche Erkennung und Szenario 2 eine fehlgeschlagene (Quelle: Modifikation von [P4]).

In beiden Szenarien wird nach [P4] der Faktor c (vgl. Gleichung 7.14) auf den Wert 4 gesetzt. Der Angriff findet jeweils bei 13,4 s statt, weil zu diesem Zeitpunkt die Vorbedingungen erfüllt sind. Sobald Gleichung 7.14 erfüllt ist und die rote Linie über der Grünen liegt, kann ein Angriff erkannt werden. Im ersten Szenario ist dies deutlich der Fall. Ein Spurwechsel wird daher nicht ausgeführt. Das Diagramm vom zweiten Szenario auf der rechten Seite zeigt jedoch, dass der gewählte Wert für c bei einer Latenz von 2 s für eine erfolgreiche Angriffserkennung zu hoch ist. Da der Schwellwert nicht überschritten wird, kommt es zu einem Spurwechsel und damit zur Kollision. Aus diesem Grund schlägt der Test fehl. [P4]

**Fazit** Die Konzepte EF, Szenario und HW-zentrierte Abbildung sind hier vereint. Die HW-zentrierte Abbildung ermöglicht nach [P4] die Simulation von Angriffen auf konkreten Angriffspfaden. Mit dem zweiten Szenario wird insbesondere deutlich, dass die Integration von Hardware-Charakteristiken bei der Evaluation von FAS notwendig ist, da sie sich auf die korrekte Funktion des Algorithmus auswirken können. Durch die Anwendung von Szenarien in einer frühen Entwicklungsphase konnte das IDS zudem als direkter Bestandteil des Spurwechselassistentenalgorithmus integriert werden. Ein solches Vorgehen kann Sicherheitslücken bereits während der Entwicklung schließen und zu späteren Zeitpunkten ausgeführte Bedrohungsanalysen präventiv ergänzen. Eine standardisierte und modellbasierte Beschreibung von Missbrauchsfällen, als Richtlinie für FAS und mit entsprechenden Schnittstellen, wäre für die Zukunft wünschenswert. [P4]

Trotz einer 3D-Co-Simulation und dem damit verbundenen Visualisierungsaufwand, wurde die Simulation auf System A mit ca. 30 Bildern pro Sekunde ausgeführt. Der Anwendungsfall konnte daher mit Ptolemy II vollständig ausgeführt werden.

### 7.2.3. Batterieelektrisches Fahrzeug

In [S1] wird die Simulation eines BEFs mit Ptolemy II untersucht. Das Prinzip der Simulation ist in Abbildung 7.14 nach [S1, 190] dargestellt.

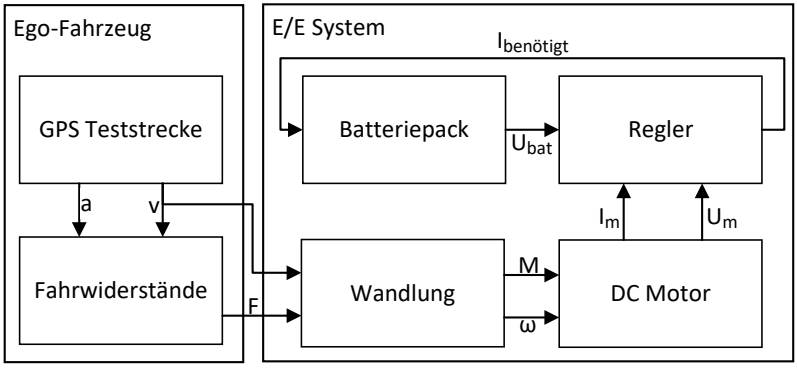



Abbildung 7.14.: Konzept zur Simulation eines BEF (Quelle: eigene Darstellung nach [S1, 190]).

In [S1] wird eine *Global-Positioning-System-Teststrecke (**GPS-Teststrecke**)* nach [190] verwendet, um das **EF** auf die für unterschiedliche GPS-Positionen geltenden Geschwindigkeiten zu beschleunigen. Dazu wird eine beliebige Route mit *Open Street Map (OSM)* [198] erstellt und als Datei im GPS-Exchange-Format (.gpx) exportiert. Diese Datei enthält die Wegpunkte der Strecke in Form von Koordinaten. Durch Zusatzwerkzeuge wie *JOSM* [137] oder die *Overpass API* [200] werden den Wegpunkten anschließend die geltenden Geschwindigkeitsbegrenzungen automatisiert zugewiesen. Das EF fährt in der Simulation diese Wegpunkte ab und regelt seine Geschwindigkeit $v$ auf die geltenden Begrenzungen ein. Der Beschleunigungswert $\boldsymbol{a}$ ist als Parameter des EFs festgelegt. Im Block **Fahrwiderstände** wird der Gesamtfahrwiderstand $\boldsymbol{F}$ berechnet. Neben $v$ und $a$ werden dafür auch fahrzeugspezifische Parameter, wie das Fahrzeuggewicht oder der cw-Wert, verwendet (siehe Unterabschnitt 6.2.3). $F$ und $v$ dienen anschließend als Eingabe für den Block **Wandlung**. Dort werden daraus die für den Block **DC Motor** (Gleichstrommotor) benötigten Eingaben berechnet. Dazu zählen das Lastmoment $\boldsymbol{M}$ (siehe Gleichung 6.7) und die Drehgeschwindigkeit $\boldsymbol{\omega}$, die sich wie folgt berechnet:

$$\omega = \frac{v * i}{r_{Rad}} \tag{7.15}$$

Dabei ist $i$ das Übersetzungsverhältnis und $r_{Rad}$ ist der Raddurchmesser. Der Block **DC Motor** ist als Differenzialgleichungssystem nach [175] implementiert. So werden der Strom durch den Motor $\boldsymbol{I_m}$ und die Motorspannung $\boldsymbol{U_m}$ berechnet, die als Eingang für den Block **Regler** dienen. Dieser bestimmt daraus zusammen mit der Batteriespannung $\boldsymbol{U_{bat}}$ den Strom $I_{benötigt}$. Damit kann schließlich der zeitabhängige Batterieladezustand im Block **Batteriepack** berechnet werden. Eine Batteriezelle des Batteriepacks wurde in [S1] mit mathematischen Ausdrücken nach [51] implementiert, da Ptolemy II keine physikalische Modellierung unterstützt. Zur Validierung wurde das entwickelte Batteriezellenmodell in [S1] außerdem mit dem Referenzmodell verglichen, wie Abbildung 7.15 zeigt. [S1]

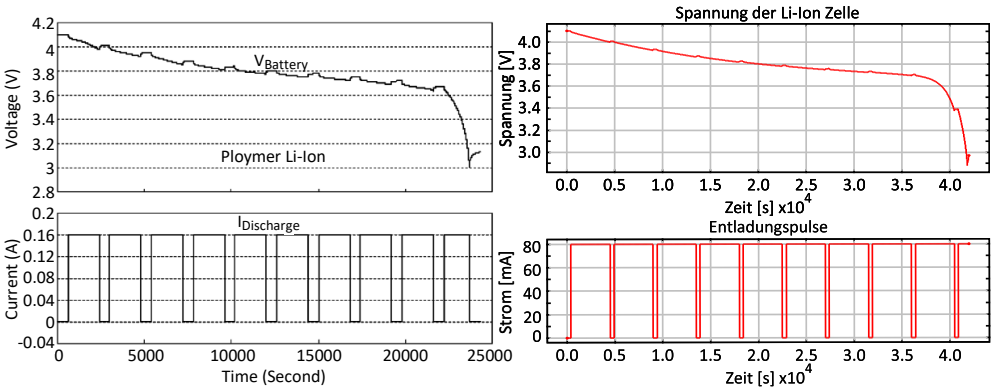



Abbildung 7.15.: Vergleich des Batteriespannungsverlaufs von [51] (links) und [S1] (rechts) bei einer gepulsten Entladung (Quelle: Modifikation von [S1]).

Die linken Diagramme sind nach [S1] dem Referenzmodell aus [51] zuzuordnen und die rechten der Implementierung aus [S1]. Sie stellen jeweils den simulierten Spannungsverlauf des implementierten Batteriezellenmodells bei einer gepulsten Entladung dar. Dabei ist eine Übereinstimmung von 99 % festzustellen. [S1]

Im Zuge dieser Dissertation wurde zusätzlich eine Laufzeitanalyse des in [S1] entwickelten Batteriezellenmodells auf System B (siehe Tabelle 2.4) durchgeführt. Dazu wurden acht Simulationsläufe für eine logische Zeit von jeweils 15 s ausgeführt. Bei jedem Simulationslauf wurde eine zusätzliche Instanz des ursprünglichen Batteriezellenmodells hinzugefügt und mitsimuliert. Die Ergebnisse sind in Tabelle 7.5 dargestellt.

Tabelle 7.5.: Laufzeiten und Speicherverbrauch in Abhängigkeit der Anzahl an Batteriezellenmodellinstanzen für eine Simulation mit Ptolemy II.

| Anzahl der Batteriezellen | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Laufzeit [s] | 16,036 | 28,239 | 42,859 | 57,968 | 73,909 | 90,89 | 105,821 | 121,566 |
| Speicherverbrauch [kB] | 564639 | 626269 | 543542 | 1029100 | 1329169 | 1224167 | 595926 | 1539698 |

Aus der Tabelle geht hervor, dass bereits für eine Batteriezelle $S_t^* = 16,036/15 = 1,069 > 1$ (vgl. Gleichung 2.8) gilt, womit die Simulation nicht skaliert.

Mithilfe des Werkzeugs VisualVM wurde ferner ermittelt, dass 44,4 % der Laufzeit auf die Evaluierung der mathematischen Ausdrücke (**Expression**) zur Beschreibung des Batterieverhaltens zurückzuführen sind (siehe Abbildung 7.16).

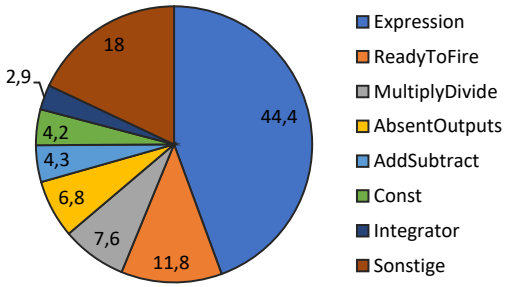



Abbildung 7.16.: Laufzeitverteilung nach Komponenten des Batteriezellenmodells in %.

In jeder Iteration wird dabei ein AST aufgebaut, um die Operatoren innerhalb der Gleichung zu evaluieren. Dies wurde bereits bei den wertdiskreten Benchmarks in Unterabschnitt 5.2.2 beobachtet. Des Weiteren fallen weitere 25,6 % auf Blöcke zur Modellierung mathematischer Logik und 11,8 % auf Datentypprüfungen an den Ports (**ReadyToFire**).

In BEFs werden höhere Spannungen benötigt, was durch eine Serienschaltung der Batteriezellen bewirkt wird. Eine entsprechend größere Kapazität wird durch eine Parallelschaltung der Zellen ermöglicht. Dadurch entstehen Batteriepacks. Eine Simulation jeder einzelnen Zelle eines Batteriepacks ist mit Ptolemy II nicht möglich (siehe Tabelle 7.5).

In [S1] wird zur Batteriezellensimulation eine entsprechende Abstraktion gewählt: Die Ausgangsspannung am Ausgang des Batteriezellenmodells wird mit einem konstanten Faktor, welcher der Anzahl serieller Zellen entspricht, multipliziert. Zur Validierung und Kalibrierung wird ein Referenzbatteriepack mit 192 Zellen aus [163] verwendet. Die Ladekurven für unterschiedliche Entladeströme stimmen dabei zu 94 % mit dem Referenzmodell überein. [S1]

Ebenso wurde in [S1] das in Ptolemy II implementierte und auf [175] basierende Motormodell (vgl. **DC Motor** in Abbildung 7.14) validiert. Dabei wurde die vom Motor aufgenommenen Leistung (**Required Road Power** / **Motorleistung**) in Abhängigkeit einer von der Teststrecke vorgegebenen Fahrzeuggeschwindigkeit (**Required Road Speed** / **Winkelgeschwindigkeit**) und dem daraus abgeleiteten Lastmoment am Motor (**Required Road Torque** / **Lastmoment**) verglichen. Die Ergebnisse zeigt Abbildung 7.17. Die linken Diagramme sind dem Referenzmodell aus [175] zuzuordnen und die rechten der Implementierung aus [S1]. Dabei ist eine Übereinstimmung von 99% festzustellen. [S1]

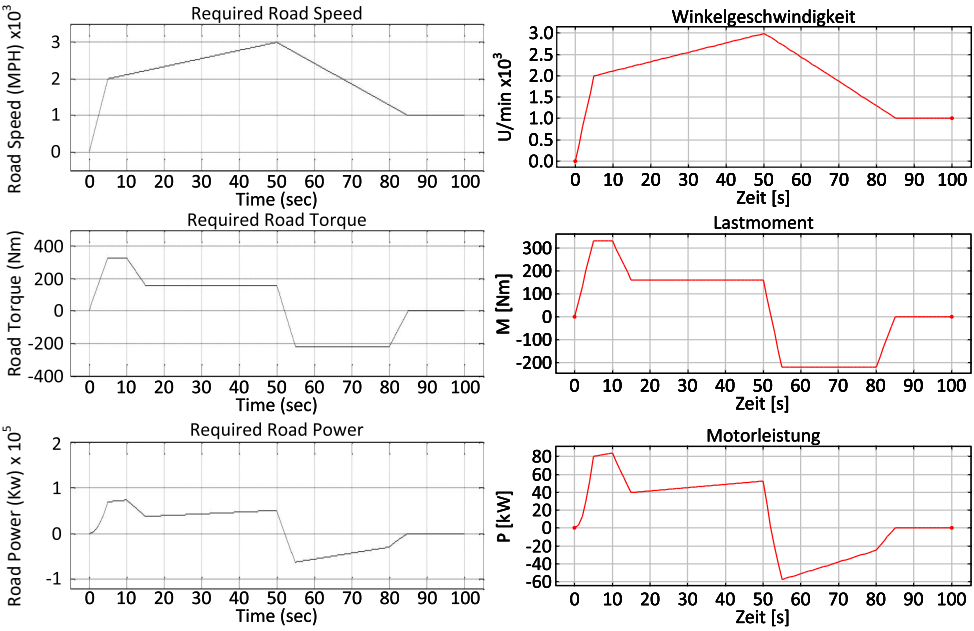



Abbildung 7.17.: Vergleich der Motormodelle aus [175] (links) und [S1] (rechts) (Quelle: Modifikation von [S1]).

Das in [S1] entstandene Ptolemy-II-Modell zur Simulation eines BEFs wurde im Zuge dieser Dissertation für eine Laufzeitanalyse auf System B (vgl. Tabelle 2.4) ausgeführt. Zunächst wurde dazu die in [S1] ausgeführte Route vom ITIV (rotes Lokalisierungssymbol) zur nächstgelegenen Tankstelle (grünes Lokalisierungssymbol) verwendet (siehe Abbildung 7.18). Sie liegt in Form einer .gpx-Datei vor. Während der Ausführung werden Serveranfragen an Overpass

gestellt, um die **Zielgeschwindigkeit** zur Laufzeit zu ermitteln. Der Algorithmus dafür wurde in [S1] aus [190] übernommen.

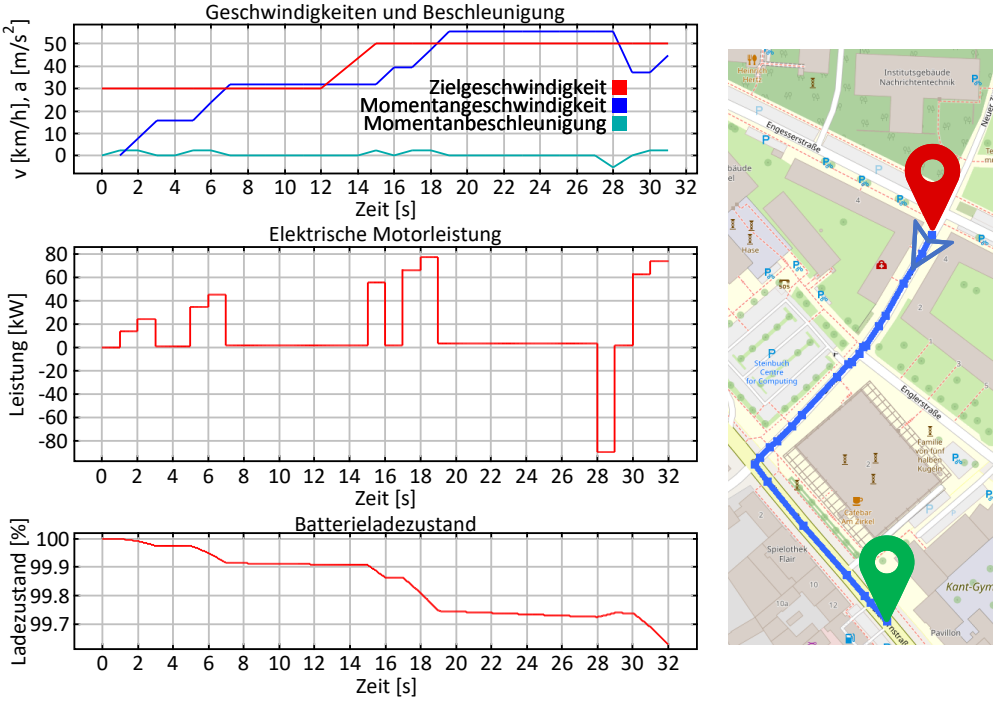



Abbildung 7.18.: Simulation der Fahrt eines BEF vom *Institut für Technik der Informationsverarbeitung (ITIV)* zur nächstgelegenen Tankstelle (Quelle: eigene Darstellung/ Simulation auf Basis von [S1]).

Auf der linken Seite zeigt Abbildung 7.18 die in dieser Dissertation entstandenen Simulationsergebnisse. Neben der in [S1] betrachteten **Motorleistung** sind zusätzlich das Szenario der ausgeführten Teststrecke (**Zielgeschwindigkeit**) und der **Batterieladezustand** dargestellt. Die Fahrt dauert in der Simulation 32 s (logische Zeit). Für die Ausführung wurden 447,367 s (physikalische Zeit) benötigt. Damit ist $S_t^* = 13,99$, womit die Simulation nicht skaliert. Dies ist größtenteils auf einen Timeout bei den Serveranfragen (Overpass API) in Abhängigkeit der Fahrzeugposition zurückzuführen. Die zwischen den Serveranfragen entstehende Totzeit von 1 s verzögert darüber hinaus auch die Einregelung der Momentangeschwindigkeit auf die Zielgeschwindigkeit. Dies erklärt die starke Quantisierung der Momentangeschwindigkeit, der Momentanbeschleunigung und damit der Leistungsaufnahme des Motors. Ab Sekunde 16 werden Anfragen von dem Overpass-Server zudem nur sporadisch beantwortet (Rückgabecode 429), da die Maximalzahl an Anfragen in einem Zeitraum begrenzt ist. Im Hinblick auf die Plausibilität ist zu erkennen, dass die elektrische Leistung des Motors sowohl mit der Momen-

tanbeschleunigung als auch mit der Momentangeschwindigkeit korreliert. Außerdem entlädt sich die Batterie bei einer Beschleunigung schneller als bei konstanter Fahrt. Beim Bremsen wird die Batterie durch Rekuperation wieder aufgeladen. Das Modell ist plausibel.

Zum Vergleich mit Simulink und zur besseren Evaluation der Skalierbarkeit ohne Serveranfragen wurde im Zuge dieser Dissertation noch ein weiteres Szenario aufgebaut. Hierbei ersetzt ein Zustandsautomat (*ModalModel*) die GPS-Teststrecke, wobei dessen Zustände die Zielgeschwindigkeiten vorgeben (siehe Abbildung 7.19). Das EF soll zunächst von 0 auf 50 km/h beschleunigen und nach Erreichen diese Geschwindigkeit für 10 s halten. Danach soll das EF auf 30 km/h abbremsen, diese Geschwindigkeit für 5 s halten und zum Stillstand kommen.

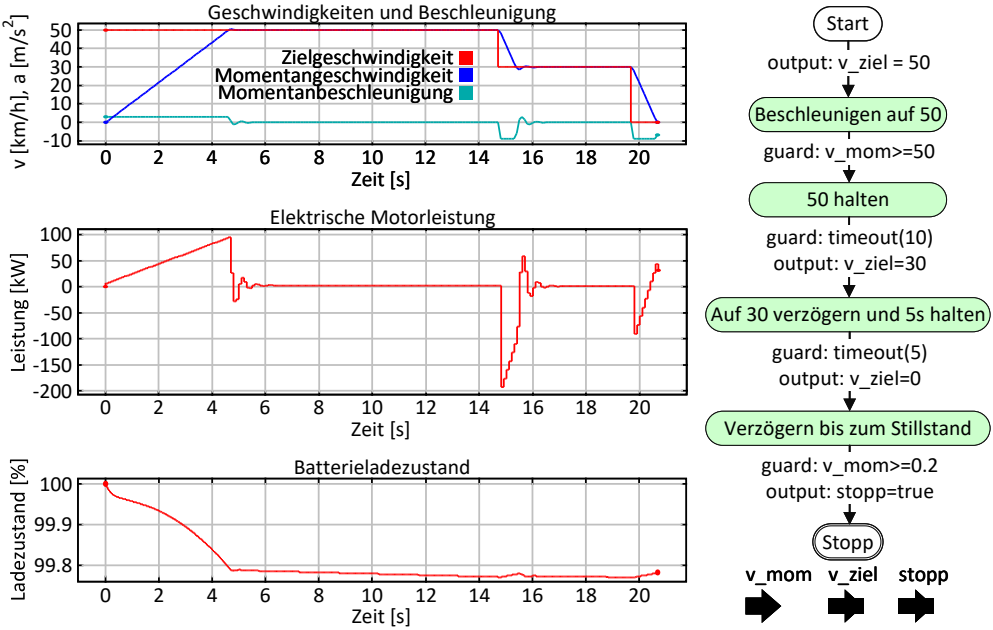



Abbildung 7.19.: Simulation der Fahrt eines BEFs innerhalb eines Szenarios mit vorgegebenen Geschwindigkeiten.

Auf der linken Seite zeigt Abbildung 7.19 die Ergebnisse der Simulation. Die Zusammenhänge zwischen den einzelnen Signalen sind hierbei, im Vergleich zu Abbildung 7.18, noch deutlicher zu erkennen. Dies liegt daran, dass die Totzeiten wegfallen und sich dadurch eine höhere zeitliche Auflösung ergibt, was auch eine bessere Regelung ermöglicht. Beim Beschleunigen wird die Leistungsaufnahme des Motors in Abhängigkeit der wirkenden Fahrwiderstände größer und erreicht eine Maximalleistung von ca. 100 kW. Zum Halten der Geschwindigkeit sind etwa 2,63 kW nötig. Der Batterieladezustand korreliert mit der Motorleistung. Insgesamt treten keine unrealistischen Werte auf und die Ergebnisse sind plausibel. Für eine abschließende Beurteilung wäre jedoch ein Vergleich mit realen Messdaten nötig. Im Hinblick auf die Performanz

lässt sich Folgendes feststellen. Mit einer Ausführungsdauer von 47,9 s und einer simulierten Zeit von 20,7 s gilt $S_t^* = 47,9/20,7 = 2,314 > 1$, womit die Simulation nicht skaliert. Eine Untersuchung mit VisualVM ergab eine ähnliche Verteilung, wie in Abbildung 7.16 dargestellt. Damit verursacht auch hier die Evaluation der mathematischen Formeln den Großteil der Laufzeit.

**Fazit** Die aus einer vorgegebenen Route automatisierte Erstellung einer Teststrecke zur Ableitung von Zielgeschwindigkeiten als Stimuli in einem Szenario hilft Energieverbräuche simulativ und mit geringem Aufwand überprüfen zu können. Die Bedingungen sind dabei von realen Straßendaten abgeleitet, was einen besonderen Nutzen bei der Bestimmung der Reichweite in Abhängigkeit der gefahrenen Route liefert. Für eine ausreichend hohe Genauigkeit müsste die Simulation allerdings noch mit Realdaten kalibriert werden.

Da Ptolemy II kein Batteriemodell besitzt, wurde in [S1] ein eigenes entwickelt und anhand von Datenblättern sowie der Literatur evaluiert. Allerdings handelt es sich dabei um ein spezifisches Modell, das einerseits nur für Li-Ion-Typen geeignet ist und andererseits für jedes weitere Batteriemodell kalibriert werden muss. Da entsprechende Parameter fehlen, ist es nicht als ein fest zugewiesenes Template bei einer E/E-Architektursimulation verwendbar. Eine Untersuchung des Balancings ist ferner nicht möglich, weil die Batteriezellen nicht einzeln modelliert sind. Des Weiteren beruhen sowohl das Batteriemodell als auch das Motormodell aus [S1] auf mathematischen Ausdrücken. Daher sind zur Kopplung dieser elektrischen Komponenten Ersatzstromwerte nötig, was die Anwendbarkeit senkt. Bei physikalischen Modellierungen wäre dies nicht nötig.

Bezüglich der Performanz ließen sich anhand des Anwendungsfalls Schwachstellen bei der Simulation von mathematischen Ausdrücken isolieren. Deswegen und weil Ptolemy II keine physikalische Modellierung für elektrische Netze unterstützt, ist Ptolemy II für den Anwendungsfall eher nicht geeignet.

### 7.2.4. Fazit

Das Ego-Fahrzeug-Konzept ist als Schnittstelle für eine Co-Simulation mit einer 3D- und Physiksimulation geeignet. Dadurch können FAS evaluiert werden, deren Signale von der Fahrdynamik oder von Sensordaten aus der Umgebung (z.B. Kameradaten) abhängen.

Die Integration von ausführbaren Szenarien hilft E/E-Architekten durch vorgegebene Schnittstellen beim Design passender Algorithmen, schränkt das Modell ein und erlaubt eine Evaluation anhand zulassungsrelevanter Umfänge. Des Weiteren konnte mit einer HW-zentrierten Abbildung ein IDS als direkter Bestandteil der FAS-Algorithmen integriert und in früher Entwicklungsphase evaluiert werden. TARA-Methoden können somit ergänzt und Sicherheitslücken bereits während der Entwicklung geschlossen werden.

Allerdings ist die Aussagekraft der Simulationsergebnisse bei der Verwendung von zeitdiskreten MAE für Regler nur begrenzt, wenn die Totzeit, die bei der Verarbeitung auf einer Recheneinheit entsteht, nicht bekannt ist.

Für die untersuchten FAS-Anwendungsfälle skaliert Ptolemy II, da die Ausführung der Co-Simulation mit ca. 30 Bildern pro Sekunde möglich ist. Mit den Ergebnissen aus Abschnitt 5.2 lässt sich darauf schließen, dass Simulink ebenfalls skaliert.

Bei der Simulation des BEFs skaliert Ptolemy II nicht, was auf Schwachstellen bei der Simulation mathematischer Ausdrücke zurückzuführen ist. Ferner unterstützt Ptolemy II keine physikalische Modellierung, weshalb Ersatzstromwerte nötig sind. Diese müssen zudem über ressourcenverbrauchende Rückführungsschleifen bereitgestellt werden.

## 7.3. Simulationen mit Simulink

In diesem Abschnitt wird ergänzend zu den zuvor evaluierten Konzepten und zum Vergleich mit Ptolemy II die in Kapitel 6 beschriebene Simulink-Abbildung evaluiert. Dazu werden Anwendungsfälle im Bereich der Simulation von BEFs untersucht. Außerdem wird hierbei der Einsatz von Templates in Bezug auf die Skalierbarkeit evaluiert.

### 7.3.1. Aufbau eines Modells für batterielektrische Fahrzeuge mit Templates

#### Referenzmodell und Ableitung von Templates

Das Modell „AC7 - Brushless DC Motor Drive“ aus der Simscape-Bibliothek [260], dient als Basis zur Ableitung von Templates und als validierte Referenz zur Simulation eines BEFs. Die Auswahl eines Modells mit einem permanenterregten Synchronmotor ist dabei auf [282, S1] zurückzuführen. Dem Modell fehlen jedoch die Aspekte einer E/E-Architektur, wie eine damit verbundene strukturelle Aufteilung in Hardware- und Software-Komponenten. Des Weiteren fehlen entsprechende Schnittstellen zum EF, welches die Geschwindigkeit und den davon abhängigen Fahrwiderstand vorgibt (vgl. Abbildung 7.14). Aus dem Fahrwiderstand lässt sich wiederum das Lastmoment am Motor ableiten (vgl. Gleichung 6.7). Ferner wird noch ein mobiler Energieträger benötigt. Abbildung 7.20 zeigt das AC7-Simulink-Modell [260], bei dem E/E-typische Komponenten hervorgehoben sind.

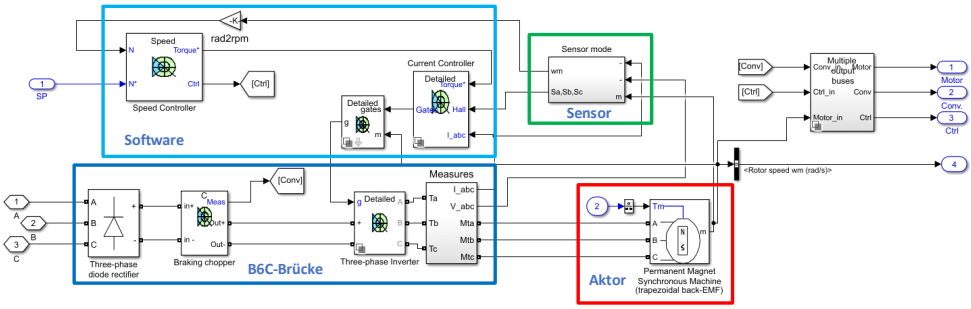



Abbildung 7.20.: Simulink AC7-Brushless-DC-Motor-Drive [260] mit Hervorhebung einer strukturellen Aufteilung in E/E-Architekturkomponenten.

Die hervorgehobenen Komponenten sollen bei der anschließenden Simulation eines BEFs als Templates dienen. Die Komponente **Software** kapselt die Geschwindigkeitsregelung des Motors (**Aktor**). Dabei liefert die Komponente **Sensor** den Istwert der Motordrehzahl. Die Komponente **B6C-Brücke** stellt die Ansteuerströme für die Motorwicklungen in Abhängigkeit der Sollgeschwindigkeit bereit.

In einem weiteren Schritt wurde das Modell, entsprechend der identifizierten E/E-Komponenten, in Subsysteme aufgeteilt und um fehlende Komponenten ergänzt. Das dabei entstandene Modell ist in Abbildung 7.21 dargestellt.

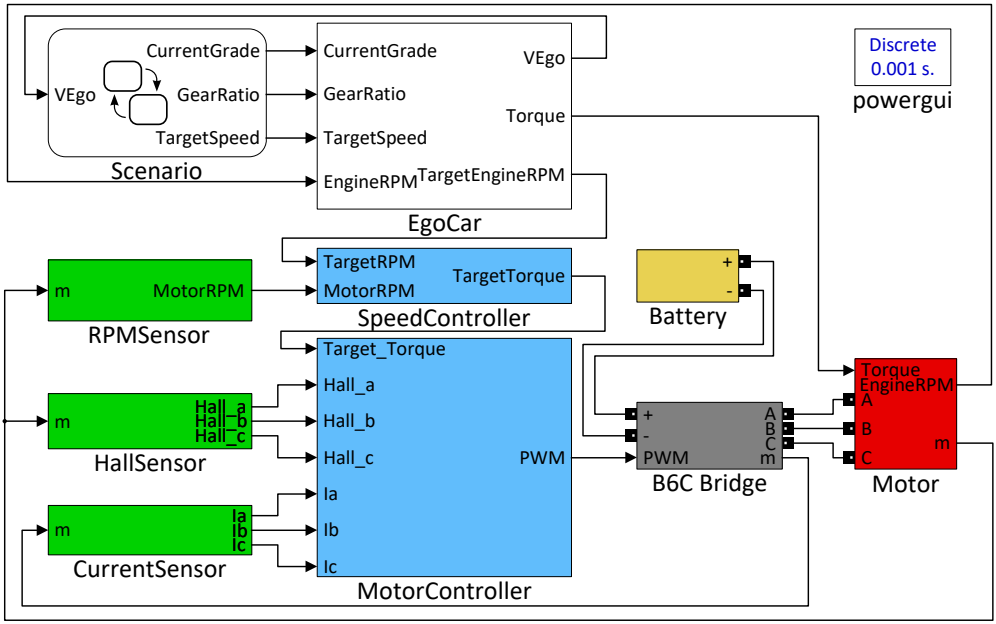



Abbildung 7.21.: Angepasstes und erweitertes Modell des Simulink-AC7-Brushless-DC-Motor-Drive aus [260].

Zu den ergänzten Komponenten zählen eine Batterie (**Battery**), ein Szenario (**Scenario**), das EF (**EgoCar**), ein Stromsensor (**CurrentSensor**) und ein Hall-Sensor (**HallSensor**).

Die Komponente **Scenario** implementiert das Fahrszenario als Stateflow-Chart. Den Aufbau zeigt Abbildung 7.22. Das Szenario aus Abbildung 7.19 wurde dabei wiederverwendet und angepasst. Das Fahrzeug soll demnach zuerst auf 50 km/h beschleunigen. Sobald es diese Geschwindigkeit erreicht hat, soll es sie für zehn Sekunden halten und danach auf 30 km/h einregeln. Zehn Sekunden danach soll das Fahrzeug auf 0 km/h abbremsen. Als Stimuli werden die Zielgeschwindigkeit **TargetSpeed**, die momentane Steigung **CurrentGrade** sowie das Übersetzungsverhältnis **GearRatio** ausgegeben.

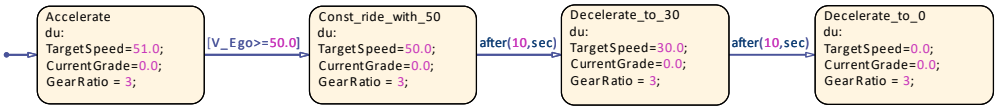



Abbildung 7.22.: Zustände des Szenarios zum Test des BEFs.

Das EF-Konzept ist in Unterabschnitt 6.2.3 beschrieben. In der Komponente **EgoCar** werden die dort aufgeführten Gleichungen der Fahrwiderstände als Matlab-Funktionen implementiert. Die vom Szenario (**Scenario**) vorgegebene Zielgeschwindigkeit (**TargetSpeed**) bestimmt zunächst die Zieldrehzahl des Motors (**TargetEngineRPM**) für die Regelung in der Komponente **SpeedController**. Dazu wird das Übersetzungsverhältnis (**GearRatio**) verwendet. Des Weiteren wird der Komponente **Motor** das Lastmoment (**Torque**) zur Verfügung gestellt. Ferner wird die Geschwindigkeit **VEgo** über die Integration der Beschleunigung des EFs bestimmt. Dabei wird ein konstanter Wert von $+4\frac{m}{s^2}$ für die Beschleunigung und $-6\frac{m}{s^2}$ für die Verzögerung angenommen. Die Werte werden als Parameter angegeben und sind variabel.
Alternativ könnte der Wert für VEgo aus dem Istwert der Motordrehzahl **EngineRPM** bestimmt werden. Allerdings wäre die Simulation dadurch instabiler, weil sich der Fehler in der Regelung durch die geschlossene Schleife verstärken kann.

Die Komponente **SpeedController** ist Teil der *Software*. Die ursprüngliche Implementierung aus Abbildung 7.20 (*Speed Controller*) wird mit einem *Proportional-Integral-Regler (PI-Regler)* ersetzt. Als Eingabe dienen der Istwert (**MotorRPM**) und der Sollwert (**TargetRPM**) der Motordrehzahl. Ausgegeben wird ein Soll-Moment (**TargetTorque**), das an die Komponente **MotorController** weitergegeben wird. Dabei ist eine Sättigung als Parameter vorgegeben, die dem maximalen Drehmoment des Motors entspricht.

Die Komponente **MotorController** enthält die Komponente *Current Controller* aus Abbildung 7.20 und ist ebenfalls Teil der Software. Sie generiert in Abhängigkeit des Soll-Drehmoments (**TargetRPM**), der Wicklungsströme ($\mathbf{I}_i$) und der Hall-Signale (**Hall_**$_i$) die passenden **PWM**-Signale für die Komponente **B6C Bridge**[1]. Die Hall-Signale werden zur Bestimmung des Istwertes vom Rotorwinkel des Motors genutzt. Zur Auflösung der **m**-Bussignale nutzen alle Sensoren das in Abbildung 6.18 dargestellte Prinzip.

Die Brückenschaltung **B6C Bridge** entspricht dem Modul *B6C-Brücke* aus Abbildung 7.20 ohne Gleichrichter (*Three-phase diode rectifier*). Dieser wird nicht benötigt, da es sich bei der verwendeten Spannungsquelle **Battery** um eine Gleichspannungsquelle handelt. Aufgrund der hohen Ströme werden Modelle von Bipolartransistoren mit isolierter Gate-Elektrode (*Insulated-Gate Bipolar Transistors (IGBTs)*) verwendet und von den **PWM**-Signalen angesteuert. Dies wird über einen Parameter des Modells festgelegt.

Die Batterie (**Battery**) stammt aus der SESPS-Bibliothek und wird als Li-Ion Typ mit 400 V Spannung sowie einer Kapazität von 94 Ah über Parameter konfiguriert.

Als **Motor** wird die Komponente *Permanent Magnet Synchronous Machine* (siehe Abbildung 7.20) verwendet und mit den Daten eines 100 kW-*Permanentmagnet-Synchronmotors (PMSM)* aus [214] konfiguriert. Die Parameter sind in Tabelle 7.6 aufgelistet.

Tabelle 7.6.: Daten des in dieser Arbeit verwendeten Permanentmagnet-Synchronmotors (Quelle: [214]).

| | | | |
|---|---|---|---|
| Spannung / Strom | $500\,V / 200\,A$ | Anzahl der Polpaare | 12 |
| Drehmoment ⌀ | $178\,Nm$ | Induktivität | $0,19\,mH$ |
| Trägheitsmoment | $0,198\,Kgm^2$ | Drehmoment Konstante | $0,627\,Nm/A$ |

[1] *Bridge-6-Controlled (B6C)* - Sechspuls-Brücken-Schaltung.

**Anpassungen, Optimierungen und Limitierungen** Da das Modell aus Abbildung 7.21 zur Kalibrierung der Templates und als Referenz für anschließend damit generierte Modelle dient, wurde die Ausführungszeit optimiert. Dazu wurden kontinuierliche Komponenten durch diskrete ersetzt, sofern es das zugrundeliegende Template erlaubte. Die zeitliche Auflösung der diskreten SESPS-Simulation (powergui) wurde außerdem von $10^{-6}$ s auf $10^{-3}$ s herabgesetzt. Dieser Wert ermöglicht z.B. noch die Einhaltung typischer Zykluszeiten von CAN-Bussystemen. Allerdings muss der Geschwindigkeitsregler die dadurch entstandene Totzeit durch höhere Beschleunigungen in Form von höheren Werten für das Soll-Moment ausgleichen. Das hat wiederum höhere Motorströme zur Folge.

Es ist außerdem zu beachten, dass bei einer diskreten Ausführung des Motormodells zusätzliche Widerstände in Serie an den Versorgungseingängen des Motors zu modellieren sind, wie in [271] unter „Limitations and Assumptions“ beschrieben. Bei einer Sample-Zeit von $t_s = 10^{-3}$ s, einer Batteriespannung U von 400 V und einem maximalen Strom $I_{max}$ von 200 A, berechnet sich der Wert zu:

$$R = \frac{U^2}{P} = \frac{U^2}{U * I * (t_s * 0,1 \frac{\%}{\mu s})} = \frac{400\,V^2}{(400\,V * 200\,A * (1000 \mu s * 0,1 \frac{\%}{\mu s}))} = 2\,\Omega \tag{7.16}$$

Der Widerstandswert erhöht sich für $t_s = 10^{-4}$ s entsprechend auf 20 Ω. Bei einigen hybrid implementierten Templates mit variabler Schrittweite wurde jedoch festgestellt, dass die so berechneten Werte zu groß sind. Die besten Ergebnisse konnten mit 1 Ω bei Modellen ohne ECU und 2 Ω bei Modellen mit ECU (siehe Unterabschnitt 7.3.2) erzielt werden. Dies gilt sowohl für eine Zeitschrittweite von $10^{-3}$ s als auch für eine von $10^{-4}$ s.

### Aufbau eines PREEvision-Modells

Das Modell aus Abbildung 7.21 entspricht schon weitestgehend der Zielabbildung. Allerdings fehlen noch Leitungssatzelemente, Übersetzungen von logische in physikalische Signale und eine ECU auf der die SWCs ausgeführt werden können. Daher wird ein PREEvision-Modell entworfen, das diese Informationen enthält. Die Templates werden dann mit diesem Modell verknüpft. Im Anschluss wird daraus das finale BEF-Simulationsmodell generiert.

Die relevanten Ebenen des so entstandenen und für [P3] verwendeten PREEvision-Modells und ihre Zusammenhänge bzw. Mappings (violette Verbindungen) zeigt Abbildung 7.23. Die oberste Ebene zeigt nach [P3] die **LA** des Modells. Sie ist aus Abbildung 7.21 abgeleitet und enthält die gleichen Komponenten. Es wird jedoch hier zwischen Umgebungssignalen und logischen Kommunikationssignalen unterschieden. Logische Aktor- und Sensorfunktionen (z.B. **Motor** oder **TargetRPM**) bilden dabei die Wandler zwischen diesen Signalarten. Somit werden Umgebungsanschlüsse an logischen Funktionen (z.B. **SpeedController**) vermieden, die bei einer Ausführung als SWC auf einer ECU nicht existieren dürfen. Eine Ausnahme bildet die Komponente **Bridge**, da sie in Hardware implementiert wird. [P3]

Dem Prototyp des Blocks **Scenario** wird nach [P3] ein PREEvision-Zustandsautomat hinzugefügt. Dessen Aufbau ist analog zu Abbildung 7.22 realisiert. Die Komponente **EgoCar** wird über eine TALIA-Repräsentation mit dem Template *EgoCar* aus Abbildung 7.21 verknüpft. [P3]

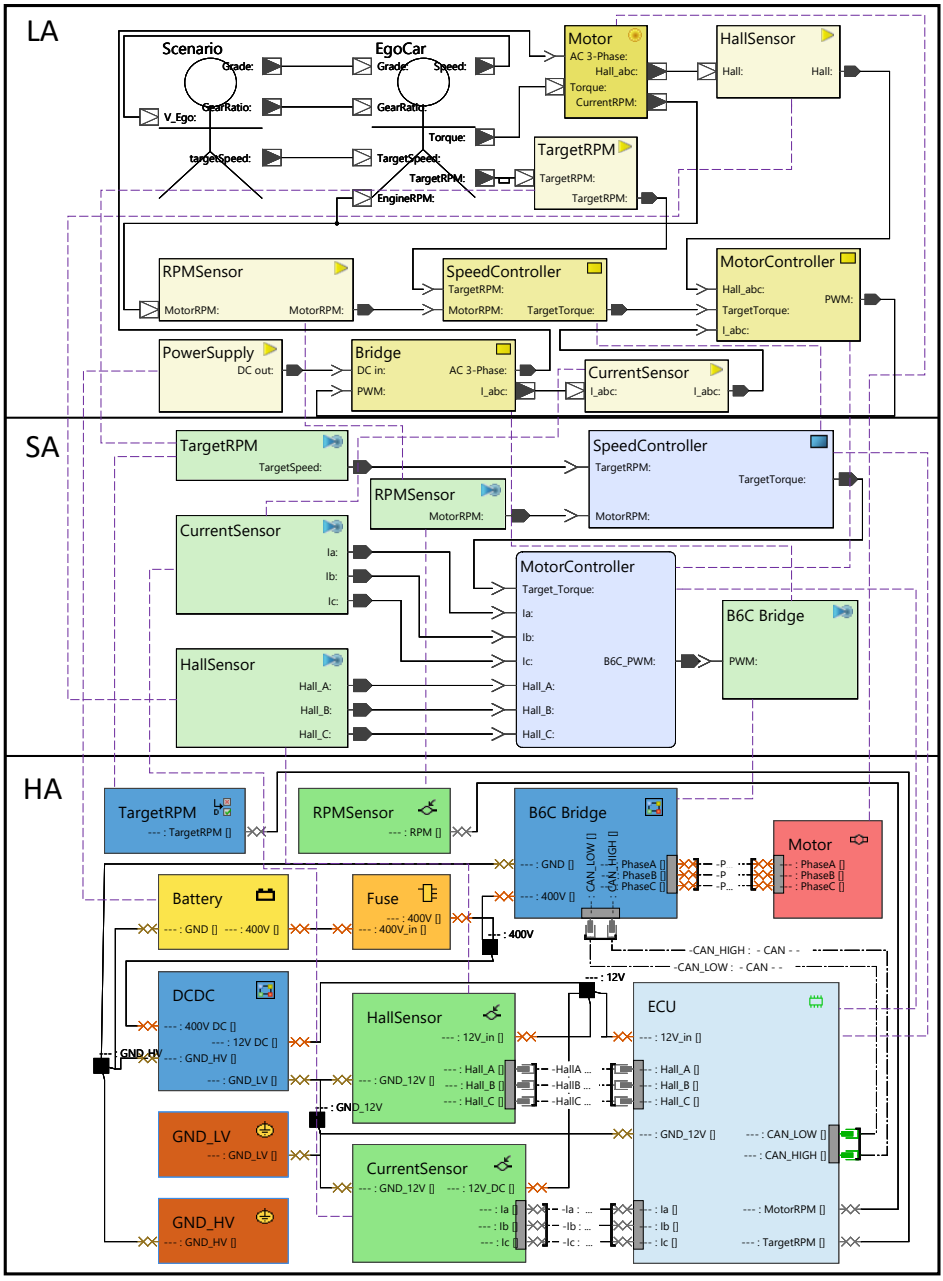



Abbildung 7.23.: PREEvision-Ebenen und Artefakte zur Simulation eines BEFs (Quelle: Modifikation von [P3]).

Um ein Signalrouting zu ermöglichen, werden nach [P3] alle Sensoren, Aktoren und Funktionen aus der LA sowie die zugehörigen logischen Signale auch auf der **SA** modelliert. Ausgenommen sind dabei Komponenten und Signale, die zur Stromversorgung verwendet werden. Für Signale, die auf der LA mehrere Datenelemente zusammenfassen, wird auf der SA zudem für jedes Datenelement ein eigenes Signal modelliert, wenn es auf der HA als konventionelle Punkt-zu-Punkt-Verbindung (Draht) realisiert wird. Die SWCs sind über TALIA-Repräsentationen mit den namentlich entsprechenden Templates aus Abbildung 7.21 (*SpeedController* und *MotorController*) verknüpft. [P3]

Auf der **HA** befinden sich nach [P3] schließlich die Modellvertreter der physikalischen Komponenten. Um die Ausführung der SWCs zu ermöglichen, wurde eine **ECU** modelliert. Ihr sind die SWCs über Mappings zugewiesen. Der ECU ist außerdem das generische Template aus Unterabschnitt 6.3.2 fest zugewiesen. Für die automatische Konfiguration des Templates werden die Informationen aus dem PREEvision-Modell genutzt. [P3]

Des Weiteren wurde für [P3] die Stromversorgung im Detail ergänzt. Neben der Batterie (**Battery**), der standardmäßig das Modell *Battery* aus der SESPS-Bibliothek als Template zugewiesen ist, wurden nach [P3] Massestellen (**GND**) und ein Sicherungskasten (**Fuse**) ergänzt. Ferner wurden Leitungen, Spleiße und Stecker modelliert. Auch diese Komponenten sind fest mit entsprechenden Templates verknüpft, sodass keine Verhaltensmodellierung dafür nötig ist. Den übrigen Komponenten werden Templates über die TALIA zugewiesen. Diese sind in Abbildung 7.21 anhand ihres Namen zu finden. Eine Ausnahme bildet jedoch der Gleichstromwandler (**DCDC**[1]). Er ist nicht in Abbildung 7.21 zu finden. Als Template wird für ihn die Komponente *Buck Converter* aus der SESPS-Bibliothek verwendet. Der Gleichstromwandler ist nötig, um aus der Batteriespannung von 400 V eine Spannung von 12 V für die übrigen Verbraucher im Niedervoltbereich zu generieren. Auf eine zweite Batterie wurde bei der Modellierung verzichtet. [P3]

### 7.3.2. Simulation eines batterieelektrischen Fahrzeugs mit ECU

Zunächst soll das Konzept zur Integration der SWCs innerhalb des ECU-Modells evaluiert werden. Aus dem in Abbildung 7.23 dargestellten PREEvision-Modell wird dafür das Simulink-Modell mittels PM generiert und anschließend manuell vereinfacht. Das resultierende Modell ist in Abbildung 7.24 dargestellt. Die Signalwandlungen in den Sensoren (grün dargestellt) und die physikalischen Leitungen an den Ausgängen der Sensoren wurden entfernt, um mögliche Seiteneffekte zu reduzieren. Die Parameter der verwendeten Templates werden aus dem Vergleichsmodell (siehe Abbildung 7.21) übernommen. Das entstandene hybride Modell weist Anteile von Stateflow, Simscape und SimEvents auf. Aufgrund unterschiedlicher Zeitausführungsmodelle, deren Auswirkung sich durch Hervorheben der Sample-Zeiten auf den Signalen sichtbar machen lässt, muss zunächst eine manuelle Anpassung in der übergeordneten Haupt-MAE gemacht werden. Dabei wird in den Solver-Einstellungen die Option „Automatically handle rate transition for data transfer“ ausgewählt, um das SimEvents-Zeitmodell mit der restlichen Simulation zu synchronisieren.

[1] In Simulink darf der Name kein „/“ enthalten. Das Zeichen ist für Pfade reserviert.

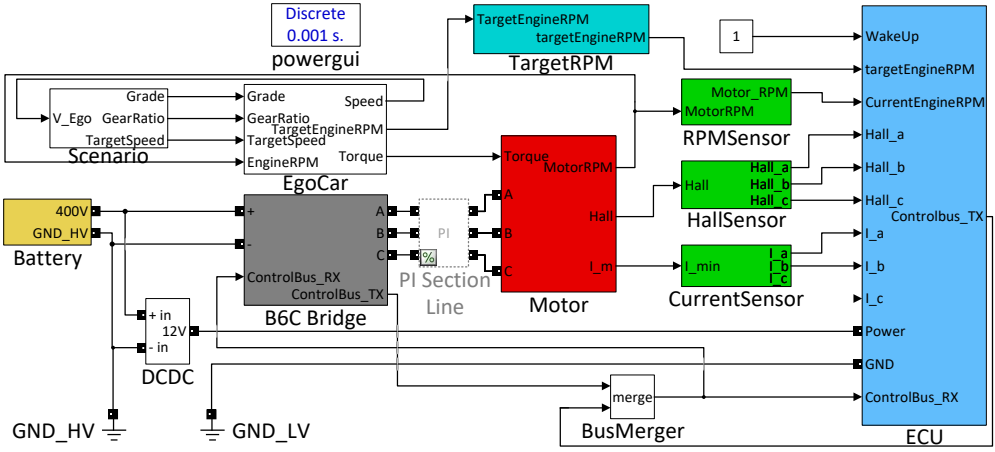



Abbildung 7.24.: Vereinfachtes, generiertes BEF-Simulink-Modell.

Abbildung 7.25 zeigt ferner die FCS der beiden SWCs **MotorController** und **SpeedController**, welche gemeinsam auf einem Kern der ECU ausgeführt werden.

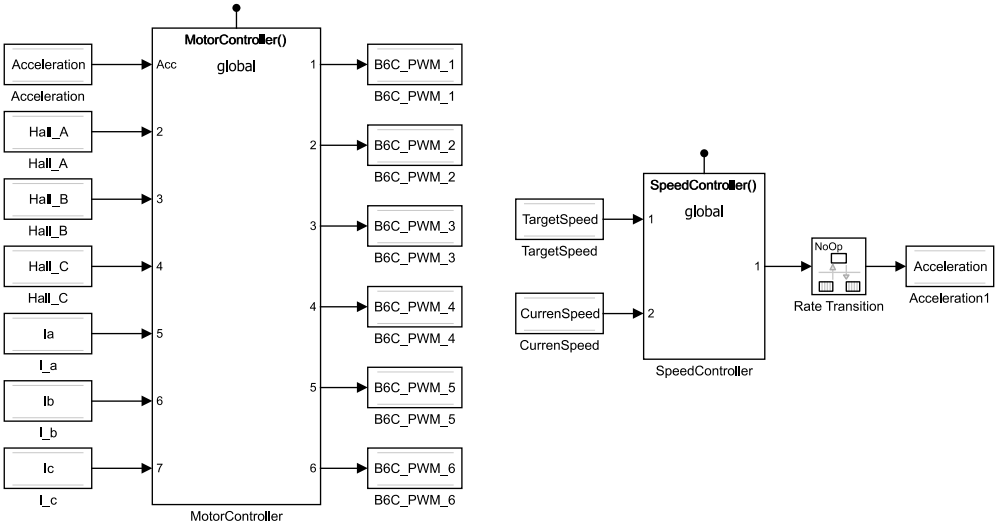



Abbildung 7.25.: Integration der SWCs in der ECU des generierten BEF-Modells.

Es gilt zu beachten, dass Daten aus einem gemeinsamen Bus- oder Multiplexer-Signal nicht zusammen in einem einzigen DSM-Block gespeichert werden können. Daher wird bei dem Bussignal PWM für jedes enthaltene Datum **B6C_PWM_i** ein eigener DSM-Block benötigt. Weitere

Einschränkungen sind, dass in einem FCS nur Sample-Zeiten verwendet werden können, die von einer höheren Modellebene geerbt werden. Außerdem können Blöcke vom Typ **Rate Transition**, die unterschiedliche Sample-Zeiten synchronisieren und in Templates genutzt werden, nicht innerhalb von FCS verwendet werden. Sie müssen daher nach außen, d.h. aus dem FCS heraus, verlagert werden. Darauf muss beim Erstellen der Templates geachtet werden.

### Evaluation des Schedulings

Für das Scheduling werden beide SWCs aus Abbildung 7.25 zyklisch und mit einer Periode von $10^{-3}$ s ausgeführt. Dies entspricht gerade der Sample-Zeit der Stromsimulation aus dem Referenzmodell (siehe powergui in Abbildung 7.21). Die SWC **SpeedController** bekommt eine doppelt so hohe Priorität als die SWC **MotorController** zugewiesen, um die Ausführungsreihenfolge aus dem Datenfluss zunächst beizubehalten. Damit entspricht die Konfiguration [P3].

Im Folgenden werden die Auswirkungen des Schedulings und der Schrittweite der Stromsimulation auf die Laufzeit und die Genauigkeit der Simulation untersucht. Alle Simulationen werden auf System B (siehe Tabelle 2.4) ausgeführt. Dabei werden das erweiterte Referenzmodell aus Abbildung 7.21 und das generierte Modell aus Abbildung 7.24 (mit ECU und darauf ausgeführten SWCs) bei unterschiedlichen Schrittweiten der Stromsimulation verglichen. Außerdem wird analog zu [P3] die Auswirkung von zusätzlich auf dem gleichen Prozessor ausgeführten Pseudo-SWCs untersucht. Sie sollen eine höhere Auslastung simulieren. Im Gegensatz zu den immer ausgeführten Standard-SWCs SpeedController und MotorController, bekommen die Pseudo-SWCs eine Periode von $10^{-2}$ s zugewiesen. Außerdem erhalten sie eine halb so hohe Priorität als die Komponente MotorController. Dies entspricht ebenfalls der Konfiguration aus [P3]. In den folgenden Grafiken und Tabellen bezieht sich die angegebene Anzahl an SWCs immer auf die Gesamtzahl aus Standard- und Pseudo-SWCs.

Tabelle 7.7 zeigt zum Überblick die Ergebnisse einer Ausführung des in Abbildung 7.22 abgebildeten Szenarios in Abhängigkeit des Modells (mit oder ohne ECU), der Anzahl an SWCs sowie der Schrittweite der Stromsimulation.

Tabelle 7.7.: Vergleich von Laufzeiten, Regelgüte und Endbatterieladezustände bei der Ausführung des in Abbildung 7.22 beschriebenen Szenarios mit unterschiedlichen Modell- und Ausführungskonfigurationen.

| Anzahl der SWCs | Ohne ECU | | 2 SWC | 2 SWC | 10 SWC | 100 SWC | 1000 SWC |
|---|---|---|---|---|---|---|---|
| Schrittweite [s] | 1e-3 | 1e-4 | 1e-3 | 1e-4 | 1e-4 | 1e-4 | 1e-4 |
| Laufzeit (normal) [s] | 3,713 | 26,259 | 8,275 | 33,591 | 49,321 | 207,648 | 4636,466 |
| Laufzeit (accelerator) [s] | 1,440 | 5,684 | 8,414 | 9,316 | 17,226 | 111,912 | 3398,295 |
| Speedup (normal->accelerator) | 2,578 | 4,620 | 0,983 | 3,606 | 2,863 | 1,855 | 1,364 |
| $S_t^*(normal)$ | 0,124 | 0,875 | 0,276 | 1,120 | 1,644 | 6,922 | 154,549 |
| $S_t^*(accelerator)$ | 0,048 | 0,189 | 0,280 | 0,311 | 0,574 | 3,730 | 113,277 |
| Mittelwert (5-15s) [km/h] | 49,977 | 50,071 | 27,259 | 49,419 | 49,623 | 47,614 | -0,002 |
| Standardabweichung (5-15s) | 0,832 | 0,197 | 0,107 | 0,476 | 0,374 | 0,230 | 1,051 |
| Endbatterieladezustand [%] | 97,970 | 99,145 | 98,828 | 98,864 | 98,866 | 98,864 | 98,757 |

Zur Durchführung des gesamten Szenarios wird eine logische Zeit von 30 s benötigt. Die Ergebnisse umfassen die bei den Ausführungen ermittelten Laufzeiten und die reziproken zeitabhängigen Scaleups $S_t^*$ für unterschiedliche Ausführungsmodi, die Mittelwerte und Standardabweichungen für den Zustand „Const_ride“ sowie die jeweiligen Endbatterieladezustände. Im Ausführungsmodus **normal** wird das Modell interpretiert ausgeführt, während im Modus **accelerator** C-Code generiert, optimiert und dann ausgeführt wird. In dem Zustand „Const_ride“ (**5-15 s**) bewegt sich das EF mit einer konstanten Geschwindigkeit von 50 km/h.

Die in Tabelle 7.7 aufgelisteten Ergebnisse werden in den folgenden Unterabschnitten im Hinblick auf unterschiedliche Schwerpunkte aufbereitet und erläutert.

**Untersuchung der Laufzeiten** Abbildung 7.26 zeigt die grafische Aufbereitung der in Tabelle 7.7 aufgelisteten Daten zur Laufzeit. Mit zunehmender Anzahl an SWCs und geringer werdender Schrittweite steigt die Laufzeit. Das generierte Modell mit ECU und zwei SWCs ist zudem bei beiden Schrittweiten langsamer als das Referenzmodell ohne ECU. Dabei wird in beiden Modellen die gleiche Logik für die Komponenten SpeedController und MotorController ausgeführt. Darüber hinaus lässt sich erkennen, dass die Ausführung im Modus **accelerator** einen durchschnittlichen Speedup von ca. 2,55 gegenüber dem Modus **normal** erzielt. Beim Referenzmodell ohne ECU liegt der Speedup bei einer Schrittweite von 1e-4 s mit ca. 4,62 deutlich über diesem durchschnittlichen Speedup. Auffällig ist auch der Speedup von 0,983 bei 2 SWCs und einer Schrittweite von 1e-3 s. Hierbei tritt der zeitliche Mehraufwand für die Code-Kompilierung im Vergleich zur eigentlichen Ausführung ins Gewicht.

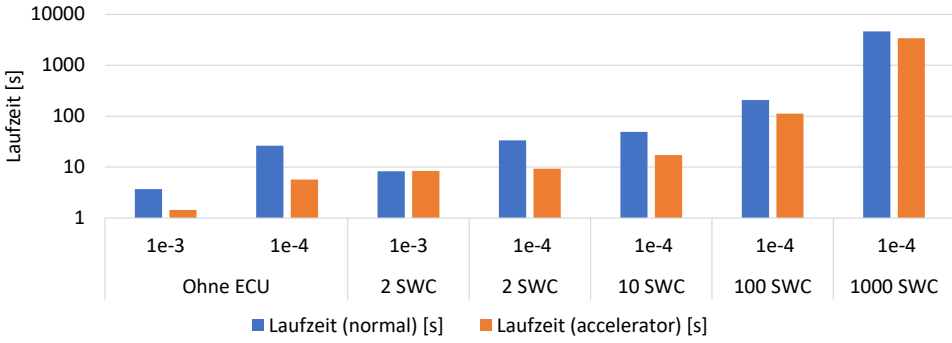



Abbildung 7.26.: Laufzeitvergleich bei der Ausführung des in Abbildung 7.22 beschriebenen Szenarios mit unterschiedlichen Modell- und Ausführungskonfigurationen.

Vergleicht man die Laufzeiten im normalen Modus zwischen dem Referenzmodell und dem generierten Modell mit 2 SWCs, so ist die Ausführung des Referenzmodells bei der Schrittweite 1e-3 s etwa 2,23 Mal schneller. Bei einer Schrittweite von 1e-4 s fällt der Unterschied mit einer 1,28-fach schnelleren Ausführung etwas geringer aus. Zur Analyse, weshalb das generierte Modell langsamer ist, wurden für beide Modelle die prozentual an der Laufzeit beteiligten Komponenten mithilfe des Simulink Profiler ermittelt. Das Ergebnis zeigt Abbildung 7.27.

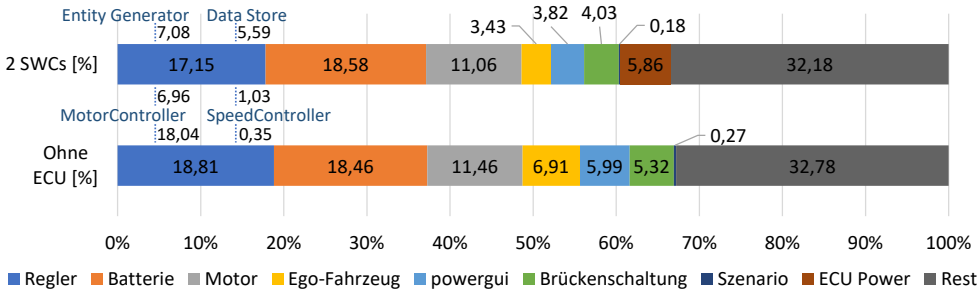



Abbildung 7.27.: Prozentuale Anteile der Simulink-Komponenten an der Laufzeit bei einer Schrittweite von 1e-4 s für das Referenzmodell (ohne ECU) und das generierte Modell mit ECU und 2 SWCs.

Es fällt auf, dass der prozentuale Laufzeitanteil der an der Regelung beteiligten Komponenten (**Regler**) im generierten Modell leicht abnimmt, obwohl dort zusätzlich *Entity Generator*- und DSM-Blöcke innerhalb der ECU-Komponente ausgeführt werden. Komponenten zur Modellierung der Leistungsaufnahme der ECU (**ECU Power**) fallen bei der Laufzeitzunahme am stärksten ins Gewicht. Setzt man die Zunahme der Laufzeit von ca. 28 % ins Verhältnis zu der um 38 % gewachsenen Modelldatei, so liegt sie aber noch im Rahmen. Außerdem wird mit dem Scheduling und der Leistungsabstraktion ein zentraler Nutzen generiert. Allerdings ist zu beachten, dass die Laufzeit mit der Anzahl an zusätzlichen Pseudo-SWCs weiter ansteigt. Im Modus *accelerator* ist die Skalierbarkeitsgrenze ($S_t^* = 1$) bei 22 SWCs erreicht, wenn ein linearer Verlauf angenommen wird.

Abbildung 7.28 zeigt zusätzlich die Generierungszeiten nach [P3].

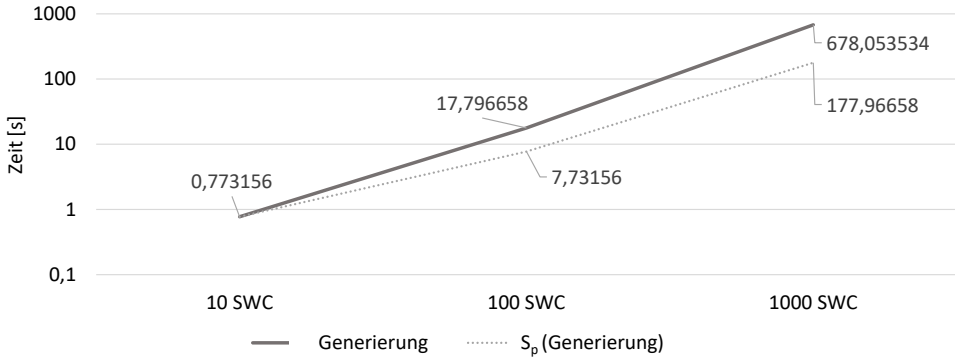



Abbildung 7.28.: Reale und mit $S_p = 1$ genäherte Generierungszeiten in Abhängigkeit der Anzahl der SWCs (Quelle: Modifikation von [P3]).

Die Kurve **Generierung** zeigt die realen Generierungszeiten. Die Kurve (**$S_p$(Generierung)**) zeigt die mit $S_p = 1$ linear genäherten Generierungszeiten. Beide Kurve sind über der Anzahl an generierten SWCs aufgetragen. Da die Kurve *Generierung* immer über der Kurve *$S_p$(Generierung)* liegt, skalieren die Generierungszeiten in keinem Fall [P3].

**Untersuchung der Genauigkeit** Abbildung 7.29 zeigt ergänzend zu Tabelle 7.7 und [P3] die Geschwindigkeitsverläufe des BEFs bei der Ausführung des in Abbildung 7.22 beschriebenen

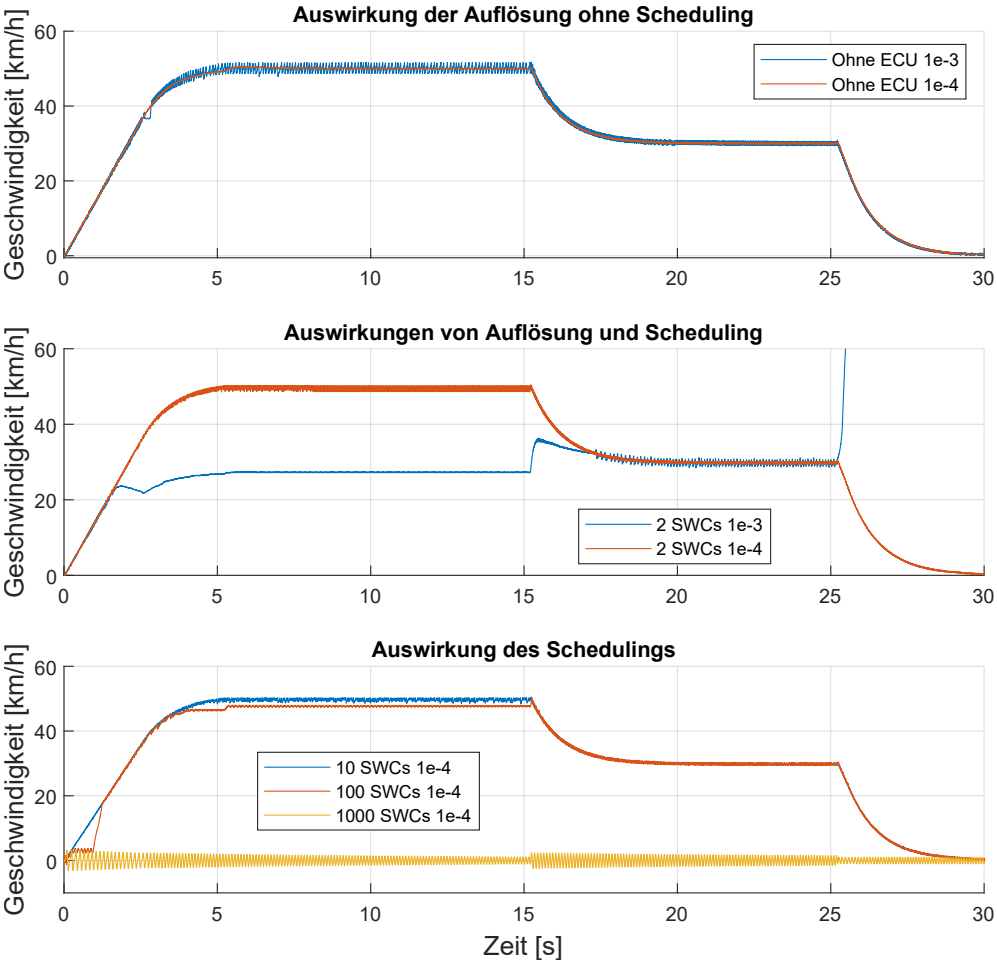



Abbildung 7.29.: Geschwindigkeitsverläufe des BEFs bei der Ausführung des in Abbildung 7.22 beschriebenen Szenarios zu unterschiedlichen Modell- und Ausführungskonfigurationen.

Szenarios und in Abhängigkeit verschiedener Modellkonfigurationen. Damit ist eine isolierte Betrachtung der Auswirkungen des Schedulings und der Auflösung (Schrittweite der Stromsimulation) auf die Genauigkeit möglich. Die Teildiagramme aus Abbildung 7.29 werden im Folgenden jeweils in eigenen Abschnitten beschrieben.

**Isolierte Betrachtung der Auflösung ohne Scheduling** Das obere Teildiagramm in Abbildung 7.29 visualisiert den Unterschied bei der Wahl der zeitlichen Auflösung der Stromsimulation, unabhängig von einem Scheduling. Für die Ausführung wird das in Abbildung 7.21 dargestellte Referenzmodell verwendet. Bei einer Schrittweite der SESPS-Stromsimulation von 1e-4 s steigt die Laufzeit gegenüber einer Schrittweite von 1e-3 s um das Siebenfache an. Allerdings ist ein deutlich vermindertes Rauschen auf dem Geschwindigkeitsverlauf zu erkennen. Konkret folgt im Zeitbereich von 5 s bis 15 s mit Tabelle 7.7 eine um den Faktor 4,2 niedrigere Standardabweichung. Das geringere Rauschen wirkt sich wiederum auf die Genauigkeit des Batterieladezustands aus, welcher nach 30 s um 1,175 % abweicht. Dies entspricht einer Steigerung der Genauigkeit um 11,85 %. Aufgrund des geringeren Rauschens fließen zudem niedrigere Ströme durch den Motor. Sie hängen von dem Soll-Moment ab, das durch den PI-Regler in der KomponenteSpeedController, in Abhängigkeit der Motordrehzahl, bestimmt wird (vgl. Target_Torque in Abbildung 7.21). Durch eine geringere Auflösung entsteht eine höhere Totzeit zwischen zwei aufeinanderfolgenden Drehzahlwerten des Motors. Dies führt zu höheren Werten für die Soll-Momente, woraus höhere Ströme folgen. Tabelle 7.8 zeigt hierzu die Mittelwerte und Standardabweichungen der positiven Motorstromphasenwerte. Hiernach ist der Mittelwert bei einer Schrittweite von 1e-3 s im Durchschnitt mehr als doppelt so hoch, als bei einer Schrittweite von 1e-4 s. Das gilt für alle drei Phasen.

Tabelle 7.8.: Vergleich der Mittelwerte und Standardabweichungen der Motorstromphasenwerte für das Referenzmodell ohne ECU bei 1e-3 s und 1e-4 s Schrittweite

| Schrittweite | Phase | Mittelwert (+) [A] | Standardabweichung | Maximalwert [A] |
|---|---|---|---|---|
| 1e-3 | A | 147,8782 | 62,7891 | 284,2524 |
| | B | 147,9006 | 64,2536 | 288,5749 |
| | C | 149,8609 | 62,9353 | 289,8562 |
| 1e-4 | A | 70,6365 | 49,9208 | 274,3116 |
| | B | 70,9432 | 48,9432 | 280,6309 |
| | C | 70,4839 | 48,8047 | 279,0925 |

Abbildung 7.30 zeigt ergänzend dazu die Verteilung der Motorstromwerte für die Phase A bei den Schrittweiten 1e-3 s und 1e-4 s. Da die Motorströme bei den Phasen B und C ähnlich verlaufen, dient Abbildung 7.30 stellvertretend auch für diese. Es ist zu erkennen, dass bei einer Zeitschrittweite von 1e-3 s jeweils zwei Verteilungsbereiche an den Rändern auftreten, da die Regelung die größeren zeitlichen Lücken ausgleichen muss und damit härter reagiert. Bei einer Zeitschrittweite von 1e-4 s ist dagegen ein deutliches Maximum in der Mitte zu erkennen. Höhere Ströme kommen seltener vor. Dies erklärt den geringeren durchschnittlichen Strom und damit den höheren Batterieladezustand am Ende, der auf die gewählte Schrittweite zurückzuführen ist.

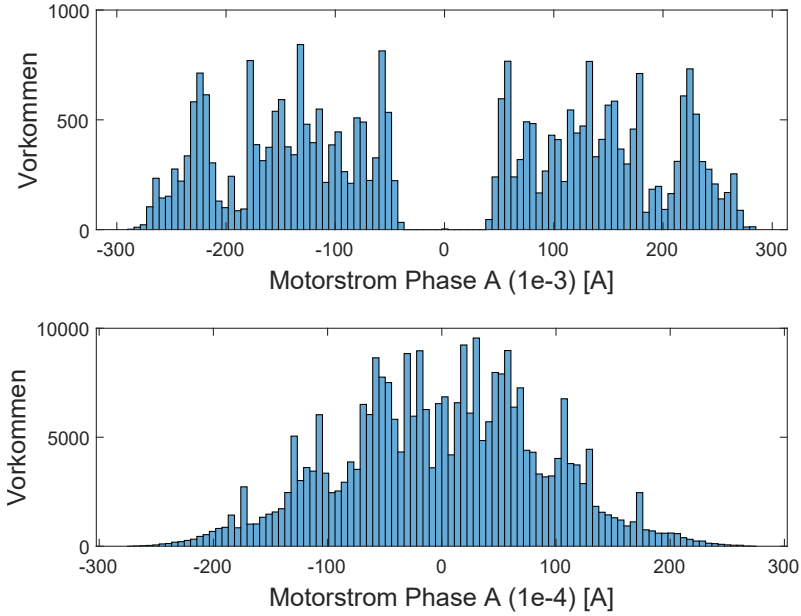



Abbildung 7.30.: Verteilung der Motorstromwerte der Phase A nach Vorkommen für das Referenzmodell ohne ECU bei 1e-3 s und 1e-4 s Schrittweite.

**Betrachtung von Auflösung und Scheduling** Das mittlere Diagramm in Abbildung 7.29 zeigt sowohl die Auswirkungen des Schedulings als auch der zeitlichen Auflösung auf die Regelung der Geschwindigkeit. Zunächst fällt hierbei auf, dass eine Regelung bei einer Schrittweite von 1e-3 s scheitert, sobald das EF ca. 25 km/h erreicht. Zur Analyse wurde ein Testmodell aufgebaut, um die Signale des Referenzmodells mit denen des generierten Modells (2 SWCs) zu vergleichen. Dabei wurde festgestellt, dass sowohl die Hall-Signale als auch die gemessenen Motorströme am Eingang der SWC MotorController um die innerhalb des Schedulings zugewiesenen Perioden verzögert werden. Auch die PWM-Signale am Ausgang der Komponente MotorController sind dabei um die entsprechende Periode verzögert, wie Abbildung 7.31 stellvertretend für ein beliebiges PWM-Signal zeigt. **0** entspricht dem Zustand **Aus**, während **1** dem Zustand **An** entspricht. Durch die Verzögerung kommt es innerhalb eines Zeitschritts in der Stromsimulation zu falschen Werten bei der Ansteuerung der IGBTs in der B6C-Brücke. Bei einer Zeitschrittweite von 1e-3 s kann der PI-Regler dies jedoch nicht ausgleichen. Zur Lösung des Problems kann die Schrittweite der Stromsimulation verkürzt werden, wie im mittleren Diagramm in Abbildung 7.29 bei einer Schrittweite von 1e-4 s zu sehen ist. Dadurch ist dann zwar eine Regelung möglich, aber die Auswirkungen des Schedulings bleiben als Rauschen sichtbar. Dies kann wiederum durch eine Anpassung des Reglers oder eine Tiefpassfilterung kompensiert werden. Alternativ könnte auch die Periode des Schedulings reduziert werden, sodass die entstandene Verzögerung geringer wird. Ferner könnte in der Praxis ein Motor mit einer geringeren Polzahl verwendet werden, da die maximale Drehfrequenz des Motors

$f_{Motor_{max}}$ von der für die Ansteuerung verwendeten maximalen PWM-Frequenz $f_{PWM_{max}}$ und der Polzahl N abhängt:

$$f_{Motor_{max}} = \frac{f_{PWM_{max}}}{N} \tag{7.17}$$

$f_{PWM_{max}}$ ist wiederum durch die Schrittweite der Simulation limitiert.

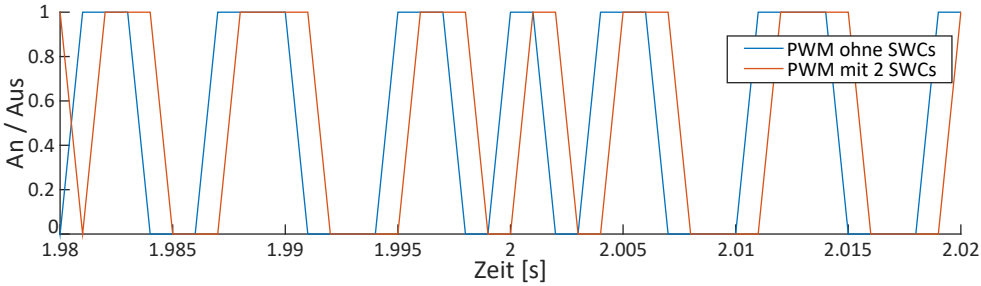



Abbildung 7.31.: Durch das Scheduling verzögertes PWM-Signal beim generierten Modell mit 2 SWCs und einer Schrittweite von 1e-3 s.

**Isolierte Betrachtung des Schedulings** Werden noch weitere Pseudo-SWCs auf der ECU ausgeführt, so verzögern sich die PWM-Signale durch das Scheduling noch stärker. Anhand des unteren Diagramms in Abbildung 7.29 sind die Auswirkungen davon erkennbar: Bei 998 zusätzlichen Pseudo-SWCs ist mit den gewählten Perioden keine Regelung mehr möglich.

### Fazit

Die Laufzeit- und Genauigkeitsanalysen haben die Grenzen der E/E-Architektursimulation am Beispiel des BEFs und in Bezug auf die Skalierbarkeit aufgezeigt. Im Vergleich zu Ptolemy II (siehe Unterabschnitt 7.2.3) stehen in Simulink physikalisch modellierte Stromleitungen bzw. Ströme zur Verfügung, die bei der Fehleranalyse unterstützt und zum Verständnis beigetragen haben (siehe Abbildung 7.30 und Abbildung 7.31). Außerdem ist die Simulation in Simulink bis zu 120,5 Mal performanter, trotz aufwendigerer Abbildung und niedrigerer Abstraktion. Dennoch kommt die Implementierung bereits bei 22 ausgeführten SWCs an ihre Grenzen. Darüber hinaus haben der Einsatz von Templates und die Simulationsmodellgenerierung mit PREEvision den Modellierungsprozess vereinfacht. Allerdings wurde auch gezeigt, dass der Nutzen evaluierter Templates aufgrund der auftretenden Emergenz, welche von der Zusammenstellung der Templates abhängt, begrenzt ist. Die Fehlerfindung gestaltet sich dann als aufwendig und erfordert ein tiefes Verständnis der Simulation, besonders wenn Ursache und Wirkung durch Rückkopplungsschleifen verschleiert sind. Zudem beeinträchtigen die durch die Parametrisierung entstandenen Freiheitsgrade die Lösungsfindung. Ferner erschwert die nötige Synchronisation und Anpassung unterschiedlicher Schrittweiten zwischen den Teilmodellen des hybriden Gesamtmodells die Modellierung. Die Synchronisation kann nicht innerhalb der einzelnen Templates erfolgen und erfordert den Eingriff des Modellierers.

### 7.3.3. Stromverlaufssimulation auf Fahrzeugleitungen

In den bisherigen Simulationen des BEFs wurde der Einfluss der Leitungscharakteristiken vernachlässigt. Sie sollen im Folgenden betrachtet werden. Dazu dient erneut das generierte Modell zur Simulation eines BEFs aus Abbildung 7.24. Die zuvor entfernte Komponente *Pi Section Line* aus der SESPS-Bibliothek wird allerdings nun zur Simulation des Kabels zwischen dem *Motor* und der *B6C-Brücke* aktiviert. So können die dort auftretenden Leitungsströme untersucht werden. Der Parameter für die Anzahl der Leiter $N$ der PSL wird auf drei festgelegt. Für eine Ausgangsspannung der Batterie $U_{Bat}$ von 400 V, einen maximalen Strom $I_{max}$ von 250 A (mit 25 % Toleranz) und einer Kabellänge $l$ von 5 m zwischen *Batterie* und *Motor*, lässt sich zusammen mit der Strombelastbarkeit von elektrischen Leitungen der nötige Querschnitt des Kabels bestimmen. Hierzu wird Spalte 5 aus Tabelle 11 der Norm *VDE 0298-4 06/13* [294] verwendet. In Kabeln mit zwei bis drei Adern wird hier ein Querschnitt $A$ von 95 $mm^2$ für einen Strom von 250 A vorgegeben. Da eine automatische Bedatung des PSL-Modells innerhalb der Generierung nicht implementiert wurde, wird zur Konfiguration das zu Simscape gehörende PLPT verwendet. Hierzu müssen die geometrischen Positionen der einzelnen Leiter im Kabel, die Kabellänge sowie Leitereigenschaften angegeben werden. Zu letztgenannten gehören der Innendurchmesser $d_{Innen} = \sqrt{\frac{4 \cdot A}{\pi}} = \sqrt{\frac{4 \cdot 95\ ,mm^2}{\pi}} \approxeq 11mm$ sowie der Außendurchmesser $d_a$ = 2 * $d_i$ der Leiter, der spezifische Widerstand des Leitermaterials (Kupfer) $\rho_{Kupfer} = 56\ \frac{m}{\Omega \cdot mm^2}$ und der Durchmesser des gesamten Kabels $d_{Kabel}$ = 1 $cm$. Aus diesen Informationen werden mit dem PLPT entsprechende $NxN$ Matrizen zur Bedatung von der PSL generiert.

Tabelle 7.9 zeigt die Ergebnisse des Vergleichs der generierten BEF-Modelle aus Abbildung 7.24 mit und ohne Kabel, bei der Ausführung von 10 SWCs und einer Schrittweite von 10e-4 s.

Tabelle 7.9.: Vergleich der generierten BEF-Modelle aus Abbildung 7.24 mit und ohne Kabel, bei der Ausführung von 10 SWCs und einer Schrittweite von 10e-4 s.

| | | Ohne Kabel | Mit Kabel | Differenz | Abweichung |
|---|---|---|---|---|---|
| Laufzeit (normal) [s] | | 49,321 | 50,302 | + 0,9810 | 1,989% |
| Laufzeit (accelerator) [s] | | 17,226 | 17,670 | + 0,4440 | 2,577% |
| Speedup | | 2,863 | 2,847 | - 0,0164 | -0,574% |
| $S_t^*(normal)$ | | 1,644 | 1,677 | + 0,0327 | 1,989% |
| $S_t^*(accelerator)$ | | 0,574 | 0,589 | + 0,0148 | 2,577% |
| Mittelwert (5-15s) [km/h] | | 49,623 | 49,628 | + 0,0049 | 0,010% |
| Standardabweichung (5-15s) | | 0,374 | 0,374 | + 0,0007 | 0,187% |
| Endbatterieladezustand [%] | | 98,866 | 98,867 | + 0,0018 | 0,002% |
| Mittelwert (Phase) | A | 81,6925 | 81,9287 | + 0,2362 | 0,289% |
| | B | 82,2436 | 81,9148 | - 0,3288 | -0,400% |
| | C | 81,9791 | 82,2826 | + 0,3035 | 0,370% |
| Standardabweichung (Phase) | A | 35,1172 | 35,4891 | + 0,3719 | 1,059% |
| | B | 35,7611 | 35,3322 | - 0,4289 | -1,199% |
| | C | 35,3436 | 35,6535 | + 0,3099 | 0,877% |

Es ist vor allem zu erkennen, dass die Laufzeit im Modus *accelerator* um ca. 2,577 % zugenommen hat. Dennoch bleibt $S_t^*(accelerator) < 1$, weswegen die Ausführung in diesem Fall skaliert. Die Standardabweichung hat um ca. 0,187 % zugenommen, sodass in der Regelung von einem geringfügig höheren Rauschen ausgegangen werden kann. Die restlichen Auswirkungen auf den Batterieladenzustand sowie die Mittelwerte und Standardabweichungen der Motorstromphasen sind mit maximal 1,2 % gering. Auf den ersten Blick scheint damit der Mehrwert der Kabelsimulation im Verhältnis zum Anstieg der Laufzeit nicht gerechtfertigt zu sein. Allerdings ist bei der Untersuchung der Ströme durch das Kabel ein weiterer Aspekt aufgefallen. Abbildung 7.32 zeigt dazu die Motorströme der Phase A beider BEF-Modelle (mit und ohne Kabel) bei der Ausführung von 10 SWCs im Zeitbereich von 0,47 s bis 0,49 s.

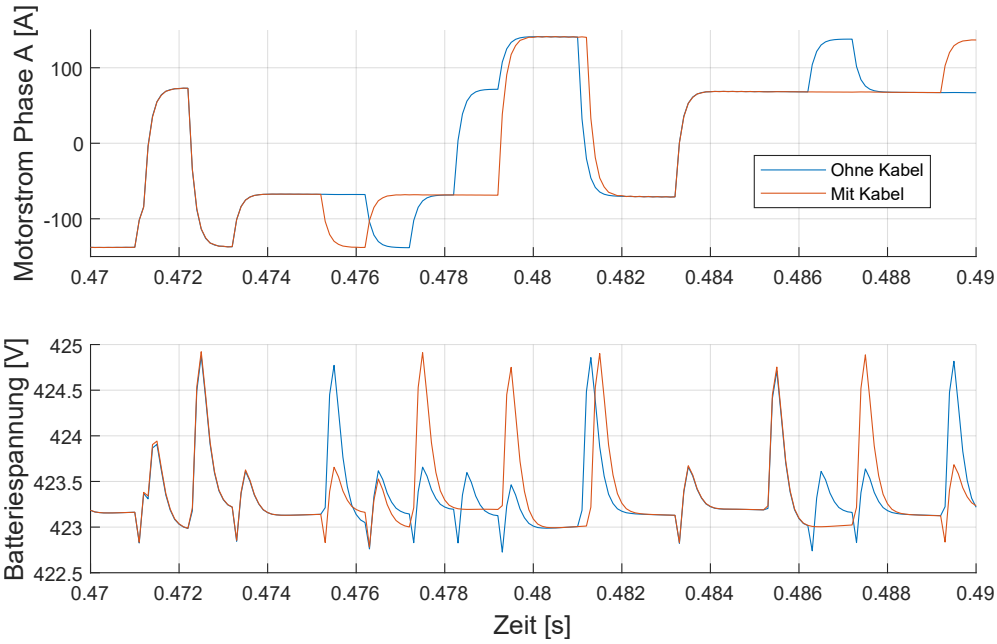



Abbildung 7.32.: Vergleich der Motorstromphase A und des Batteriespannungsverlaufs der generierten BEF-Modelle aus Abbildung 7.24, mit und ohne Kabel, bei der Ausführung von 10 SWCs und einer Schrittweite von 10e-4 s.

Der Zeitbereich wurde gewählt, da ab ca. 0,475 s erstmals ein Unterschied bei den Verläufen auftritt. Es ist zu erkennen, dass sich zeitliche Verschiebungen und im weiteren Verlauf auch Unterschiede in der Steigung und Amplitude ergeben. Diese wirken sich, wie im unteren Teildiagramm von Abbildung 7.32 dargestellt, auf den Batteriespannungsverlauf und damit auf die Stabilität der Versorgungsspannung aus. Der Aspekt ist vor allem dann wichtig, wenn es für elektronische Komponenten, wie ECUs, keine eigene Batterie bzw. ein gesondertes Versorgungsnetz gibt. Wird ein Gleichstromwandler ohne entsprechende Filter und Anpassun-

gen verwendet, so kann es zu Spannungseinbrüchen oder Überspannungen an der Elektronik kommen, wenn die entsprechenden Schutzschaltungen nicht dafür ausgelegt sind. In [305] wird des Weiteren die Auswirkung der verwendeten Kabellänge auf die Einschalt- und Ausschaltverluste von IGBTs beschrieben. Eine solche Untersuchung ist mit der gewählten Abbildung ebenfalls möglich, erfordert jedoch detailliertere Modelle der Transistoren.

#### Fazit und Limitierungen

Die Simulation von Kabeln führt im Vergleich zu den beobachtbaren Auswirkungen des Ladezustands oder der Regelung zu einem verhältnismäßig hohen Laufzeitzuwachs. Dennoch kann dieser, je nach Anwendung, gerechtfertigt sein. Zum Beispiel, wenn es um die Auslegung von Schutzschaltungen für elektronische Verbraucher oder die Optimierung von Energieverlusten in Abhängigkeit der Kabellänge geht.

Da die SESPS-Bibliothek eigentlich für die Simulation von Hochspannungsnetzen ausgelegt ist, können nicht alle Komponenten sinnvoll für die Leitungssatzsimulation verwendet werden. Der in der Bibliothek enthaltene Block *Distributed Parameters Line* ist z.B. für Leitungen im Kilometerbereich ausgelegt. Wird er verwendet, so sinkt die für die Stromsimulation nötige Zeitschrittweite im Verhältnis zur Zehnerpotenz der gewählten Leitungslänge in der Einheit km, was bei kleinen Leitungslängen zu sehr hohen Laufzeiten führt. Außerdem werden die Leitungserwärmung sowie mechanische Beanspruchungen und Verlegewege der Kabel im Fahrzeug nicht beachtet. Letztgenanntes würde für die Beurteilung der *elektromagnetischen Verträglichkeit (EMV)* benötigt werden. Ein weiterer Punkt ist, dass zur Parametrisierung von SESPS-Kabelkomponenten standardmäßig das PLPT verwendet werden muss. Dadurch lässt sich eine automatisierte Bedatung bei der Generierung nur aufwendig realisieren.

### 7.3.4. Einschränkungen und Integrationsreife

In den vorausgegangenen Abschnitten wurde die Abbildung von PREEvision nach Simulink hinsichtlich Performanz, Nutzen und Genauigkeit untersucht. Dazu wurde das AC7-Modell aus der SESPS-Bibliothek als Grundlage und Referenz verwendet. Es wurde mit einer entsprechenden Architektur, einem Szenario, dem EF und einer ECU zum Scheduling von SWCs erweitert. Aus den Komponenten bzw. Subsystemen des erweiterten Modells wurden dann Templates abgeleitet. Anschließend wurde ein PREEvision-Architekturmodell erstellt und mit den Templates verknüpft. Das schließlich in PREEvision generierte, hybride Simulink-Simulationsmodell integriert die PREEvision-Ebenen LA, SA und HA. Dabei werden drei MAEs kombiniert, nämlich Simulink, Simscape und SimEvents. Neben der Schwierigkeit, dass parallel ablaufende Prozesse entsprechend synchronisiert sein müssen, um Kausalität und Konsistenz zu gewährleisten, führt die verwendete permanenterregte Synchronmaschine mit dem zugehörigen PI-Regler aufgrund zahlreicher Rückkopplungsschleifen und einem nicht-linearen Charakter zu einem besonders komplexen Modell [204, 52].

### Performanz

Bei den Untersuchungen wurde festgestellt, dass die Grenzen der Skalierbarkeit im Modus *normal* bei einer variablen Schrittweite von 1e-4 s, ohne Kabelsimulation und mit zwei auf einer ECU ausgeführten SWCs erreicht werden. Nach [38] wurde deshalb zur Steigerung der Skalierbarkeit der Modus *accelator* untersucht, wodurch die Anzahl der SWCs bei gleicher Schrittweite auf ca. 22 gesteigert werden konnte. Außerdem wurde nach [38] die Simulink *Parallel Computing Toolbox* installiert, um eine Leistungssteigerung durch eine parallele Ausführung auf einem HPC-Cluster zu erreichen. In Simulink kann diese Toolbox jedoch nur verwendet werden, „um mehrere Simulationen eines Modells parallel auszuführen“ [266]. D.h. eine automatisierte Aufteilung des Simulationsmodells in Teilmodelle, die auf unterschiedlichen Computern ausgeführt werden, ist nicht möglich. Für die Simulation des BEFs lässt sich die Parallel Computing Toolbox somit nicht anwenden. Des Weiteren wurden die Schrittweiten zur Laufzeitoptimierung der Templates iterativ so groß wie möglich gewählt, um ausreichend genaue Ergebnisse zu erhalten, respektive eine funktionierende Regelung. Aufgrund der Einschränkungen in der Performanz sollten größere Modelle mit mehr als 22 SWCs eher in späteren Entwicklungsphasen durch Ausführung von generiertem Code auf einer (virtuellen) Hardware evaluiert werden.

### Anwendbarkeit und Validierbarkeit

Die gefundene Grenze bzgl. der Performanz deckt sich mit den Grenzen der Anwendbarkeit. Einerseits erleichtern Templates, Szenarien und Modellgenerierung die Anwendbarkeit und beschleunigen den Entwicklungsprozess, da Verhaltensmodellierung und Validierung wegfallen, sofern sie einmal durchgeführt wurden. Andererseits führt aber die Kombination der Templates zu einem emergenten System, welches zwei Hauptherausforderungen mit sich bringt. Die Erste tritt bei der Fehlersuche auf. Dabei sind Ursache und Wirkung aufgrund von Rückführungsschleifen auf den ersten Blick nicht zu erkennen. Wird bei der Fehlersuche eine zu niedrige PWM-Frequenz zur Ansteuerung festgestellt, so müssen im weiteren Verlauf Vermutungen angestellt werden, die auf Erfahrung beruhen. Dabei gibt es entsprechend viele Freiheitsgrade. So könnte die Ursache auf das Scheduling oder eine zu niedrige Batteriespannung zurückzuführen sein. Dass der Grund eine zu geringe Schrittweite in der Stromsimulation ist, was an dem hybriden Charakter des Modells in Kombination mit dem Scheduling liegt, erfordert entsprechende Kenntnisse des E/E-Architekten in der Simulation und über den Simulator. Sind diese, wie in [311] beschrieben, nicht vorhanden, so ist eine Anwendung nicht möglich. Die zweite Herausforderung ergibt sich bei der Einhaltung der Datenkonsistenz zwischen den Parametern beliebig zusammengestellter Templates. Ändert sich bspw. die Polzahl des Motors, so muss neben dem Template des Motors auch das Template des Reglers auf die Polzahl angepasst werden. Außerdem werden in Templates teilweise Komponenten verwendet, die nicht mit einer bei der Generierung übergeordneten MAE vereinbar sind. Eine mögliche Lösung dafür sind Integrationsmodelle nach [27]. Auf die Abbildung übertragen könnten diese Integrationsmodelle als „Super-Templates“ realisiert werden, bei denen für unterschiedliche Templates gleiche Parameter in Konfigurationsmodulen zusammengefasst werden. Eine Validierung des generierten Gesamtmodells lässt sich dabei zwar einschränken, aber nicht vermeiden. In jedem

Fall muss nämlich die MAE-übergreifende Zeitschrittsynchronisierung zwischen den Templates überprüft werden.

### Einschränkungen durch Simulink

Bei der Realisierung und Überprüfung der Abbildung sind Einschränkungen in Abhängigkeit des verwendeten Simulators Simulink aufgefallen. So können unterschiedliche Daten aus Bussignalen nicht zusammen in einem einzigen DSM-Block zwischengespeichert werden. Dies gilt auch für die unterschiedlichen Signale aus einem Multiplexer-Block (*Mux*). Kombiniert ein Signal also mehrere Datenelemente, so muss es aufgelöst werden und für jedes seiner Datenelemente ein eigener DSM-Block angelegt werden. Insbesondere bei analogen, elektrischen Signalen müssen mehrfach zugeordnete logische Daten bei der Abbildung aufgelöst werden.

In Bezug auf das Prozessormodell und die verwendeten Blöcke vom Typ *Entity Generator* ist ferner zu beachten, dass Blöcke vom Typ *Input Switch* mit maximal 127 Eingängen konfiguriert werden können. Gibt es im Modell also mehr als 127 zyklische ausgeführte SWCs und damit Entity-Generator-Blöcke, so müssen diese auf mehrere Input-Switch-Blöcke aufgeteilt und kaskadiert werden, was sich auf die Laufzeit der Simulation auswirkt.

Ferner kann für die in FCS enthaltenen Blöcke keine individuelle Sample-Zeit angegeben werden. Stattdessen wird sie von der nächsthöheren Modellhierarchieebene geerbt. Im Falle der SWC-FCS wird die Sample-Zeit also durch das Scheduling bestimmt. SWC-Templates dürfen daher nicht aus Blöcken mit unterschiedlich zugewiesenen Sample-Zeiten aufgebaut sein.

Darüber hinaus müssen in jedem Fall „algebraische Schleifen“ vermieden werden, da sonst das Modell nicht ausgeführt werden kann. Diese betreffen den Simscape-Anteil der Simulation. Wird ein Strom oder eine Spannung mit einem Mess-Block ausgelesen und damit in ein logisches Simulink-Signal gewandelt, so darf dieses logische Signal nicht wieder über einen entsprechenden Strom- oder Spannungsgenerator in das gleiche Netz zurückgeführt werden. Es würde sonst eine Schleife entstehen, die entweder vom Gleichungslöser nicht aufgelöst werden kann oder durch sehr kleine Schrittweiten zu unverhältnismäßig langen Laufzeiten und Abbrüchen führt. Dies schränkt die Sensorabstraktion ein. Zum Beispiel, wenn ein elektrisches Signal vorliegt, das zur Übertragung über ein Bussystem in ein logisches Signal gewandelt wird, danach verarbeitet wird und anschließend wieder als elektrisches Signal in das gleiche Netz zurückgeführt wird.

Schließlich müssen Namen noch innerhalb eines Blockes einzigartig sein und dürfen keine Umlaute beinhalten. Dies ist bei der Generierung aus PREEvision heraus zu beachten, da die Namen von den dort modellierten Komponenten übernommen werden.

### Zentrale Einschränkungen der Abbildung

Neben den bereits in Abschnitt 6.3 beschriebenen Limitierungen, wurden im Zuge der Evaluation weitere Einschränkungen aufgedeckt, die im Folgenden beschrieben werden.

Bisher wird bei der Abbildung die Bedatung der Kabel nicht beachtet, da die dafür notwendigen Matrizen mit einem in Simscape integrierten Werkzeug (PLPT) berechnet werden. Für

einen skalierten Einsatz der Simulation müsste die Bedatung entsprechend automatisiert in der Generierung realisiert werden.

Des Weiteren wurden in der Abbildung keine besonderen Datentypen, wie Boolesche Werte oder Hexadezimalzahlen, sowie deren gültige Wertebereiche beachtet. Gerade bei Booleschen Werten ist der Spannungspegel einer logischen Eins für die richtige Interpretation des elektrischen Signals entscheidend. Außerdem beeinflusst der Spannungspegel die Leistungsaufnahme. Die Zuordnung von Wertebereichen auf die Spannungsbereiche ist zudem für die Genauigkeit der A/D- bzw. D/A-Wandlung entscheidend.

#### Integrationsreife

Bei der Abbildung wurde mit Simulink ein vergleichsweise performanter Simulator (vgl. Kapitel 5) ausgewählt. Außerdem wurden dabei Methoden zur Steigerung der Skalierbarkeit nach dem Stand der Technik berücksichtigt und umgesetzt (Anforderung 4). Dennoch haben die Untersuchungen gezeigt, dass die Simulation von großmaßstäblichen PREEvision-E/E-Architekturmodellen aufgrund der Performanz und der Anwendbarkeit nicht sinnvoll ist. Anhand von Szenarien, die Anwendungsfälle beschreiben und helfen das E/E-Architekturmodell auf die nötigen Komponenten einschränken, konnte dennoch ein Mehrwert für E/E-Architekten geschaffen werden. So konnte mit der Simulation das Verständnis erhöht und die Auslegung von Reglern verbessert werden. Des Weiteren konnten Maßnahmen gegen (Cyber-) Angriffe evaluiert, Stromverbräuche und Ströme beurteilt sowie Vor- und Nachteile einer verwendeten Technologie untersucht werden. Durch den Einsatz von Templates und der Modellgenerierung wird zudem der Entwicklungsaufwand reduziert, da die Verhaltensmodellierung und die Validierung der Teilmodelle (Templates) im Prozess wegfallen. Dadurch werden die Anforderungen 2 und 3 erfüllt. Zu bedenken ist allerdings auch, dass zusätzlich zu Simulink viele Toolboxen benötigt werden, um die E/E-Architektur in ein hybrides Simulationsmodell zu überführen. Dazu zählen Simscape, SimEvents und Stateflow. Eine Co-Simulation wurde mit Simulink im Vergleich zu Ptolemy II noch nicht untersucht. Trotz des gezeigten Nutzens ist ein industrieller Einsatz aufgrund der beschriebenen Einschränkungen in der Performanz und der Anwendbarkeit nur bedingt möglich. Anforderung 1 ist damit nicht erfüllt.

#### Schlussfolgerungen

In Bezug auf Forschungsfrage 4 wurden anhand verschiedener Simulationen und dem Einsatz zweier Simulatoren die Grenzen einer E/E-Architektursimulation hinsichtlich der Skalierbarkeit aufgezeigt. Dabei schränken vor allem die emergenten Modelleigenschaften hybrider und zusammengesetzter Simulationsmodelle die Anwendbarkeit ein, sodass die Frage nach der Performanz bei der Skalierbarkeit eine der Anwendbarkeit untergeordnete Rolle spielt. Die in dieser Arbeit anhand von Anwendungsfällen definierten Szenarien helfen durch ihre Schnittstellen die Simulationsmodellgröße entsprechend dem Verkürzungsmerkmal nach [241] einzuschränken. Dadurch können emergente Eigenschaften reduziert und so die Anwendbarkeit gesteigert werden. Ebenso kann die Performanz durch die Einschränkung der Modellgröße erhöht werden. Zusätzlich können die Vorteile einer HW-zentrierten Generierung mit Templates aus dieser Arbeit zur Steigerung der Anwendbarkeit und Validierbarkeit genutzt werden.

# 8. Zusammenfassung und Ausblick

## 8.1. Zusammenfassung

In dieser Arbeit wurden die Grenzen der Skalierbarkeit einer modellbasierten E/E-Architektursimulation untersucht. Dazu wurde ein bestehender Ansatz zur automatisierten Synthese von Simulationsmodellen aus PREEvision-E/E-Architekturmodellen nach Bucher [43] verwendet. Dieser wurde entsprechend den in der Arbeit identifizierten Anforderungen hinsichtlich großmaßstäblicher Modelle angepasst und erweitert. Nach Auswahl eines performanten Simulators (Simulink) wurde die Implementierung umgesetzt. Anschließend wurden industriell relevante Anwendungsfälle bestimmt. Anhand von ihnen wurden sowohl die Konzepte als auch die Skalierbarkeitsgrenzen evaluiert.

Zunächst wurde der in [43] verwendete Simulator Ptolemy II [280] mit Alternativen verglichen, die ebenso eine integrierte und hybride Simulation erlauben. Dafür wurden acht aus 93 ermittelten Simulatoren herausgesucht und genauer analysiert. Außerdem wurden industrielle Anforderungen definiert. Es wurde deutlich, dass Ptolemy II diese deutlich schlechter als die Alternativen erfüllt. Als beste Alternative wurde schließlich Simulink [265] identifiziert, zusammen mit den Blocksets SimEvents, Simscape und Stateflow. Anhand von synthetischen, skalierbaren Benchmarks wurde Simulink abschließend mit Ptolemy II bzgl. der Performanz verglichen. Hierzu wurden Benchmarks aus der Literatur verwendet oder eigene entwickelt, um die jeweiligen MAEs getrennt zu analysieren. Mit den Benchmarks wurde deutlich, dass Simulink performanter als Ptolemy II ist. Es wurde ferner gezeigt, dass Ptolemy II hauptsächlich schlecht bei größeren Modellen und länger gewählten logischen Zeiten skaliert. Der größte Schwachpunkt ist dabei die zeitkontinuierliche MAE, welche bei E/E-Architektursimulationen vor allem für elektrische Verbindungen und analoge Signale benötigt wird. Des Weiteren wurden algorithmische Defizite bei der parallelen Ausführung und der Evaluation von Zustandsübergängen aufgezeigt. Zudem fehlt Ptolemy II im Vergleich zu Simulink eine Möglichkeit die Modelle kompiliert und automatisch optimiert auszuführen.

Nach der Festlegung von Simulink als Zielsimulator wurde ein Konzept entwickelt, um E/E-Architekturmodelle aus PREEvision automatisiert in Simulink-Modelle zu überführen. Dabei wurden die Herausforderungen, die bei der Simulation von E/E-Architekturen als CPS entstehen, adressiert. Sie beziehen sich auf die Performanz, die Anwendbarkeit und die Validierbarkeit. Außerdem müssen die E/E-Architekturebenen in einem hybriden Simulationsmodell integriert werden.

Die verwendeten Methoden schließen modellbasierte und ausführbare Szenarien zur Reduktion der Modellgröße über festgelegte Schnittstellen ein. Szenarien erlauben ferner eine Bereitstellung von Stimuli und eine Evaluation der Systemsignale mit entsprechenden Evaluationsparametern. Mit dem eingeführten Konzept des Ego-Fahrzeugs werden darüber hinaus me-

chanische Kopplungen zwischen Sensoren und Aktoren über die Umgebung und die Fahrdynamik beachtet. Zusätzlich wird damit die Integration von Co-Simulationen ermöglicht.

Zur Generierung der Simulink-Modelle werden parametrisierbare, pre-validierte und für schnelle Ausführungen optimierte Teilmodelle, Templates genannt, verwendet. Templates adressieren damit sowohl die Validierbarkeit als auch die Performanz. Sie können innerhalb einer HW-zentrierten Abbildung den dort vorhandenen Leistungssatzkomponenten, wie Batterien, Stecker oder Kabel, fest zugewiesen sein. Die Verbindungen solcher Leistungssatzkomponenten werden physikalisch, in Form von elektrischen Signalen, simuliert. Durch die fest zugewiesenen Templates können Simulationsmodelle auch ohne explizite Verhaltensmodellierung und Validierung generiert werden, was die Anwendbarkeit erhöht. Alternativ können Templates aber den Komponenten auch manuell zugewiesen werden. Dies geschieht mittels der im Zuge dieser Arbeit in [P3] eingeführten TALIA und ist für die Komponenten aus der LA und SA obligatorisch.

Zur Bildung hybrider Modelle und der Integration der PREEvision-Ebenen werden die HWCs Sensoren, Aktoren und ECUs verwendet. Sensoren und Aktoren integrieren die Umgebungssignale aus der LA mit der HA, während ECUs die SWCs aus der SA mit der HA integrieren. Dazu müssen an den Ein- und Ausgängen der HWCs im Simulationsmodell die elektrischen Signale in logische gewandelt werden und umgekehrt. Die Ausführung der SWCs wird innerhalb einer in der Arbeit eingeführten und an AUTOSAR angelehnten LU gesteuert. Dabei wird der Datenfluss durch eine Zwischenspeicherung aller an den Ausgängen der SWCs anliegenden Daten aufgebrochen. So kann ein Scheduling realisiert werden. Des Weiteren kann auch die Leistungsaufnahme des Prozessors anhand der Auslastung der Kerne simuliert werden.

Um Simulink-Simulationsmodelle automatisiert, objektorientiert und mithilfe von Java-Stellvertretermodellen in Java aufzubauen, wurde ein Framework entworfen. Es wird bei der Generierung verwendet und nutzt die ME-API-J als Schnittstelle.

Für die Evaluation wurden zunächst Anwendungsfälle für eine Simulation analysiert und anhand passender Kriterien ausgewählt. Dabei wurden aktuelle automobile Trends, funktionale E/E-Domänen, PREEvision und weitere Industriebereiche analysiert. Außerdem wurde eine Umfrage durchgeführt. Zur Evaluation wurden zwei Hauptanwendungsfälle, die mehrere Anwendungsfälle integrieren, abgeleitet und untersucht. Im ersten Fall wurden möglichst komplexe FAS, d.h. mit hohem SAE-Level, in Ptolemy II simuliert. Ziel davon war primär die frühe Evaluation der in dieser Arbeit entstandenen Konzepte, solange die Implementierung des Simulink-Modellgenerators noch nicht abgeschlossen war. Dazu gehören das Szenario, das EF und die HW-zentrierte Abbildung. Sekundär konnten weitere Performanzanalysen von Ptolemy II durchgeführt werden. Im zweiten Fall wurde ein BEFs sowohl in Ptolemy II als auch in Simulink simuliert. Dadurch konnten beide Simulatoren anhand eines konkreten Anwendungsfalls verglichen werden. In Simulink lag der Fokus, zusätzlich zur Untersuchung des Energiebedarfs, auf der Beurteilung der eingesetzten Technologien (Dimensionierung, Performanz) und des Schedulings. Die im Modell verwendete permanenterregte Synchronmaschine und der zugehörige PI-Regler bilden dabei aufgrund zahlreicher Rückkopplungsschleifen und einem nicht-linearen Charakter sowie der nötigen Integration der PREEvision-Ebenen LA, SA und HA in einem hybriden Modell ein besonders komplexes Simulationsmodell. Hierbei konnte auch der Einsatz der Templates evaluiert werden.

Es wurde gezeigt, dass durch den Einsatz von Templates und der Modellgenerierung der Entwicklungsaufwand reduziert werden kann, da insbesondere Verhaltensmodellierungen und Validierungen der Teilmodelle wegfallen. Außerdem helfen Szenarien die Modellgröße auf die nötigen Komponenten einzuschränken. Insgesamt kann ein Mehrwert für E/E-Architekten geschaffen werden, indem bspw. das Verständnis erhöht, Strom- und Signalverläufe beurteilt oder Vor- und Nachteile einer verwendeten Technologie untersucht werden können. Laufzeit- und Genauigkeitsanalysen haben jedoch die Grenzen der Skalierbarkeit hinsichtlich Performanz und Anwendbarkeit am Beispiel der BEF-Simulation aufgezeigt. Im Vergleich zu Ptolemy II ist die Simulation eines BEFs mit Simulink und den entsprechenden Blocksets, trotz aufwendigerer Abbildung und niedrigerer Abstraktion, bis zu 120,5 Mal schneller. Allerdings kommt die Abbildung bereits bei 22 SWCs im beschleunigten Modus *accelerator* an ihre Performanzgrenzen. Ab diesem Punkt ist die für die Simulation benötigte physikalische Zeit größer als die simulierte logische Zeit. Da Kompositionen von Templates zu emergenten Systemen führen können, deckt sich diese Performanzgrenze aber auch mit der Grenze der Anwendbarkeit. Zum einen, wegen der Schwierigkeiten bei der Fehlerfindung. Rückkopplungsschleifen erschweren hier das Erkennen von Ursache und Wirkung. Ferner setzt die durch Parametrisierung eintretende Vielzahl an Freiheitsgraden ein tiefes Verständnis der Simulation und des Simulators bei der Lösungsfindung voraus. Dieses ist jedoch bei E/E-Architekten i.d.R. nicht vorhanden [311]. Wenn Templates zudem nicht vom E/E-Architekten selbst entwickelt wurden, wird die Fehlerfindung aufgrund des fehlenden Wissens weiter erschwert. Zum anderen ist bei generierten hybriden Simulationsmodellen eine Synchronisation und Anpassung der Zeitschrittweiten zwischen den Templates nötig. Ebenso muss die Datenkonsistenz auch bei Parametern gewährleistet sein, die mehrere Templates betreffen. Beides kann nicht über Templates erfolgen und erfordert den Eingriff des Modellierers.

## 8.2. Ausblick

In der bisherigen Implementierung in Simulink wurden Bussysteme nur als eine Zusammenfassung logischer Signale betrachtet. Eine Co-Simulation mit einem entsprechenden Bussimulator, wie in Unterabschnitt 6.3.1 mit CANoe [288] beschrieben, würde den Einbezug von CAN-Matrizen und eine Verwendung von Busdaten als Stimuli erlauben. Dabei kann zusätzlich die Signalwandlung in elektrische Signale untersucht werden.

Außerdem wäre eine automatisierte Bedatung der Kabel innerhalb der Generierung nötig, sodass die Erstellung der notwendigen Matrizen nicht manuell mit einem zusätzlichen Werkzeug (PLPT) durchgeführt werden muss. Dadurch könnte die Datenkonsistenz bei Im- und Exporten gewährleistet werden.

Ferner werden bisher keine Datentypen wie Boolesche Werte oder Hexadezimalzahlen sowie deren Wertebereiche betrachtet. Dadurch ist keine automatisierte Zuordnung von Werten zu Spannungspegeln möglich. Diese werden z.B. für die A/D- bzw. D/A-Wandlung benötigt.

Die manuelle Synchronisation von Parametern und Zeitschrittweiten zwischen den Templates sollte automatisiert werden. [27] beschreibt hierfür Integrationsmodelle, die als „Super-Templates“ realisiert werden könnten und übergeordnete Konfigurationsmodule für eine Menge an Templates darstellen.

Der Einsatz von Templates kann zu Konvertierungsfehlern an den Ports führen, wenn auf der TALIA ein „Simulink-Port“ und ein „ConnectionPort“ miteinander verbunden werden, da sie beide in PREEvision als logischer Port repräsentiert sind. Es müssten deshalb entsprechende Portunterscheidungen in PREEvision modelliert werden können.

Zeitlogik und mathematische Ausdrücke auf den Zustandsübergängen von Zustandsautomaten werden bisher mit Strings, d.h. nicht modellbasiert, abgebildet. Zum Erhalt der Datenkonsistenz zwischen dem E/E-Architekturmodell und dem Simulationsmodell sowie zur Reduktion der Fehleranfälligkeit sollte eine modellbasierte Beschreibung umgesetzt werden.

Des Weiteren würde sich eine Co-Simulation der Simulink-Implementierung mit dem Simulator DYNA4 [289] anbieten, da dieser ebenfalls auf Simulink aufbaut und damit entsprechende Templates liefert, die in die E/E-Architektursimulation integriert werden könnten.

Nicht betrachtete Anwendungsfälle aus Abschnitt 7.1 könnten hinsichtlich eines Mehrwerts für E/E-Architekten untersucht und implementiert werden. Dazu zählen die Berücksichtigung der EMV, die Analyse der Versorgungsspannungsstabilität oder die Simulation von Isolationsspannungsüberwachungssystemen.

# A. Anhang

## A.1. Betrachtete Simulatoren

### A.1.1. Proprietär

20-Sim, ANSYS, AnyLogic, AmeSim, Arena, Autonomie, CANoe, CarMaker, CarSim, Carla, Comsol Multiphysics, DYNA4, Dymola, Easy 5, ExtendSim, Keysight Ptolemy, LabView, MapleSim, Matlab / Simulink, MATSim, MLDesigner, MS4ME, MSArchitect, PLECS, Saber, Simio, Simscale, Simprocess, Simul8, SimulationX, Simulia, VeMoF, VissSim, extendSim DE, visualSim, MapleSim

### A.1.2. Quelloffen

Ascend, AutoFOCUS3, APP4MC (Amalthea), BMS, CAP-Open COCO, CEDAR, CoCoVila, CoSMos, COOJA, CPN Tools, Deeds, Desmo-J, Elmer, Facsimile, FreeMat, Gecko, gprmax, HADES, Into-CPS-Toolchain, JAAMSIM, James II, Jmodelica, Simjulia, JAS-mine, Jsim, JuliaSim, Logisim, MARTE-based Simulator, MASON, MIXR, Mobility Testbed, Mosaik, ns-3 Simulator, Octave GNU, Omnet++, OMSimulator, OpenDS, OpenPass, OpenSim, Opnet, POLARIS, Ptolemy II, Qucs, SciLab / Xcos, SimJS, Simantics System Dynamics, SimForge, SystemLab, SUMO, Tejas, The crescendo Tool, Toolsys, Tortuga, Triquetrum, Vadere, Cossim, RinSim, Jades

## A.2. Programmstrukturanalyse von Ptolemy II

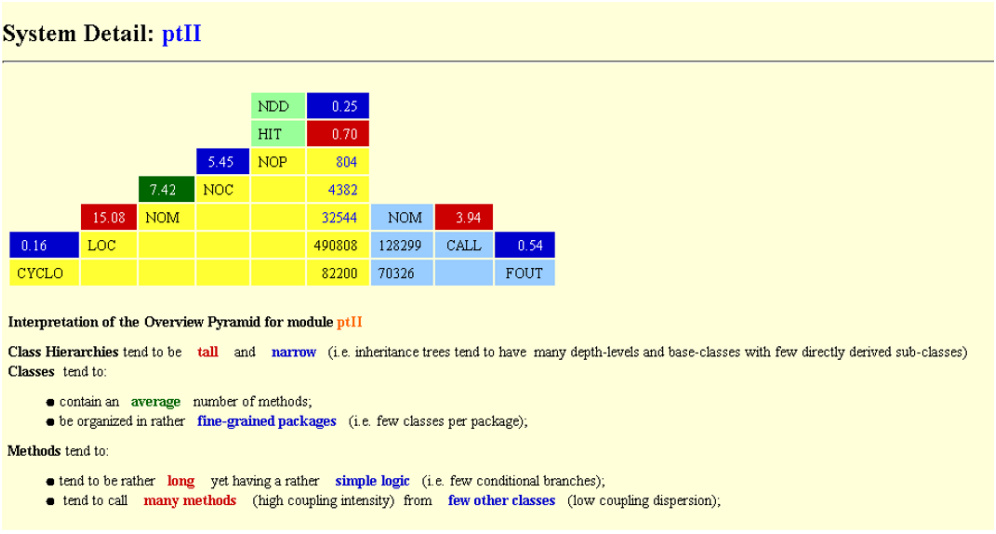



Abbildung A.1.: Überblickspyramide von Ptolemy II.

## A.3. Rohdaten zu den synthetischen Benchmarks

Die Laufzeit *Tot* bezeichnet im Folgenden bei beiden Simulatoren die gesamte Ausführungsdauer der Simulation. Bei Simulink ist hier die Kompilierung und ggf. auch die Generierung von Code mit eingeschlossen. Die Laufzeit *Self* (nur bei Simulink) bezeichnet die reine Ausführungsdauer der Simulation.

In Bezug auf den Speicherverbrauch von Simulink bezeichnet *Allok* den gesamten Speicher, der für eine Funktion und die sie aufrufende Funktion alloziert ist. *Self* schließt dagegen aufgerufene Funktionen nicht mit ein. *Peak* umfasst den maximalen Speicherbedarf zu einem Zeitpunkt der Ausführung. *Freed* ist der Speicher, der nach der Ausführung der Simulation frei wird.
Bei Ptolemy II bezeichnet (Tot) den in der JVM verfügbaren Speicher, während *Frei* den noch freien Speicher nach der Ausführung der Simulation bezeichnet. Der Speicherverbrauch errechnet sich über die Differenz der beiden Werte und wird *Used* bezeichnet.

## A.3.1. Kontinuierliche Benchmarks

Tabelle A.1.: Ptolemy-II-Benchmark: Parallele Sinusgeneratoren.

| Anzahl Sinusgen. | 10 | | | | 100 | | | | 250 | | | | 500 | 1000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | 0,216 | | | | 0,286 | | | | 0,367 | | | | 0,506 | 0,969 |
| Öffnungszeit [s] | 2,317 | | | | 1,25 | | | | 4,398 | | | | 8,4 | 19,567 |
| Simulierte Zeit [s]↓ | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | | | |
| | Tot | Tot | Free | Used | Tot | Tot | Free | Used | Tot | Tot | Free | Used | | |
| 10 | 0,101 | 172032 | 131785 | 40247 | 0,633 | 1595904 | 1314516 | 281388 | 1,333 | 2054144 | 1759185 | 294959 | | |
| 100 | 0,582 | 211968 | 187461 | 24507 | 5,149 | 1595904 | 635867 | 960037 | 13,467 | 2010624 | 1335105 | 675519 | | |
| 1000 | 4,1 | 145408 | 93831 | 51577 | 40,417 | 1604608 | 142776 | 1461832 | 109,269 | 2068480 | 1700405 | 368075 | | |
| 10000 | 12,473 | 155648 | 131264 | 24384 | 123,272 | 1626624 | 254000 | 1372624 | 337,034 | 2091520 | 1759192 | 332328 | | |

Tabelle A.2.: Simulink-Benchmark: Parallele Sinusgeneratoren.

| Anzahl Sinusgen. | 10 | | | | | | 100 | | | | | | 250 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | 0,416 | | | | | | 0,685 | | | | | | 1,143 | | | | | |
| Öffnungszeit [s] | 0,377 | | | | | | 0,952 | | | | | | 1,636 | | | | | |
| Simulierte Zeit [s]↓ | Laufzeit [s] | Speicherverbrauch [kb] | | | | | Laufzeit [s] | Speicherverbrauch [kb] | | | | | Laufzeit [s] | Speicherverbrauch [kb] | | | | |
| | Tot | Self | Allok | Freed | Self | Peak | Tot | Self | Allok | Freed | Self | Peak | Tot | Self | Allok | Freed | Self | Peak |
| 10 | 0,507 | 0,175 | 12416 | 4108 | 3320 | 4108 | 0,796 | 0,236 | 13720 | 12352 | 1364 | 4136 | 1,001 | 0,434 | 7744 | 4140 | 3544 | 4108 |
| 100 | 0,602 | 0,014 | 13356 | 12320 | 188 | 4108 | 0,853 | 0,317 | 12824 | 12316 | 64 | 5460 | 1,034 | 0,475 | 4904 | 4252 | 500 | 4140 |
| 1000 | 0,718 | 0,171 | 21400 | 20596 | 3096 | 5464 | 0,994 | 0,45 | 4328 | 4204 | -4 | 4096 | 2,404 | 1,714 | 22264 | 21104 | -584 | 5468 |
| 10000 | 0,733 | 0,219 | 20616 | 20512 | 4 | 4112 | 1,872 | 1,249 | 14136 | 12436 | 1072 | 4096 | 3,47 | 2,691 | 10288 | 9712 | 376 | 4112 |

| Anzahl Sinusgen. | 500 | | | | | | 1000 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | 1,857 | | | | | | 4,348 | | | | | |
| Öffnungszeit [s] | 2,999 | | | | | | 5,399 | | | | | |
| Simulierte Zeit [s]↓ | Laufzeit [s] | Speicherverbrauch [kb] | | | | | Laufzeit [s] | Speicherverbrauch [kb] | | | | |
| | Tot | Self | Allok | Freed | Self | Peak | Tot | Self | Allok | Freed | Self | Peak |
| 10 | 1,033 | 0,667 | 12604 | 15400 | -3044 | 4108 | 2,178 | 1,463 | 26044 | 17132 | 12476 | 8568 |
| 100 | 1,496 | 0,888 | 4492 | 4140 | 4 | 4112 | 5,59 | 4,737 | 25872 | 16120 | 2336 | 4036 |
| 1000 | 3,862 | 3,064 | 5888 | 4260 | 1404 | 4124 | 10,76 | 9,858 | 18360 | 17124 | 7560 | 4040 |
| 10000 | 6,595 | 5,769 | 8376 | 6924 | -1340 | 4112 | 22,14 | 21,26 | 16552 | 13568 | -5200 | 8196 |

Tabelle A.3.: Ptolemy-II-Benchmark: Integratorkette.

| Anzahl Integratoren | 5 | | | | 10 | | | | 20 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | 0,214 | | | | 0,227 | | | | 0,218 | | | |
| Öffnungszeit [s] | 0,467 | | | | 0,627 | | | | 0,7261 | | | |
| Simulierte Zeit [s]↓ | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | |
| | Tot | Tot | Free | Verbraucht | Tot | Tot | Free | Verbraucht | Tot | Tot | Free | Verbraucht |
| 10 | 0,15 | 2007040 | 1868944 | 138096 | 0,2 | 2026496 | 1398271 | 628225 | 0,466 | 2010112 | 1535450 | 474662 |
| 100 | 2,138 | 2007040 | 1702927 | 304113 | 96,869 | 2038784 | 1735961 | 302823 | 79290 | 3594240 | 33239 | 3561001 |
| 1000 | 27,134 | 2016768 | 1734795 | 281973 | NA | NA | NA | NA | NA | NA | NA | NA |

Tabelle A.4.: Simulink-Benchmark: Integratorkette.

| Anzahl Integratoren | 5 | | | | | | 10 | | | | | | 20 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | 0,463 | | | | | | 0,471 | | | | | | 0,486 | | | | | |
| Öffnungszeit [s] | 0,403 | | | | | | 0,400 | | | | | | 0,481 | | | | | |
| Simulierte Zeit [s]↓ | Laufzeit [s] | Speicherverbrauch [kb] | | | | | Laufzeit [s] | Speicherverbrauch [kb] | | | | | Laufzeit [s] | Speicherverbrauch [kb] | | | | |
| | Tot | Self | Allok | Freed | Self | Peak | Tot | Self | Allok | Freed | Self | Peak | Tot | Self | Allok | Freed | Self | Peak |
| 10 | 0,448 | 0,103 | 12620 | 12316 | 12 | 2732 | 0,811 | 0,162 | 20864 | 22776 | 76 | 4108 | 0,702 | 0,143 | 17808 | 17776 | -5460 | 4108 |
| 100 | 0,673 | 0,163 | 26016 | 25928 | -2688 | 5460 | 0,717 | 0,172 | 12372 | 12316 | -2672 | 6840 | 0,722 | 0,183 | 17964 | 12316 | -2712 | 4124 |
| 1000 | 0,851 | 0,35 | 4576 | 3736 | 796 | 4460 | 1,023 | 0,427 | 16428 | 16396 | -13652 | 5464 | 0,815 | 0,315 | 5576 | 5348 | -2540 | 2736 |

## A.3.2. Zeitdiskrete Benchmarks

Tabelle A.5.: Simulink-Benchmark: Ereignisliste.

| Ereignisse | | 100 (10 Ereignisse in 0,1 s) | | | | | | 1000 (10 Ereignisse in 0,01 s) | | | | | | 10000 (10 Ereignisse in 0,001 s) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ↓Modell | | Laufzeit [s] | | Speicherverbrauch [kb] | | | | Laufzeit [s] | | Speicherverbrauch [kb] | | | | Laufzeit [s] | | Speicherverbrauch [kb] | | | |
| Simulierte Zeit [s]↓ | | Tot | Self | All | Freed | Self | Peak | Tot | Self | All | Freed | Self | Peak | Tot | Self | All | Freed | Self | Peak |
| Einfach | 10 | 0,863 | 0,175 | 35380 | 73476 | -22896 | 7644 | 0,808 | 0,128 | 34808 | 27692 | 16 | 7644 | 0,902 | 0,204 | 74176 | 72908 | 7772 | 7644 |
| | 20 | 0,788 | 0,123 | 42944 | 50028 | 260 | 7644 | 0,836 | 0,145 | 73148 | 72888 | 0 | 7644 | 1,02 | 0,328 | 51432 | 50492 | 7704 | 7644 |
| 2 Parallel | 10 | 0,898 | 0,134 | 54156 | 49968 | -22372 | 7644 | 0,956 | 0,14 | 50208 | 52096 | 192 | 7644 | 1,091 | 0,272 | 100864 | 99936 | 16236 | 32096 |
| | 20 | 0,949 | 0,145 | 73212 | 73416 | -15548 | 7640 | 0,96 | 0,163 | 50064 | 49968 | 7640 | 7644 | 1,342 | 0,518 | 76988 | 77020 | 4 | 8216 |

| Ereignisse | | 100000 (10 Ereignisse in 0,0001 s) | | | | | | 1000000 (10 Ereignisse in 0,00001 s) | | | | | | 10000000 (10 Ereignisse in 0,000001 s) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ↓Modell | | Laufzeit [s] | | Speicherverbrauch [kb] | | | | Laufzeit [s] | | Speicherverbrauch [kb] | | | | Laufzeit [s] | | Speicherverbrauch [kb] | | | |
| Simulierte Zeit [s]↓ | | Tot | Self | All | Freed | Self | Peak | Tot | Self | All | Freed | Self | Peak | Tot | Self | All | Freed | Self | Peak |
| Einfach | 10 | 1,665 | 0,887 | 105572 | 104244 | 920 | 19000 | 8,022 | 6,872 | 386588 | 379604 | -2040 | 207036 | 68,151 | 67,222 | 3130904 | 3061800 | 44952 | 2982256 |
| | 20 | 2,92 | 2,197 | 121956 | 119608 | -5280 | 19000 | 21,089 | 20,079 | 458728 | 370220 | 80856 | 241872 | 208,78 | 207,862 | 3757112 | 3395064 | 331324 | 3461772 |
| 2 Parallel | 10 | 2,432 | 1,427 | 187052 | 187060 | -7648 | 35920 | 13,278 | 12,171 | 730140 | 812548 | -90296 | 560464 | 136,146 | 134,761 | 6064156 | 6020216 | 5352 | 5783696 |
| | 20 | 4,731 | 3,681 | 192712 | 190036 | -18748 | 35924 | 36,535 | 35,514 | 871148 | 833248 | 14872 | 680828 | 372,547 | 368,278 | 8401692 | 7833820 | 522704 | 5935768 |

Tabelle A.6.: Ptolemy-II-Benchmark: Ereignisliste.

| Ereignisse | | 100 (10 Ereignisse in 0,1 s) | | | | 1000 (10 Ereignisse in 0,01 s) | | | | 10000 (10 Ereignisse in 0,001 s) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ↓Modell | | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | |
| Simulierte Zeit [s]↓ | | Tot | Tot | Free | Verbrauch | Tot | Tot | Free | Verbrauch | Tot | Tot | Free | Verbrauch |
| Einfach | 10 | 0,02 | 2010112 | 1691607 | 318505 | 0,049 | 2010112 | 1651708 | 358404 | 0,116 | 2010112 | 1586410 | 423702 |
| | 20 | 0,033 | 2010112 | 1665007 | 345105 | 0,045 | 2010112 | 1615922 | 394190 | 0,115 | 2010112 | 1560170 | 449942 |
| 2 Parallel | 10 | 0,184 | 2940928 | 1831334 | 1109594 | 0,05 | 2940928 | 1685913 | 1255015 | 0,117 | 2940928 | 1605725 | 1335203 |
| | 20 | 0,02 | 2940928 | 1728743 | 1212185 | 0,05 | 2940928 | 1652522 | 1288406 | 0,217 | 2940928 | 1547398 | 1393530 |

| Ereignisse | | 100000 (10 Ereignisse in 0,0001 s) | | | | 1000000 (10 Ereignisse in 0,00001 s) | | | | 10000000 (10 Ereignisse in 0,000001 s) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ↓Modell | | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | |
| Simulierte Zeit [s]↓ | | Tot | Tot | Free | Verbrauch | Tot | Tot | Free | Verbrauch | Tot | Tot | Free | Verbrauch |
| Einfach | 10 | 0,6 | 2010112 | 1918992 | 91120 | 5,55 | 2062336 | 1532890 | 529446 | 56,867 | 2715136 | 1256794 | 1458342 |
| | 20 | 1,134 | 2010112 | 1661836 | 348276 | 11,317 | 2696704 | 2136518 | 560186 | 135,297 | 2940928 | 1918024 | 1022904 |
| 2 Parallel | 10 | 1,001 | 2940928 | 1278318 | 1662610 | 9,816 | 2937344 | 952398 | 1984946 | 106,804 | 2960384 | 1545216 | 1415168 |
| | 20 | 2 | 2940928 | 783144 | 2157784 | 20,141 | 2921984 | 1275377 | 1646607 | 250,452 | 3910656 | 1460071 | 2450585 |

Tabelle A.7.: Simulink-Benchmark: Ereignisliste parallel.

| Instanzen | 100 | | | | | | 200 | | | | | | 400 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | 0,376 | | | | | | 0,694 | | | | | | 1,236 | | | | | |
| Öffnungszeit [s] | 0,697 | | | | | | 1,178 | | | | | | 1,748 | | | | | |
| | Laufzeit [s] | | Speicherverbrauch [kb] | | | | Laufzeit [s] | | Speicherverbrauch [kb] | | | | Laufzeit [s] | | Speicherverbrauch [kb] | | | |
| | Tot | Self | All | Freed | Self | Peak | Tot | Self | All | Freed | Self | Peak | Tot | Self | All | Freed | Self | Peak |
| | 13,945 | 13,094 | 165092 | 146576 | 25456 | 38900 | 28,012 | 27,112 | 150648 | 145236 | 5416 | 73004 | 58,835 | 57,927 | 317108 | 286848 | 29896 | 240140 |

| Instanzen | 800 | | | | | | 1600 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | 2,565 | | | | | | 6,407 | | | | | |
| Öffnungszeit [s] | 3,167 | | | | | | 6,182 | | | | | |
| | Laufzeit [s] | | Speicherverbrauch [kb] | | | | Laufzeit [s] | | Speicherverbrauch [kb] | | | |
| | Tot | Self | All | Freed | Self | Peak | Tot | Self | All | Freed | Self | Peak |
| | 120,474 | 119,536 | 619796 | 560536 | 56864 | 460248 | 252,186 | 251,025 | 1219820 | 1110848 | 106068 | 978036 |

Tabelle A.8.: Ptolemy-II-Benchmark: Ereignisliste parallel

| Instanzen | 100 | | | | 200 | | | | 400 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | 0,409 | | | | 0,537 | | | | 0,524 | | | |
| Öffnungszeit [s] | 1,25 | | | | 2,6 | | | | 4,333 | | | |
| | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | |
| | Tot | Tot | Free | Verbrauch | Tot | Tot | Free | Verbrauch | Tot | Tot | Free | Verbrauch |
| | 5,389 | 2010112 | 1610330 | 399782 | 13,434 | 2721792 | 1464163 | 1257629 | 42,771 | 2689024 | 1990848 | 698176 |

| Instanzen | 800 | | | | 1600 | | | |
|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | 0,961 | | | | 1,694 | | | |
| Öffnungszeit [s] | 9,217 | | | | 23,85 | | | |
| | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | |
| | Tot | Tot | Free | Verbrauch | Tot | Tot | Free | Verbrauch |
| | 151,517 | 2598912 | 886734 | 1712178 | 511,456 | 2428928 | 787597 | 1641331 |

### A.3.3. Wertdiskrete Benchmarks

Tabelle A.9.: Simulink-Benchmark: Zustände und Übergänge.

| Anzahl Zustände | | 10 | | | | | | 100 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | | 3,555 | | | | | | 4045,979 | | | | | |
| Öffnungszeit [s] | | 2,047 | | | | | | 15,131 | | | | | |
| ↓ Simulierte Zeit [s] | | Laufzeit [s] | Speicherverbrauch [kb] | | | | | Laufzeit [s] | Speicherverbrauch [kb] | | | | |
| Periode [s] ↓ | | Tot | Self | All | Freed | Self | Peak | Tot | Self | All | Freed | Self | Peak |
| 10 | 0,1 | 1,745 | 0,485 | 5188 | 107912 | -103564 | 4104 | 28,76 | 0,329 | 39744 | 50280 | -7608 | 7644 |
| | 0,01 | 1,386 | 0,2 | 66936 | 19476 | 7768 | 15284 | 29,039 | 0,346 | 22904 | 27412 | 32 | 15284 |
| | 0,001 | 1,278 | 0,161 | 42384 | 34648 | -7568 | 7644 | 29,03 | 0,417 | 50236 | 50232 | 7640 | 7644 |
| 100 | 0,1 | 1,3 | 0,158 | 50672 | 50108 | 4 | 15284 | 28,921 | 0,338 | 27348 | 4388 | 7640 | 7644 |
| | 0,01 | 1,297 | 0,169 | 35052 | 49972 | 0 | 7644 | 29,128 | 0,411 | 50376 | 50788 | 15284 | 15284 |
| | 0,001 | 1,452 | 0,295 | 73004 | 72188 | -6864 | 7644 | 30,067 | 1,25 | 27712 | 27284 | 0 | 7644 |

Tabelle A.10.: Ptolemy-II-Benchmark: Zustände und Übergänge.

| Anzahl Zustände | | 10 | | | | 100 | | | |
|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | | 0,326 | | | | 3,283 | | | |
| Öffnungszeit [s] | | 0,511 | | | | 16,617 | | | |
| ↓ Simulierte Zeit [s] | | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | |
| Periode [s] ↓ | | Tot | Tot | Free | Vebraucht | Tot | Tot | Free | Verbraucht |
| 10 | 0,1 | 0,251 | 2606080 | 1557310 | 1048770 | 1,835 | 2166784 | 1894925 | 271859 |
| | 0,01 | 1,586 | 2576896 | 1677036 | 899860 | 5,675 | 2131968 | 1355068 | 776900 |
| | 0,001 | 4,082 | 2759168 | 2381636 | 377532 | 49,134 | 2096640 | 1266422 | 830218 |
| 100 | 0,1 | 0,4 | 2759680 | 2677928 | 81752 | 5,383 | 2094080 | 1777902 | 316178 |
| | 0,01 | 3,883 | 2428928 | 1678889 | 750039 | 48,567 | 2096640 | 1480341 | 616299 |
| | 0,001 | 38,834 | 2320384 | 1739166 | 581218 | 484,973 | 2198528 | 1333794 | 864734 |

Tabelle A.11.: Simulink-Benchmark: Hierarchiestufen Zufallsgenerator.

| Hierarchiestufen | | 10 | | | | | | 100 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | | 2,450 | | | | | | 155,296 | | | | | |
| Öffnungszeit [s] | | 1,439 | | | | | | 2,097 | | | | | |
| ↓ Simulierte Zeit [s] | | Laufzeit [s] | | Speicherverbrauch [kb] | | | | Laufzeit [s] | | Speicherverbrauch [kb] | | | |
| Periode [s] ↓ | | Tot | Self | All | Freed | Self | Peak | Tot | Self | All | Freed | Self | Peak |
| 10 | 0,1 | 0,598 | 0,068 | 57660 | 57628 | 0 | 7644 | 1,312 | 0,286 | 8060 | 5808 | 864 | 4112 |
| | 0,01 | 1,28 | 0,163 | 50156 | 50064 | 36 | 15284 | 1,056 | 0,255 | 27784 | 27076 | 22896 | 27032 |
| | 0,001 | 0,946 | 0,128 | 53672 | 49968 | 8216 | 7644 | 1,018 | 0,22 | 27100 | 27096 | 7676 | 7640 |
| 100 | 0,1 | 0,934 | 0,118 | 57700 | 27052 | -7632 | 7648 | 1,07 | 0,249 | 10120 | 9200 | 1176 | 4108 |
| | 0,01 | 0,95 | 0,132 | 34720 | 49972 | 4 | 7644 | 1,043 | 0,237 | 19440 | 27044 | 0 | 7640 |
| | 0,001 | 1,042 | 0,236 | 50504 | 42328 | 7704 | 76444 | 1,035 | 0,234 | 4156 | 4124 | 0 | 4108 |

Tabelle A.12.: Ptolemy-II-Benchmark: Hierarchiestufen Zufallsgenerator.

| Anzahl Hierarchiestufen | | 10 | | | | 100 | | | |
|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | | 0,408 | | | | 1,195 | | | |
| Öffnungszeit [s] | | 1,666 | | | | 1,493 | | | |
| ↓ Simulierte Zeit [s] | | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | |
| Periode [s] ↓ | | Tot | Tot | Free | Vebraucht | Tot | Tot | Free | Verbraucht |
| 10 | 0,1 | 0,147 | 2010112 | 1700607 | 309505 | 1,1 | 2480640 | 2105748 | 374892 |
| | 0,01 | 0,62 | 2010112 | 1616832 | 393280 | 8,275 | 2212352 | 2113087 | 99265 |
| | 0,001 | 2,5 | 2478592 | 2250785 | 227807 | 81,852 | 2277888 | 1365442 | 912446 |
| 100 | 0,1 | 0,21 | 2478592 | 1689872 | 788720 | 8,26 | 2302976 | 1477910 | 825066 |
| | 0,01 | 2,142 | 2285056 | 2003867 | 281189 | 86,818 | 2215936 | 1853576 | 362360 |
| | 0,001 | 21,511 | 2213888 | 1442955 | 770933 | 844,621 | 2256384 | 1797362 | 459022 |

Tabelle A.13.: Simulink-Benchmark: Hierarchiestufen Zeitlogik.

| Hierarchiestufen | | 10 | | | | | | 100 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | | 2,228 | | | | | | 154,344 | | | | | |
| Öffnungszeit [s] | | 0,636 | | | | | | 2,350 | | | | | |
| Simulierte Zeit [s] | Periode [s] | Laufzeit [s] | | Speicherverbrauch [kb] | | | | Laufzeit [s] | | Speicherverbrauch [kb] | | | |
| | | Tot | Self | All | Freed | Self | Peak | Tot | Self | All | Freed | Self | Peak |
| 10 | 0,1 | 1,2 | 0,194 | 80552 | 80680 | -15412 | 7648 | 1,213 | 0,16 | 65416 | 51056 | -7648 | 7648 |
| | 0,01 | 1,13 | 0,139 | 42372 | 42340 | 0 | 7648 | 1,29 | 0,18 | 67552 | 50064 | 428 | 7644 |
| | 0,001 | 2,156 | 0,155 | 79836 | 76180 | 8 | 7648 | 1,133 | 0,127 | 80808 | 75624 | -14284 | 15280 |
| 100 | 0,1 | 1,125 | 0,155 | 51008 | 57664 | 8296 | 7644 | 1,164 | 0,154 | 34748 | 50000 | 7644 | 7648 |
| | 0,01 | 1,115 | 0,123 | 34724 | 27168 | -128 | 7648 | 1,366 | 0,176 | 50060 | 50284 | 14992 | 15280 |
| | 0,001 | 1,555 | 0,176 | 36888 | 34988 | -5764 | 7644 | 1,209 | 0,149 | 75464 | 27188 | 112 | 7668 |

Tabelle A.14.: Ptolemy-II-Benchmark: Hierarchiestufen Zeitlogik.

| Hierarchiestufen | | 10 | | | | 100 | | | |
|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | | 0,318 | | | | 0,849 | | | |
| Öffnungszeit [s] | | 0,151 | | | | 0,831 | | | |
| Simulierte Zeit [s] | Periode [s] | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | |
| | | Tot | Tot | Free | Vebraucht | Tot | Tot | Free | Verbraucht |
| 10 | 0,1 | 0,985 | 2229760 | 1721145 | 508615 | 57,784 | 2252800 | 1931968 | 320832 |
| | 0,01 | 1,7 | 2309632 | 1575737 | 733895 | 605,4 | 2296320 | 1950291 | 346029 |
| | 0,001 | 14,757 | 2101760 | 1493583 | 608177 | 5825,729 | 2610688 | 1301928 | 1308760 |
| 100 | 0,1 | 1,898 | 2113024 | 1664408 | 448616 | 581,269 | 2232320 | 1606263 | 626057 |
| | 0,01 | 14,799 | 2362880 | 1764400 | 598480 | 5976,777 | 2524160 | 435180 | 2088980 |
| | 0,001 | 148,057 | 2326528 | 1761511 | 565017 | 83984,958 | 2383872 | 909051 | 1474821 |

Tabelle A.15.: Simulink-Benchmark: Parallele Zustände.

| Parallele Zustände | | 10 | | | | | | 100 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | | 1,383 | | | | | | 9,300 | | | | | |
| Öffnungszeit [s] | | 1,081 | | | | | | 2,044 | | | | | |
| Simulierte Zeit [s] | Periode [s] | Laufzeit [s] | | Speicherverbrauch [kb] | | | | Laufzeit [s] | | Speicherverbrauch [kb] | | | |
| | | Tot | Self | All | Freed | Self | Peak | Tot | Self | All | Freed | Self | Peak |
| 10 | 0,1 | 1,776 | 0,165 | 74372 | 52508 | 1376 | 7644 | 1,82 | 0,093 | 16540 | 16508 | -128 | 7640 |
| | 0,01 | 1,243 | 0,121 | 72916 | 50088 | 7512 | 7644 | 2,121 | 0,14 | 61484 | 31704 | 11504 | 11504 |
| | 0,001 | 1,342 | 0,147 | 65332 | 65388 | -96 | 7644 | 1,889 | 0,125 | 27068 | 16096 | -3772 | 7648 |
| 100 | 0,1 | 1,289 | 0,163 | 73312 | 65688 | -7836 | 7644 | 2,099 | 0,143 | 52696 | 60136 | 0 | 7648 |
| | 0,01 | 1,199 | 0,147 | 27076 | 49964 | 0 | 7644 | 1,962 | 0,144 | 50020 | 55112 | -15288 | 7644 |
| | 0,001 | 1,23 | 0,168 | 42840 | 50172 | 68 | 7644 | 1,727 | 0,155 | 50172 | 50136 | 0 | 7644 |

Tabelle A.16.: Ptolemy-II-Benchmark: Parallele Zustände.

| Parallele Zustände | | 10 | | | | 100 | | | |
|---|---|---|---|---|---|---|---|---|---|
| Generierungszeit [s] | | 0,357 | | | | 0,541 | | | |
| Öffnungszeit [s] | | 0,133 | | | | 0,367 | | | |
| Simulierte Zeit [s] | Periode [s] | Laufzeit [s] | Speicherverbrauch [kb] | | | Laufzeit [s] | Speicherverbrauch [kb] | | |
| | | Tot | Tot | Free | Vebraucht | Tot | Tot | Free | Verbraucht |
| 10 | 0,1 | 0,05 | 2789888 | 821768 | 1968120 | 0,617 | 2163712 | 1488380 | 675332 |
| | 0,01 | 0,633 | 2648576 | 1322918 | 1325658 | 5,861 | 2512384 | 1775046 | 737338 |
| | 0,001 | 5,731 | 2775040 | 1289055 | 1485985 | 58,85 | 2518528 | 1558345 | 960183 |
| 100 | 0,1 | 0,583 | 2741248 | 506883 | 2234365 | 5,822 | 2356224 | 2033965 | 322259 |
| | 0,01 | 5,7 | 2532352 | 673547 | 1858805 | 58,517 | 2617856 | 1170932 | 1446924 |
| | 0,001 | 57,4 | 2569728 | 703322 | 1866406 | 604,187 | 2792448 | 1734613 | 1057835 |

## A.4. Umfrage zu Anwendungsfällen

<table>
<tr><th colspan="2">Questionaire</th></tr>
<tr><td>Problem description</td><td>• What is the demand for simulation, what is the problem and why is simulation the best way to cope with it? Which decisions will have to be made?</td></tr>
<tr><td>Idea / test description</td><td>• What is the Idea for solving the problem? What solution is expected?</td></tr>
<tr><td>Measurand(s)</td><td>• What is the output of the simulation? What is measured?</td></tr>
<tr><td>Stimuli & disturbances</td><td>• What are the inputs and which parameters (controllable / uncontrollable) affect them?</td></tr>
<tr><td>Input model artifacts, relations and relevant attriubtes</td><td>
<table>
<tr><th>Names</th><th>Change over time / iteration step</th><th>Behavior needed</th></tr>
<tr><td>Artifact 1</td><td>☐</td><td>☐</td></tr>
<tr><td>- Attribute 1.1</td><td>☐</td><td></td></tr>
<tr><td>- Attribute 1.2</td><td>☐</td><td></td></tr>
<tr><td>Artifact 2</td><td>☐</td><td>☐</td></tr>
<tr><td>- Attribute 2.1</td><td>☐</td><td></td></tr>
<tr><td>- Attribute 2.2</td><td>☐</td><td></td></tr>
<tr><td>Artifact 3</td><td>☐</td><td>☐</td></tr>
<tr><td>- Attribute 3.1</td><td>☐</td><td></td></tr>
<tr><td>- Attribute 3.2</td><td>☐</td><td></td></tr>
</table>
</td></tr>
<tr><td>Environment</td><td>• Which further components form the surrounding / environment? => System</td></tr>
<tr><td>Affected layers<br>(mark gray)</td><td>
<table>
<tr><td>Use Cases</td><td>Customer Features</td><td>Requirements</td><td rowspan="2">Tests</td></tr>
<tr><td colspan="3">Logical Function Architecture</td></tr>
<tr><td>Software Behavior</td><td colspan="2">Service-Oriented Software Design</td><td rowspan="2">Diagnostics</td></tr>
<tr><td>Diagnostics</td><td colspan="2">Software Architecture Design</td></tr>
<tr><td>High Performance Computer</td><td>Hardware Network Topology</td><td>Communication</td><td rowspan="2">Analysis</td></tr>
<tr><td rowspan="2">Hardware Component Architecture</td><td colspan="2">Electric Circuit</td></tr>
<tr><td colspan="2">Wiring Design</td><td rowspan="2">Plan</td></tr>
<tr><td colspan="2">Geometry Topology</td><td>Harness Design</td></tr>
</table>
</td></tr>
<tr><td>Model changes and affects on further modeling</td><td>• How can the results be interpreted and influence the futher development process? Which model changes could be triggered by the result?</td></tr>
<tr><td>Support in decision making / benefit of an integration in Pv</td><td>• What is the concrete benefit for an E/E architect in terms of decision making? Particularly, what is the benefit of an integration in PREEvision?</td></tr>
<tr><td>Characteristics of the results (optional)</td><td>• In which form should the results be presented (e.g. graph)? What accuracy is necessary (e.g. deviation)? Which factors influence the accuracy?</td></tr>
</table>

Abbildung A.2.: Fragebogen zur Umfrage nach Anwendungsfällen für eine E/E-Architektursimulation bei PREEvision-Produktmanagern.

## A.5. Ptolemy-II-Modell zur Analyse eines IDS

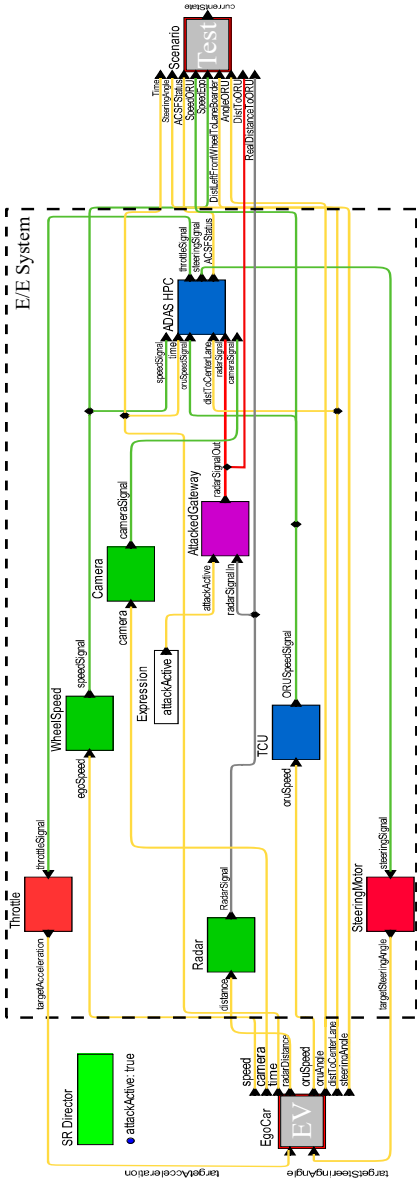



Abbildung A.3.: Ptolemy-II-Toplevel-Modell zur Analyse eines IDS (Quelle: [P4]).

## A.6. PREEvision-Modell einer zukünftigen E/E-Architektur

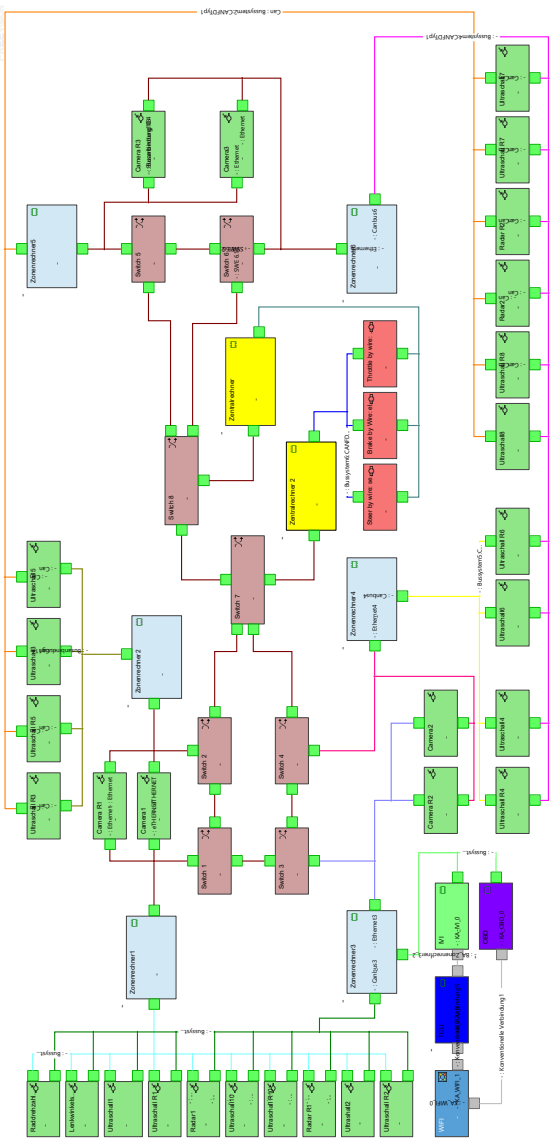


Abbildung A.4.: PREEvision-Modell einer zukünftigen E/E-Architektur (Quelle: eigene Darstellung nach [S5]).

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| **ABM** | Agentenbasierte Modellierung |
| **A/D** | Analog/Digital |
| **ADL** | Architecture Description Language |
| **AEBS** | Advanced Emergency Braking System |
| **API** | Application Programming Interface |
| **AST** | Abstract Syntax Tree |
| **AUTOSAR** | AUTomotive Open System ARchitecture |
| **B6C** | Bridge-6-Controlled |
| **BB** | Building Block |
| **BEF** | Batterieelektrisches Fahrzeug |
| **BLA** | Behavioral Logical Architecture |
| **BOM** | Base Object Model |
| **CAN** | Controller Area Network |
| **CMB** | Chandy/Misra/Bryant |
| **CPU** | Central Processing Unit |
| **CPS** | Cyber-physisches System |
| **CPMS** | Cyber-physisches mechatronisches System |
| **CSF** | Co-Simulations-Framework |
| **D/A** | Digital/Analog |
| **DDE** | Distributed Discrete Event |
| **DE** | Discrete Event |
| **DEVS** | Discrete Event System Specification |
| **DSM** | Data Store Memory |
| **DSR** | Data Store Read |
| **DSW** | Data Store Write |
| **E/E** | Elektrik/Elektronik |
| **ECU** | Electronic Control Unit |
| **EECM** | Electrical Equivalent Circuit Model |
| **EF** | Ego-Fahrzeug |
| **EKG** | Elektrokardiogramm |
| **EMV** | Elektromagnetische Verträglichkeit |
| **EMF** | Eclipse Modeling Framework |
| **EOL** | Epsilon Object Language |
| **ESP** | Elektronisches Stabilitätsprogramm |
| **EV** | Ego Vehicle |
| **EVA** | Eingabe-Verarbeitung-Ausgabe |
| **FAS** | Fahrerassistenzsystem |
| **FCS** | Function Call Subsystem |

| | |
|---|---|
| **FEM** | Finite-Elemente-Methode |
| **FIFO** | First-In First-Out |
| **FMEA** | Fehlermöglichkeits- und Einflussanalyse |
| **FMI** | Functional Mockup Interface |
| **FMU** | Functional Mockup Unit |
| **FSM** | Finite State Machine |
| **GPS** | Global Positioning System |
| **HA** | Hardware-Architektur |
| **HIL** | Hardware in the Loop |
| **HLA** | High Level Architecture |
| **HPC** | High Performance Computer |
| **HS** | Hybride Simulation |
| **HW** | Hardware |
| **HSM** | Hardware Security Module |
| **HWC** | Hardware Component |
| **IAF** | Interaktionsfolge |
| **ID** | Identifikator |
| **IDS** | Intrusion Detection System |
| **IGBT** | Insulated-Gate Bipolar Transistor |
| **IMS** | Isolation Monitoring System |
| **INCOSE** | International Council on Systems Engineering |
| **ISIC** | International Standard Industrial Classification |
| **ISO** | International Organization for Standardization |
| **ISTQB** | International Software Testing Qualifications Board |
| **ITIV** | Institut für Technik der Informationsverarbeitung |
| **JDK** | Java Development Kit |
| **JVM** | Java Virtual Machine |
| **KI** | Künstliche Intelligenz |
| **LA** | Logische Architektur |
| **LAN** | Local Area Network |
| **LIN** | Local Interconnect Network |
| **LP** | Logischer Prozess |
| **LU** | Laufzeitumgebung |
| **ME-API-J** | Matlab Engine API for Java |
| **MBSE** | Model Based Systems Engineering |
| **MIL** | Model in the Loop |
| **MAE** | Modellausführungseinheit |
| **ML** | Maschinelles Lernen |
| **MoML** | Modeling Markup Language |
| **MPI** | Message Passing Interface |
| **NACE** | Nomenclature statistique des activités économiques dans la Communauté européenne |
| **NCAP** | New Car Assessment Programme |
| **NVR** | Non-Volatile RAM |
| **OEM** | Original Equipment Manufacturer |

| | |
|---|---|
| **OICA** | Organisation Internationale des Constructeurs de Automobiles |
| **OMG** | Object Management Group |
| **OPC** | Open Packaging Conventions |
| **ORU** | Other Road User |
| **OS** | Operating System |
| **OSI** | Open Systems Interconnection |
| **OSM** | Open Street Map |
| **OTA** | Over the Air |
| **PI** | Proportional-Integral |
| **PLPT** | Power Line Parameters Tool |
| **PSL** | Pi Section Line |
| **PMSM** | Permanentmagnet-Synchronmotor |
| **POJO** | Plain Old Java Object |
| **PM** | PREEvision-Metrik |
| **PVM** | Parallel Virtual Machine |
| **PWM** | Pulsweitenmodulation |
| **QFD** | Quality Function Deployment |
| **QSS** | Quantized State System |
| **RAM** | Random Access Memory |
| **RCP** | Rich Client Platform |
| **RMI** | Remote Method Invocation |
| **RTE** | Runtime Environment |
| **SA** | Software-Architektur |
| **SAE** | Society of Automotive Engineers |
| **SES** | System Entity Structure |
| **SESPS** | Simscape Electrical Specialized Power Systems |
| **SD** | Systemdynamik |
| **SIL** | Software in the Loop |
| **SOA** | Serviceorientierte Architektur |
| **SOI** | System of Interest |
| **SOTIF** | Safety Of The Intended Functionality |
| **SUMO** | Simulation of Urban MObility |
| **SW** | Software |
| **SWC** | Software Component |
| **TTE** | Time Triggered Ethernet |
| **TALIA** | Template And Layer Integration Architecture |
| **TARA** | Threat Analysis and Risk Assessment |
| **TPM** | Trusted Platform Module |
| **UNECE** | United Nations Economic Commission for Europe |
| **UNO** | United Nations Organization |
| **UML** | Unified Modeling Language |
| **VDI** | Verein Deutscher Ingenieure |
| **vHIL** | virtual Hardware in the Loop |
| **VIL** | Vehicle in the Loop |
| **XML** | Extensible Markup Language |

# Literatur- und Quellennachweise


[1] ARRK Engineering GmbH. *Theseus-FE: Passenger Thermal Comfort Simulation*. URL: https://www.theseus-fe.com/application-areas/automotive/passenger-comfort (besucht am 25. 06. 2021).

[2] AUTomotive Open System ARchitecture. *Specification of ECU State Manager*. 2017. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_ECUStateManager.pdf (besucht am 08. 07. 2021).

[3] Ali, A. H. und Abdullah, M. Z. "A Survey on Vertical and Horizontal Scaling Platforms for Big Data Analytics". In: *International Journal of Integrated Engineering* 11.6 (2019). ISSN: 2229838X. DOI: 10.30880/ijie.2019.11.06.015.

[4] Aliwa, E., Rana, O., Perera, C. und Burnap, P. "Cyberattacks and Countermeasures for In-Vehicle Networks". In: *ACM Computing Surveys* 54.1 (2021), S. 1–37. ISSN: 0360-0300. DOI: 10.1145/3431233.

[5] Allgemeiner Deutscher Automobil-Club e.V. *Leichte Beute: Autos und Motorräder mit Keyless*. URL: https://www.adac.de/rund-ums-fahrzeug/ausstattung-technik-zubehoer/assistenzsysteme/keyless/ (besucht am 06. 12. 2022).

[6] Almeida, E. S. de, Medeiros, A. C. und Frery, A. C. "How good are MatLab, Octave and Scilab for computational modelling?" In: *Computational & Applied Mathematics* 31.3 (2012), S. 523–538. DOI: 10.1590/S1807-03022012000300005.

[7] Amdahl, G. M. "Validity of the single processor approach to achieving large scale computing capabilities: IBM Sunnyvale, California". In: *AFIPS spring joint computer conference*. URL: https://inst.eecs.berkeley.edu/~n252/paper/Amdahl.pdf (besucht am 30. 03. 2021).

[8] Angermann, A., Beuschel, M., Rau, M. und Wohlfarth, U. *Fundamentals, Toolboxes, and Examples: Grundlagen, Toolboxen, Beispiele*. 8th ed. Berlin/Boston: Walter de Gruyter GmbH, 2014. ISBN: 9783486859102. DOI: 10.1524/9783486859102.

[9] Appel, M., Oruganti, P. S., Ahmed, Q., Wilkerson, J. und Sekar, R. "A Safety and Security Testbed for Assured Autonomy in Vehicles". In: *SAE Technical Paper Series*. SAE Technical Paper Series. SAE International400 Commonwealth Drive, Warrendale, PA, United States, 2020. DOI: 10.4271/2020-01-1291.

[10] Arizona State University. *CoSMoS – Arizona Center of Integrative Modeling and Simulation*. URL: https://acims.asu.edu/software/cosmos/ (besucht am 28. 05. 2021).

[11] Assadollahi, R. und Gelowicz, S. "KI im Fahrzeug: Welche Szenarien denkbar sind". In: *AUTOMOBIL INDUSTRIE* (2018). URL: https://www.automobil-industrie.vogel.de/ki-im-fahrzeug-welche-szenarien-denkbar-sind-a-724967/?p=2 (besucht am 07. 12. 2022).

[12] Association for Standardisation of Automation and Measuring Systems. *OpenDRIVE: Concept Document*. 2020. URL: https://www.asam.net/index.php?eID=dumpFile&t=f&f=3907&token=fffa694711f0cd3cc37e61f38587b3a308e9a720 (besucht am 28. 07. 2021).

[13] Association for Standardisation of Automation and Measuring Systems. *OpenSCENARIO: User Guide*. 2022. URL: https://www.asam.net/index.php?eID=dumpFile&t=f&f=4908&token=ae9d9b44ab9257e817072a653b5d5e98ee0babf8 (besucht am 19. 09. 2022).

[14] Bach, J., Otten, S. und Sax, E. "Model based scenario specification for development and test of automated driving functions". In: *2016 IEEE Intelligent Vehicles Symposium (IV)*. 2016, S. 1149–1155. DOI: 10.1109/IVS.2016.7535534.

[15] Baier, C. und Katoen, J.-P. *Principles of model checking*. Cambridge, Mass.: MIT Press, 2008. ISBN: 9780262026499.

[16] Banks, J., Carson II, J. S., Nelson, B. L. und Nicol, D. M. *Discrete-Event System Simulation (Fifth Edition)*. Pearson, 2010. ISBN: 978-0-13-81037-2.

[17] Bargende, M., Reuss, H.-C., Wagner, A. und Wiedemann, J. *19. Internationales Stuttgarter Symposium*. Wiesbaden: Springer Fachmedien Wiesbaden, 2019. ISBN: 978-3-658-25938-9. DOI: 10.1007/978-3-658-25939-6.

[18] Baudin, R. *Run time comparison of MATLAB, Scilab and GNU Octave on various benchmark programs*. 2016. URL: http://roland65.free.fr/benchmarks/benchmarks-0.2.pdf (besucht am 27. 05. 2021).

[19] Baumann, T., Pfitzinger, B. und Jestädt, T. "Profiling Simulation Performance: The Example of the German Toll System". In: *Advances in ICT for Business, Industry and Public Sector*. Hrsg. von M. Mach-Król, C. M. Olszak und T. Pełech-Pilichowski. Bd. 579. Studies in Computational Intelligence. Cham: Springer International Publishing, 2015, S. 19–34. ISBN: 978-3-319-11327-2. DOI: 10.1007/978-3-319-11328-9_2.

[20] Baumgart, S., Fröberg, J. und Punnekkat, S. "Enhancing Model-Based Engineering of Product Lines by Adding Functional Safety". In: *MASE@MoDELS*. URL: https://www.semanticscholar.org/paper/Enhancing-Model-Based-Engineering-of-Product-Lines-Baumgart-Fr%C3%B6berg/107c0fc35399ef813dc46418423bc4c9b1f29de6/107c0fc35399ef813dc46418423bc4c9b1f29de6 (besucht am 06. 12. 2022).

[21] Becker, S. *Coupled model transformations for QoS enabled component-based software design: Zugl.: Oldenburg, Univ., Diss., 2008*. Bd. 1. The Karlsruhe series on software design and quality. Karlsruhe: Univ.-Verl., 2008. ISBN: 9783866442719.

[22] Beltrame, T. und Cellier, F. E. "Quantised State System Simulation in Dymola/Modelica Using the DEVS Formalism". In: *Modelica 2006*. 2006. URL: https://modelica.org/events/modelica2006/Proceedings/sessions/Session1c3.pdf (besucht am 25. 05. 2021).

[23] Benjamin, P., Patki, M. und Mayer, R. "Using Ontologies for Simulation Modeling". In: *Proceedings of the 2006 Winter Simulation Conference*. Hrsg. von L. F. Perrone. Orlando, Fla.: IEEE, 2006, S. 1151–1159. ISBN: 1-4244-0501-7. DOI: 10.1109/WSC.2006.323206.

[24] Berg, I. *Genauigkeit numerischer Integrationsverfahren - Schrittweite und Globaler Fehler*. URL: https://beltoforion.de/de/runge-kutta_vs_euler/ (besucht am 05. 04. 2021).

[25] Berger, C., Block, D., Hons, C., Kühnel S., Leschke, A., Plotnikov, D. und Rumpe B. "Large-Scale Evaluation of an Active Safety Algorithm with EuroNCAP and US NCAP Scenarios in a Virtual Test Environment – An Industrial Case Study". In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. 2015, S. 2280–2286. ISBN: 2153-0017. DOI: 10.1109/ITSC.2015.368.

[26] Berlin, C. *E/E-Architekturen: Frischzellenkur*. 2019. URL: https://www.automotiveit.eu/exklusiv/frischzellenkur-210.html (besucht am 28. 06. 2021).

[27] Bernardi, S., Marrone, S., Merseguer, J., Nardone, R. und Vittorini, V. "Towards a model-driven engineering approach for the assessment of non-functional properties using multi-formalism". In: *Software & Systems Modeling* 18.3 (2019), S. 2241–2264. ISSN: 1619-1366. DOI: 10.1007/s10270-018-0663-8.

[28] Birta, L. G. und Arbez, G. *Modelling and Simulation: Exploring Dynamic System Behaviour*. 3rd ed. 2019. Simulation Foundations, Methods and Applications. Cham: Springer International Publishing, 2019. ISBN: 9783030188696. DOI: 10.1007/978-3-030-18869-6.

[29] Blender Foundation. *blender*. URL: https://www.blender.org/ (besucht am 19. 12. 2021).

[30] Bondi, A. B. "Characteristics of scalability and their impact on performance". In: *Proceedings of the 2nd international workshop on Software and performance*. Hrsg. von M. Woodside. New York, NY: ACM, 2000, S. 195–203. ISBN: 158113195X. DOI: 10.1145/350391.350432.

[31] Borgeest, K. *Elektronik in der Fahrzeugtechnik: Hardware, Software, Systeme und Projektmanagement*. 4., aktualisierte und erweiterte Auflage. ATZ/MTZ-Fachbuch. Wiesbaden: Springer Vieweg, 2021. ISBN: 978-3-658-23663-2. DOI: 10.1007/978-3-658-23664-9.

[32] Bosl, A. *Einführung in MATLAB/Simulink: Berechnung, Programmierung, Simulation*. 3., vollständig überarbeitete Auflage. München: Hanser, 2020. ISBN: 9783446465466. DOI: 10.3139/9783446465466.

[33] Braess, H.-H. *Vieweg Handbuch Kraftfahrzeugtechnik*. 7th ed. ATZ/MTZ-Fachbuch Ser. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2013. ISBN: 9783658016913.

[34] Brailsford, S. C., Eldabi, T., Kunc, M., Mustafee, N. und Osorio, A. F. "Hybrid simulation modelling in operational research: A state-of-the-art review". In: *European Journal of Operational Research* 278.3 (2019), S. 721–737. ISSN: 03772217. DOI: 10.1016/j.ejor.2018.10.025.

[35] Braun, L. *Modellbasierte Design-Space-Exploration nicht-funktionaler Auslegungskriterien des Fahrzeugenergiebordnetzes*. 2018. DOI: 10.5445/IR/1000081298.

[36] Braun, W., Casella, F. und Bachmann, B. "Solving large-scale Modelica models: new approaches and experimental results using OpenModelica". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, 2017, S. 557–563. DOI: 10.3384/ecp17132557.

[37] Bresser, T. *Validierung und Verifikation (inkl. Testen, Model-Checking und Theorem Proving)*. 2016. URL: https://docplayer.org/11279539-Validierung-und-verifikation-inkl-testen-model-checking-und-theorem-proving.html (besucht am 25. 07. 2021).

[38] Bridge, S. *Speeding Up Simulation*. 2018. URL: https://www.matlabexpo.com/content/dam/mathworks/mathworks-dot-com/images/events/matlabexpo/uk/2018/speeding-up-simulation.pdf (besucht am 07. 08. 2021).

[39] Brito, A. V., Negreiros, A. V., Roth, C., Sander, O. und Becker, J. "Development and Evaluation of Distributed Simulation of Embedded Systems Using Ptolemy and HLA". In: *2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications*. IEEE, 102013, S. 189–196. ISBN: 978-0-7695-5138-8. DOI: 10.1109/DS-RT.2013.28.

[40] Bröhl, A.-P., Hrsg. *Das V-Modell: Der Standard für die Softwareentwicklung mit Praxisleitfaden*. 2. Aufl. Software - Anwendungsentwicklung - Informationssysteme. München: Oldenbourg, 1995. ISBN: 3486234706.

[41] Broy, M. *Cyber-Physical Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. ISBN: 978-3-642-14498-1. DOI: 10.1007/978-3-642-14901-6.

[42] Brüggemann, H. und Bremer, P. *Grundlagen Qualitätsmanagement: Von den Werkzeugen über Methoden zum TQM*. 3. Auflage. Wiesbaden: Springer Vieweg, 2020. ISBN: 978-3-658-28779-5. DOI: 10.1007/978-3-658-28780-1.

[43] Bucher, H. *Integrierte modell- und simulationsbasierte Entwicklung zur dynamischen Bewertung automobiler Elektrik/Elektronik-Architekturen*. 2020. DOI: 10.5445/IR/1000126919.

[44] Bundesanstalt für Straßenwesen. *Lane Change Test for ACSF*. URL: https://wiki.unece.org/download/attachments/27459841/ACSF-04-06%20%20%28D%29%20-%20ACSF-Lane%20Change%20Test.pdf?api=v2-81-32e.pdf (besucht am 07. 12. 2022).

[45] Burkacky, O., Deichmann, J. und Stein, J. P. *Automotive software and electronics 2030: Mapping the sector's future landscape*. 2019. URL: https://www.mckinsey.com/~/media/mckinsey/industries/automotive%20and%20assembly/our%20insights/mapping%20the%20automotive%20software%20and%20electronics%20landscape%20through%202030/automotive-software-and-electronics-2030-final.pdf (besucht am 10. 05. 2021).

[46] Canedo, A. *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010: 8 - 12 March 2010, Dresden, Germany ; proceedings*. Piscataway, NJ: IEEE, 2010. ISBN: 9781424470549.

[47] Cassandras, C. G. und Lafortune, S., Hrsg. *Introduction to Discrete Event Systems*. Boston, MA: Springer US, 2008. ISBN: 978-0-387-33332-8. DOI: 10.1007/978-0-387-68612-7.

[48] Cellier, F. E. *Entwurf und Entwicklung einer Dymola/Modelica Bibliothek für diskrete ereignisorientierte Systeme unter Verwendung der DEVS Methodologie*. 2005. URL: https://people.inf.ethz.ch/fcellier/SA/DEVS.html (besucht am 28. 05. 2021).

[49] Cellier, F. E. "Qualitative Modeling and Simulation: Promise or Illusion". In: *1991 Winter Simulation Conference*. 1991. DOI: 10.5555/304238.304429.

[50] Chandy, K. M. und Misra, J. "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs". In: *IEEE Transactions on Software Engineering* SE-5.5 (1979), S. 440–452. ISSN: 0098-5589. DOI: 10.1109/TSE.1979.230182.

[51] Chen, M. und Rincon-Mora, G. A. "Accurate electrical battery model capable of predicting runtime and I-V performance". In: *IEEE Transactions on Energy Conversion* 21.2 (2006), S. 504–511. ISSN: 1558-0059. DOI: 10.1109/TEC.2006.874229.

[52] Chen, W. Z., Zhang, X. Y. und Sui, X. M. "Simulation of Permanent Magnet Synchronous Motor Field Oriented Vector Control System". In: *Applied Mechanics and Materials* 672-674 (2014), S. 1234–1237. DOI: 10.4028/www.scientific.net/AMM.672-674.1234.

[53] Choi, S. H., Lee, S. J. und Kim, T. G. "Multi-fidelity modeling & simulation methodology for simulation speed up". In: *Proceedings of the 2014 ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*. Hrsg. von J. A. Hamilton, G. F. Riley und R. M. Fujimoto. New York, NY: ACM, 2014, S. 139–150. ISBN: 9781450327947. DOI: 10.1145/2601381.2601385.

[54] Chwif, L., Paul, R. J. und Barretto, M. R. P. "Discrete event simulation model reduction: A causal approach". In: *Simulation Modelling Practice and Theory* 14.7 (2006), S. 930–944. ISSN: 1569190X. DOI: 10.1016/j.simpat.2006.05.001.

[55] Collins, M. *Formal Methods*. 1998. URL: https://formal.kastel.kit.edu/~beckert/teaching/Verification-SS06/01intro.pdf (besucht am 25. 07. 2021).

[56] Controllab Products. *20-sim (Version: 4.8.2)*. 2021. URL: https://www.20sim.com/ (besucht am 04. 12. 2022).

[57] Czichos, H. *Mechatronik*. Wiesbaden: Springer Fachmedien Wiesbaden, 2019. ISBN: 978-3-658-26293-8. DOI: 10.1007/978-3-658-26294-5.

[58] Daimler AG. *Drive Pilot*. URL: https://www.daimler.com/innovation/case/autonomous/drive-pilot-2.html (besucht am 18. 08. 2021).

[59] Damiron, C. und Krahl, D. "A global approach for discrete rate simulation". In: *2014 Winter Simulation Conference (WSC 2014)*. Hrsg. von A. Tolk. Piscataway, NJ: IEEE, 2014, S. 2966–2977. ISBN: 978-1-4799-7486-3. DOI: 10.1109/WSC.2014.7020136.

[60] Dannenberg, J. und Bromberger, L. "Studie: Künftig 40 Prozent der Entwicklungskosten für Software". In: *AUTOMOBIL INDUSTRIE* (2020). URL: https://www.automobil-industrie.vogel.de/studie-kuenftig-40-prozent-der-entwicklungskosten-fuer-software-a-944011/ (besucht am 07. 12. 2022).

[61] Dargam, F., Hernández, J. E., Zaraté, P., Liu, S., Ribeiro, R., Delibašić, B. und Papathanasiou, J. *Decision Support Systems III - Impact of Decision Support Systems for Global Environments: Euro Working Group Workshops, EWG-DSS 2013, Thessaloniki, Greece, May 29-31, 2013, and Rome, Italy, July 1-4, 2013, Revised Selected and Extended Papers*. Bd. 184. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2014. ISBN: 978-3-319-11363-0. DOI: 10.1007/978-3-319-11364-7.

[62] Davis, A. M. "The art of requirements triage". In: *Computer* 36.3 (2003), S. 42–49. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1185216.

[63] DeWitt, D. und Gray, J. "Parallel database systems". In: *Communications of the ACM* 35.6 (1992), S. 85–98. ISSN: 0001-0782. DOI: 10.1145/129888.129894.

[64] Deckert, A. und Klein, R. "Agentenbasierte Simulation zur Analyse und Lösung betriebswirtschaftlicher Entscheidungsprobleme". In: *Journal für Betriebswirtschaft* 60.2 (2010), S. 89–125. ISSN: 0344-9327. DOI: 10.1007/s11301-010-0058-6.

[65] Deichmann, J., Klein, B., Scherf, G. und Stützle, R. *The race for cybersecurity: Protecting the connected car in the era of new regulation*. 2019. URL: https://web.archive.org/web/20061103062416/http:/www.eads.com/xml/content/OF00000000400004/0/74/41485740.pdf (besucht am 10. 05. 2021).

[66] Deicke, M., Hardt, W. und Martinus, M. "Simulation hardwarespezifischer Komponenten von ECU- Software in der virtuellen Absicherung". In: *Energieeffiziente Antriebstechnologien: Hybridisierung – Downsizing – Software und IT*. Hrsg. von W. Siebenpfeiffer. Wiesbaden: Springer Fachmedien Wiesbaden, 2013, S. 196–200. ISBN: 978-3-658-00790-4. DOI: 10.1007/978-3-658-00790-4_28.

[67] Deloitte Development LLC. *Dloitte. Insights: Software is transforming the automotive world*. 2020. URL: https://www2.deloitte.com/us/en/insights/focus/future-of-mobility/pure-play-software-in-automotive-industry.html (besucht am 29. 04. 2021).

[68] Deutsches Forschungszentrum für Künstliche Intelligenz. *OpenDS: the flexible open source driving simulation*. URL: https://opends.dfki.de/ (besucht am 21. 04. 2021).

[69] Diebig, M. *Entwicklung einer Methodik zur simulationsbasierten Dimensionierung von Kfz-Bordnetzen*. 2016. DOI: 10.17877/DE290R-17350.

[70] Duffy, V. G. *Digital human modeling: First International Conference on Digital Human Modeling, ICDHM 2007, held as part of HCI International 2007, Beijing, China, July 22 - 27, 2007 ; proceedings*. Bd. 4561. Lecture Notes in Computer Science. Berlin und Heidelberg: Springer, 2007. ISBN: 9783540733218. DOI: 10.1007/978-3-540-73321-8.

[71] ESI Group. *Scilab Model Order Reduction*. URL: https://www.scilab.org/software/atoms/model-reduction (besucht am 27. 05. 2021).

[72] ESI Group. *Scilab/Xcos*. 2021. URL: https://www.scilab.org/software/xcos (besucht am 04. 12. 2022).

[73] Eckardt, S. "Neue Möglichkeiten der Individualisierung für eigenes Auto". In: *elektroniknet.de* (2020). URL: https://www.elektroniknet.de/automotive/infotainment/neue-moeglichkeiten-der-individualisierung-fuer-eigenes-auto.179952.html (besucht am 07. 12. 2022).

[74] Eclipse Foundation. *Capella MBSE Tool: Open Source Solution for Model-Based Systems Engineering*. URL: https://www.eclipse.org/capella/ (besucht am 31. 07. 2021).

[75] Eclipse Foundation. *Eclipse Epsilon*. URL: https://www.eclipse.org/epsilon/ (besucht am 05. 06. 2021).

[76] Eclipse Foundation. *Eclipse Triquetrum*. URL: https://projects.eclipse.org/projects/science.triquetrum (besucht am 26. 05. 2021).

[77] Eclipse Foundation. *SUMO - Simulation of Urban MObility*. URL: https://www.eclipse.org/sumo/ (besucht am 01. 11. 2022).

[78] Eigner, M., Roubanov, D. und Zafirov, R. *Modellbasierte virtuelle Produktentwicklung*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. ISBN: 978-3-662-43815-2. DOI: 10.1007/978-3-662-43816-9.

[79] Eldabi, T., Brailsford, S., Djanatliev, A., Kunc, M., Mustafee, N. und Osorio, A. F. "Hybrid Simulation Challenges and Opportunities: A Life-Cycle Approach". In: *2018 Winter Simulation Conference WSC 2018*, S. 1500–1514. DOI: 10.1109/WSC.2018.8632465.

[80] Elektro-Auto-Journal. *Anzahl Akkuzellen in E-Autos*. URL: https://e-auto-journal.de/anzahl-akkuzellen-in-e-autos/ (besucht am 19. 08. 2021).

[81] Elektronik-Kompendium. *OSI-Schichtenmodell*. URL: https://www.elektronik-kompendium.de/sites/kom/0301201.htm (besucht am 15. 08. 2021).

[82] Elsevier B.V. *Scopus - Document search*. URL: https://www.scopus.com/search/form.uri?display=basic#basic (besucht am 26. 05. 2021).

[83] Esfandiari, R. S. und Lu, B. *Modeling and analysis of dynamic systems*. Third edition. Boca Raton, London und New York: CRC Press Taylor & Francis Group, 2018. ISBN: 9781138726420.

[84] FRAUNHOFER-ALLIANZ BATTERIEN. *ENTWICKLUNGSPERSPEKTIVEN FÜR ZELLFORMATE VON LITHIUM-IONENBATTERIEN IN DER ELEKTROMOBILITÄT*. 2017. URL: https://www.batterien.fraunhofer.de/content/dam/batterien/de/documents/Allianz_Batterie_Zellformate_Studie.pdf (besucht am 19. 12. 2021).

[85] Faragó, D. "Model Checking and Model-Based Testing : Improving Their Feasibility by Lazy Techniques, Parallelization, and Other Optimizations". Dissertation. Karlsruhe: Karlsruher Institut für Technologie, 2016. URL: https://publikationen.bibliothek.kit.edu/1000059473/3891208.

[86] Felderer, M., Zech, P., Breu, R., Büchler, M. und Pretschner, A. "Model-based security testing: a taxonomy and systematic classification". In: *Software Testing, Verification and Reliability* 26.2 (2016), S. 119–148. ISSN: 09600833. DOI: 10.1002/stvr.1580.

[87] Fernandez, J., Kofman, E. und Bergero, F. "A parallel Quantized State System Solver for ODEs". In: *Journal of Parallel and Distributed Computing* 106 (2017), S. 14–30. ISSN: 07437315. DOI: 10.1016/j.jpdc.2017.02.011.

[88] Fleischmann, A., Oppl, S., Schmidt, W. und Stary, C. *Ganzheitliche Digitalisierung von Prozessen: Perspektivenwechsel – Design Thinking – Wertegeleitete Interaktion*. Wiesbaden: Springer Vieweg, 2018. ISBN: 978-3-658-22648-0.

[89] Franceschini, R., Bisgambiglia, P.-A., Touraille, L., Bisgambiglia, P. und Hill, D., Hrsg. *A survey of modelling and simulation software frameworks using Discrete Event System Specification: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany*. 2014. DOI: 10.4230/OASICS.ICCSW.2014.40.

[90] Fraunhofer FOKUS. *In-Car Infotainment: Solutions for easy to deploy and high-performance HMIs*. URL: https://www.fokus.fraunhofer.de/en/asct/in-car_infotainment (besucht am 24. 06. 2021).

[91] Fraunhofer-Institut für Produktionsanlagen und Konstruktionstechnik. *MBSE-Reifegradanalyse - Fraunhofer IPK*. URL: https://www.ipk.fraunhofer.de/de/kompetenzen/digital-engineering/modellbasiertes-systems-engineering/reifegradanalyse.html (besucht am 22. 07. 2021).

[92] Friedenthal, S., Moore, A. und Steiner, R. *Systems Modeling Language (SysML) Tutorial*. 2009. URL: https://www.omgsysml.org/INCOSE-OMGSysML-Tutorial-Final-090901.pdf (besucht am 28. 03. 2021).

[93] Fujimoto, R. M. *Parallel and distribution simulation systems*. New York: Wiley, 2000. ISBN: 0471183830.

[94] Fujimoto, R. *PARALLEL AND DISTRIBUTED SIMULATION*. Piscataway, NJ und Madison, Wis.: IEEE und Omnipress, 2015. ISBN: 9781467397414. (Besucht am 05. 04. 2021).

[95] Gebauer, D. "Ein modellbasiertes, graphisch notiertes, integriertes Verfahren zur Bewertung und zum Vergleich von Elektrik/Elektronik-Architekturen". Dissertation. Karlsruhe: Karlsruher Institut für Technologie, 2016. URL: https://d-nb.info/1121683487/34 (besucht am 09. 07. 2021).

[96] Geiger, A., Lenz, P. und Urtasun, R. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012. URL: http://www.cvlibs.net/datasets/kitti/ (besucht am 19. 12. 2021).

[97] Geiger, W. und Kotte, W. *Handbuch Qualität: Grundlagen und Elemente des Qualitätsmanagements: Systeme — Perspektiven*. 5., vollständig überarbeitete und erweiterte Auflage. Wiesbaden: Friedr. Vieweg & Sohn Verlag | GWV Fachverlage GmbH Wiesbaden, 2008. ISBN: 9783834894298. DOI: 10.1007/978-3-8348-9429-8.

[98] Genivi Alliance. *Future Vehicle EE and Software Architecture*. 2019. URL: https://www.grcc.vip/article-4760-.html (besucht am 07. 12. 2022).

[99] Gerster, M. "Daimler stärkt Software-Kompetenz: Mercedes erhält eigenes Betriebssystem". In: *Automobilwoche* (2020). URL: https://www.automobilwoche.de/article/20200108/BCONLINE/200109971/daimler-staerkt-software-kompetenz-mercedes-erhaelt-eigenes-betriebssystem (besucht am 07. 12. 2022).

[100] GitHub. *Conqat*. URL: https://github.com/vimaier/conqat/tree/master/org.conqat.engine.simulink (besucht am 05. 06. 2021).

[101] GitHub. *JSpice SPICE-inspired analog circuit simulator*. URL: https://github.com/knowm/jspice (besucht am 17. 12. 2021).

[102] GitHub. *Massif: MATLAB Simulink Integration Framework for Eclipse*. 2019-02-13. URL: https://viatra.github.io/massif/ (besucht am 05. 06. 2021).

[103] GitHub. *ScalableTestSuite*. URL: https://github.com/casella/ScalableTestSuite (besucht am 28. 05. 2021).

[104] Glöckler, M. *Simulation mechatronischer Systeme*. Wiesbaden: Springer Fachmedien Wiesbaden, 2018. ISBN: 978-3-658-20702-1. DOI: 10.1007/978-3-658-20703-8.

[105] Grave, R. "Die Fahrzeugarchitektur des autonomen Fahrens". In: *heise online* (2016). URL: https://www.heise.de/ratgeber/Die-Fahrzeugarchitektur-des-autonomen-Fahrens-3568991.html (besucht am 27. 11. 2021).

[106] Grösser, S. "Definition: System Dynamics". In: *Springer Fachmedien Wiesbaden GmbH* (2018). URL: https://wirtschaftslexikon.gabler.de/definition/system-dynamics-47445 (besucht am 13. 08. 2021).

[107] Guo, S. "An introduction to Surrogate modeling, Part I: fundamentals". In: *Towards Data Science* (2020). URL: https://towardsdatascience.com/an-introduction-to-surrogate-modeling-part-i-fundamentals-84697ce4d241 (besucht am 24. 07. 2021).

[108] Gustafson, J. L. "Reevaluating Amdahl's law". In: *Communications of the ACM* 31.5 (1988), S. 532–533. ISSN: 0001-0782. DOI: 10.1145/42411.42415.

[109] Hagedorn, N. "Sensoren – die vielen Sinne eines Autos". In: *Das Internetstudio GmbH* (2020). URL: https://greencarmagazine.de/sensoren-die-vielen-sinne-eines-autos/ (besucht am 10. 05. 2021).

[110] Hart, L. *Introduction To Model-Based System Engineering (MBSE) and SysML*. 2015. URL: https://www.incose.org/docs/default-source/delaware-valley/mbse-overview-incose-30-july-2015.pdf (besucht am 27. 03. 2021).

[111] Haskins, C. "4.6.1 A historical perspective of MBSE with a view to the future". In: *INCOSE International Symposium* 21.1 (2011), S. 493–509. ISSN: 2334-5837. DOI: 10.1002/j.2334-5837.2011.tb01220.x.

[112] Heegaard, P. E. "Speed-up techniques for simulation". In: *Teletronikk* (1995). URL: https://www.researchgate.net/publication/2830951_Comparison_Of_Speed-Up_Techniques_For_Simulation (besucht am 07. 12. 2022).

[113] Hejase, M., Kurt, A., Aldemir, T. und Ozguner, U. "Identification of Risk Significant Automotive Scenarios Under Hardware Failures". In: *Electronic Proceedings in Theoretical Computer Science* 269 (2018), S. 59–73. DOI: 10.4204/EPTCS.269.6.

[114] Hesse, W. und Mayr, H. C. "Modellierung in der Softwaretechnik: eine Bestandsaufnahme". In: *Informatik-Spektrum* 31.5 (2008), S. 377–393. ISSN: 0170-6012. DOI: 10.1007/s00287-008-0276-7.

[115] Hettig, C. F., Orth, P., Deppe, M., Pajenkamp, T., Granrath, C. und Andert, J. "Toolchain for architecture development, modeling and simulation of battery electric vehicles". In: *20. Internationales Stuttgarter Symposium*. Hrsg. von M. Bargende, H.-C. Reuss und A. Wagner. Proceedings Ser. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2020, S. 471–484. ISBN: 978-3-658-30995-4.

[116] Hill, M. D. "What is scalability?" In: *ACM SIGARCH Computer Architecture News* 18.4 (1990), S. 18–21. ISSN: 0163-5964. DOI: 10.1145/121973.121975.

[117] Hinton, H. M. "Under-Specification, Composition and Emergent Properties". In: (1997). DOI: 10.1145/283699.283743. URL: https://dl.acm.org/doi/pdf/10.1145/283699.283743 (besucht am 13. 08. 2021).

[118] Hoedt, J. "Fahrdynamikregelung fr fehlertolerante X-By-Wire-Antriebstopologien". Dissertation. Darmstadt: Technische Universitat Darmstadt, 2013. URL: https://tuprints.ulb.tu-darmstadt.de/3631/7/Dissertation_Hoedt.pdf (besucht am 24. 06. 2021).

[119] Hofacker, A. "Infotainmentsysteme mittels Simulation statt im Klimawindkanal testen". In: *Springer Professional* (2016). URL: https://www.springerprofessional.de/simulation---berechnung/infotainment/infotainmentsysteme-simulation-statt-klimawindkanaltest/10657792 (besucht am 07. 12. 2022).

[120] Huppertz, H. *Fahrgeschwindigkeit*. URL: https://www.kfz-tech.de/Biblio/Formelsammlung/Fahrgeschwindigkeit.htm (besucht am 07. 12. 2022).

[121] IBM. *Rhapsody*. 2021. URL: https://www.ibm.com/products/systems-design-rhapsody (besucht am 31. 07. 2021).

[122] INTO-CPS Association. *INTO-CPS Toolchain*. URL: https://into-cps.org/ (besucht am 27. 05. 2021).

[123] Imagine That Inc. *ExtendSim Discrete Rate Technology*. URL: https://extendsim.com/products/line/rate (besucht am 27. 05. 2021).

[124] Imagine That Inc. *ExtendSim System Requirements*. URL: https://extendsim.com/support/sysreq (besucht am 27. 05. 2021).

[125] Imagine That Inc. *ExtendSim (Version: 10)*. 2021. URL: https://extendsim.com/ (besucht am 04. 12. 2022).

[126] Institute of Electrical and Electronics Engineers. *IEEE 1516-2010: Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules*. 2010. URL: https://standards.ieee.org/standard/1516-2010.html (besucht am 14. 08. 2021).

[127] International Council on Systems Engineering. *Systems Engineering Vision 2020: INCOSE-TP-2004-004-02*. 2007. URL: http://www.ccose.org/media/upload/SEVision2020_20071003_v2_03.pdf (besucht am 27. 03. 2021).

[128] International Software Testing Qualifications Board. *ISTQB Definition: Testfall*. URL: https://glossary.istqb.org/en/term/test-case-3 (besucht am 04. 09. 2022).

[129] Internationale Organisation für Normung / Internationale Elektrotechnische Kommission / Institute of Electrical and Electronics Engineers. *ISO/IEC/IEEE 42010:2011(en): Systems and software engineering - Architecture description*. 2012. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:42010:en (besucht am 14. 09. 2022).

[130] Internationale Organisation für Normung / Internationale Elektrotechnische Kommission. *ISO/IEC 29500-2:2012*. URL: https://www.iso.org/standard/61796.html (besucht am 05. 06. 2021).

[131] Internationale Organisation für Normung / Publicly Available Specification. *ISO/PAS 21448:2019*. 2019. URL: https://www.iso.org/standard/70939.html (besucht am 29. 11. 2021).

[132] Internationale Organisation für Normung / Society of Automotive Engineers International. *ISO/SAE 21434:2021*. 2021. URL: https://www.iso.org/standard/70918.html (besucht am 29. 11. 2021).

[133] Internationale Organisation für Normung / Society of Automotive Engineers International. *ISO/SAE Road Vehicles - Cybersecurity Engineering*. 2020. URL: https://www.iso.org/standard/70918.html (besucht am 07. 12. 2022).

[134] Internationale Organisation für Normung. *ISO 26262-1:2018*. 2018. URL: https://www.iso.org/standard/68383.html (besucht am 29. 11. 2021).

[135] JMonkeyEngine.org. *JMonkeyEngine*. 2021-03-11. URL: https://jmonkeyengine.org/ (besucht am 21. 04. 2021).

[136] Jain, R. *The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation, and modeling*. [Nachdr.] New York: Wiley, 1991. ISBN: 0471503363.

[137] *Java Open Street Map (JOSM)*. 2021. URL: https://josm.openstreetmap.de/ (besucht am 07. 12. 2022).

[138] Jefferson, D. R. "Virtual time". In: *ACM Transactions on Programming Languages and Systems* 7.3 (1985), S. 404–425. ISSN: 0164-0925. DOI: 10.1145/3916.3988.

[139] Kahani, N., Bagherzadeh, M. und Cordy, J. R. “Synthesis of state machine models”. In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. Hrsg. von E. Syriani. ACM Digital Library. New York,NY,United States: Association for Computing Machinery, 2020, S. 274–284. ISBN: 9781450370196. DOI: 10.1145/3365438.3410936.

[140] Karner, M. “Co-Simulation of Cross-Domain Automotive Systems”. Dissertation. Graz: Technische Universität Graz, 2012. URL: https://diglib.tugraz.at/co-simulation-of-cross-domain-automotive-systems-2011 (besucht am 07. 12. 2022).

[141] Kaup, F., Gottschling, P. und Hausheer, D. “PowerPi: Measuring and modeling the power consumption of the Raspberry Pi”. In: *39th Annual IEEE Conference on Local Computer Networks*. 2014, S. 236–243. ISBN: 0742-1303. DOI: 10.1109/LCN.2014.6925777.

[142] Kawahara, K., Matsubara, Y. und Takada, H. “A Simulation Environment and preliminary evaluation for Automotive CAN-Ethernet AVB Networks”. In: *OMNET* (2014). URL: https://arxiv.org/pdf/1409.0998 (besucht am 23. 07. 2021).

[143] Kecher, C. und Salvanos, A. *UML 2.5: Das umfassende Handbuch*. 5., aktualisierte und erweiterte Auflage. Rheinwerk Computing. Bonn: Rheinwerk Computing, 2015. ISBN: 978-3-8362-2977-7.

[144] Keuchel, S., Zaleski, O. und Estorff, O. “Numerische Auslegung von Infotainment-Systemen in Fahrzeugen”. In: *Tagungsband - DAGA 2018*. Berlin: Deutsche Gesellschaft für Akustik e.V. (DEGA), 2018. ISBN: 9783939296133.

[145] Keysight Technologies. *Keysight Ptolemy Simulator*. URL: https://www.keysight.com/de/de/lib/resources/technical-specifications/keysight-ptolemy-simulator-2063172.html (besucht am 26. 05. 2021).

[146] Khlopov, D. und Mangold, M. “Automatic model reduction of differential algebraic systems by proper orthogonal decomposition”. In: *Computers & Chemical Engineering* 97 (2017), S. 104–113. ISSN: 00981354. DOI: 10.1016/j.compchemeng.2016.11.004.

[147] Knerr, T. *OSM2World*. URL: http://osm2world.org/ (besucht am 21. 08. 2021).

[148] Königs, S. F. “Konzeption und Realisierung einer Methode zur templategestützten Systementwicklung”. Dissertation. Berlin: Technische Universität Berlin, 2013. URL: https://d-nb.info/1029818762/34 (besucht am 07. 12. 2022).

[149] Kopacek, P. und Zauner, M. “Modellbildung und Simulation”. In: *Leitfaden der technischen Informatik und Kommunikationstechnik*. Hrsg. von P. Kopacek und M. Zauner. SpringerTechnik. Vienna: Springer Vienna, 2004, S. 209–212. ISBN: 978-3-211-00765-5. DOI: 10.1007/978-3-7091-0600-6_9.

[150] Kords, M. *Anzahl der Elektroautos in Deutschland von 2011 bis 2021*. 2021. URL: https://de.statista.com/statistik/daten/studie/265995/umfrage/anzahl-der-elektroautos-in-deutschland/ (besucht am 07. 12. 2022).

[151] Kounev, S., Lange, K.-D. und Kistowski, J. von. *Systems Benchmarking: For Scientists and Engineers*. 1st ed. 2020. Cham: Springer International Publishing, 2020. ISBN: 9783030417055. DOI: 10.1007/978-3-030-41705-5.

[152] Krempl, S. "Security by Design im Auto: Neue UN-Vorgaben für Cybersicherheit von Fahrzeugen". In: *heise online* (2020). URL: https://www.heise.de/news/Security-by-Design-Neue-UN-Vorgaben-fuer-Cybersicherheit-im-Auto-4767180.html (besucht am 07. 12. 2022).

[153] Krings, J. und Seyfferth, J. "Digital Auto Report". In: *Strategy&* (2019). URL: https://www.strategyand.pwc.com/de/en/industries/automotive/digital-auto-report.html (besucht am 12. 08. 2021).

[154] Kuhlmann, K., Schaub, D., Schünemann, M. und Haugwitz, C. "Methodische Auswahl des Antriebsstrangs hinsichtlich Motoren und Getriebe für ein Elektrofahrzeug". In: *Vocabularius Ex quo‹*. Hrsg. von B. Schnell und K. Grubmüller. Texte und Textgeschichte. Berlin, Boston: De Gruyter, 2018, S. 4–40. ISBN: 9783110924992. DOI: 10.1515/9783110924992-003.

[155] Kuhnert, F. und Sürmer, C. "Five trends transforming the Automotive Industry". In: *PwC Autofacts* (2018). URL: https://www.pwc.com/gx/en/industries/automotive/assets/pwc-five-trends-transforming-the-automotive-industry.pdf (besucht am 29. 04. 2021).

[156] Kumar, V., Rastogi, V. und Pathak, P. M. "Simulation for whole-body vibration to assess ride comfort of a low–medium speed railway vehicle". In: *SIMULATION* 93.3 (2017), S. 225–236. ISSN: 0037-5497. DOI: 10.1177/0037549716679254.

[157] Laroque, C. *Einführung in die Simulation*. 2011. URL: https://www.inf.tu-dresden.de/content/institutes/iai/ms/lehre/webdateien_simulation/5_DES.pdf (besucht am 09. 04. 2021).

[158] Laschewski-Grossbaier, R. und Mack, T. "Intelligent und sicher: Smarte Aktoren im Automobil". In: *elektroniknet.de* (2019). URL: https://www.elektroniknet.de/automotive/smarte-aktoren-im-automobil.162769.html (besucht am 10. 05. 2021).

[159] Law, A. M. *Simulation modeling and analysis*. 5. ed. McGraw-Hill series in industrial engineering and management science. New York, NY: McGraw-Hill Education, 2015. ISBN: 9780073401324.

[160] Lee, J. und Fishwick, P. A. “Dynamic exchange language layer for RUBE”. In: *Enabling Technologies for Simulation Science VI*. Hrsg. von A. F. Sisti und D. A. Trevisani. SPIE Proceedings. SPIE, 2002, S. 359–366. DOI: 10.1117/12.474932.

[161] Li, W., Mani Ramamurthy, Pieter J., M. und Teresa Hubscher-Younger. “Simulating a Multicore Scheduler of Real-Time Control Systems in Simulink”. In: *Risco Martin, D'Ambrogio et al. (Hg.) 2016 – Summer Computer Simulation Conference SCSC*. DOI: 10.22360/SummerSim.2016.SCSC.016.

[162] Lilja, D. J. *Measuring Computer Performance: A practitioner's guide*. 1. paperback version. Cambridge: Cambridge University Press, 2009. ISBN: 9780521646703. DOI: 10.1017/CBO9780511612398.

[163] Lima, P. *2018 Nissan Leaf battery real specs: PushEVs - Push Electric Vehicles Forward*. 2018. URL: https://pushevs.com/2018/01/29/2018-nissan-leaf-battery-real-specs/ (besucht am 10. 08. 2021).

[164] LinkedIn Corporation. *LinkedIn*. URL: https://www.linkedin.com/ (besucht am 18. 08. 2021).

[165] Liu, L. und Frey, G. *EFFICIENT SIMULATION OF HYBRID CONTROL SYSTEMS IN MODELICA/DYMOLA: 6th Vienna Conference on Mathematical Modelling, Vienna, February 11 - 13, 2009, Vienna University of Technology, Austria ; full papers CD volume*. Bd. 35. ARGESIM report. Vienna: ARGESIM Publ. House, 2009. ISBN: 9783901608353.

[166] Lock, A. und Zerfokwski, D. “Neue E/E-Architekturen mit Vehicle Computern bringen neue Chancen!” In: *all-electronics* (2020). URL: https://www.all-electronics.de/automotive-transportation/neue-ee-architekturen-mit-vehicle-computern-bringen-neue-chancen.html (besucht am 04. 05. 2021).

[167] Loper, M. L. *Modeling and Simulation in the Systems Engineering Life Cycle*. London, 2015. DOI: 10.1007/978-1-4471-5634-5.

[168] MLDesign Technologies Inc. *ML Designer (Version: 4)*. 2021. URL: https://www.mldesigner.com/ (besucht am 04. 12. 2022).

[169] MPI Forum. *MPI: A Message-Passing Interface Standard*. URL: https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf (besucht am 24. 07. 2021).

[170] Marinescu, C., Marinescu, R., Mihancea, P. F., Ratiu, D. und Wettel, R. “iPlasma: An Integrated Platform for Quality Assessment of Object-Oriented Design”. In: *21st IEEE International Conference on Software Maintenance (ICSM'05)*. 2005, S. 77–80. ISBN: 1063-6773.

[171] Master Cloud. *Scalability – Scale Out/In vs Scale Up/Down (Horizontal Scaling vs Vertical Scaling)*. URL: http://www.nitrix-reloaded.com/2016/10/01/scalability-scale-outin-vs-scale-updown-horizontal-scaling-vs-vertical-scaling/ (besucht am 29. 03. 2021).

[172] Math, R. *OpenDS Tutorial*. 2019. URL: https://umtl.cs.uni-saarland.de/lectures/AutomotiveUISeminar19/OpenDS_Tutorial.pdf (besucht am 21. 04. 2021).

[173] Mattern, F. und Mehl, H. "Diskrete Simulation - Prinzipien und Probleme der Effizienzsteigerung durch Parallelisierung". In: *Informatik Spektrum*. Bd. 12. 1989, S. 198–210. DOI: 10.1515/9783110924992-003.

[174] McAslan, S. "ECU: Hypervisor verwalten virtuelle Maschinen". In: *all-electronics* (2019). URL: https://www.all-electronics.de/ecu-hypervisor-verwalten-virtuelle-maschinen/ (besucht am 07. 12. 2022).

[175] McDonald, D. "Electric Vehicle Drive Simulation with MATLAB / Simulink". In: *Proceedings of the 2012 North-Central 2012* (2012). DOI: 10.9790/1676-1204014753.

[176] Mehl, H. *Methoden verteilter Simulation*. Programm Angewandte Informatik. Wiesbaden: Vieweg+Teubner Verlag, 1994. ISBN: 9783322906090. DOI: 10.1007/978-3-322-90609-0.

[177] Mehling, F. *Isolationsfehler in Bordnetzen/Ladeinfrastruktur erkennen und beherrschen*. 2013. URL: http://www.linowsee.de/media/files/fachtagung-windenergie-2013/6-fachtagung-17_09_-18_09_2013/Mehling.pdf (besucht am 24. 06. 2021).

[178] Meyer, J. "Modellgetriebene Skalierbarkeitsanalyse von selbst-adaptiven komponentenbasierten Softwaresystemen in der Cloud". Masterarbeit. Paderborn: Universität Paderborn, 2011. URL: https://www.hni.uni-paderborn.de/fileadmin/Fachgruppen/Softwaretechnik/Lehre/Ba_Ma_Arbeiten/masterarbeit.pdf (besucht am 31. 03. 2021).

[179] Michael, V. und Winner, H. *Stand der Technik und der Wissenschaft: Modellvalidierung im Anwendungsbereich der Fahrdynamiksimulation*. 2017. URL: http://tuprints.ulb.tu-darmstadt.de/6662/1/Viehof%2C%20Winner%20-%20Stand%20der%20Technik%20und%20Wissenschaft%20Validierung.pdf (besucht am 07. 12. 2022).

[180] Mikelsons, L. und Samlaus, R. "Towards Virtual Validation of ECU Software using FMI". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, 2017, S. 307–311. DOI: 10.3384/ecp17132307.

[181] Mirabilis Design Inc. *University License Agreement*. 2018. URL: https://www.mirabilisdesign.com/university-license/ (besucht am 27. 05. 2021).

[182] Mirabilis Design Inc. *Visualsim*. 2021. URL: https://www.mirabilisdesign.com/visualsim/ (besucht am 04. 12. 2022).

[183] Modelica Association. *Functional Mock-up Interface*. URL: https://fmi-standard.org/ (besucht am 14. 08. 2021).

[184] Modelica Association. *Modelica Language*. URL: https://modelica.org/modelicalanguage.html (besucht am 27. 05. 2021).

[185] Montecchi, L., Lollini, P. und Bondavalli, A. "A Template-Based Methodology for the Specification and Automated Composition of Performability Models". In: *IEEE Transactions on Reliability* 69.1 (2020), S. 293–309. ISSN: 0018-9529. DOI: 10.1109/TR.2019.2898351.

[186] Moradi, F., Nordvaller, P. und Ayani, R. "Simulation Model Composition using BOMs". In: *2006 Tenth IEEE International Symposium on Distributed Simulation and Real-Time Applications*. IEEE, 102006, S. 242–252. ISBN: 0-7695-2697-7. DOI: 10.1109/DS-RT.2006.34.

[187] Mustafee, N., Brailsford, S., Djanatliev, A., Eldabi, T., Kunc, M. und Tolk, A. "Purpose and benefits of hybrid simulation: Contributing to the convergence of its definition". In: *WSC'17*. Hrsg. von W. K. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer und E. H. Page. Piscataway, NJ: IEEE, 2017, S. 1631–1645. ISBN: 978-1-5386-3428-8. DOI: 10.1109/WSC.2017.8247903.

[188] Nanda, P., Swain, S. und Mohapatra, D. "Generation of Test Scenarios Using Activity Diagram". In: *Proceedings of SPIT-IEEE Colloquium and International Conference - Volume: 4*. 2008. URL: https://www.researchgate.net/publication/232957703_Generation_of_Test_Scenarios_Using_Activity_Diagram (besucht am 07. 12. 2022).

[189] Nattermann, R. und Anderl, R. "Approach for a Data-Managmenet-System and a Proceeding-Model for the Development of Adaptronic Systems". In: *Volume 3: Design and Manufacturing, Parts A and B*. ASMEDC, 2010, S. 379–387. ISBN: 978-0-7918-4427-4. DOI: 10.1115/IMECE2010-37828.

[190] Ngueagni Dongmo, B. K. "Implementierung einer GPS abhängigen Geschwindigkeitsregelung zur Simulation einer virtuellen Teststrecke". Bachelorarbeit. Kaiserslautern: Hochschule Kaiserslautern, 2020.

[191] Nicol, D. M. "The cost of conservative synchronization in parallel discrete event simulations". In: *Journal of the ACM* 40.2 (1993), S. 304–333. ISSN: 0004-5411. DOI: 10.1145/151261.151266.

[192] Nidzwetzki, J. K. *Entwicklung eines skalierbaren und verteilten Datenbanksystems: Auf Basis von Apache Cassandra und SECONDO*. 1. Aufl. 2016. BestMasters. Wiesbaden: Springer Fachmedien Wiesbaden, 2016. ISBN: 978-3-658-12443-4. DOI: 10.1007/978-3-658-12444-1.

[193] Noguchi, Y., Hosoi, T., Yoshioka, T., Yoshinaga, K. und Sakiyama, K. "Application of the Simulation Technology for the Automotive Wire Harness Development". In: *Fujikura Technical Review (45)*. 2016. URL: https://www.fujikura.co.jp/eng/rd/gihou/backnumber/pages/__icsFiles/afieldfile/2016/02/18/45e_08.pdf (besucht am 07. 12. 2022).

[194] Object Management Group. *Meta Object Facility (MOF) Core Specification*. 2019. URL: https://www.omg.org/spec/MOF/2.5.1/PDF (besucht am 28. 03. 2021).

[195] Object Management Group. *Unified Modeling Language, v2.5.1*. 2017. URL: https://www.omg.org/spec/UML/2.5.1/PDF (besucht am 09. 06. 2021).

[196] Oertel, M. *Architectures of High Performance Computing*. 2018. URL: https://assets.vector.com/cms/content/events/2018/VeCo18/presentations/VeCo18_Oertel_01.pdf (besucht am 28. 06. 2021).

[197] Ong, B. W. und Schroder, J. B. "Applications of time parallelization". In: *Computing and Visualization in Science* 23.1-4 (2020). ISSN: 1432-9360. DOI: 10.1007/s00791-020-00331-4.

[198] OpenStreetMap-Stiftung. *OpenStreetMap*. URL: https://www.openstreetmap.org/#map=6/51.330/10.453 (besucht am 10. 08. 2021).

[199] Organisation Internationale des Constructeurs d'Automobiles. *Future Certification of Automated Driving Systems: GRVA-02-27*. 2019. URL: https://unece.org/DAM/trans/doc/2019/wp29grva/GRVA-02-27e.pdf (besucht am 12. 08. 2021).

[200] *Overpass API*. URL: http://overpass-api.de/ (besucht am 10. 08. 2021).

[201] PEGASUS Projekt. *Abschlussbericht für das Gesamtprojekt*. 2020. URL: https://www.pegasusprojekt.de/files/tmpl/pdf/PEGASUS_Abschlussbericht_Gesamtprojekt.PDF (besucht am 28. 07. 2021).

[202] Pace, D. K. "Fidelity, Resolution, Accuracy, and Uncertainty". In: *Modeling and Simulation in the Systems Engineering Life Cycle*. Hrsg. von M. L. Loper. Simulation Foundations, Methods and Applications. London: Springer London, 2015, S. 29–37. ISBN: 978-1-4471-5633-8. DOI: 10.1007/978-1-4471-5634-5_3.

[203] Page, B. *Diskrete Simulation: Eine Einführung mit Modula-2*. Springer-Lehrbuch. Berlin und Heidelberg: Springer, 1991. ISBN: 9783642768620. DOI: 10.1007/978-3-642-76862-0.

[204] Palensky, P., Widl, E. und Elsheikh, A. "Simulating Cyber-Physical Energy Systems: Challenges, Tools and Methods". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44.3 (2014), S. 318–326. ISSN: 2168-2216. DOI: 10.1109/TSMCC.2013.2265739.

[205] Pallierer, R. und Schmelz, B. "Adaptive AUTOSAR für Hochleistungsrechner im Auto". In: *elektroniknet.de* (2017). URL: https://www.elektroniknet.de/automotive/software-tools/adaptive-autosar-fuer-hochleistungsrechner-im-auto.147883.html (besucht am 07. 12. 2022).

[206] Parente, F. R., Santonico, M., Zompanti, A., Benassai, M., Ferri, G., D'Amico, A. und Pennazza, G. "An Electronic System for the Contactless Reading of ECG Signals". In: *Sensors (Basel, Switzerland)* 17.11 (2017). DOI: 10.3390/s17112474.

[207] Pawlaszczyk, D., Straßburger, S. und Timm, I. J. *Skalierbare agentenbasierte Simulation: Werkzeuge und Techniken zur verteilten Ausführung agentenbasierter Modelle: Dissertation*. Ilmenau: Univ.-Verl. Ilmenau, 2009. ISBN: 9783939473596.

[208] Pawlik, V. *Wichtigste Kriterien beim Autokauf in Deutschland in den Jahren 2017 bis 2020*. 2021. URL: https://de.statista.com/statistik/daten/studie/171605/umfrage/wichtige-kriterien-beim-autokauf/ (besucht am 03. 11. 2021).

[209] Peng, J., He, H. und Feng, N. "Simulation Research on an Electric Vehicle Chassis System Based on a Collaborative Control System". In: *Energies* 6.1 (2013), S. 312–328. DOI: 10.3390/en6010312.

[210] Pietruszka, W. D. *MATLAB® und Simulink® in der Ingenieurpraxis: Modellbildung, Berechnung und Simulation*. 5., neu bearb. u. erw. Auflage 2021. Wiesbaden: Springer Fachmedien Wiesbaden, 2021. ISBN: 9783658297404.

[211] Platzer, A. und Quesel, J.-D. "KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description)". In: *Automated reasoning*. Hrsg. von A. Armando, P. Baumgartner und G. Dowek. Lecture notes in computer science Lecture notes in artificial intelligence. Berlin: Springer, 2008, S. 171–178. ISBN: 978-3-540-71070-7.

[212] Pohl, K. *Requirements engineering: Grundlagen, Prinzipien, Techniken*. 2., korrigierte Aufl. Heidelberg: dpunkt-Verl., 2008. ISBN: 3898645509.

[213] Pohle, I. *Embedded Programmierung*. Hrsg. von MicroConsult. URL: https://www.microconsult.de/832-0-Embedded-Programmierung-Fachwissen.html (besucht am 17. 06. 2021).

[214] Powersim Inc. *Control Design of a 100–kW PMSM Drive*. URL: https://powersimtech.com/resources/application-notes/control-design-of-a-100%E2%80%90kw-pmsm-drive/ (besucht am 06. 08. 2021).

[215] Ptolemaeus, C. *System design, modeling, and simulation: Using Ptolemy II*. 1. ed., version 1.02. Berkeley, Calif.: UC Berkeley EECS Dept, 2014. ISBN: 9781304421067.

[216] Rabe, M., Spieckermann, S. und Wenzel, S. *Verifikation und Validierung für die Simulation in Produktion und Logistik*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. ISBN: 978-3-540-35281-5. DOI: 10.1007/978-3-540-35282-2.

[217] Rahm, E. *Mehrrechner-Datenbanksysteme: Grundlagen der verteilten und parallelen Datenbankverarbeitung*. 1. Aufl. Bonn und Paris: Addison-Wesley, 1994. ISBN: 3893197028.

[218] Rauber, T. und Rünger, G. *Parallele Programmierung*. 2., neu bearb. und erw. Aufl. eXamen.press. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. ISBN: 978-3-540-46549-2. DOI: 10.1007/978-3-540-46548-5.

[219] Raue, S. "Systemorientierung in der modellbasierten modularen E/E-Architekturentwicklung". Dissertation. Tubingen: Eberhard Karls Universitat Tubingen, 2019. DOI: 10.15496/PUBLIKATION-29213. URL: https://publikationen.uni-tuebingen.de/xmlui/handle/10900/87828.

[220] Reder, S. T. "Compileroptimierung und parallele Code-Generierung für zeitkritische eingebettete Multiprozessorsysteme". Dissertation. Karlsruhe: Karlsruher Institut für Technologie, 2020. DOI: 10.5445/IR/1000124294.

[221] Reif, K. *Sensoren im Kraftfahrzeug*. Wiesbaden: Springer Fachmedien Wiesbaden, 2016. ISBN: 978-3-658-11210-3. DOI: 10.1007/978-3-658-11211-0.

[222] Reussner, R., Becker, S., Happe, J., Heinrich, R., Koziolek, A. und Koziolek, H., Hrsg. *Modeling and simulating software architectures: The Palladio approach*. Cambridge, Massachusetts und London, England: The MIT Press, 2016. ISBN: 9780262034760.

[223] Roddeck, W. *Grundprinzipien der Mechatronik: Modellbildung und Simulation mit Bondgraphen*. Wiesbaden: Springer Vieweg, 2013. ISBN: 9783834821942. DOI: 10.1007/978-3-8348-2194-2.

[224] Rueda, A. R. "A Novel Optimization Methodology of Modular Wiring Harnesses in Modern Vehicles: Weight Reduction and Safe Operation". Dissertation. Terrassa: Universitat Politècnica de Catalunya, 2017. URL: https://upcommons.upc.edu/handle/2117/106293?locale-attribute=en (besucht am 07. 12. 2022).

[225] Ruf, F. "Auslegung und Topologieoptimierung von spannungsstabilen Energiebordnetzen". Dissertation. München: Technische Universität München, 2015. URL: https://mediatum.ub.tum.de/1220402 (besucht am 07. 12. 2022).

[226] Rugh, J., Bharathan, D. und Chaney, L. *Predicting Human Thermal Comfort in Automobiles*. 2005. URL: https://www.nrel.gov/docs/fy05osti/38236.pdf (besucht am 25. 06. 2021).

[227] Saboora, K. und Man-Hoe, K. “Thermal Comfort in the Passenger Compartment Using a 3-D Numerical Analysis and Comparison with Fanger’s Comfort Models”. In: *Energies* 13.3 (2020), S. 690. DOI: 10.3390/en13030690.

[228] Sagdeo, V., Hrsg. *The complete Verilog book*. Boston: Kluwer Academic Publ, 1998. ISBN: 0-7923-8188-2. DOI: 10.1007/b116655.

[229] Sanghavi, A. “What is formal verification?” In: *EE Times Asia* (2010). URL: https://archive.eetasia.com/www.eetasia.com/STATIC/PDF/201005/EEOL_2010MAY21_EDA_TA_01.pdf?SOURCES=DOWNLOAD (besucht am 25. 07. 2021).

[230] Sanz, V., Urquia, A., Cellier, F. E. und Dormido, S. “Hybrid system modeling using the SIMANLib and ARENALib Modelica libraries”. In: *Simulation Modelling Practice and Theory* 37 (2013), S. 1–17. ISSN: 1569190X. DOI: 10.1016/j.simpat.2013.05.005.

[231] Sargent, R. “Verification and validation of simulation models”. In: *Proceedings of Winter Simulation Conference*. IEEE, 1994, S. 77–87. ISBN: 0-7803-2109-X. DOI: 10.1109/WSC.1994.717077.

[232] Schäfer, C. und Denkelmann, R. “Zukunftsfähige E/E-Architektur Leistungsversorgung und Datenübertragung für autonomes Fahren”. In: *ATZelektronik* 13.6 (2018), S. 16–21. ISSN: 1862-1791.

[233] Schäuffele, J. und Zurawka, T. *Automotive Software Engineering: Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen*. 6. Auflage. ATZ / MTZ-Fachbuch. Wiesbaden: Springer Vieweg, 2016. ISBN: 978-3-658-11814-3. DOI: 10.1007/978-3-658-11815-0.

[234] Schmitt, R. und Pfeifer, T. *Qualitätsmanagement: Strategien - Methoden - Techniken*. 5., aktualisierte Auflage. München: Hanser, 2015. ISBN: 9783446440821. DOI: 10.3139/9783446440821.

[235] Schnellbach, A. “Fail-Operational Automotive Systems”. Dissertation. Graz: Graz University of Technology, 2016. URL: https://diglib.tugraz.at/download.php?id=5aa2484fb4bcb&location=browse (besucht am 07. 12. 2022).

[236] Shankar, D. “Conceptual Performance Analysis Eliminates Distributed System Design Risks”. In: *SAE Technical Paper Series*. SAE Technical Paper Series. SAE International400 Commonwealth Drive, Warrendale, PA, United States, 2007. DOI: 10.4271/2007-01-1280.

[237] Simonot-Lion, F. und Song, Y.-Q. “Design and Validation Process of In-Vehicle Embedded Electronic Systems”. In: *Embedded Systems Handbook*. Hrsg. von R. Zurawski. CRC Press, 2005, S. 995–1018. ISBN: 9780429122392. DOI: 10.1201/9781420038163-47. (Besucht am 25. 06. 2021).

[238] Simulation Interoperability Standards Organization. *SISO-STD-003-2006: Base Object Model (BOM) Standard Spezifikation*. 2006. URL: https://www.sisostds.org/StandardsActivities/SupportGroups/BOMPSG-BaseObjectModel.aspx (besucht am 27. 07. 2021).

[239] Sjöstedt, C.-J. *Modeling and Simulation of Physical Systems in a Mechatronic Context: Dissertation*. Stockholm: KTH School of Industrial Engineering, 2009. ISBN: 978-91-7415-361-3.

[240] Society of Automotive Engineers. *SAE J3016: Automated-Driving Graphic*. URL: https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic (besucht am 18. 08. 2021).

[241] Stachowiak, H. *Allgemeine Modelltheorie*. Wien: Springer, 1973. ISBN: 3211811060.

[242] Städter, P., Schälte, Y., Schmiester, L., Hasenauer, J. und Stapor, P. L. "Benchmarking of numerical integration methods for ODE models of biological systems". In: *Scientific reports* 11.1 (2021), S. 2696. DOI: 10.1038/s41598-021-82196-2.

[243] Standard Performance Evaluation Corporation. *Glossar*. URL: https://www.spec.org/spec/glossary/ (besucht am 31. 03. 2021).

[244] Stein, S. *Begriff und Definition Emergenz*. URL: https://emergenz.hpfsc.de/html/node8.html (besucht am 26. 09. 2021).

[245] Steinbrink, C., Lehnhoff, S., Rohjans, S., Strasser, T. I., Widl, E., Moyo, C., Lauss, G., Lehfuss, F., Faschang, M., Palensky, P., van der Meer, A. A., Heussen, K., Gehrke, O., Guillo-Sansano, E., Syed, M. H., Emhemed, A., Brandl, R., Nguyen, V. H., Khavari, A., Tran, Q. T., Kotsampopoulos, P., Hatziargyriou, N., Akroud, N., Rikos, E. und Degefa, M. Z. "Simulation-Based Validation of Smart Grids – Status Quo and Future Research Trends". In: *Industrial applications of holonic and multi-agent systems*. Hrsg. von V. Mařík, W. Wahlster, T. Strasser und P. Kadera. Bd. 10444. Lecture notes in computer science Lecture notes in artificial intelligence. Cham: Springer, 2017, S. 171–185. ISBN: 978-3-319-64634-3. DOI: 10.1007/978-3-319-64635-0_13.

[246] Streichert, T. und Traub, M. *Elektrik/Elektronik-Architekturen im Kraftfahrzeug: Modellierung und Bewertung von Echtzeitsystemen*. 1. Aufl. VDI-Buch. s.l.: Springer-Verlag, 2012. ISBN: 3642254772.

[247] Su, Z., Wang, D., Yang, Y., Jiang, Y., Chang, W., Fang, L., Li, W. und Sun, J. "Code Synthesis for Dataflow Based Embedded Software Design". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2021), S. 1. ISSN: 0278-0070. DOI: 10.1109/TCAD.2021.3055487.

[248] Surmund, S., Walkowiak, C., Nguyen, D. und Beck, M. "Neue Szenarien für autonome Fahrsysteme". In: *ATZextra* 23.S2 (2018), S. 42–45. ISSN: 2195-1462. DOI: 10.1007/s35778-018-0022-y.

[249] Sutter, H. *The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software*. 2005. URL: http://www.gotw.ca/publications/concurrency-ddj.htm (besucht am 30. 03. 2021).

[250] Synopsis. *Coverity Scan - Static Analysis of Ptolemy II*. URL: https://scan.coverity.com/projects/ptolemy-ii (besucht am 26. 05. 2021).

[251] Synopsis. *Coverity Scan - Static Analysis of Scilab*. URL: https://scan.coverity.com/projects/881 (besucht am 27. 05. 2021).

[252] T2informatik GmbH. *Was ist ein Verhaltensdiagramm?* URL: https://t2informatik.de/wissen-kompakt/verhaltensdiagramm/ (besucht am 28. 03. 2021).

[253] Technische Hochschule Mittelhessen. *Systemmodell: Definition*. URL: https://wiki.thm.de/Systemmodell (besucht am 03. 08. 2021).

[254] Techopedia. *Service-Oriented Architecture Security (SOA Security): Definition*. URL: https://www.techopedia.com/definition/29809/service-oriented-architecture-security-soa-security (besucht am 25. 06. 2021).

[255] TelematikWissen. *Predictive Maintenance: ein neues Betätigungsfeld der Telematik?* URL: https://telematikwissen.de/predictive-maintenance/ (besucht am 24. 03. 2021).

[256] Terstegen, S. und Duckwitz, S. *Simulationsbasierte Projektplanung*. Hrsg. von derobino. 2014. URL: https://iaw-aachen.de/files/iaw/handreichungen/handreichung_2014-6_simulation.pdf (besucht am 25. 06. 2021).

[257] The Anylogic Company. *AnyLogic: Softwarewerkzeuge für Simulationsmodellierung & geschäftliche Lösungen*. 2021. URL: https://www.anylogic.de/#tab5 (besucht am 27. 05. 2021).

[258] The Anylogic Company. *AnyLogic (Version: 8.7)*. 2020. URL: https://www.anylogic.de/ (besucht am 04. 12. 2022).

[259] The Mathworks Inc. *5G R&D at Huawei: An Insider Look*. 2017. URL: https://www.mathworks.com/content/dam/mathworks/tag-team/Objects/h/80861v00_Huawei_QA.pdf (besucht am 18. 08. 2021).

[260] The Mathworks Inc. *AC7 - Brushless DC Motor Drive During Speed Regulation*. 2021-07-10. URL: https://de.mathworks.com/help/physmod/sps/ug/ac7-brushless-dc-motor-drive-during-speed-regulation.html (besucht am 10. 07. 2021).

[261] The Mathworks Inc. *AUTOSAR Support in MATLAB and Simulink: Automotive Industry Standards*. URL: https://de.mathworks.com/solutions/automotive/standards/autosar.html (besucht am 31. 08. 2021).

[262] The Mathworks Inc. *Evaluate Transitions: MATLAB & Simulink - MathWorks Deutschland*. 2021. URL: https://de.mathworks.com/help/stateflow/ug/evaluate-transitions.html (besucht am 31. 05. 2021).

[263] The Mathworks Inc. *How can I speed up simulation of my Simulink model? MATLAB Answers - MATLAB Central*. 2013. URL: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2179 (besucht am 31. 05. 2021).

[264] The Mathworks Inc. *MATLAB Engine API for Java: MATLAB & Simulink - MathWorks France*. URL: https://fr.mathworks.com/help/matlab/matlab_external/get-started-with-matlab-engine-api-for-java.html (besucht am 20. 04. 2021).

[265] The Mathworks Inc. *Matlab/Simulink (Version: R2020.b)*. 2020. URL: https://de.mathworks.com/products/simulink.html (besucht am 04. 12. 2022).

[266] The Mathworks Inc. *Parallel Computing Toolbox*. URL: https://de.mathworks.com/products/parallel-computing.html (besucht am 25. 09. 2021).

[267] The Mathworks Inc. *Programmatic Modeling Basics: MATLAB & Simulink - MathWorks Deutschland*. URL: https://de.mathworks.com/help/simulink/ug/approach-modeling-programmatically.html#d123e586 (besucht am 07. 06. 2021).

[268] The Mathworks Inc. *Simscape Limitations*. URL: https://fr.mathworks.com/help/physmod/simscape/ug/limitations.html (besucht am 23. 04. 2021).

[269] The Mathworks Inc. *Simulink User's Guide*. URL: https://de.mathworks.com/help/pdf_doc/simulink/simulink_ug.pdf (besucht am 31. 05. 2021).

[270] The Mathworks Inc. *Stateflow User's Guide*. URL: https://de.mathworks.com/help/stateflow/ (besucht am 07. 12. 2022).

[271] The Mathworks Inc. *Three-phase permanent magnet synchronous machine with sinusoidal or trapezoidal back electromotive force, or five-phase permanent magnet synchronous machine with sinusoidal back electromotive force: Simulink - MathWorks Deutschland*. URL: https://de.mathworks.com/help/physmod/sps/powersys/ref/permanentmagnetsynchronousmachine.html (besucht am 06. 08. 2021).

[272] The Mathworks Inc. *Workspace Variables and MAT-Files*. URL: https://www.mathworks.com/help/matlab/workspace.html (besucht am 11. 09. 2022).

[273] Tischer, M. "AUTOSAR Adaptive: Das Rechenzentrum im Fahrzeug". In: *elektronikautomotive.de* (2018). URL: https://assets.vector.com/cms/content/know-how/_technical-articles/AUTOSAR/AUTOSAR_Adaptive_ElektronikAutomotive_201809_PressArticle_DE.pdf (besucht am 28. 06. 2021).

[274] Tjonnaas, J. und Johansen, T. A. "Adaptive Optimizing Dynamic Control Allocation Algorithm for Yaw Stabilization of an Automotive Vehicle using Brakes". In: *2006 14th Mediterranean Conference on Control and Automation*. IEEE, 62006, S. 1–6. ISBN: 0-9786720-1-1. DOI: 10.1109/MED.2006.328748.

[275] Trampe, H. *Modellierung und Simulation: Übersicht*. 2013. URL: https://hps.vi4io.org/_media/teaching/wintersemester_2012_2013/ms-1213-trampe-modellierungsimulationuebersicht-hausarbeit.pdf (besucht am 16. 12. 2021).

[276] Tschirner, N. C. "Rahmenwerk zur Integration des modellbasierten Systems Engineering in die Produktentstehung mechatronischer Systeme". Dissertation. Paderborn: Universität Paderborn, 2016. URL: https://digital.ub.uni-paderborn.de/urn/urn:nbn:de:hbz:466:2-27332 (besucht am 05. 05. 2021).

[277] Tschudi, Y. *Künstliche Intelligenz im Fahrzeug: Marktanalyse und Ausblick*. 2019. URL: https://www.all-electronics.de/kuenstliche-intelligenz-im-fahrzeug-marktanalyse-und-ausblick/ (besucht am 07. 12. 2022).

[278] Tulpule, P., Rezaeian, A., Karumanchi, A. und Midlam-Mohler, S. "Model Based Design (MBD) and Hardware In the Loop (HIL) validation: Curriculum development". In: *2017 American Control Conference (ACC)*. IEEE, 5/24/2017 - 5/26/2017, S. 5361–5366. ISBN: 978-1-5090-5992-8. DOI: 10.23919/ACC.2017.7963788.

[279] UC Berkeley EECS Dept. *Ptolemy Classic*. URL: https://ptolemy.berkeley.edu/ptolemyclassic/index.htm (besucht am 26. 05. 2021).

[280] UC Berkeley EECS Dept. *Ptolemy II (Version: 11.0.1)*. 2018. URL: https://ptolemy.berkeley.edu/ptolemyII/ptII11.0/index.htm (besucht am 04. 12. 2022).

[281] UC Berkeley EECS Dept. *The Ptolemy Project*. URL: https://ptolemy.berkeley.edu/ (besucht am 26. 05. 2021).

[282] Un-Noor, F., Padmanaban, S., Mihet-Popa, L., Mollah, M. und Hossain, E. "A Comprehensive Study of Key Electric Vehicle (EV) Components, Technologies, Challenges, Impacts, and Future Direction of Development". In: *Energies* 10.8 (2017), S. 1217. DOI: 10.3390/en10081217.

[283] United Nations Economic Commission for Europe. *Proposal for amendments to ECE/TRANS/WP.29/GRVA/2020/3*. 2020. URL: http://www.unece.org/fileadmin/DAM/trans/doc/2020/wp29grva/ECE-TRANS-WP29-2020-079-Revised.pdf (besucht am 24. 03. 2021).

[284] United Nations Economic Commission for Europe. *UNECE Regel 131 - Advanced Emergency Braking System (AEBS): EUR-Lex - 42014X0719(01) - EN - EUR-Lex*. URL: https://eur-lex.europa.eu/eli/reg/2014/131/oj (besucht am 28. 09. 2021).

[285] Upstream Security Ltd. *Global Automotive Cybersecurity Report 1021*. 2020. URL: https://info.upstream.auto/hubfs/Security_Report/Security_Report_2021/Upstream_Security-Global_Automotive_Cybersecurity_Report_2021.pdf?_hsmi=101240621&_hsenc=p2ANqtz-9wQ7--I_haaPkzVomThYNiCKsB4EAzYHQ3KuA_9FQbwVck2lzLRlKQ6iQJiJmHMrB_eo3jobYe4NKHOQPxmpTNmiC_fQ (besucht am 12. 08. 2021).

[286] Vangheluwe, H. *The Discrete EVent System specication (DEVS) formalism*. Hrsg. von McGill School of Computer Science. 2005. URL: https://www.cs.mcgill.ca/~hv/classes/MS.00.Fall/lecture.DEVS/devs.pdf (besucht am 07. 12. 2022).

[287] Vasilache, S. und Tanaka, J. "Synthesis of State Machines from Multiple Interrelated Scenarios Using Dependency Diagrams". In: *Journal of Systemics, Cybernetics and Informatics*. Hrsg. von International Institute of Informatics and Cybernetics, IIIC. Bd. 3. 2004. URL: https://www.iplab.cs.tsukuba.ac.jp/paper/journal/si_jsci2006.pdf (besucht am 07. 12. 2022).

[288] Vector Informatik GmbH. *CANoe*. URL: https://www.vector.com/int/en/products/products-a-z/software/canoe/ (besucht am 19. 08. 2021).

[289] Vector Informatik GmbH. *DYNA4*. URL: https://www.vector.com/int/en/products/products-a-z/software/dyna4/ (besucht am 19. 08. 2021).

[290] Vector Informatik GmbH. *PREEvision Homepage*. URL: https://www.vector.com/de/de/produkte/produkte-a-z/software/preevision (besucht am 07. 12. 2022).

[291] Vector Informatik GmbH. *PREEvision Manual_9.5.6_EN*. 2020.

[292] Vector Informatik GmbH. *PREEvision Systemanforderungen*. URL: https://assets.vector.com/cms/content/products/preevision/docs/system-requirements/PREEvision_SystemRequirements_9.5.x_EN.pdf (besucht am 25. 11. 2021).

[293] Verband der Automobilindustrie. *VDA Elektromobilität*. URL: https://www.vda.de/de/themen/innovation-und-technik/elektromobilitaet/startseite-elektromobilitaet.html (besucht am 29. 09. 2021).

[294] Verband der Elektrotechnik, Elektronik und Informationstechnik e. V. *DIN VDE 0298-4:2013-06: Verwendung von Kabeln und isolierten Leitungen für Starkstromanlagen*. URL: https://www.vde-verlag.de/normen/0298016/din-vde-0298-4-vde-0298-4-2013-06.html (besucht am 23. 09. 2021).

[295] Verein Deutscher Ingenieure / Verband der Elektrotechnik, Elektronik und Informationstechnik e. V. *VDI/VDE 2206: Entwicklung cyber-physischer mechatronischer Systeme (CPMS)*. URL: https://www.vdi.de/fileadmin/pages/vdi_de/redakteure/richtlinien/dateien/V-Modell_nach_VDIVDE_2206.pdf (besucht am 05. 05. 2021).

[296] VisualVM. *Documentation*. URL: https://visualvm.github.io/documentation.html (besucht am 21. 04. 2021).

[297] Wagenbrenner, J. K. *Zonal EE Architecture:Towards a Fully Automotive Ethernet–Based Vehicle Infrastructure*. Hrsg. von Visteon. 2019. URL: https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/other/eipatd-presentations/2019/D1-04_KLAUS-Zonal_EE_Architecture.pdf (besucht am 04. 05. 2021).

[298] Wang, J., Huang, G. und Sun, Y. "Optimal Control of Complex HVAC Systems: Event-driven or Time-driven Optimization?" In: *CLIMA 2016 - proceedings of the 12th REHVA World Congress*. Hrsg. von P. K. Heiselberg. 2016. URL: https://www.semanticscholar.org/paper/Optimal-Control-of-Complex-HVAC-Systems-%3A-or-Wang-Huang/4c868a1a8bfecbd933f340d986aa14b0f85fe146 (besucht am 06. 04. 2021).

[299] Wang, W. und Yang, Y. "The Speedup of Discrete Event Simulations by Utilizing CPU Caching". In: *Discrete Event Simulations - Development and Applications*. Hrsg. von E. W. C. Lim. InTech, 2012. ISBN: 978-953-51-0741-5. DOI: 10.5772/50397.

[300] Weilkiens, T. *Popular SysML/MBSE Modeling Tools*. URL: https://mbse4u.com/sysml-tools/ (besucht am 31. 07. 2021).

[301] Weiss, G., Schleiss, P., Drabek, C., Ruiz, A. und Radermacher, A. "Safe Adaptation for Reliable and Energy-Efficient E/E Architectures". In: *Comprehensive Energy Management - Safe Adaptation, Predictive Control and Thermal Management*. Hrsg. von D. Watzenig und B. Brandstätter. SpringerBriefs in Applied Sciences and Technology Ser. Cham: Springer International Publishing, 2017, S. 1–18. ISBN: 978-3-319-57445-5. DOI: 10.1007/978-3-319-57445-5_1.

[302] Weiss, P., Weichslgartner, A., Reimann, F. und Steinhorst, S. "Fail-Operational Automotive Software Design Using Agent-Based Graceful Degradation". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2020, S. 1169–1174. ISBN: 1558-1101. DOI: 10.23919/DATE48585.2020.9116322.

[303] Winner, H., Hakuli, S., Lotz, F. und Singer, C. *Handbook of Driver Assistance Systems*. Cham: Springer International Publishing, 2016. ISBN: 978-3-319-12351-6. DOI: 10.1007/978-3-319-12352-3.

[304] Winner, H., Wachenfeld, W. und Junietz, P. *Safety Assurance for Highly Automated Driving: The PEGASUS Approach*. 2016. URL: https://www.pegasusprojekt.de/files/tmpl/pdf/Automation%20Vehicle%20Symposium%202016%20Folien.pdf (besucht am 07. 12. 2022).

[305] Wintrich, A. "IGBTs in elektrischen Antrieben mit Kabellast richtig schalten". In: *ELEKTRONIKPRAXIS* (2017). URL: https://www.analog-praxis.de/igbts-in-elektrischen-antrieben-mit-kabellast-richtig-schalten-a-942026/ (besucht am 08. 09. 2021).

[306] World Wide Web Consortium, W3C. *SRML - Simulation Reference Markup Language*. 2002. URL: https://www.w3.org/TR/2002/NOTE-SRML-20021218/ (besucht am 27. 07. 2021).

[307] Zehrer, G. "Isolationsüberwachung an DC-Ladestationen". In: *elektrotechnik* (2018). URL: https://www.elektrotechnik.vogel.de/isolationsueberwachung-an-dc-ladestationen-a-783361/ (besucht am 24. 06. 2021).

[308] Zeigler, B. P. und Sarjoughian, H. S. *Guide to Modeling and Simulation of Systems of Systems*. 2nd ed. 2017. Simulation Foundations, Methods and Applications. Cham: Springer International Publishing, 2017. ISBN: 978-3-319-64133-1. DOI: 10.1007/978-3-319-64134-8.

[309] Zerfowski, D. und Buttle, D. "Paradigmenwechsel im Automotive-Software-Markt". In: *ATZ - Automobiltechnische Zeitschrift* 121.9 (2019), S. 28–35. ISSN: 0001-2785. DOI: 10.1007/s35148-019-0095-y.

[310] Zhang, C., Li, K., Mcloone, S. und Yang, Z. "Battery modelling methods for electric vehicles - A review". In: *2014 European Control Conference (ECC)*. 2014, S. 2673–2678. DOI: 10.1109/ECC.2014.6862541.

[311] Zheng, X. und Julien, C. "Verification and Validation in Cyber Physical Systems: Research Challenges and a Way Forward". In: *2015 IEEE/ACM 1st International Workshop on Software Engineering for Smart Cyber-Physical Systems*. IEEE, 52015, S. 15–18. ISBN: 978-1-4673-7088-2. DOI: 10.1109/SEsCPS.2015.11.

[312] Zimmer, D. "An Application of Sol on Variable-Structure Systems with Higher Index". In: *Proceedings of the 7 International Modelica Conference Como, Italy*. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, 2009, S. 225–232. DOI: 10.3384/ecp09430026.

[313] Zimmermann, W. und Schmidgall, R. *Bussysteme in der Fahrzeugtechnik*. Wiesbaden: Springer Fachmedien Wiesbaden, 2014. ISBN: 978-3-658-02418-5. DOI: 10.1007/978-3-658-02419-2.

[314] Züricher Hochschule für Angewandte Wissenschaften. *Von der Mechatronik zu Cyber-physikalischen Systemen*. URL: https://blog.zhaw.ch/industrie4null/2017/02/06/von-der-mechatronik-zu-cyber-physikalischen-systemen/ (besucht am 22. 03. 2021).

# Betreute studentische Abschlussarbeiten

[S1] Hu, Y. "Aufbau eines Simulationsmodells für elektrisch betriebene Kraftfahrzeuge in Ptolemy II". Bachelorarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2020.

[S2] Kozarev, A. "Entwicklung einer Testbench sowie des generischen Verhaltensmodells einer ECU zur Evaluation von Simulatoren". Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2019.

[S3] Lam, J.-S. "Kopplung der Simulatoren SUMO und Ptolemy II zur Bereitstellung von Verkehrsdaten als Stimuli für eine E/E Architektur-Simulation". Bachelorarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2020.

[S4] Mahl, M. "Aufbau und Evaluation des Simulationsmodells einer Multicore AUTOSAR ECU für E/E-Architektursimulationen". Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2021.

[S5] Miao, Y. "Konzeption einer Hardwarearchitektur für zukünftige, sichere und skalierbare E/E Architekturen". Bachelorarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2020.

[S6] Schwarz, J. "Szenarienbasierte Entwicklung einer virtuellen und reaktivenTestumgebung zur Ausführung einer domänenübergreifenden Simulation von automobilen E/E Architekturen". Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2020.

[S7] Tsai, T. H. "Coupling of Simulators OpenDS and Ptolemy II for Preparation of Camera Data in E/E Architecture". Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2020.

[S8] Yang, Y. "Optimization and speedup of a simulator using the example of advanced driver assistance functions". Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2019.

# Eigene Publikationen


[P1] Bucher, H., Neubauer, K. und Becker, J. "Automated Assessment of E/E-Architecture Variants Using an Integrated Model- and Simulation-Based Approach". In: *SAE Technical Paper Series*. SAE Technical Paper Series. SAE International400 Commonwealth Drive, Warrendale, PA, United States, 2019. DOI: 10.4271/2019-01-0111.

[P2] Neubauer, K., Bucher, H., Haas, B. und Becker, J. *Model-Based Development and Simulative Verification of Logical Vehicle Functions Using Executable UN/ECE Regulations*. ACM Digital Library. San Diego,CA,United States: Society for Computer Simulation International, 2020. ISBN: 9781713814290.

[P3] Neubauer, K., Masing, L., Mahl, M., Becker, J., Kramer, M. und Reichmann, C. "Template-Driven and Hardware-Centric Cross-Domain E/E Architecture Simulation". In: *Proceedings of the 2021 32nd International Workshop on Rapid System Prototyping: Shortening the path from specification to prototype*. Piscataway, NJ: IEEE, 2021, S. 29–35. ISBN: 978-1-6654-6956-2. DOI: 10.1109/RSP53691.2021.9806231.

[P4] Neubauer, K., Rumez, M., Tremmel, H., Hoppe, A., Kriesten, R., Nenninger, P., Sax, E. und Becker, J. "Virtual Verification of E/E Architectures for Secure Automated Driving Functions". In: *2021 IEEE International Symposium on Systems Engineering (ISSE)*. Piscataway, NJ: IEEE, 2021, S. 1–8. ISBN: 978-1-6654-3168-2. DOI: 10.1109/ISSE51541.2021.9582552.