Modellierung und Export von Multicore-Eigenschaften für Simulationen während der Steuergeräteentwicklung für Fahrzeuge

Master-Thesis von

Tsvetan Spasov

an der Fakultät für Informatik
Institut für Programmstrukturen und Datenorganisation (IPD)

Erstgutachter:	Prof. Dr. Ralf H. Reussner
Zweitgutachter:	Prof. Dr.-Ing. Anne Koziolek
Betreuender Mitarbeiter:	Dipl.-Ing. Daniel Zimmermann
Zweiter betreuender Mitarbeiter:	Dr. Max Kramer, M.Sc. Martin Schend

15. Juni 2018 – 30. Januar 2019

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Änderungen entnommen wurde.

PLACE, DATE

. . . . . .

(Tsvetan Spasov)

Zusammenfassung

Derzeit führen die sehr hohen Anforderungen in nahezu allen Bereichen des Fahrzeugs zu einer immer komplexer werdenden Steuergerätearchitektur. Dies betrifft unter anderem sowohl sicherheitskritische wie auch Fahrerassistenz- und Komfortsysteme. Dies führt zu einer stetig steigenden Anzahl der Funktionalitäten, die auf einem Steuergerät ausgeführt werden müssen. Damit steigt der Bedarf an Rechenleistung der dafür eingesetzten elektronischen Steuereinheiten. Um diese Anforderungen zu bewältigen, werden bei der Entwicklung sicherheitskritischer, echtzeitfähiger, eingebetteter Systeme immer häufiger Steuergeräte mit Prozessoren mit mehreren Kernen eingesetzt. Bei diesem Ansatz entstehen notwendigerweise Echtzeitanforderungen, die im Idealfall bereits in der frühen Entwicklungsphase berücksichtigt werden sollten. Die Erfüllung dieser Anforderungen werden oft erst spät während des Entwicklungsprozesses überprüft.

Um Systeme mit Mehrkern-Steuergeräten modellieren zu können und deren Echtzeitanforderungen beschreiben zu können wurden in dieser Arbeit entscheidende Parameter identifiziert. Dabei wurden Lösungsansätze bereits realisierter Architektur-Entscheidungen berücksichtigt, die in der Lage sind, das Echtzeitverhalten einer E/E-Architektur zu simulieren. Mit Hilfe eines Modellierungswerkzeugs wurden die identifizierten Multicore-Parameter modelliert und in einer standardisierten Form bereitgestellt. Dadurch wurde eine Simulation sowie eine frühzeitige Verifikation der Echtzeitanforderungen ermöglicht. Somit wurde eine mögliche Werkzeugkette entworfen, prototypisch implementiert und anschließend evaluiert. Die Evaluation wurde auf Basis eines Fallbeispiels durchgeführt, welches unter Berücksichtigung anhand der zuvor identifizierten Multicore-Parameter erstellt wurde. Außerdem wurden mögliche Optimierungsansätze für die Integration und Interoperabilität der Werkzeuge herausgearbeitet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

In diesem Kapitel werden die Motivation, das Umfeld und die Ziele der Arbeit erläutert.

1.1. Motivation

Heutzutage werden sogenannte eingebettete Systeme in viele elektrische und elektronische Geräte integriert. Die Absicht dahinter ist häufig eine Verbesserung des menschlichen Lebens in Bezug auf Sicherheit, Schutz, Komfort, Autonomie und Produktivität. Diese verschiedensten Ziele sollen durch Funktionalitäten erreicht werden, die häufig von eingebetteten Systemen realisiert werden. Dabei können auch zeitliche Vorgaben eine wesentliche Rolle spielen. Gerade im Automobilbereich gibt es eine signifikante Anzahl solcher zeitkritischen Funktionalitäten.

Diese Funktionalitäten werden heutzutage mit Hilfe von elektronischen Steuereinheiten (ECUs) implementiert. Folglich stieg in der jüngeren Vergangenheit die Anzahl der im Fahrzeug eingebetteten ECUs drastisch an. Mehr als 2000 Funktionen wurden in den High-End-Fahrzeugen auf über 70 ECUs realisiert [Web09]. Auf Grund dieser Tatsache nimmt der Bedarf an Rechenleistung dieser elektronischen Steuereinheiten beständig zu.

Die üblichen Maßnahmen, diese Anforderungen zu erfüllen, beschränken sich in der Regel auf die Miniaturisierung von Transistoren und die Erhöhung der Taktgeschwindigkeit des Prozessors [Wan17]. Beide dieser Lösungsansätze stoßen mittlerweile an ihre Grenzen. So hat beispielsweise die Taktgeschwindigkeit einen Maximalwert erreicht, da auf Grund physikalischer Gegebenheiten die Hitze in den immer kleiner werdenden Einheiten nicht mehr abtransportiert werden kann [Wal16].

Eine vielversprechende Möglichkeit, trotz dieser Probleme Leistungssteigerungen zu erzielen, liegt im zunehmenden Einsatz von Steuergeräten mit Multicore-Prozessoren. Verringerung der Komplexität der Architektur, ein besserer Umgang mit ressourcenintensiven Anwendungen und eine Verbesserung der Sicherheit sind einige Vorteile, die man sich von der Verwendung von Multicore-Prozessoren verspricht [Nav+10].

In ihrer Dissertation fasst Mircea Negrean drei Anwendungsfälle zusammen, die hauptsächlich die Einführung von Multicore-Architekturen für elektronische Steuergeräte im Automobil vorantreiben. Diese werden im Folgenden zitiert und in Abbildung 1.1 veranschaulicht:

1. Die Notwendigkeit, die Anzahl der ECUs zu reduzieren, löst die Aggregation von mehreren kleineren ECUs in eine Multicore-ECU. Das bedeutet, dass bisher vertriebene Software-Anwendungen auf einem einzigen Chip zusammengefasst werden.

2. Die Notwendigkeit, mehr Funktionen in die Steuergeräte zu integrieren, erfordert automatisch mehr Rechenleistung. So müssen beispielsweise in relativ leistungsstarken

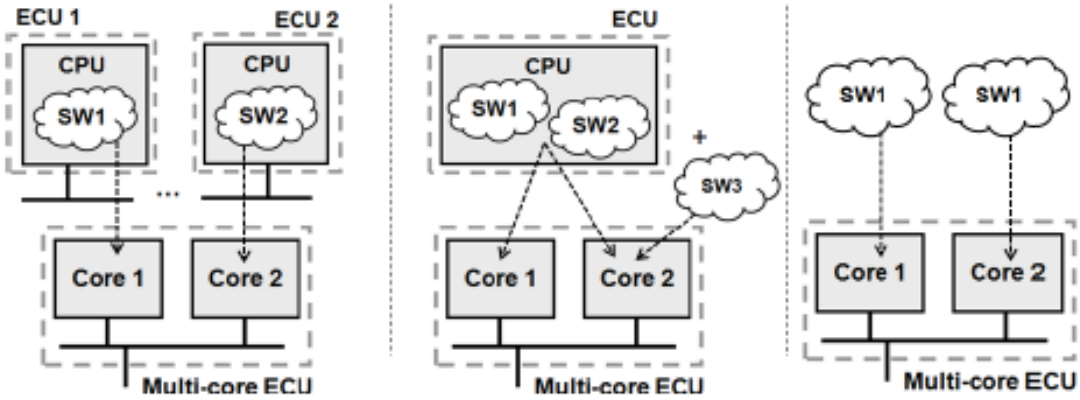

Abbildung 1.1.: Multi-Core Systeme - Anwendungsfälle [Neg15]

Bereichen, wie fortschrittlichen Fahrerassistenzsystemen, immer mehr Funktionen implementiert werden. In diesem Fall können Multicore-Architekturen verwendet werden, um komplexe Berechnungen über mehrere Kerne zu parallelisieren und gleichzeitig die Integration zusätzlicher Softwarefunktionen zu ermöglichen.

3. Die Notwendigkeit, eine hohe Leistung bei gleichzeitig hoher Zuverlässigkeit, d.h. Redundanz bei Systemausfällen, zu gewährleisten. Dies kann erreicht werden, indem die gleiche Software auf verschiedenen Kernen ausgeführt wird.

Multicore-Systeme stellen eine Verbindung aus Rechenressourcen und darauf laufender Software dar. Um die beschriebenen Anwendungsfälle zu ermöglichen, müssen sie innerhalb streng definierter Zeitgrenzen auf Eingangssignale reagieren, damit essenzielle Funktionen wie die des Airbags zweckmäßig ausgeführt werden. Dies muss während des kompletten Design-Prozesses beachtet werden. Aspekte wie Echtzeitbeschränkungen, gemeinsame Ressourcennutzung und gemeinsamer Speicherzugriff sollten bestenfalls nicht erst nach der abgeschlossenen Implementierung erkannt werden. Die Interaktion zwischen den Kernen z.B. über gemeinsame Speicher-Ressourcen erzeugt eine zeitliche Abhängigkeit zwischen verschiedenen Aufgaben (Tasks), die auf verschiedenen Kernen laufen. Diese Abhängigkeit kann dazu führen, dass ein Task mit niedriger Priorität auf einem Kern einen Task mit hoher Priorität auf einem anderen Kern verlangsamen, was zu schwerwiegenden Konsequenzen führen kann. So ein Fall darf daher zur Designzeit nicht unentdeckt bleiben. Aus diesem Grund muss die Modellierung bzw. Entwicklung komplexerer Softwarearchitekturen durch einen passenden Designprozess unterstützt werden. Dabei ist Expertise zum Timing-Verhaltens in Multicore-Systemen von großer Relevanz. [Neg15]

In der Vergangenheit wurden die meisten Softwareanwendungen für Steuergeräte mit einem einzelnen Kern entwickelt. Es ist die Aufgabe heutiger Systemdesigner, diese auf Multicore-Systemen einsetzbar zu machen.

Aus diesem Grund bedarf es einer durchgängigen Werkzeugkette, welche die Modellierung von Architekturen mit Multicore-ECUs unter Berücksichtigung von Echtzeitbeschränkungen unterstützt. Diese sollte dem Entwickler objektives Feedback über die Qualität

der Softwarearchitektur geben. Um dieses Ziel zu erreichen, soll beispielhaft an einem Modellierungswerkzeug erforscht werden, ob Timing-sensitive Multicore-Architekturen in einer ununterbrochenen Werkzeugkette entwickelt und durch frühzeitige Simulation das Design verifiziert und validiert werden können.

1.2. Umfeld der Arbeit

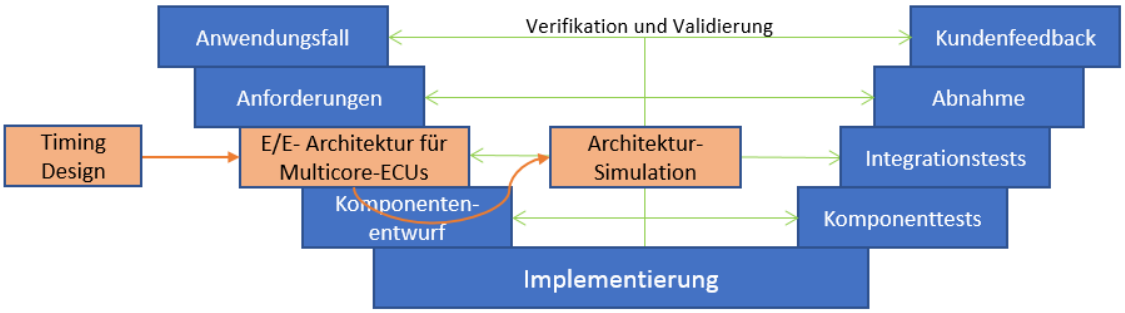

Abbildung 1.2.: Allgemeines V-Modell mit Ergänzungen zur Arbeit.

Die Domäne der Abschlussarbeit ist die Automobilindustrie. Allgemein gesagt sind in dieser Industrie viele Funktionalitäten gefragt, welche zum großen Teil mit Hilfe von E/E-Architekturen beschrieben werden können. Dazu gehört z.B. das elektronische Verstellen von Sitzen oder Außenspiegeln, aber vor allem auch essenzielle Funktionalitäten wie Bremsen oder Airbags. Für einige wesentliche Funktionalitäten ist das Timing äußerst wichtig, wie z.B. für die bereits erwähnten Bremsen oder Airbags. Um die wachsenden Anforderungen nach neuen Funktionalitäten und gesteigerter Leistung zu bewältigen, gewinnt die Multicore-Technologie auch in der Automobilentwicklung an großer Bedeutung. Das engere Umfeld dieser Arbeit besteht aus den Themenfeldern der Timings und Multicore-Technologie sowie der Kombination aus beiden. Multicore-Architekturen mit Echtzeitbeschränkungen müssen gesondert berücksichtigt werden, worauf in dieser Arbeit ein besonderer Fokus liegt.

Die Softwareentwicklung in dieser Industrie folgt im Wesentlichen dem V-Modell [Bar+06]. Das V-Modell ermöglicht es, Fehler und Probleme im Entwicklungsprozess frühzeitig zu erkennen und korrigierend einzugreifen. So können beispielsweise Probleme mit der Multicore-Architektur oder Verletzungen der Echtzeitanforderungen vor der Implementierung erkannt werden.

Das Umfeld dieser Arbeit besteht im Wesentlichen daraus, eine Möglichkeit zu erforschen, wie man oben beschriebenen Timing- und Multicore-Aspekte in der Automobil-Entwicklung gemäß des V-Modell-Konzeptes möglichst früh überprüfen kann. Dies ist in Abbildung 1.2 gezeigt. Wie in dieser Abbildung dargestellt wird, soll diese Überprüfung Mithilfe von Architektur-Simulationen umgesetzt werden.

1.3. Ziel der Arbeit

Das Ziel dieser Masterarbeit ist es zu erforschen, inwiefern eine durchgängige Modellierung und Simulation von Multicore-Eigenschaften während der Steuergeräteentwicklung möglich ist. Das kann durch die Kopplung eines Werkzeugs zur Modellierung von sogenannten E/E-Architekturen mit einem Werkzeug, das die frühzeitige Simulationen dieser Architekturen hinsichtlich deren Echtzeitbeschränkungen ermöglicht, erreicht werden. Nun liegen die Herausforderungen dieser Arbeit zu untersuchen, ob alle zur Simulation notwendigen Informationen modelliert sowie in einer standardisierten Form bereitgestellt werden können. Daher wird das im Rahmen dieser Arbeit verwendete Modellierungswerkzeug anhand der identifizierten Multicore-Eigenschaften erweitert. Der Erfolg dieser Arbeit wird durch die Evaluation der Simulierbarkeit der bereitgestellten Strukturen gezeigt. Das Ziel dieser Masterarbeit lässt sich zu folgenden fünf Punkten zusammenfassen:

1. **Identifikation der Multicore-Eigenschaften, die für die entsprechenden Simulationen benötigt werden.** Dazu gehört die Analyse von Standards wie Automotive Open System Architecture (AUTOSAR), welches das kommende defacto-Standard für Automotive-Steuergeräte ist, sowie Automobilprojekten, die Multicore-Technologie für Steuergeräte bereits nutzen und eine Softwarearchitektur auf Basis von Multicore-ECUs simulieren können.

2. **Fallbeispiel (Use-Case) Festlegung.** Aus den Erkenntnissen und Resultaten von Punkt eins wird ein praktisches Fallbeispiel aus dem Bereich der Fahrzeugentwicklung festgelegt. Das Fallbeispiel soll zum einem die Software- sowie Hardware-Architektur des Systems und zum anderen die identifizierten Multicore-Eigenschaften berücksichtigen.

3. **Konzept zur Modellierung der identifizierten Eigenschaften.** Anhand des festgelegten Fallbeispiels werden zuerst die bestehenden Strukturen mit Hilfe eines E/E-Architekturwerkzeugs modelliert. Es wird auch ein Konzept zur Modellierung der identifizierten Multicore-Eigenschaften unter Berücksichtigung der Echtzeitbeschränkungen erarbeitet.

4. **Kopplung von existierenden Werkzeugen zur Modellierung und Simulation.** Damit eine Werkzeugkette für die Entwicklung von E/E-Architekturen mit der Berücksichtigung von Multicore-Steuergeräte realisiert werden kann, müssen die einzelnen Werkzeuge in geeigneter Weise miteinander verbunden werden. Im Automotiv-Kontext könnte dies in Form von AUTOSAR Dateien stattfinden. Um das zu ermöglichen, wird eine sogenannte Modell-zu-Modell-Transformation unter Anwendung eines Transformationsregelwerkes durchgeführt.

5. **Evaluation der umgesetzten Werkzeugkette.** Es soll evaluiert werden, ob das modellierte Fallbeispiel im ausgewählten Simulationswerkzeug simulierbar ist. Das Ergebnis dieser Evaluation zeigt, ob mit AUTOSAR die M2M-Transformation vollständig und korrekt durchgeführt wird. Gegebenenfalls soll ermittelt werden, um welche der Eigenschaften dieses Format in Zukunft erweitert werden müsste, damit

alle für die betrachteten Simulationen nötigen Informationen übertragen werden können.

1.4. Aufbau der Arbeit

Diese Masterarbeit besteht aus vier Teilen. Im ersten Teil werden die Multicore-Technologie, das Timing-Problemfeld und weitere für diese Arbeit wichtigen Grundlagen dargestellt. Im zweiten Teil wird eine Analyse der verwendeten Werkzeuge und Austauschformate durchgeführt. Dabei werden auch die notwendigen Multicore-Systemparametern identifiziert. Im Rahmen dieser Analyse wird ein Fallbeispiel definiert. Im dritten Teil wird ein Konzept für eine von Modellierung bis Simulation durchgängige Werkzeugkette erarbeitet. Im vierten Teil wird das entwickelte Konzept mit Hilfe des Fallbeispiels evaluiert.

2. Grundlagen

In diesem Kapitel werden die wichtigsten Begrifflichkeiten und Grundlagen, die zum Verständnis dieser Abschlussarbeit notwendig sind, erläutert.

2.1. Aktuelle E/E-Architekturen für die Automobilindustrie

Besonders Oberklassenwagen weisen heutzutage eine immense Anzahl verteilter elektronischer Steuergeräte (ECUs) auf, da die verschiedensten Funktionen elektronisch gelöst werden. Diese müssen sowohl miteinander als auch mit verschiedensten anderen Komponenten im Fahrzeug kommunizieren. Die E/E(Electric/Electronic) -Architektur verbindet die überwältigende Mehrheit dieser E/E-Systeme und diese bilden somit eine gemeinsame Netzwerkinfrastruktur.

Zu den Komponenten, die von einer E/E-Architektur beschrieben werden, gehören unter anderem Sensoren, Aktoren, Kabelräume für Daten und Strom sowie Steuergeräte mit eingebetteter Software. Aufgabe der E/E-Architektur ist es unter anderem, Strom, Daten und Signale bestmöglich zu verteilen. Aktoren agieren dabei gemäß den von Sensoren und Verarbeitungskomponenten verursachten Signalen. In der Automotive-Domäne bestehen E/E-Hardwarearchitekturen aus ECUs und weiteren Hardware-Elementen, welche in den Fahrzeugen verteilt sind. Durch Busse mit spezifischen Protokollen wird die Kommunikation zwischen den Steuergeräten ermöglicht. [Fik16]

Die Komplexität dieser Architekturen ist in der Regel so groß, dass sie unmöglich von einzelnen Softwareentwickler überschaut werden kann und wird in der Praxis von einer Reihe von Werkzeugen aus verschiedenen Abstraktionsebenen und komplexitätsreduziert dargestellt, um die Übersichtlichkeit zu ermöglichen.

2.1.1. Migration von Single-Core zu Multicore

Wie in der Einführung schon erwähnt, aufgrund dem exponentiellen Anstieg der Softwarebasierten E/E-Funktionen in Fahrzeugen, die zuvor auf vielen ECUs ausgeführt wurden, führen Automobilhersteller und Zulieferer nach und nach ECUs mit Multicore Prozessoren in ihre elektronischen Architekturen ein.

Multicore Prozessoren vereinen mehrere Prozessorkerne auf einem Chip. Während das Design der Multicore-Hardwarearchitekturen in Hinblick auf Verarbeitungseinheiten weitgehend festgelegt ist, liegt die größte Herausforderung mehr und mehr in der Software, welche die Multicore-Komponenten nutzen muss. In diesem Zusammenhang stehen System-Designer vor dem Problem, die bestehenden Werkzeugen zur Modellierung einer E/E-Architektur, die momentan auf Single-Core-Steuergeräte laufen, effizienter auf Multi-Core-Steuergeräte zu migrieren [KM07].

Robert Hilbrich, der Systemarchitekt in der Abteilung Fraunhofer FIRST und FOKUS war, zeigt drei Strategien für diese Migration auf [Hil12]. In seiner Masterarbeit fasst Andre Christian Roßbach diese, wie in im folgenden Abschnitt dargestellt, zusammen [Roß14]:

1. **Single-core Ansatz:** Die Anwendung wird einfach auf einen Kern abgebildet. Dies wird jedoch nicht das volle Potenzial des Multicore-Systems ausschöpfen und wahrscheinlich die Geschwindigkeit reduzieren, da die Taktrate des Multicore-Ziels geringer ist. Außerdem ist dies wirtschaftlich nicht sinnvoll, da das Ziel darin besteht, weniger ECUs zu haben und mehrere Funktionen demselben ECU zuzuordnen.

2. **Eine Anwendung - ein Multicore Prozessor:** Die Anwendung wird zerlegt und über alle Kerne des Hardware-Targets verteilt. Nur diese Anwendung nutzt die gesamte ECU und es ist daher vergleichsweise einfach, die Echtzeitbeschränkungen über Tools zu überprüfen. Die Anwendung wird jedoch nicht vollständig sequentiell und damit nur bis zu einem bestimmten Betrag skalierbar sein. Darüber hinaus kann das Design-Muster (insbesondere für die Echtzeit) nicht mehr verwendet werden.

3. **Multifunktionsintegration:** Mehrere Anwendungen teilen sich das gleiche Multicore-Ziel. Dies ermöglicht die optimale Nutzung des Potenzials der Multicore-Prozessoren. Dies wird aber auch die Komplexität des Designs erheblich erhöhen. Daher muss insbesondere der Zugriff auf gemeinsame Ressourcen überprüft werden [Neg15].

2.2. Ansätze zur E/E-Architektur Beschreibung

2.2.1. Modellbasierte Entwicklung

Einer der wichtigsten Trends bei der Beschreibung einer E/E-Architektur ist der Trend zu mehr Abstraktion und damit die Fähigkeit, die Komplexität während der gesamten Entwicklung besser zu managen [EJ09]. Aus diesem Grund ist die modellbasierte Entwicklung ein sehr wichtiger Trend im Design einer E/E-Architektur. In modellbasierten Designs wird ein Modell der Anwendung verwendet, um komplexe Designs auf höheren Abstraktionsebenen darzustellen. Dabei werden es nur spezielle Informationen zum Verständnis und zur Analyse großer Designs angezeigt [OR10]. In [Pre+07] verdeutlichen die Autoren die Vorteile einer nahtlosen modellbasierten Entwicklungs-Werkzeugkette für das Automotive Software Engineering. [Mac15]

2.2.2. Metamodellierung

Damit ein Modell eines Systems gebildet werden kann, muss eine Modellierungssprache vorliegen, die die abstrakte Syntax dieses Modells vorgibt. [Rei05]. Diese Aufgabe wird vom Metamodell übernommen, welches man daher umgangssprachlich als "Modell eines Modells"bezeichnen kann.

Dieses Prinzip kann wiederum auf das Metamodell selbst angewendet werden, da dieses ebenfalls spezifiziert werden muss. Dementsprechend gibt es das Meta-Meta-Modell, welches diese Funktion für das Metamodell erfüllt.

Die *Object Management Group* (OMG) führte den Begriff der *Meta Object Facility* (MOF) ein [Gro16], welche eine spezielle Metadaten-Architektur beschreibt. Diese besteht aus verschiedenen, mindestens zwei, Ebenen.

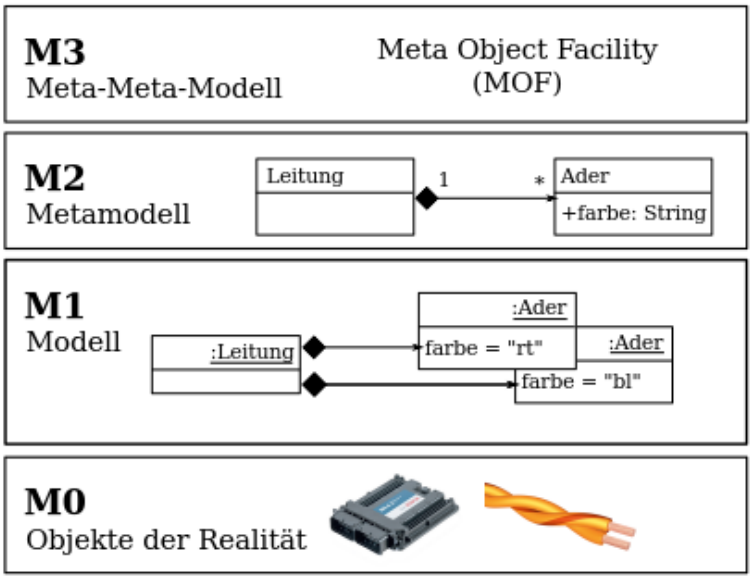

Abbildung 2.1.: Schematische Darstellung der Metamodellierung-Architektur [Sch17]

Die in Abbildung 2.1 dargestellte, dieser Arbeit zugrundeliegende, Metamodellierung-Architektur lässt sich mit vier Ebenen beschreiben. Ihre Benennung als *M0*- bis *M3*-Ebene ist dabei an die Spezifikation der *MOF* angelehnt.

- **M0-Ebene**
  Diese unterste Ebene repräsentiert die zu modellierenden Objekte der Realität. Diese Objekte können z.B. Stecker, Verbinder, Kabel oder Prozessoren sein.

- **M1-Ebene**
  Die M1-Ebene liefert abstrakte Beschreibungen zu modellierenden Objekte. Diese Beschreibungen werden auf die für das Anwendungsgebiet wesentlichen Aspekte und Eigenschaften reduziert.

- **M2-Ebene**
  Diese Ebene definiert das Metamodell. Hier wird vorgegeben, welche Elemente modelliert werden können. Dies bezieht sich auf die Beschaffenheit, den Einsatz und die Beziehungen dieser Elemente.

- **M3-Ebene**
  Analog zur Ebene M2 definiert die M3-Ebene die Beschaffenheit des Metamodells. Das Ergebnis solcher Definitionsprozesse wird dementsprechend Meta-Metamodell genannt. Außerdem kann diese Ebene sich selbst beschreiben.

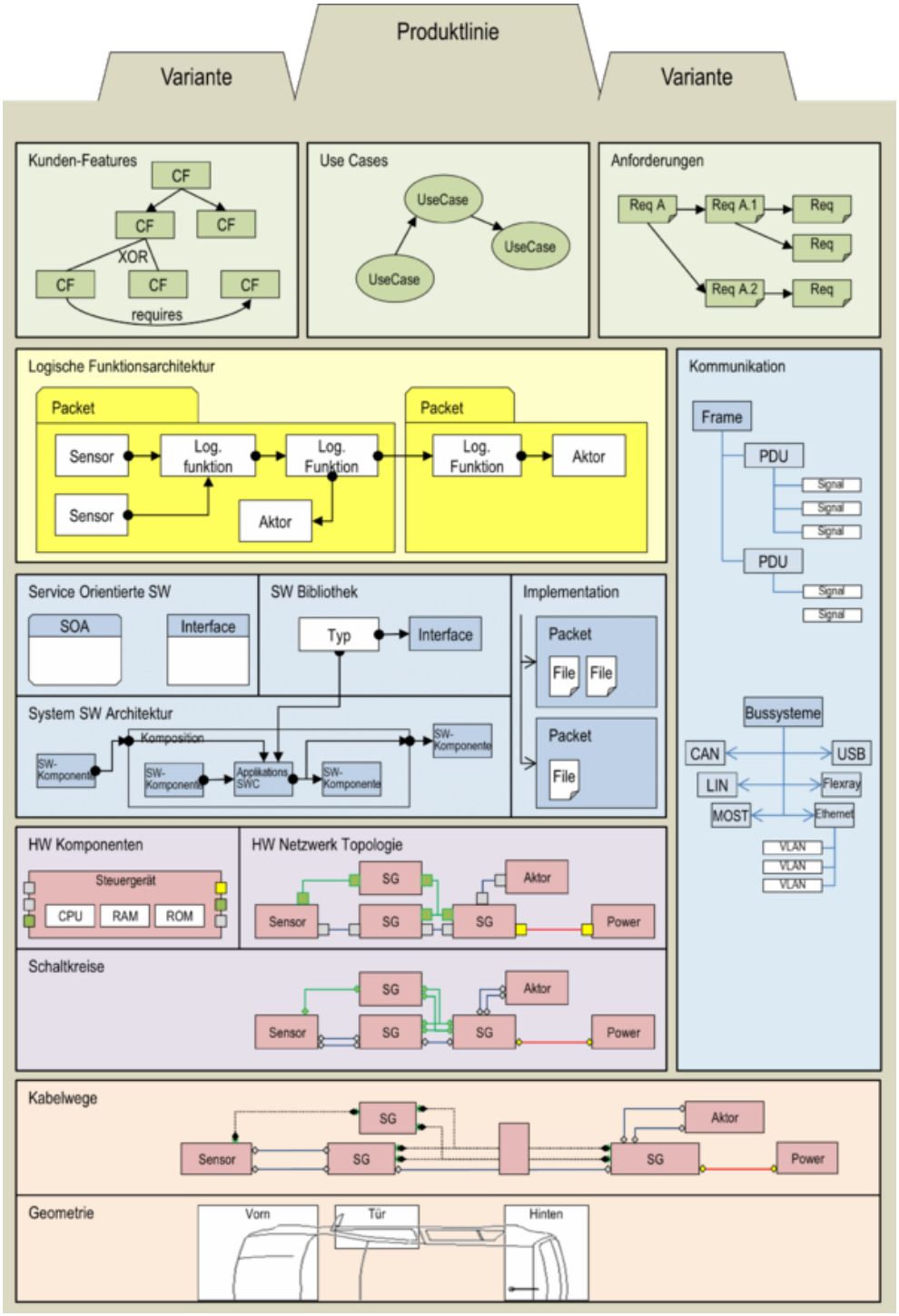

Abbildung 2.2.: Ebenen einer E/E-Architektur [Gmb]

2.2.3. Modellbasierter Ansatz mit PREEvision

PREEvision ist ein Werkzeug, welches eine modellbasierte Entwicklung von E/E-Architekturen unterstützt. Es wird von der Firma Vector Informatik GmbH vertrieben. PREEvision besitzt ein eigenes MOF-basiertes Metamodell namens EEA-ADL (Electric Electronic Analysis Architecture Description Language).

Um mit komplexen E/E-Architekturen umgehen zu können, folgt PREEvision drei bewährten systemtechnischen Prinzipien - Abstraktion, Zerlegung und Wiederverwendung - und unterstützt die Modellierung von E/E-Architekturen mit den in Abbildung 2.2 gezeigten Architekturebenen.

1. **Kunden-Features:** Diese Schicht unterstützt die Erstellung und Sammlung von kundenrelevanten Funktionen.

2. **Use-Cases:** Diese Schicht unterstützt die Erstellung und Modellierung von Anwendungsfällen und Akteuren, um das Verhalten eines Systems aus Sicht des Benutzers zu beschreiben.

3. **Anforderungen:** Diese Schicht unterstützt die Erstellung und Sammlung von Anforderungen. Anforderungen verfeinern Kundenmerkmale oder beschreiben technische Spezifikationen.

4. **Logische-Funktionsarchitektur:** Diese Schicht unterstützt das logische Design von Fahrzeugfunktionen als ersten Schritt zur Einrichtung der Implementierung von Kunden-Features. Es stellt die Zerlegung der Funktionalität des Systems über logische Komponenten dar.

5. **System SW Architektur:** Diese Schicht unterstützt die Modellierung von Softwarekomponenten und ihre Schnittstellen. Auch eine serviceorientierte Modellierung ist möglich.

6. **Implementation:** Diese Schicht unterstützt die Verwaltung von Implementierungsdaten für Softwarekomponenten. Implementierungsmodelle können als Dateien gespeichert werden. Grundfunktionen wie Wiederverwendung, Versionierung und Mappings sind für alle Dateien verfügbar.

7. **HW Netzwerk Topologie:** Diese Schicht unterstützt die Modellierung von Hardware-Netzwerken und Verbindungskonzepten bezüglich Hardwarekomponenten auf höchster Abstraktionsebene.

8. **HW Komponenten:** Diese Schicht unterstützt die Modellierung von Hardwarekomponenten und deren internen und externen Verbindungen.

9. **Schaltkreise:** Diese Schicht unterstützt die Verfeinerung der im Hardwarekomponenten-Paket modellierten logischen Verbindungen und die Beschreibung der elektrischen Potentiale. Die elektrische Schaltung ist die Grundlage für die Entwicklung der Drähte auf der nächsten Schicht.

10. **Kabelwege:** Diese Schicht unterstützt die Implementierung von logischen und schematischen Konnektoren. Alle physikalischen Eigenschaften, wie z.B. Kosten, können dargestellt werden.

11. **Geometrie:** Diese Schicht unterstützt die Modellierung des geometrischen Layouts des Systems. Mit Positionskoordinaten und Geometrieabmessungen der Bauteile liefert das Modell detaillierte Informationen. Auch 3D-Modelle von CAD-Tools können importiert werden.

12. **Kommunikation:** Diese Schicht unterstützt die Verwaltung aller verfügbaren Prototypen und Signale des vollständiges Modell über alle Ebenen hinweg. Signale können in das System importiert oder manuell erstellt werden. Sie können auch durch Signalrouting erzeugt werden.

2.3. Automotive Open System Architecture (AUTOSAR) als Austauschformat

Als die mikrokontrollerbasierte Automobilelektronik in der Vergangenheit an Popularität gewann, war die Koordination zwischen diesen Mikrokontrollern eine große Herausforderung. Für jede Funktionalität war ein separates Steuergerät vorgesehen. Mit den steigenden funktionalen Anforderungen stieg auch die Anzahl der Steuergeräte. Eine größere Anzahl von Steuergeräten führte zu einer erhöhten Komplexität des Netzwerks und damit auch von dessen Entwicklungskosten. Die Lösung wurde 1983 als Controller Area Network (CAN) entwickelt [Pow18]. Wenn mehrere Hersteller eine gemeinsame Funktionalität für die beteiligten Steuergeräte entwickeln, ist zusätzlichen Aufwand und Kosten für Integration und Tests nötig. Um eine Lösung für solche Probleme der Automobilhersteller im Jahr 2003 zu finden, haben Automobilhersteller, Zulieferer und Werkzeughersteller kooperiert, um gemeinsam eine standardisierte Softwarearchitektur für den gesamten Entwicklungsprozess bei der Fahrzeugentwicklung zu etablieren. Das Ergebnis dieser Zusammenarbeit stellt AUTOSAR dar.

AUTOSAR ermöglicht eine hardwareunabhängige Softwareentwicklung, indem es die Anwendungssoftware und die zugehörige Infrastruktur in Form von Basissoftware trennt. PREEvision unterstützt diese Basissoftware von AUTOSAR nicht.

AUTOSAR standardisiert insgesamt die Spezifikation der Softwarearchitektur, die Schnittstellen und eine Methodik für den Entwicklungsprozess.

Zur Bewältigung der stetigen Zunahme von Komplexität einer E/E-Architektur und der anderen zuvor beschriebenen Herausforderungen, beschäftigt sich das AUTOSAR-Konsortium seit mehr als 10 Jahren mit der Integration der Multicore-Technologie. Mit der Version 4.0 der AUTOSAR Spezifikation, die im Jahr 2009 veröffentlicht wurde, begann die Unterstützung für die verteilte Ausführung auf eingebetteten Multicore-Prozessoren [AUT14].

E/E-Architekturen beschreiben sehr oft Echtzeitsysteme. Um eine sichere Anwendung dieser Architekturen zu garantieren, müssen geeignete Lösungen zur Untersuchung des

Timing-Verhaltens zur Designzeit notwendig. Dieser Aspekt gewinnt zusätzlich an Bedeutung, je größer der Anteil an Multicore-Steuergeräten im Architektur-Netzwerk wird.

AUTOSAR hat jedoch eine formale Sprache und Methodik für Timing-Design im Automobilbereich entwickelt. So ist es mit der Timing Extension Spezifikation des AUTOSAR-Standards von Ende 2009 möglich, das Sollverhalten des Systems festzulegen [AK17].

2.3.1. Grundlagen der AUTOSAR-Softwarearchitektur

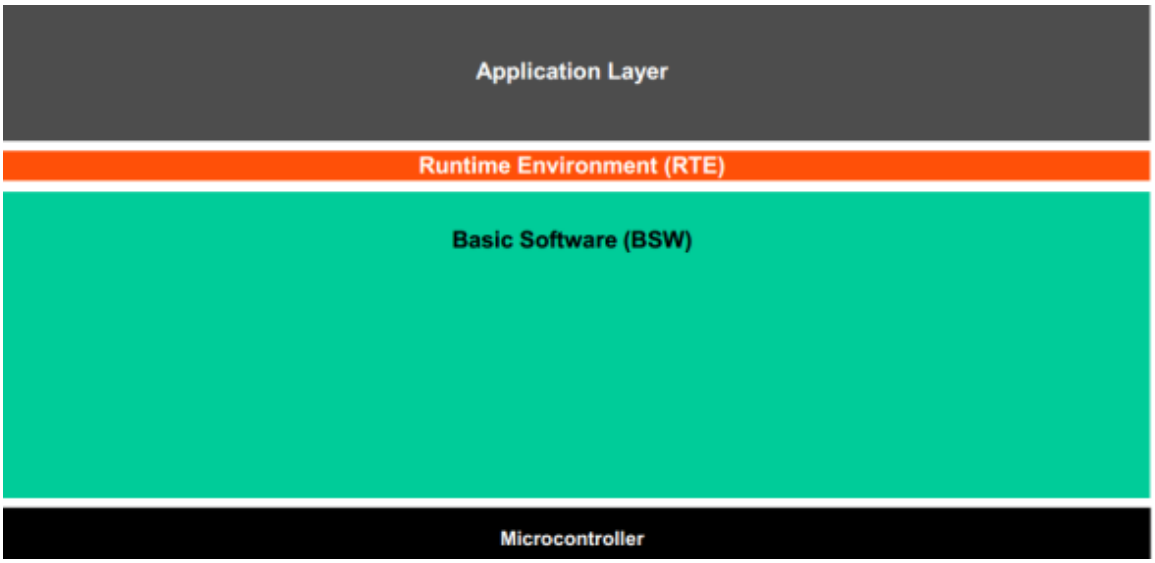

Abbildung 2.3.: AUTOSAR-Architekturschichten

Durch seine standardisierte Schichtarchitektur reduziert AUTOSAR die Komplexität im Systementwurfsprozess: die Anwendungsschicht, die Basic Software-Schicht und die Laufzeitumgebung (RTE). Diese AUTOSAR-Schichtarchitektur ermöglicht die Entkopplung der Funktionalität der unterstützenden Hard- und Softwaredienste, was in Abbildung 2.3 zu sehen ist. Der Zweck jeder einzelnen Schicht wird im Folgenden ausgeführt:

- **Anwendungsschicht:** Die Anwendungsschicht stellt ein Standardbeschreibungsformat für die Anwendung bereit, welche aus den Softwarekomponenten besteht. Diese Schicht ist gänzlich unabhängig von der Hardware.
- **Laufzeitumgebung (RTE):** Die Laufzeitumgebung ist eine mittlere Schicht und befindet sich zwischen der Anwendungsschicht und der Basissoftware. Sie realisiert die Kommunikation zwischen und innerhalb von Softwarekomponenten sowie zwischen Softwarekomponenten und Basissoftware. Für eine Softwarekomponente ist es irrelevant, ob eine Intra-ECU- oder Inter-ECU-Kommunikation vorliegt. Das Ziel von Inter-ECU-Kommunikation ist es, dass alle Softwarekomponenten aller Steuergeräte so miteinander kommunizieren können, als befänden sie sich im selben großen Steuergerät. Die Intra-ECU-Kommunikation bezieht auf die Softwarekomponenten innerhalb derselben Steuereinheit. Dabei dürfen sie nicht direkt miteinander, sondern nur über die RTE kommunizieren.

- **Basis Software (BSW):** Die Basissoftware ist eine Abstraktionsschicht der Hardware. Ihre Aufgabe ist es, den direkten Zugriff der Applikationsschicht auf die Hardware unterbinden.

2.3.2. Übersicht über den AUTOSAR-Ansatz

Für die Systementwicklung schlägt AUTOSAR einen Ansatz vor, welcher in [AK14] spezifiziert wurde. Dieser basiert auf dem Konzept der AUTOSAR-Softwarekomponente.

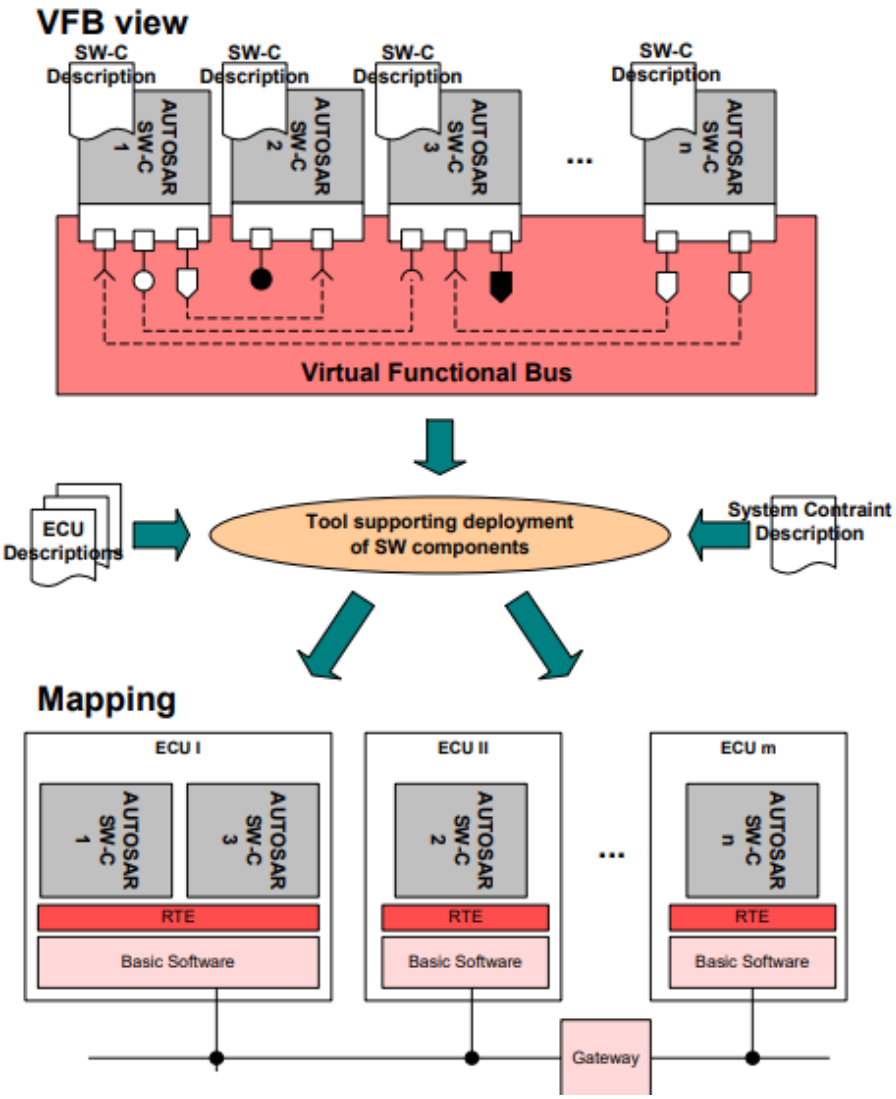

Abbildung 2.4.: AUTOSAR-Grundansatz [AK08]

In [Wan17] wird dieser Ansatz wie folgt dargestellt:

- **Software Component Description:** Softwarekomponenten (SWCs) bei AUTOSAR bündeln Anwendungen. Diese Komponenten laufen auf der AUTOSAR-Infrastruktur. Nahezu alles, was ein Entwickler verstehen muss, damit er seine Komponente ein System einpflegen kann, wird über eine Softwarekomponentenbeschreibung zu Verfügung gestellt. So kann die Softwarekomponentenbeschreibung einer SWC, die Daten/Dienst, ihre gesendeten und empfangenen Daten, ihr internes Verhalten im AUTOSAR XML (ARXML)-Format beschreiben.
- **Virtual Functional Bus (VFB):** Mittels des Virtual Functional Bus werden die Softwarekomponenten integriert und stehen miteinander in Verbindung. So wird durch diesen die abstrakte Beschreibung der Kommunikation und der Deployment-Phase unabhängig voneinander verfügbar gemacht. Dies ermöglicht die Ortsveränderung der Komponenten. Durch den Aspekt der Virtualität kann die Software ohne Berücksichtigung spezifischer Hardware-Besonderheiten entwickelt werden.
- **ECU description and System constraints:** Im Prozessschritt ËCU description System constraintsërfolgt die ECU-Beschreibung abgekoppelt von der Softwarekomponenten-Beschreibung. Hier werden zudem Systemeinschränkungen definiert.
- **ECU Mapping:** In diesem Prozessschritt müssen die Softwarekomponenten auf das ECU-Netzwerk abgebildet werden. Dies bedeutet außerdem, dass die Konfiguration und Generierung der Laufzeitumgebung und Basissoftware spezifischen ECUs zugeordnet werden.

2.3.3. Modellierungsdetails von AUTOSAR

In diesem Teil werden die für diese Arbeit relevanten Komponenten der Modellierungsdetails der Anwendungsschicht der AUTOSAR-Architektur, die zuvor in der Abbildung 2.3 dargestellt wurden, erläutert. Die Basissoftware-Sicht und Laufzeitumgebung werden im Rahmen dieser Arbeit nicht betrachtet, da PREEvision diese nicht unterstützt.

Anwendungsschicht

Die Anwendungsschicht besteht aus den Softwarekomponenten des anwendungsspezifischen Teils des Systems. Diese Softwarekomponenten kommunizieren über Ports mit den entsprechenden Elementen ihrer Umgebung. Um direkt miteinander kommunizieren zu können, brauchen Softwarekomponenten speziell dafür vorgesehene Schnittstellen. Softwarekomponenten werden üblicherweise in atomar und kompositorisch unterteilt. Bei kompositorischen Softwarekomponenten werden verschiedenen Komponenten logisch zusammengefasst. Diese können sogar auf unterschiedlichen Steuergeräten verstreut sein. Die Komponenten-Beschreibung enthält den *ComponentType*, *PortPrototypes*, *PortInterfaces*, *DataElementPrototype* und das *InternalBehavior*, welches durch *Runnables* definiert wird. AUTOSAR definiert diese Aspekte in seiner Software-Komponentenvorlage [AK14]. In Abbildung 2.5 wird eine typische AUTOSAR-Softwarekomponente dargestellt.

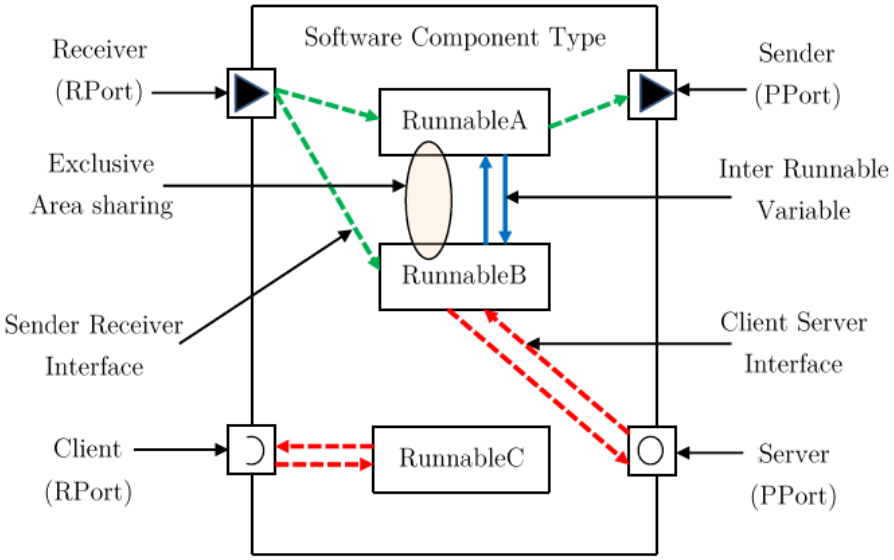

Abbildung 2.5.: Grafische Darstellung einer Softwarekomponente in AUTOSAR [Pow18]

- *Runnable:* Die atomaren Funktionskomponenten in den Softwarekomponenten werden durch die Runnables repräsentiert. Dass die Funktionskomponenten atomar sind, bedeutet, dass Sie nicht weiter unterteilt werden können. Die Runnables setzen sich aus Teilen funktionalem Quellcodes zusammen. Sie können mit anderen Runnables, die sich in derselben Softwarekomponente befinden, mittels Inter Runnable Variables (IVR) kommunizieren. Mit Runnables, die in anderen Softwarekomponenten platziert sind, verständigen sie sich über Interfaces und Ports.

  Die hauptsächlichen Aufgaben bzw. Funktionen von Runnables sind das Lesen, Ausführen und Schreiben. Das Ausführen und die Planung von Runnables kann unabhängig von anderen ausführbaren Einheiten derselben atomaren Softwarekomponente erfolgen.

  Die Ausführung eines Runnables findet im Rahmen eines Betriebssystem-Tasks statt und wird durch ein Laufzeitumgebungs-Ereignis ausgelöst.

  Runnables können durch die unterschiedlichsten Ereignisse (Events) ausgelöst werden. Darunter fallen z.B. Timing-Ereignisse, welche die periodischen Runnables auslösen. Weitere Beispiele sind Moduswelchsel-Events, Data Received Events, Operation Invoked Events usw.

- *Ports:* Ports implementieren eine Vielfalt an Schnittstellen, um auf benötigte Informationen zuzugreifen. Softwarekomponenten besitzen zwei Arten von Ports: bereitgestellte (*PPort*) und erforderliche Ports (*RPort*).

- *Interfaces:* Die Schnittstelle bzw. das Interface definiert, wie Informationen zwischen Softwarekomponenten und/oder Basissoftware-Modulen ausgetauscht werden. Durch diese wird die Kommunikation über Ports erst möglich gemacht. *Sen-*

derReceiver Interface, *ClientServer Interface* und *ModeSwitch Interface* stellen die bedeutendsten Arten von Schnittstellen dar.

2.3.4. AUTOSAR XML

ARXML steht für AUTOSAR XML und stellt das von AUTOSAR verwendete XML-basierte Austauschformat dar. In diesem werden Konfigurations- und Designdetails eines Systems dargestellt. Dadurch wird die Wiederverwendung von den zuvor erläuterten Softwarekomponenten-Beschreibungen möglich gemacht. Demzufolge kann man Systeme als eigenständige Softwarekomponenten-Module entwickeln und dann anschließend mittels einer ARXML-Datei wiederverwenden.

Mittels ARXML kann man vollständige E/E-Architektursystem definieren, ohne dass dazu weitere Dateiformate eingesetzt werden zu müssen.

2.3.5. *AdminData* in AUTOSAR

AdminData ist ein Konzept, das AUTOSAR anbietet, mit welchen fehlende Elemente des Metamodells ergänzt werden können.

AdminData ermöglicht es, administrative Informationen eines Elements auszudrücken. Diese administrativen Informationen entsprechen in der Handhabung Metadaten wie beispielsweise einer ID oder dem Zustand einer Datei [AKb]. Grundsätzlich gibt es vier Metadaten-Arten, die im Folgenden genannt werden:

- Die Sprache bzw. die verwendete Sprache
- Revisionsinformationen. Darunter beispielsweise Revisionsnummer, Zustand, Release-Zeitpunkt und Veränderungen. Dabei ist zu beachten, dass diese Informationen sowohl allgemein als auch auf ein spezielles Unternehmen bezogen sein können.
- Unternehmenseigene Dokument-Metadaten

AdminData ist ein für diese Arbeit besonders wichtiges Konzept, da hiermit fehlende Aspekte im AUTOSAR-Metamodell ergänzt werden können.

2.3.6. AMALTHEA

Genau wie AUTOSAR ist AMALTHEA ein XML-basiertes Austauschformat. Der Schwerpunkt dieses Formats im Gegensatz zu AUTOSAR liegt bei eingebetteten Multi-Core-Systemen. Das Besondere dabei ist, dass AMALTHEA die Basissoftware-Ebene tiefer abdeckt. Da PREEvision diese Basissoftware-Ebenen nicht in derselben Weise unterstützt, sondern sich auf höhere Abstraktionsebenen beschränkt, eignet sich die Verwendung von AMALTHEA im Rahmen dieser Arbeit nicht (vgl. 2.6.1).

2.4. Technische Grundlagen für Modelltransformation

Eine Modelltransformation stellt eine Umwandlung eines Modells in ein anderes Modell dar. Dies ist z.B. notwendig, wenn ein Werkzeug wie PREEvision mit zwei unterschiedlichen Metamodellen umgehen muss. Der Ablauf einer Modelltransformation wird in Abbildung 2.6 dargestellt und im Folgenden folgenden erläutert.

Durch die Anwendung eines Transformationsregelwerks wird im Zuge einer Modell-zu-Modell-Transformation ein Quellmodell in ein Zielmodell umgewandelt. Das Quell- und das Zielmodell sind jeweils als Instanzen ihrer Metamodelle zu betrachten. Im Falle dieser Arbeit sind dies das AUTOSAR-Metamodell und das EEA-Metamodell von PREEvsion. Die Modelltransformation findet in der in Unterabschnitt 2.2.2 beschriebenen Abstraktionsebene M1 statt. Zur Festlegung der Transformationsvorschriften wird das Transformationsregelwerk eingesetzt. Dieses verwendet die in den Quell- und Zielmetamodellen spezifizierten Metaklassen.

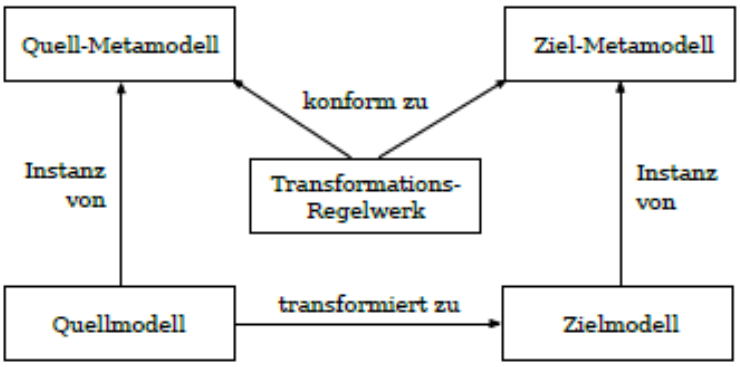

Abbildung 2.6.: Struktur einer Modell-zu-Modell Transformation [CS13]

PREEvision unterstützt den Import und Export verschiedener Austauschformate. (z.B. für Anforderungen, die Daten des Kabelbaums oder Standards wie AUTOSAR, welche für diese Arbeit relevant ist). Daher ist es möglich, mit PREEvision erzeugten Modellierungen in anderen Werkzeuge schnell und einfach weiterzuverwenden.

2.4.1. Transformationsregelwerk

Die Vorschriften zur Abbildung von Elementen aus dem Quellmodell in das Zielmodell werden durch das Transformationsregelwerk vorgegeben. Die entsprechenden Regeln liegen in der Modell-zu-Modell-Transformationssprache M2TOS vor [Rei05].

2.4.2. Mikrostruktur von Regeln

Wie in Abbildung 2.7 zu sehen ist, enthält eine Regel Regeltest, die *Left Hand Side (LHS)* und Regeltransformation, die *Right Hand Side (RHS)*. Die LHS repräsentiert die zu transformierende Elemente des Quellmodells und die RHS die entsprechenden Elemente im

Zielmodell. Das Regeldiagramm besteht aus Object, Link und die Instanzen der entsprechenden *MOF* Klassen und *MOF* Relationen. Falls der Regeltest zutrifft, dann wird die Regeltransformation ausgeführt.

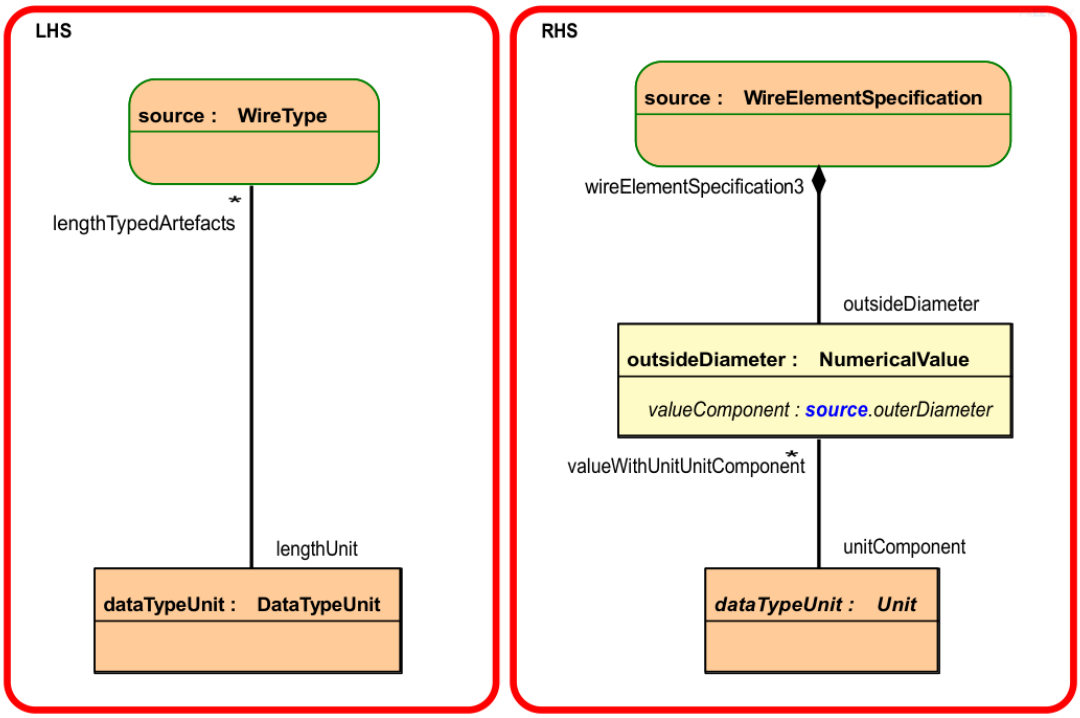

Abbildung 2.7.: Beispiel einer Transformationsregel aus [Sch17]

In Abbildung 2.7 wird die grundsätzliche Aufbau einer Regel dargestellt. Die Muster von LHS und RHS besteht aus Objekten, die über Links verbunden sind. Ein Objekt hat einen Namen und einen Typ. Das obere Objekt in LHS hat den Namen *Source* und den Typ *WireType*.

Über das Schlüsselword *source* wird es als Bezugspunkt der Regel gekennzeichnet. Durch Namensgleichheit zwischen Objekten der *LHS* und *RHS* wird eine Zuordnung zwischen dem Quell- und Zielmodell hergestellt. In Diagramm sind Objekten mit Zuordnung rot. Im Gegensatz dazu besitzen die ohne Zuordnung die Farbe von gelb. Für Objekte der *RHS* bedeutet dies dann, dass sie aus einem Regeladditionsanteil stammen, also neu erzeugt werden und keine Beziehung zum Quell-Modell haben.

Links verbinden Objekte und sind durch ihre Rollennamen typisiert. So verbindet in der *RHS* der Link *wireElementSpecification3-outsideDiameter* das Objekt von Typ *WireElement-Specification* mit dem vom *NumericalValue*.

In der Abbildung werden alle Objekte im Quellmodell vom Typ *WireType* mit assoziierter *DataTypeUnit* im Zielmodell durch eine *WireElementSpecification* repräsentiert. Diese enthält einen *NumericalValue* als Komposition, welchem wiederum ein Objekt vom Typ *Unit* zugeordnet ist, dass den *DataTypeUnit* abbildet.

2.5. Multicore Technologie bei der Fahrzeugentwicklung

In diesem Abschnitt werden zuerst der Stand der Technik und dann das während dieser Arbeit verwendeten Simulationswerkzeug vorgestellt.

2.5.1. Stand der Technik

Im letzten Jahrzehnt hat sich die Automobilindustrie auf die Multicore-Technologie umgestellt und mehrere Spezifikationen und Forschungsunterlagen veröffentlicht. Diese werden in der Tabelle 2.1 aufgelistet. Diese gibt dadurch wird einen Überblick über den neuesten Stand der Technik.

Jahr	Name	Beschreibung
2011	AUTOSAR 4.0 Spezifikation	Enthält Spezifikationen für Multicore Architekturkonzepte mit AUTOSAR [AKa]
2011-2015	ARAMiS	Feststellung der generellen Eignung von Mehrkernprozessoren in sicherheitskritischen Anwendungen [Kon11]
2012	MICROSAR	Vector Informatik veröffentlicht MICROSAR für AUTOSAR 4.0 mit Lösungen zur Multicore Technologie [Inf12]
2013	AUTOSAR 4.1 Spezifikation	Das AUTOSAR Konsortium veröffentlicht Spezifikation zur Verteilung der BSW und generelle Spezifikation zu Multicore Systemen mit AUTOSAR [AKa]
2015	IEEE	Enthält Bestandsaufnahme über Multicore-CPUs und Betriebssysteme für den Automotive-Bereich und Leitlinien für die Migration von Altsoftware. [Mac+15]
2014-2017	AMALTHEA	Entwicklung einer open source Tool Plattform für AUTOSAR umd Multicore. Ziel ist es Interoperabilität, Erweiterbarkeit und einen standardisierten Datenaustausch zwischen Tools und Projekten zu schaffen [AMA17]
2016-2019	ARAMiS II	Erforschung und Optimierung von Entwicklungsprozessen und Plattformen für Multicore-Architekturen [Kon16]

Tabelle 2.1.: Forschungsprojekte und Spezifikationen zur Multicore Technologie bei der Fahrzeugentwicklung. Verändert und ergänzt nach [Fri17]

Die in Tabelle 2.1 gezeigten Arbeiten stehen alle in Beziehung mit dieser Abschlussarbeit. Von diesen ist AMALTHEA das wichtigste Projekt, da das in dieser Arbeit verwendete Werkzeug zur Timing-Simulation aus dem selben Forschungsprojekt namens ITEA2 stammt [ITE].

2.5.2. Simulationssoftware für Systemmodelle

Die Simulation der Systeme ist notwendig, um den zeitlichen Ablauf entsprechend den Echtzeitanforderungen zu validieren. Zunächst wird der Grund für ein Simulationswerkzeug

angezeigt. Zweitens wird das Simulationswerkzeug skizziert, das auch später verwendet wird.

Timing Architects Simulationswerkzeug

Das Timing Architects Tool Suite (TA Tool Suite) wurde 2011 im Kontext von Forschungsprojekten an der Fachhochschule Regensburg, der Continental Automotive GmbH und der Technischen Universität München entwickelt. Heute wird es zusätzlich von der Firma Vector Informatik vertrieben. Das TA Simulationswerkzeug deckt das System Design, die Simulation und Analyse, die Architektur- und Modulentwicklung sowie die Verifikation auf dem Zielsystem ab. [Roß14]

Das TA Tool Suite wurde mit dem Ziel entwickelt, Entwickler dabei zu unterstützen, manuelle und fehleranfällige Architekturentscheidungen automatisiert zu bewerten. Das kann zum Beispiel eine Verteilung von Software auf verschiedene Rechenkerne sein. TA Tool Suite ist an sich eine Sammlung von fünf verschiedenen Modulen. Im Folgenden werden die einzelnen Modulen kurz vorgestellt:

- Der *Timing Architects Explorer* dient dazu, die ganze Applikation, die Hardware und das Betriebssystem modellbasiert zu visualisieren und editieren.
- Der Timing Architects Designer erlaubt es auch Applikationen modellbasiert zu betrachten. Der Fokus liegt auf den Kommunikationsinterfaces und dem Datenaustausch zwischen Runnables verschiedener Software-Komponenten. Dadurch ist die Optimierungen der Software-Kommunikation möglich.
- Im *Timing Architects Simulator* werden basierend auf dem Modell sogenannte Simulationstraces erzeugt. Eine Simulation ermöglicht eine frühzeitige Überprüfung der Timing Aspekte bereits während der Architekturerstellung. Es wird sichergestellt, dass bestimmte Echtzeitanforderungen mittels eines Modells eingehalten werden. Das kann anhand einer Liste von Ereignisse, die das Verhalten der Applikation darstellen, durchgeführt werden .
- Letztendlich findet der *Timing Architects Optimizer* automatisch Lösungen und der User kann eine mögliche bessere Lösung auswählen und verwenden. Dadurch bietet Timing Architects Simulationswerkzeug die Möglichkeit sowohl die Performanz als auch das Reaktion-Verhalten von Multicore Systemen zu verbessern.
- *Timing Architects Inspector* ist für die Überprüfung des Zeitverhaltens der Anwendungssoftware und des Betriebssystems verantwortlich. Auf diese Weise können bereits aufgezeichnete Trace-Messungen von Drittanbietern in Bezug auf Reaktionszeiten, Auslastung und andere Kennzahlen analysiert werden.

Was ist aber eine Zeitsimulation im Kontext von Timing Architects Simulationswerkzeug?

Eine Zeitsimulation betrachtet das dynamische Verhalten eines Softwaresystems, indem sie dessen Scheduling-Verhalten simuliert. Die Simulation basiert auf einem Modell, das im Wesentlichen eine abstrakte Beschreibung des untersuchten Systems ist.

In einem frühen Entwicklungsstadium, z.B. einem Timing-Simulation kann helfen, den zukünftigen Verarbeitungsbedarf des Softwaresystems abzuschätzen. Bereits jetzt können diese Informationen über das zu entwickelnde System genutzt werden, um Designentscheidungen zu treffen, z.B. über die benötigte Hardware oder wenn bereits verfügbare Hardwareplattformen den Ressourcenbedarf decken. Mit einem erhöhten Detaillierungsgrad, der in späteren Entwicklungsschritten zur Verfügung steht, kann das Simulationsmodell Schritt für Schritt verfeinert werden. Hierdurch kann das Verhalten des realen Systems mit jeder Information genauer bewertet werden.

2.6. Auswahl der verwendeten Technologien

In diesem Abschnitt wird die Auswahl der in dieser Arbeit verwendeten Technologien vorgestellt und begründet.

2.6.1. Auswahl des Austauschformats

Die Wahl für das Austauschformat fiel auf AUTOSAR, da sich dieses als gängigstes Format in der Automobilindustrie etabliert hat. Dadurch wird eine Kompatibilität mit einer Vielzahl an Werkzeugen und Produkten garantiert. Aus diesem Grund wurde die Entscheidung gegen AMALTHEA getroffen, obwohl dieses im selben Forschungsprojekt entwickelt wurde wie das hier eingesetzte Simulationswerkzeug Timing Architects (vgl. 2.5.2).

2.6.2. Auswahl des Modellierungswerkzeugs

Zur Modellierung der Multicore-Eigenschaften wurde das Werkzeug PREEvision ausgewählt. Dieses unterstützt zum einen das AUTOSAR, zum anderen wird es mit Vector Informatik GmbH von einem der wichtigsten Anbieter von E/E-Architekturen entwickelt, was wichtig in Bezug auf die Zukunftsperspektiven der Arbeit ist.

2.6.3. Auswahl des Simulationswerkzeugs

Da es ebenfalls AUTOSAR unterstützt, wurde Timing Architects (TA) als Simulationswerkzeug ausgewählt. Außerdem ist eine wichtige Eigenschaft von TA, dass es spezifische Elemente und Vorgaben für Timing und Multicoresystem-Modelle mitbringt.

2.7. Fazit

In diesem Kapitel wurden die Grundlagen für diese Arbeit erläutert. Zuerst wurden die Eigenschaften einer E/E-Architektur vorgestellt. Anschließend wurden die ausgewählten Tools und Technologien eingeführt, namentlich AUTOSAR als Austauschformat, das Konzept der Modelltransformation und den Aspekt der Multicore-Technologie bei der Fahrzeugentwicklung als eines der zentralen Themen dieser Arbeit. Abschließend wurde die Auswahl der verwendeten Werkzeuge und Technologien präsentiert und erläutert.

3. Analyse

In diesem Kapitel wird eine Analyse der in dieser Arbeit zu entwickelnden Werkzeugkette vorgenommen. Konkret besteht diese Analyse daraus, die Anforderungen zu identifizieren, die von den Simulationswerkzeugen erfüllt werden sollten. Anschließend wird der Modellierungsansatz für E/E-Architekturen PREEvision und sein zugrundeliegendes Systemmodell analysiert. Die Modellierung und die Analyseansätze stellen die Grundlagen für die Werkzeugkette dar, die durch diese Arbeit zur Simulation von Echtzeit-Multicore-Systemen beigetragen wird. Außerdem werden die Möglichkeiten für eine frühzeitige Modellierung von Echtzeitbeschränkungen analysiert, die AUTOSAR durch seine Timing Extension-Spezifikation bereitstellt. Mittels der identifizierten Multicore-Eigenschaften wird ein Fallbeispiel vorgestellt. Anhand dieses Beispiels wird anschließend ein Konzept für die zu entwickelnde Werkzeugkette entwickelt (vgl. Kapitel 4).

3.1. Systemmodell von AMALTHEA und Identifizierung der Multicore-Parameter

In diesem Abschnitt wird das Systemmodell von AMALTHEA dargestellt und anschließend als Grundlage für die Identifizierung der Multicore-Parameter verwendet.

3.1.1. Systemmodell von AMALTHEA

Da AMALTHEA in seinem Systemmodell die höchste Übereinstimmung mit dem Simulationswerkzeug vorweist und öffentlich zugänglich ist, und wird in der Abbildung 3.1 dargestellt. Es wird von AMALTHEA in [AMA14] wie folgt beschrieben, wobei hier Ergänzungen nach [Roß14] hinzugefügt werden:

1. **Software-Beschreibung:** Die Software-Beschreibung stellt Informationen über das statische oder dynamische Verhalten der Software bereit. Dazu gehören beispielsweise Aufgaben (Tasks), Softwarekomponenten, Schnittstellen und Variablen. Außerdem können spezifische Eigenschaften der Software beschrieben werden. Dazu gehört z.B. der lesende oder schreibende Zugriff auf Variablen.

2. **Hardware/ECU Beschreibung:** Hier werden die abstrakten Informationen über die Hardware bereitgestellt. Beispiele sind Anzahl und Eigenschaften der Kerne, der verfügbare Speicher und die Zugriffszeiten der Kerne auf dem Speicher. Der Detailgrad der Modellierung kann in dieser Beschreibung stark variieren. Er reicht von der Anzahl der Rechenkerne und deren Verarbeitungsleistung bis hin zu einer detaillierten Speichertopologie und Verhaltensbeschreibungen.

3. **Timing-Einschränkungen:** Zu den Timing-Einschränkungen gehören Timing-Informationen wie End-to-End-Verzögerungen, Latenzen, Synchronisation und Ausführungszeiten von Events. Sie können formal auch durch die von AUTOSAR spezifizierten *Timing Constraints* beschrieben werden.

4. **Mapping Einschränkungen:** Zu den Mapping-Einschränkungen gehören alle spezifischen Informationen über die Prozessorkerne. Diese sind beispielsweise I/O-Verbindungen oder die maximale Taktrate. Um eine optimale Performance zu gewährleisten, müssen diese Informationen beschränkt werden. Allerdings müssen aus Sicherheitsgründen einige Funktionen auf bestimmten Kernen angesiedelt sein. Solche Fälle werden in diesem Submodell dargestellt.

5. **Software Mapping:** Das Software-Mapping enthält alle Informationen, welche für die Zuordnung von Softwareeinheiten (z.B. Tasks, Runnables, etc.) zu den Prozessorkernen relevant sind. Zusätzlich stellt es auch die Zuordnung von Daten und Anweisungen zum Speicher zu Verfügung.

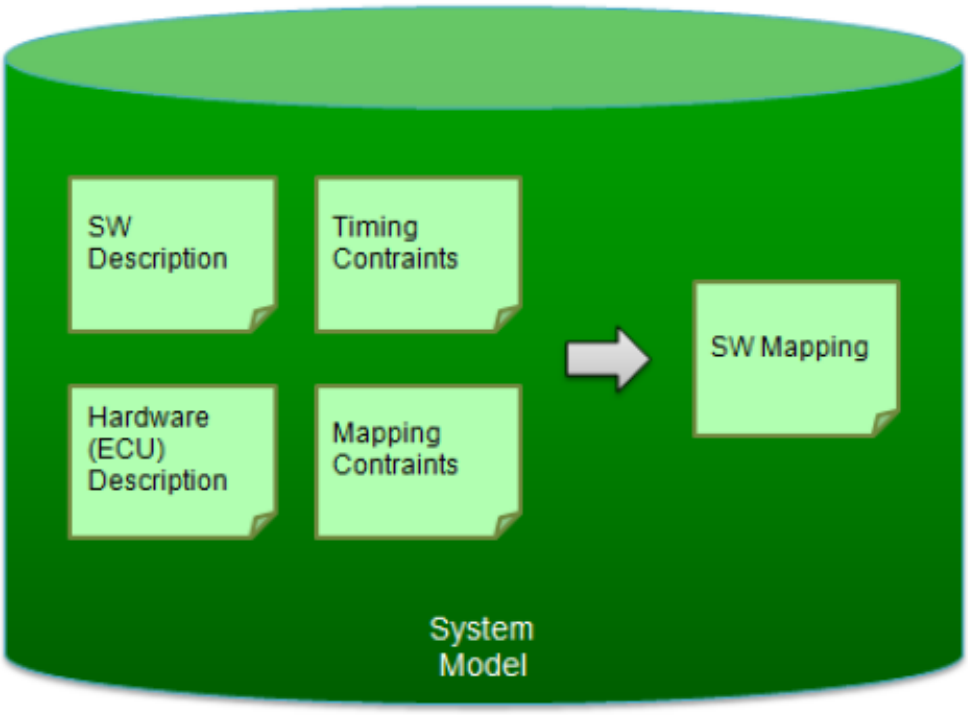

Abbildung 3.1.: Systemmodell von AMALTHEA [AMA14]

3.1.2. Identifizierte Multicore-Parameter

Bei der Berücksichtigung des zuvor vorgestellten Systemmodells sowie durch Literaturrecherche [Neg15] [Roß14] [KM07] wurden die folgenden Komponenten für die Beschreibung

eines Multicore-Systems herausgearbeitet. Diese sind im Wesentlichen die Beschreibung der Softwarestruktur, das Verhalten der festgelegten Softwarekomponenten, die darunterliegende Hardware, die Konfiguration des Betriebssystems sowie das Mapping von Softwarekomponenten zu den Prozessorkernen. Diese sind in der Tabelle 3.1 dargestellt.

Softwarestruktur	Software-Komponenten, Ports und Schnittstellen
Komponentenverhaltens	Runnables, Auftreten von Events und Aufgaben in periodischer, sporadischer und einzigartiger Weise
Multicore-ECUs	Anzahl der Kerne unter dem Prozessor des Steuergeräts, Prozessorbeschreibungen wie z.B. Taktfrequenzen
Konfiguration des Betriebssystems	OS-Anwendungen-Konfiguration, Zuordnung der OS-Objekte wie z.B. Tasks, Interrupts, etc.
Software-Mapping	Zuordnung von Softwarekomponenen zu den Prozessorkernen
Echtzeit-Fähigkeit	Eventketten und Timing-Beschränkungen

Tabelle 3.1.: Abstraktes Systemmodell im Multicore-Kontext

Um die Echtzeit-Fähigkeit eines Systems simulieren bzw. verifizieren zu können, wird zunächst ein Systemmodell mit besonderen Fokus auf Timing benötigt. Dann sollen die folgenden Systemattribute hinzugefügt werden, um ein verifizierbares Zeitmodell zu erstellen. Diese Systemattribute realisieren die in Abschnitt 3.2 erläuterten Prinzipien:

1. **Eventketten:** Beschreibungsevents werden verwendet, um das Systemverhalten zu beschreiben, das während der Laufzeit des Systems beobachtet werden kann. Eine Eventkette soll einen Stimuli-Reaktion-Zusammenhang zwischen den zugeordneten Beschreibungsevents spezifizieren.

2. **Timing Beschränkungen:**

   Latency begrenzt die Zeitdauer einer Eventkette.

   Offset gibt einen zeitlichen Offset zwischen den Events an.

   Triggering beschreibt das Auslösen des referenzierten Events.

   Input- bzw. Output-Synchronität (Synchronization) schränkt den maximalen Abstand zwischen zusammengehörenden Events ein. Es ist sowohl möglich, die Synchronität von Stimuli vorzugeben, als auch die Synchronität von Reaktionen.

   Alter (Age) gibt das Alter der Daten an, wenn sie an einer Softwarekomponente auf ihrem gewünschtem Port ankommen.

   Ausführungsreihenfolge legt die Reihenfolge der Ausführung einer Reihe von ausführbaren Entitäten fest.

   Ausführungszeit definiert minimale und maximale Ausführungszeitbeschränkungen von ausführbaren Entitäten.

Prozessorauslastung legt die maximale Prozessorauslastung fest.

3.2. AUTOSAR Timing Extensions

Wie zuvor erwähnt, mit der Version 4.0 enthält AUTOSAR AUTOSAR Timing Extensions (TIMEX) zur Formulierung zeitbezogener Anforderungen und Eigenschaften von Fahrzeugsystemen, einschließlich einer Methodik zur Bewältigung dieser Anforderungen. Diese Erweiterung ist in der offiziellen AUTOSAR-Spezifikation enthalten [AK17]. Die Timing Extension deckt verschiedene Ebenen der Implementierung ab. Die verschiedenen Ebenen sind [AK17]:

- Das *Virtual Functional Bus Timing (VfbTiming)* ist verantwortlich für die Spezifikation von Beschränkungen auf logischer Ebene. Dabei werden weder die Verteilung, noch das interne Verhalten oder die Ausführungsplattform von Softwarekomponenten berücksichtigt.

- Das *Software Component Timing (SwcTiming)* ist dafür zuständig, das interne Verhalten von Softwarekomponenten, insbesondere das Verhalten von Runnables, zu definieren.

- Das *System Timing (SystemTiming)* beinhaltet den Einsatz der Softwarekomponenten. Es kann demzufolge die Kommunikation zwischen den Komponenten entweder auf einen lokalen Kommunikationsprozess oder eine externe Kommunikationstechnologie abbilden.

- Das *Basic Software Module Timing (BswModuleTiming)* erfasst alle Einschränkungen in Bezug auf die Module der Basissoftware. Das bedeutet, dass es die Treiber und das Laufzeitsystem eines bestimmten elektronischen Steuergerätes definiert. Das *BswModuleTiming* ist außerdem ähnlich wie das *SwcTiming*, bezieht sich jedoch auf Basis-Softwaremodule, die während des Konfigurationsprozesses eines Steuergeräts bestimmt werden.

- Das *Electronic Control Unit Timing (EcuTiming)* ist die konkreteste Darstellungsebene für zeitliche Beschränkungen. Es enthält alle Informationen, die gebraucht werden, um den Zeitpunkt eines einzelnen elektronischen Steuergeräts zu bestimmen. Es liefert also eine vollständige Konfiguration des Steuergeräts inklusive der Zuordnung von Nachrichten und Signalen zu internen und externen Kommunikationsmitteln. Im Vergleich zum *SystemTiming* enthält diese Ansicht zusätzlich alle Informationen über die grundlegenden Softwaremodule und deren Zusammenspiel.

Die AUTOSAR bieten grundlegende Möglichkeiten, um Timing-Informationen zu beschreiben und zu spezifizieren. Das sind Timing-Beschreibungen, die durch Beschreibungsevents und Beschreibungseventketten ausgedrückt werden, und Timing-Constraints, die diesen Beschreibungsevents und Beschreibungseventketten auferlegt werden.

Timing-Beschreibungsevent

Das Timing-Beschreibungsevent (in der AUTOSAR-Spezifikation als *TimingDescriptionEvent* bezeichnet, vgl. [AK17]) repräsentiert ein zur Laufzeit beobachtbares spezifisches Systemverhalten.

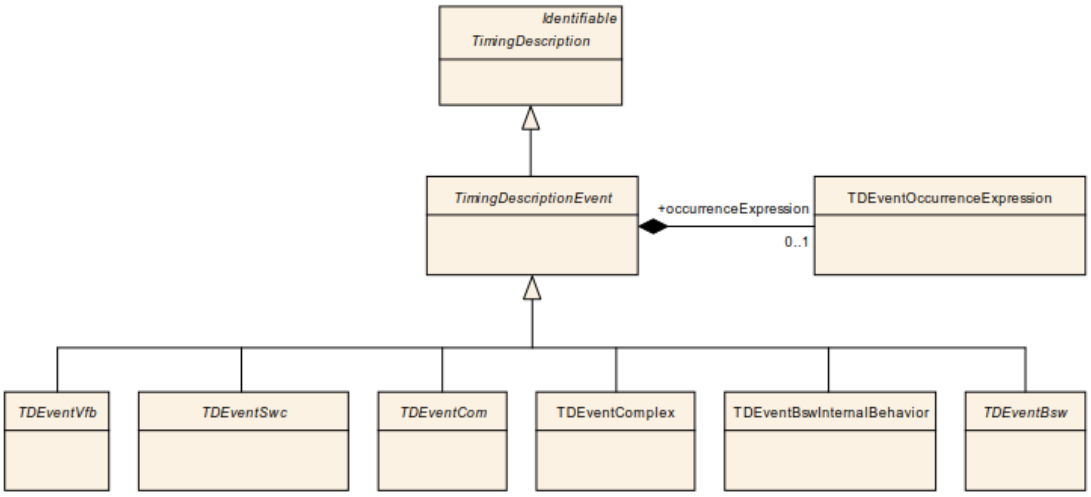

Abbildung 3.2.: Beschreibungsevent Metamodell-Definition [AK17]

Wie in Abbildung 3.2 dargestellt, definiert TIMEX eine Reihe von vordefinierten Timing-Events. Jedes Beschreibungsevent ist einer der zuvor dargestellten Timing-Ansichten zugeordnet.

Das Beschreibungsevent verdeutlicht somit, welche Events für eine bestimmte AUTOSAR-Ansicht relevant sind. AUTOSAR beschreibt beispielsweise das System in der VFB-Ansicht als eine Zusammensetzung von Softwarekomponenten. Ein vernünftiges Timing-Event, das in dieser Ansicht beobachtet bzw. beschrieben werden kann, ist dann *TDEventVariableDataPrototype*.

Timing-Beschreibungseventkette

Eine Timing-Beschreibungseventkette (*TimingDescriptionEventChain* [AK17]) spezifiziert die Beziehung zwischen zwei Beschreibungsevents. Der Zusammenhang zwischen einer Stimulation eines Systems und der Reaktion kann mit Hilfe einer solchen Beschreibungseventkette ausdrücklich beschrieben werden. Wie in Abbildung 3.3 dargestellt, wird eine Beschreibungseventkette durch mindestens zwei Beschreibungsevents definiert: den Stimulus und die Reaktion.

Beschreibungseventketten enthalten mehrere Komponenten, die in Eventkettensegmente unterteilt sind. Diese beeinflussen das Timing. Wie in Abbildung 3.4 abgebildet, hat eine Beschreibungseventkette einen Startpunkt. Dieser ist meist ein Port einer Komponente. Eine Beschreibungseventkette hat des weiteren einen Endpunkt. Dieser ist in der Regel auch ein Port einer Komponente oder eine Komponente selbst. Zwischen Startpunkt und Endpunkt muss ein gerichteter Weg der internen und externen Beziehungen der Kompo-

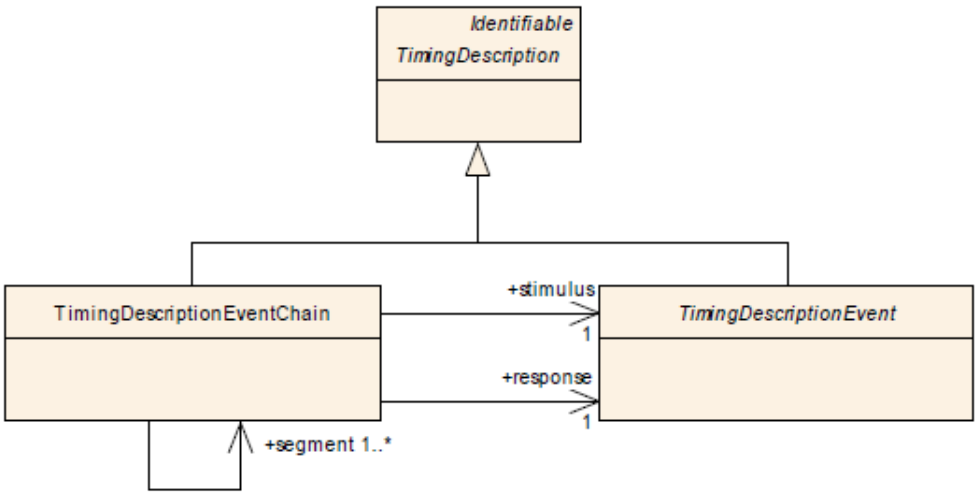

Abbildung 3.3.: Eventkette Definition [AK17]

nenten gegeben sein, um eine gültige Beschreibungseventkette zu bilden (siehe Abbildung 3.4]).

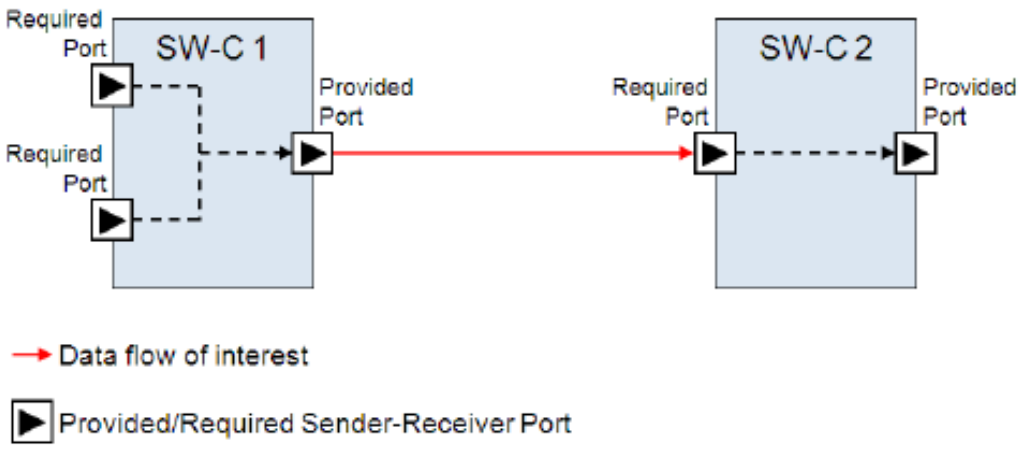

Abbildung 3.4.: VFB-Sicht Beispiel [AK17]

Timing-Beschränkungen

Durch Timing-Beschreibungsevents und Timing-Beschreibungseventketten können nur kausale Zusammenhänge zwischen beobachtbarem Systemverhalten vorgegeben werden. Um auch ein einzelnes Beschreibungsevent oder eine Gruppe von Beschreibungsevents mit zeitlicher Einschränkung zu klassifizieren, stellt TIMEX einen Satz von Zeitbeschränkungen zur Verfügung. Beispiele für diesen Anwendungsfalls sind ein Zeitraum, eine Latenzzeit oder ein Zeitintervall. Solche Zeitbeschränkungen können auf Timing-Beschreibungsevents, Timing-Beschreibungseventketten oder eine geordnete Liste von

ausführbaren Entitäten (*Runnables*) angewendet werden. [AK17] Im Rahmen von TIMEX gibt es folgende Einschränkungen:

- Eine Beschränkung der Art ***EventTriggeringConstraint*** wird einem Beschreibungsevent auferlegt. Sie dient der Angabe eines Aktivierungsmusters, also beispielsweise periodisch oder sporadisch.
- Ein ***LatencyTimingConstraint*** wird ebenfalls einer Beschreibungseventkette auferlegt. Es hat die Funktion, die Zeitspanne zwischen dem Auftreten von zwei beliebigen Events zu begrenzen.
- Ein ***AgeConstraint*** wird einem *TDEventVariableDataPrototype* auferlegt. Es dient dazu, die Zeitspanne zu begrenzen, in der ein Variablenwert empfangen wird.
- Ein ***SynchronizationTimingConstraint*** wird entweder einem Beschreibungsevent oder einer Beschreibungseventkette auferlegt. Es dient dazu, die Zeit zwischen dem Auftreten von zwei oder mehr Events zu einzuschränken.
- Ein ***OffsetTimingConstraint*** wird auf zwei Beschreibungsevent angewendet. Es dient zur Angabe eines Offsets bzw. einer Verzögerung zwischen dem Auftreten zweier Events.
- Ein ***ExecutionOrderConstraint*** wird einer beliebigen Anzahl von ausführbaren Entitäten auferlegt. Es hat die Funktion, eine Sequenz zwischen ebendiesen anzugeben.
- Ein ***ExecutionTimeConstraint*** wird einer ausführbaren Einheit auferlegt. Es dient der Angabe einer minimalen und maximalen Zeit.

Eine Zeitbeschränkung kann in zwei verschiedenen Rollen verwendet werden, *timingRequirement* und *timingGuarantee*. Im Fall der erstgenannten Rolle ist das System verpflichtet, den durch eine Einschränkung festgelegten Zeitpunkt zu erfüllen. Im Fall der letztgenannten Rolle kann das System den angegebenen Zeitpunkt garantieren.

Vorteile von Timing-Beschränkungen

Zeitliche Anforderungen und Garantien sind im Entwicklungsprozess von Embedded-Anwendungen von größter Bedeutung. Die Systembeschränkungen werden jedoch in einer Vielzahl von verschiedenen Tools [SAG12] und Timing-Sprachen [Gro] beibehalten. Mit TIMEX können Entwickler ihre Einschränkungen selbst festlegen und aufgrund des standardisierten Formats in verschiedenen Werkzeugen verwenden.

So ist es beispielsweise über TIMEX unkompliziert möglich, Anforderungen zwischen mehreren Werkzeugen wie PREEvision und Timing Architects Simulationswerkzeug auszutauschen. Dies würde es ermöglichen, automatisch gemessene Metriken im Timing Architects Simulationswerkzeug ohne manuelle Schritte zu validieren.

3.3. Analyse des Modellierungswerkzeugs PREEvision

Im Kapitel 2.2.3 wurden die dedizierten Abstraktionsebenen für die Modellierung eines E/E-Systems vorgestellt. In dem folgenden Abschnitt wird das bestehende Metamodell von PREEvision genauer analysiert. Ziel ist es die bestehenden bzw. fehlenden Strukturen für die zuvor identifizierten Multicore-Parameter herauszuarbeiten.

Software Architektur

Für die Modellierung der Software-Architektur unterstützt PREEvision alle wichtigsten Aspekte des AUTOSAR Software Component Templates [AK14]. Dadurch können Softwarekomponenten hierarchisch strukturiert werden. Es kann auch eine Komposition von Softwarekomponenten modelliert werden. Eine Komposition ist ein gruppierendes Artefakt zur Strukturierung von Softwarekomponenten sowie deren Ports. Sender- und Empfängerport liefern die Informationen über verfügbare Operationen und Datenelemente einer Softwarekomponente. Mit Hilfe einer Schnittstellenzuordnung kann die Zuordnung zwischen einem Port und einer Schnittstelle definiert werden. Das interne Verhalten einer Softwarekomponente wird in PREEvision durch ihre lauffähigen Einheiten (Runnables) und die Events, auf die sie reagiert, beschrieben. Eine Runnable ist das kleinste Codefragment, das für eine Softwarekomponente beschrieben wird, das typischerweise in der C-Sprache implementiert wird. Datenzugriffe können ebenfalls modelliert werden und beschreiben, wie eine Runnable auf Daten zugreift oder Daten über Ports bereitstellt.

Hardware Architektur

Die Hardwarekomponentenschicht unterstützt zwei Aspekte der Hardwaremodellierung. Zum einen können externe logische Verbindungen zwischen Hardwarekomponenten (z.B. ECUs, Sensoren, Aktoren, etc.) und deren Vernetzung definiert werden, wie z.B. konventionelle Verbindungen, Bussysteme und sogar Stromversorgungs- und Masseverbindungen. Auf der anderen Seite der Inhalt der ECUs, die inneren Details, wie z.B. die Prozesseinheiten, die Kerne oder Speicher, können modelliert werden. Die Modellierung von mehrere Kerne unter dem gleichen ECU ist für diese Abschlussarbeit von einer großen Bedeutung, da wir uns jetzt nämlich in dem Kontext von Multicore-ECUs befinden.

Mapping in PREEvision

Nachdem das Soft-und Hardware-Design abgeschlossen wurden, unterstützt PREEvision auch das Software/Hardware-Mapping. Unter Software/Hardware-Mapping versteht man den Einsatz jeder einzelnen Softwarekomponente auf einem Steuergerät. Das Mapping kann zusätzlich auf einem bestimmten Kern zugewiesen werden, was die letzte Anforderung von der Tabelle 3.1 abdeckt.

Timing-Beschränkungen

PREEvision unterstützt ein Konzept zur Festlegung von Zeitvorgaben für die Definition von zeitkritischem Verhalten.

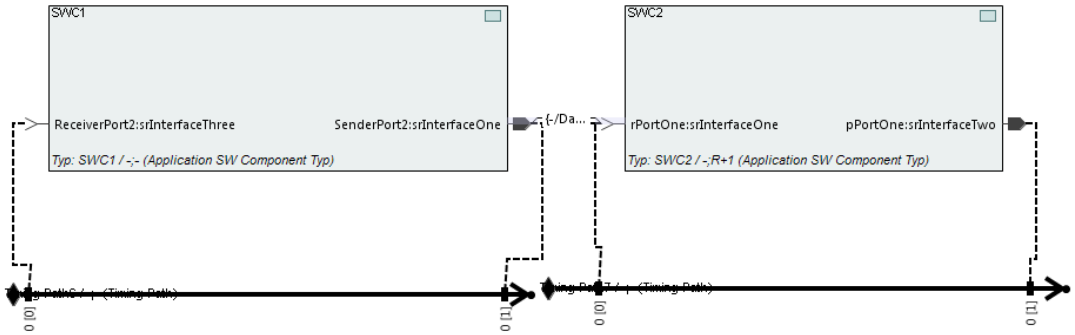

Abbildung 3.5.: Timing-Einschränkungen in PREEvision

Die Timing-Beschränkungen in PREEvision definieren eine so genannte End-to-End-Zeitspanne (in der Abbildung 3.5 als *TimingPath* dargestellt). Diese Zeitspanne stellt den Orientierungswert für die gewünschte Gesamtausführungszeit einer Funktion dar. Das Gesamtzeitverhalten kann in einzelne Zeitintervalle unterteilt werden (*TimingPathSlots*). Auf der Abbildung sind diese mit einer unterbrochenen Linie dargestellt, die von einem Punkt, der auf dem Timing Path liegt, anfängt. Die *TimingPathSlots* referenzieren einen bestimmten Port der entsprechenden Softwarekomponente und daher bildet die davor vorgestellte *VfbTiming*-Sicht fast vollständig ab. Die für einen *TimingPathSlot* angegebene Zeit ist die Zeitdifferenz zwischen einem Punkt und einem anderen. Sie kennzeichnet die Zeit, die - im Vergleich zur gewünschten Gesamtausführungszeit - vergehen darf, bis eine Reaktion oder ein anderes Ereignis eintritt.

Fehlende Betriebssystemschicht

PREEvision unterstützt die Modellierung von Tasks bzw. die Konfiguration der Betriebssystemschicht derzeit nicht, da es den Fokus auf eine höhere Abstraktionsebene legt. Abbildung 3.6 stellt eine beispielhafte Betriebssystemkonfiguration schematisch dar. Die Zuordnung von Runnables auf OS-Tasks, die Zuweisung dieser Tasks zu einer OS-Applikation sowie deren Zuordnung zu den Prozessorkernen ist sehr wichtig, damit die Konsistenz des Datenaustausch gesichert bzw. überprüft werden kann. [Inf12] [Infb] [AMA17]

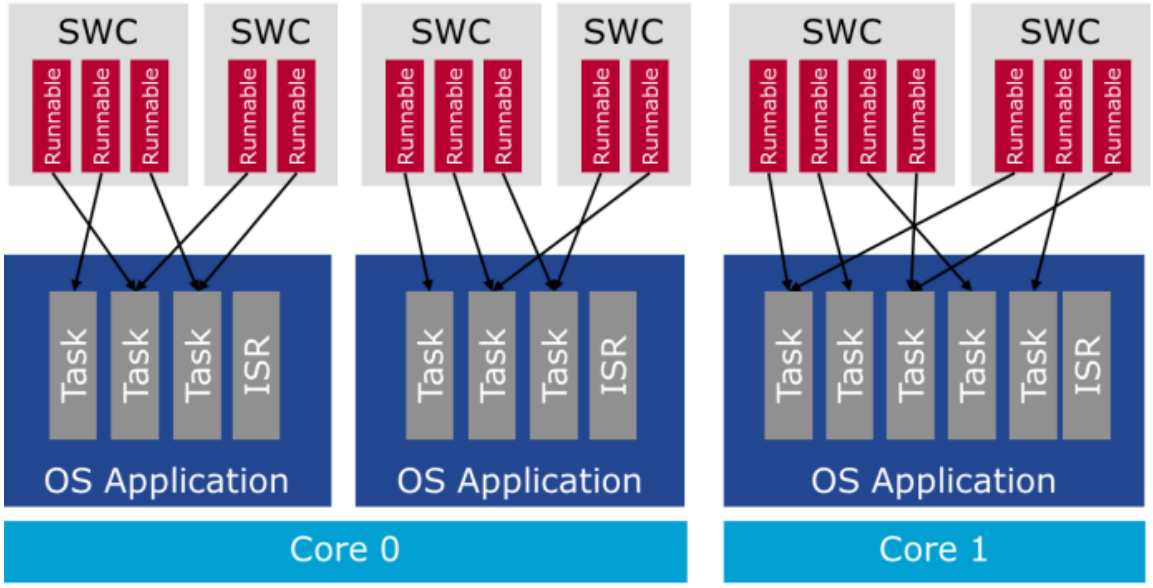

Abbildung 3.6.: Zuordnung von Softwarekomponenten, Aufgaben (Tasks) und OS-Instanzen zu den Prozessorkernen (schematische Darstellung) [Infb]

3.4. Fallbeispiel (UseCase) Festlegung

Das folgende Fallbeispiel wird definiert, um zu untersuchen, ob die in Unterabschnitt 3.1.2 aufgeführten, zur Simulation notwendigen Informationen mit PREEvision modelliert und im AUTOSAR-Format exportiert werden können.

Die Analyse der Systemmodelle von AMALTHEA, AUTOSAR und PREEvision hat dazu beigetragen, die wichtigsten Punkte zu erfassen, um das System so zu beschreiben, dass die Simulation dieses Systems danach möglich ist (Abbildung 3.7). Zunächst müssen die Soft- und Hardwarekomponenten und ihre Beziehungen modelliert werden. Die Beschreibung der Hardware beinhaltet die ECUs mit ihren Prozessoren bzw. Kernen. Die Beschreibung der Software umfasst die Modellierung von Softwarekomponenten, deren gegenseitige Abhängigkeiten, allgemeine Ein- und Ausgänge, deren internes Verhalten (Runnables) und das Auftreten von Events. Wie sich bereits gezeigt hat, kann PREEvision mit diesen Aspekten umgehen. Die Zuordnung von Softwarekomponenten auf den Kernen des Prozessors sind der nächste wichtige Punkt, der unser Systemmodell abstrakt vervollständigen muss. Mit TIMEX können Entwickler Zeitliche Anforderungen und Garantien direkt festlegen und aufgrund des standardisierten Formats in verschiedenen Werkzeugen verwenden.

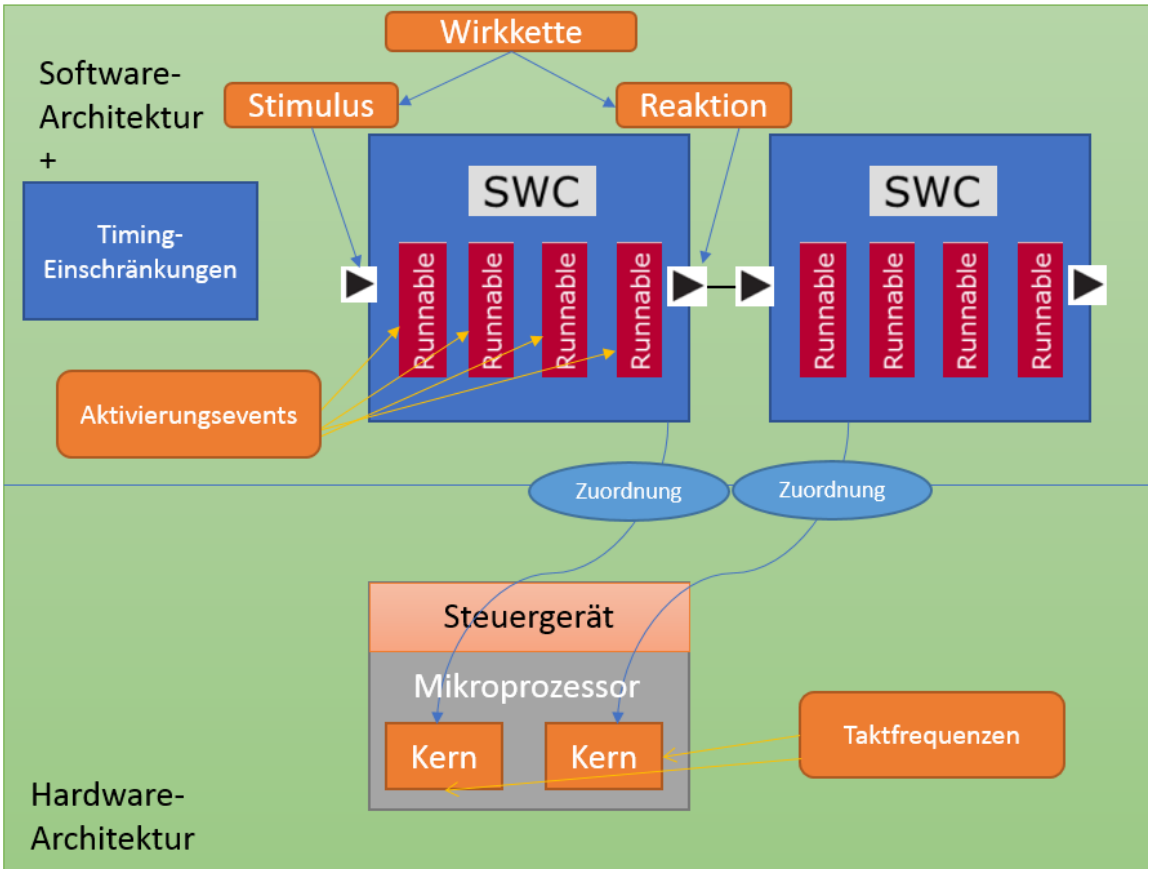

Abbildung 3.7.: Fallbeispiel anhand der identifizierten Multicore-Eigenschaften

3.5. Zusammenfassung

In diesem Kapitel wurden durch die Analyse des Systemmodells von AMALTHEA sowie durch Literaturrecherche [Neg15] [Roß14] [KM07] die notwendigen Multicore-Eigenschaften identifiziert. Anschließend wurden die Möglichkeit für Echtzeitbeschränkungen mittels AUTOSAR analysiert. Daraufhin wurde das Modellierungswerkzeug PREEvision genauer untersucht. Dabei wurden die Herausforderungen für das zu entwickelnde Konzept herausgearbeitet. Zuletzt wurde ein Fallbeispiel vorgestellt, welches die in der Analyse ermittelten Besonderheiten abdeckt.

4. Konzept für die Entwicklung der Werkzeugkette

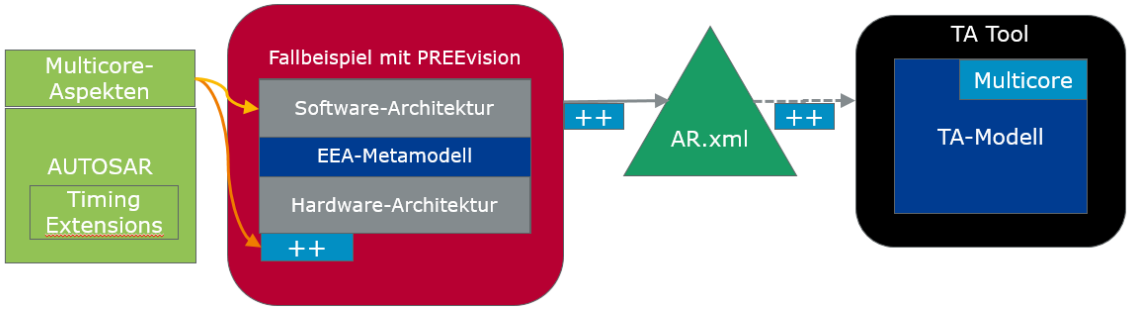

Abbildung 4.1.: Werkzeugkette Konzept

Abbildung 4.1 zeigt das Konzept für die Werkzeugkette. Der Workflow beginnt mit dem System-Design. Das im Abschnitt 3.4 vorgestellte Fallbeispiel wird mit dem Modellierungswerkzeug PREEvision modelliert. Dabei wird eine Erweiterung des Metamodells von PREEvision vorgeschlagen (vgl. linkes ++). Eine Erweiterung der AUTOSAR-Exportschnittstelle von PREEvision wird ebenfalls umgesetzt. Anschließend wird die korrespondierende AR-XML exportiert (vgl. mittleres ++). Das wird mit AUTOSAR exportierten Systemmodell in das Simulationswerkzeug importiert. Zuvor wurde festgestellt, dass PREEvision die Betriebssystemkonfiguration nicht unterstützt. Dabei sucht das Simulationswerkzeug eine Konfiguration für das Betriebssystem. Dies wird durch das im Rahmen dieser Arbeit erarbeitete Konzept für das AUTORSAR-kompatible *SWC-Prozessorkerne*-Mapping und die nachfolgende Synthese ermöglicht (vgl. rechtes ++). Bei der Synthese wird versucht, durch die Kombination von Eingaben, in diesem Fall SWC-Prozessorkerne, sowohl eine Zuordnung von mit PREEvision modellierten Runnables auf OS-Tasks als auch eine Zuweisung dieser Tasks zu der generierten OS-Applikation zu erzeugen (vgl. [Aho08] und [Mey14]). Die Erzeugung einer korrekten Konfiguration führt dazu, dass das System lauf- bzw. simulationsfähig ist. Die nachfolgende Simulation berücksichtigt die entsprechenden *TIMEX*-Echtzeitbeschränkungen.

4.1. Erweiterung des PREEvision-Metamodells

Das bestehende Konzept für die Modellierung von Timing-Beschränkungen (Abschnitt 3.3) ist zu allgemein und wird erweitert, dass es mit dem TIMEX-Konzept vereinbar ist. Der in 3.4 festgelegte Umfang des in dieser Arbeit betrachteten Fallbeispiels umfasst die

Modellierung des internen Verhaltens von Software-Komponenten sowie deren Kommunikation untereinander. Dies entspricht in AUTOSAR der Vfb- bzw. SWC-Sicht, weswegen diese im weiteren Verlauf betrachtet werden. Die folgenden Unterpunkte zeigen die Metamodelländerungen in PREEvision. Zur Verdeutlichung der neu hinzugekommenen Metamodellbestandteile, sind diese in den Abbildungen farblich gekennzeichnet.

4.1.1. Modellierung von Beschreibungsevents und Eventketten

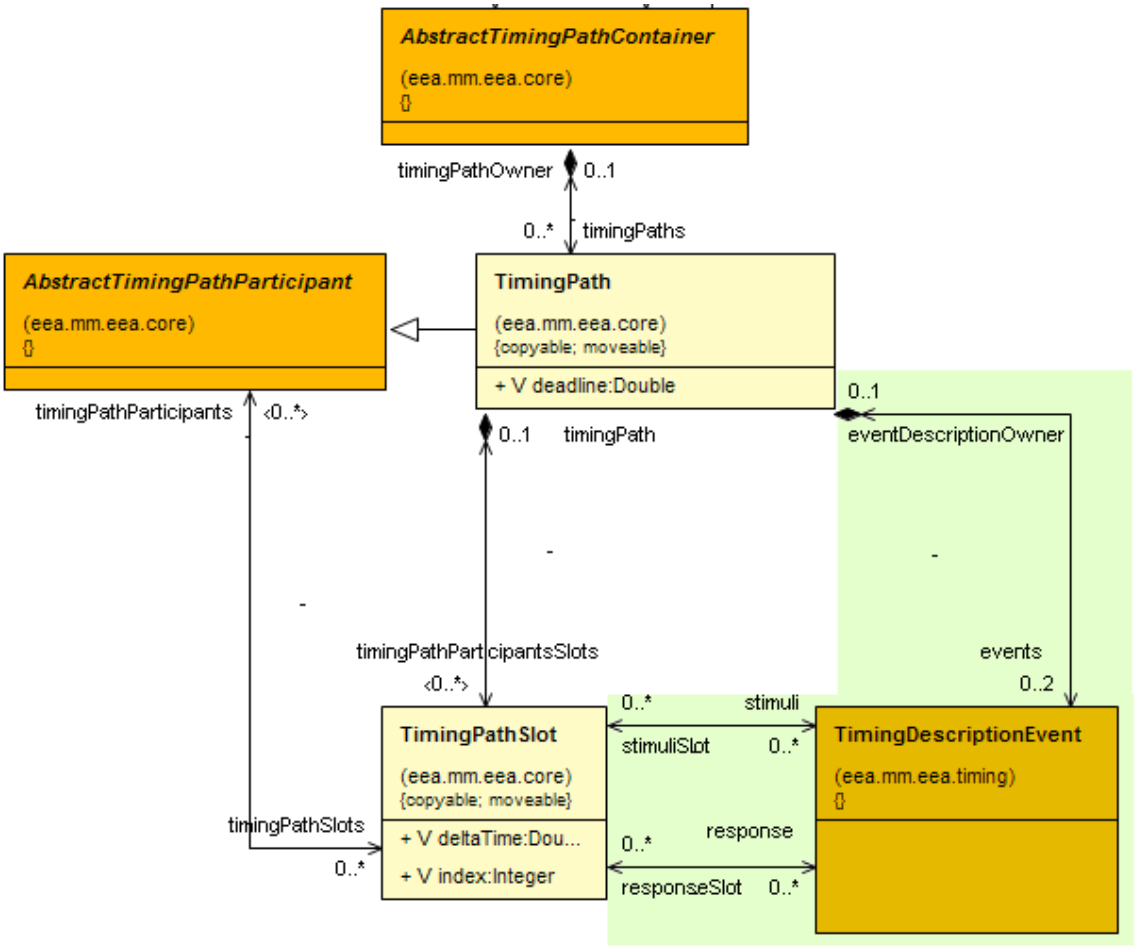

Abbildung 4.2.: Metamodell der Beschreibungsevents und Eventkettn

Im PREEvision Metamodell sollen verschiedene Arten von Events beschrieben werden. Ein Beschreibungsevent kann Stimulus oder Reaktion eines Slots sein. Das wurde mit der abstrakten Klasse *TimingDescriptionEvent* abgebildet. Abbildung 4.2 zeigt die hinzugefügte Klasse sowie die benötigten Assoziationen und Kompositionen im Bezug der schon existierenden *TimingPath* und *TimingPathSlot*. Dadurch ist die Unterscheidung zwischen den verschiedenen Beschreibungsevents auch möglich.
Damit die *TimingPath* als eine Eventkette betrachtet werden kann, sind einige Annahmen und Anpassungen des aktuellen Metamodells notwendig.

Ein *TimingPath*-Objekt muss immer nur zwei TimingPathSlots beinhalten. Um eine einfache Eventkette zu definieren, genügt ein TimingPath-Objekt und zwei TimingPathSlots, die auf zwei Beschreibungsevents (*TimingDescriptionEvents*) verweisen, ein Stimulus-Beschreibungsevent und ein Reaktion-Beschreibungsevent. In diesem Fall wäre es nur eine Kette, die mit ihrem Stimuli als erstes Beschreibungsevent und der Reaktion als zweites Beschreibungsevent. Wenn mehr Beschreibungsevents erforderlich sind, ist es notwendig, einen neuen *TimingPath* hinzuzufügen. Die Vererbung zwischen den Klassen *AbstractTimingPathParticipant* und *TimingPath* und die Assoziation *timingPathParticipants* ermöglichen die Segmentierung von mehreren *TimingPaths*.

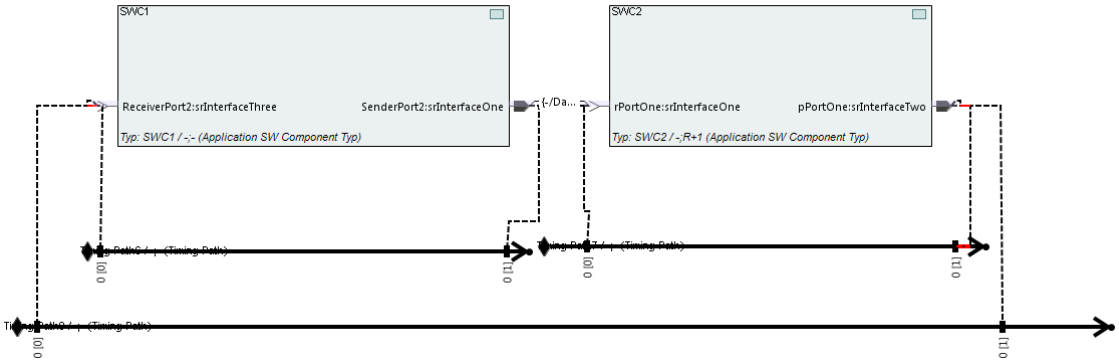

Abbildung 4.3.: Segmentierung von *TimingPaths* im PREEvision

Abbildung 4.3 zeigt wie eine beispielsweise Darstellung der Segmentierung von Eventketten im PREEvision aussieht.

Beschreibungsevents auf logischer Ebene

Die *TimingPathSlots* beziehen sich auf die Ports einer Softwarekomponenten und daher sind eine gute visuelle Darstellung von Eventketten auf logischer Ebene.

Abbildung 4.4 zeigt die Metamodellerweiterung für die Festlegung von Beschreibungsevents, die sich auf die Beziehungen zwischen den Ports einer Softwarekomponente beziehen. Die Metaklasse *TDEventVariableDataPrototype* erbt von der abstrakten Klasse *TimingDescriptionEvent*, die an den erforderlichen und bereitgestellten Sender-/Empfängerports beobachtbar sind. *TDEventVariableDataPrototype* hat ein Attribut *tdEventVariableDataPrototypeType*. Dies beschreibt den spezifischen Eventstyp eines TDEventvariableDataPrototype durch die Enum-Klasse *TDEventVariableDataPrototypeTypeEnum* wie folgt:

1. *VARIABLE-DATA-PROTOTYPE-SENT* gibt den Zeitpunkt, zu dem der referenzierte Daten-Element erfolgreich von der sendenden Softwarekomponente versendet wurde.

2. *VARIABLE-DATA-PROTOTYPE-RECEIVED* gibt den Zeitpunkt, zu dem der referenzierte Daten-Element erfolgreich übertragen wurde.

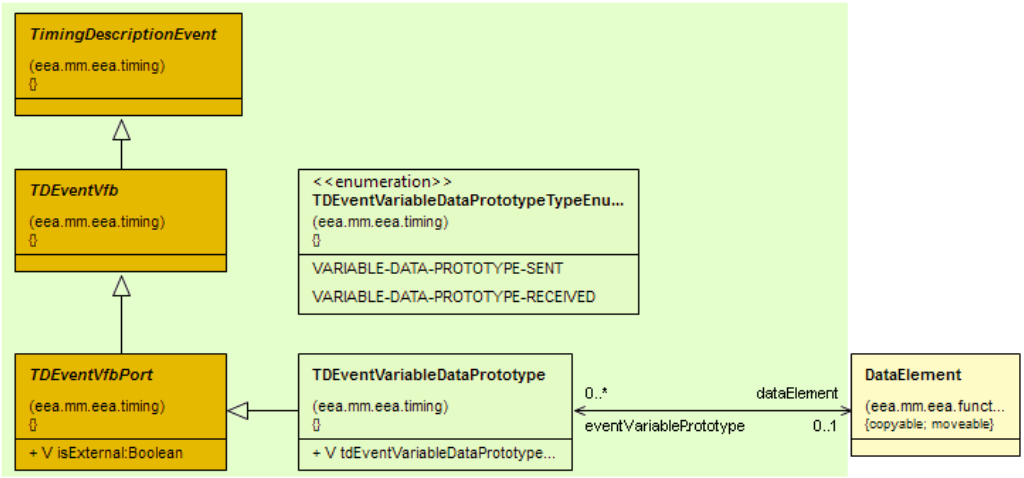

Abbildung 4.4.: Metamodell der Beschreibungsevents auf logischer Ebene

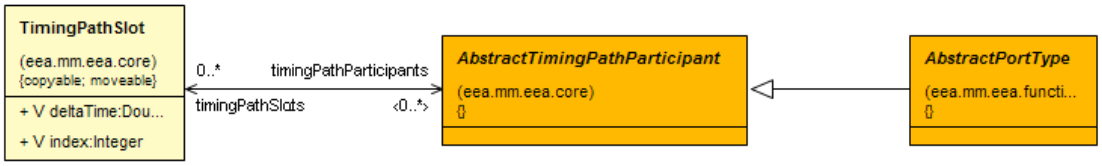

Abbildung 4.5.: Darstellung der Assoziation zwischen *TimingPathSlots* und *AbstractPortType*

Abbildung 4.5 zeigt das bereits bestehende Metamodellkonstrukt, mit dem die neue *TDEventVariableDataPrototype*-Metaklasse an den PortTypen angebunden werden. Bei der Erstellung eines Slot wird der entsprechende Port automatisch dem Port zugewiesen.

Die oberen zwei abstrakten Klassen (*TDEventVfb* und *TDEventVfbPort*) sind für die zukünftigen Erweiterungen des Metamodells gedacht. Solche können weitere Beschreibungsevents sein und die von AUTOSAR spezifizierte Möglichkeit für Wiederwervendung von solche. Diese wurden im Rahmen dieser Arbeit nicht berücksichtigt wurden.

Beschreibungsevents des internen Verhaltens einer Softwarekomponente

Die Metaklasse *TDEventSwcInternalBehavior* wird zur Angabe von Events verwendet, nämlich Aktivierung, Start, Beendigung von ausführbaren Entitäten sowie variable Zugriffe, die in der Ansicht des internen Verhaltens von Softwarekomponenten beobachtet werden. Abbildung 4.6 zeigt die neue konkrete Metaklasse. Das Attribut *tdEventSwcInternalBehaviorType* und dafür definierte *TDEventSwcInternalBehaviorTypeEnum*-Klasse beschreibt den spezifischen Eventstyp. Die Assoziationen *runnable* und *variableAccess* werden verwendet, um das Scope des auftretenden Events festzulegen.

Wie in dem vorstehend beschriebenen Konzept wurde die Metaklasse *TDEventSwc* um eine zukünftige und vollständige Abbildung der TIMEX [AK17] zu ermöglichen.

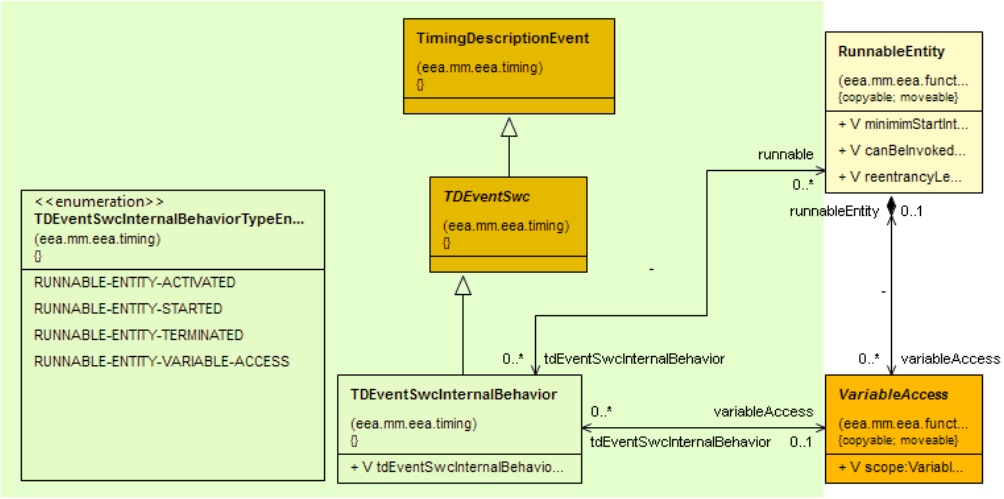

Abbildung 4.6.: Metamodell der Beschreibungsevents des internen Verhaltens einer Software Komponente

4.1.2. Modellierung von Timing-Beschränkungen

Im folgenden Abschnitt werden die Metamodellerweiterungen beschrieben, die notwendig sind, um die in Kapitel 3.2 analysierten Timing Beschränkungen abbilden zu können.

Latenzzeit Beschränkung

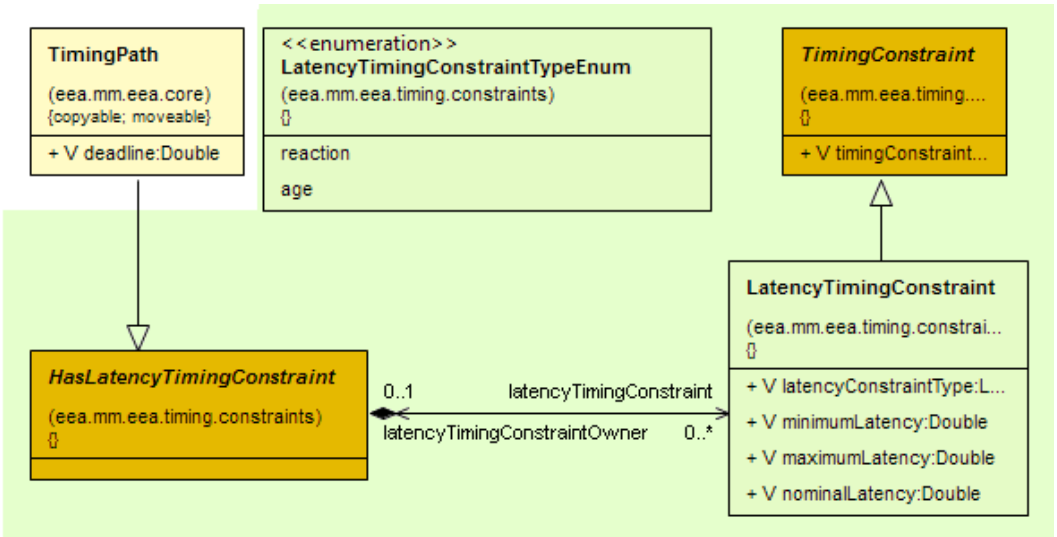

Abbildung 4.7.: Metamodell für Latenzzeit Beschränkung

Abbildung 4.7 zeigt die Metamodellerweiterung für die Modellierung vom *LatencyTimingConstraint*. Die Vererbung zwischen dem *TimingPath* und der *HasLatencyTimingConstraint* bzw. die Komposition-Beziehung zwischen der *HasLatencyTimingConstraint* und der

LatencyTimingConstraint ermöglichen jedoch die Erstellung der Latenzzeit unter einem *TimingPath*.

Das Alter einer bestimmten Reaktion und die Reaktion auf einen bestimmten Stimulus und wird durch die Klasse *LatencyTimingConstraintTypeEnum* bzw. das Attribut *latencyConstraintType* representiert. Die anderen drei Attribute der Klasse *LatencyTimingConstraint* sind *minimumLatency*, *maximumLatency* und *nominalLatency* repräsentieren die minimale, maximale sowie die nominelle Zeit zwischen dem Auftreten des Stimulus und dem Auftreten der entsprechenden Reaktion der zugehörigen Eventkette.

Ausführungszeit Beschränkung

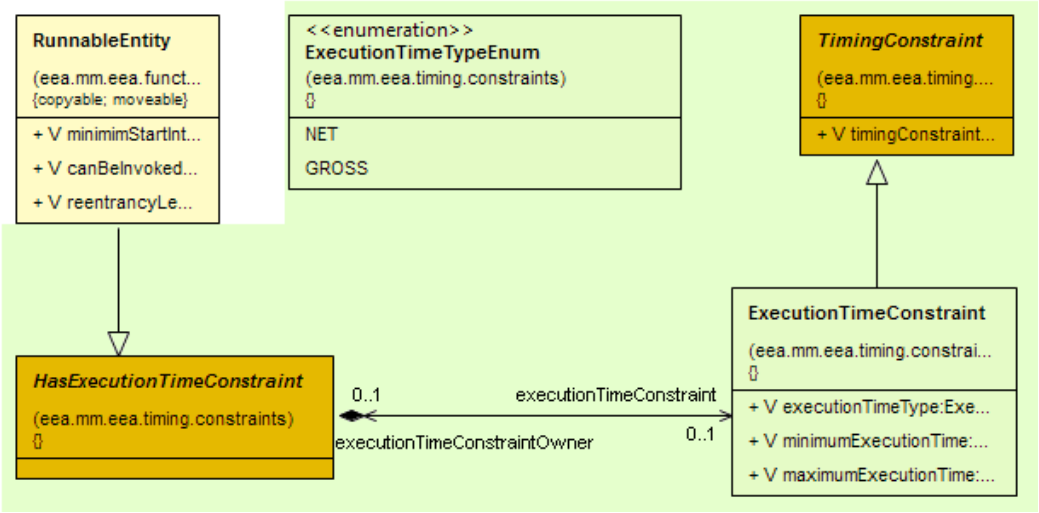

Abbildung 4.8.: Metamodell für Ausführungszeit Beschränkung

Abbildung B.8 zeigt die Metamodellerweiterung für die Modellierung von Ausführungszeit Beschränkung. Die Vererbung zwischen dem *RunnbaleEntity* und der *HasExecutionTimeConstraint* bzw. die Kompositionsbeziehung zwischen der *HasExecutionTimeConstraint* und der *ExecutionTimeConstraint* ermöglichen die Erstellung der Ausführungszeit unter einem *RunnbaleEntity*. Über die Attribute *minimumExecutionTime* und *maximumExecutionTime* kann eine minimale und maximale Ausführungszeit definiert werden. Die *ExecutionTimeTypeEnum* und das Attribut *executionTimeType* definieren zwei Arten von Ausführungszeit-Semantik. Die *NET*-Ausführungszeit ist die Zeit, die benötigt wird, um die Runnable ohne Unterbrechung und ohne externe Aufrufe auszuführen. Die *GROSS*-Ausführungszeit ist die Zeit, die für die Ausführung der Runnable ohne Unterbrechung, einschließlich externer Anrufe an andere Runnables.

Delay Beschränkung

Abbildung 4.9 zeigt die Metamodellerweiterung für die Modellierung von Delay Beschränkung. Im Vergleich zu den davor vorgestellten Metamodellerweiterungen wird diese Beschränkung unter einem neuen Container (*TimingConstraintsPackage*) erstellt, da diese

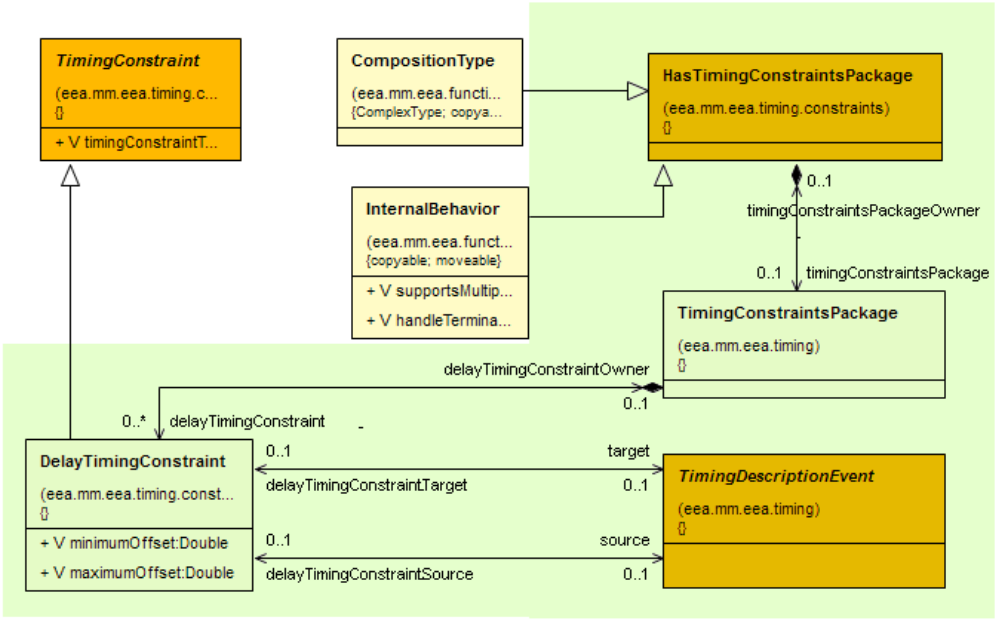

Abbildung 4.9.: Metamodell für Delay Beschränkung

Beschränkung sich nicht nur auf einem Beschreibungsevent oder einer Eventkette bezieht. Der Container wird jedoch unter den bestehenden *InternalBehavior* und *CompositionType* erstellt. Eine *CompositionType* dient zur Strukturierung von Softwarekomponenten sowie deren Ports. Dafür wurde die abstrakte *HasTimingConstraintsPackage* definiert.

Eine *DelayTimingConstraint* begrenzt den zeitlichen Abstand zwischen dem Auftreten von zwei *TimingDescriptionEvents*, ohne dass eine direkte funktionale Abhängigkeit zwischen den auftretenden Events erforderlich ist. Damit die Quelle und Ziel festgelegt werden können, wurden die zwei Assoziation-Beziehungen *target* und *source* definiert.

Wenn das Ziel-Event eintritt, wird erwartet, dass es frühestens mit dem *minimumOffset* und spätestens mit dem *maximumOffset* relativ nach dem Auftreten des Quell-Event eintritt.

Alter Beschränkung

Abbildung 4.10 zeigt die Metamodellerweiterung für die Modellierung von Alter Beschränkung. Die Container-Struktur ist ähnlich wie bei der zuvor vorgestellen Metaklasse *DelayTimingConstraint*. Die Metaklasse *AgeConstraint* wird verwendet, um ein minimales und maximales Alter festzulegen, wenn Lese- oder Schreibdaten auf dem aktuellen Port der Softwarekomponente empfangen werden. Aus diesem Grund wird der Scope einer solchen Beschränkung ein *TDEventVariableDataPrototype* sein (vorgestellt in Abschnitt 4.1.1). Jedes Mal, wenn ein Ereignis eintritt, muss der VariableDataPrototype das angegebene Datenalter haben.

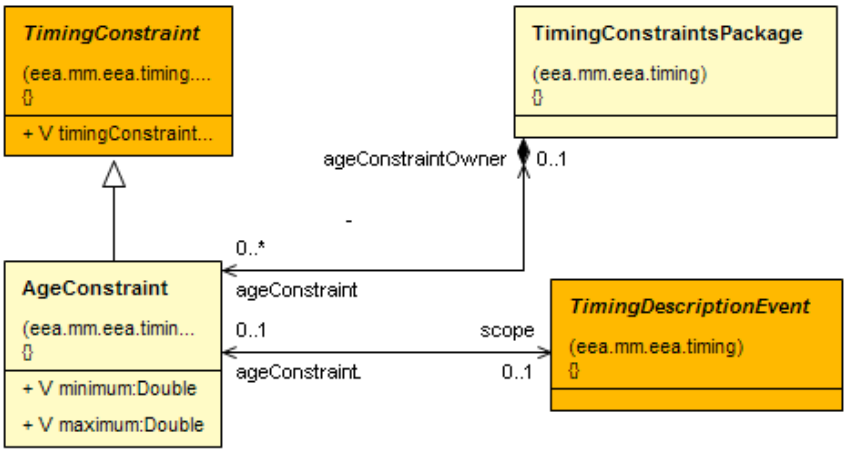

Abbildung 4.10.: Metamodell für Alter Beschränkung

EventTriggering Beschränkung

Abbildung 4.11 zeigt die Metamodellerweiterung für die Modellierung von EventTriggering Beschränkung. Die Klasse *EventTriggeringConstraint* wird verwendet, um das Auftreten eines bestimmten *TimingDescriptionEvent* anzugeben. Dies wird durch die Assoziation *scope* festgelegt.

AUTOSAR bietet fünf verschiedene Arten des Auftretens von Ereignissen. Sie können periodisch, sporadisch, konkret, geplatzt und willkürlich sein. Im Rahmen dieser Abschlussarbeit wurde erstmal eine konkrete Klasse abgebildet.

PeriodicEventTriggering beschreibt das periodische Auftreten von Events, das durch das Attribut *period* gegeben wird. Das Attribute *jitter* gibt die maximale Abweichung der Periode des Auftretensverhaltens. Das Attribut *minimumInterArrivalTime* gibt den minimalen Abstand zwischen den nachfolgenden Eintreten des Events an. Wenn der Wert des Attributes *minimumInterArrivalTime* kleiner als der Wert des Attributes *period* minus dem Wert des Attributes *jitter* ist, dann hat *minimumInterArrivalTime* keinen Einfluss auf die Eigenschaften der periodischen Constraints.

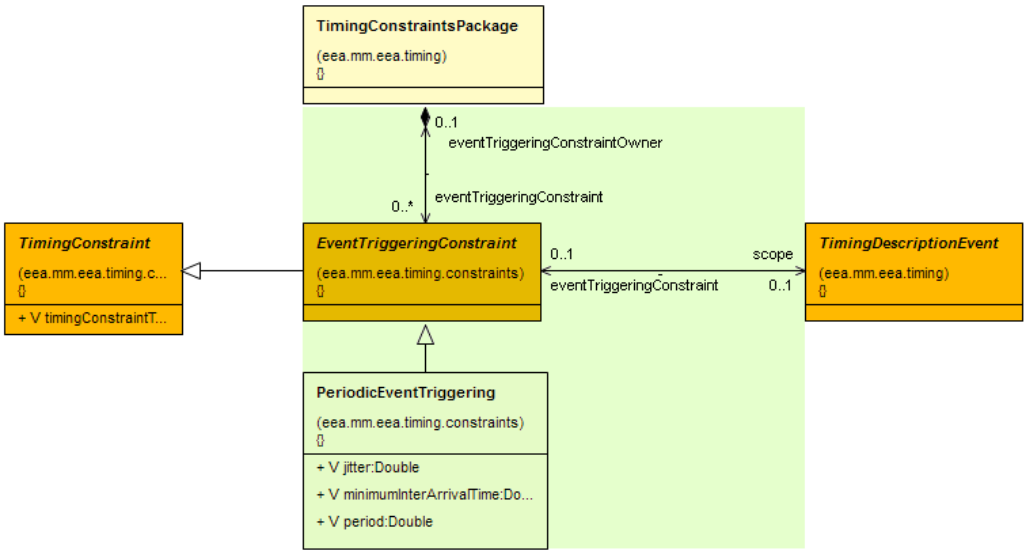

Abbildung 4.11.: EventTriggeringConstraint

4.2. Methoden zur Integration und Interoperabilität der Werkzeugkette

In diesem Abschnitt der Konzeptentwicklung werden beispielhaft einige Teile der Transformationsregeln dargestellt, die während dieser Abschlussarbeit umgesetzt wurden (vgl. Unterabschnitt 2.4.1). Diese ermöglichen es die in dieser Arbeit definierte Aspekte des Systemmodells, die für die Simulation notwendig sind, im AUTOSAR-Format auszutauschen.

4.2.1. Transformation der Abbildung auf Multicore-ECUs

Um die Übersichtlichkeit der Arbeit zu optimieren, werden die im Rahmen dieser Arbeit implementierten umfangreichen Transformationsregeln der Abbildung einer Softwarekomponente auf einem Kern nicht an dieser Stelle, sondern im Anhang dargestellt (siehe Anhang B). Hier wird eine vereinfachte Abbildung 4.12 verwendet, um das implementierte Konzept vorzustellen.

PREEvision unterstützt bisher die Modellierung eines Mapping auf einzelne Kerne einer ECU. Dies wird auf dem oberen Teil der Abbildung 4.12 dargestellt. Dort ist zu sehen, wie eine spezifische Softwarekomponente über das SwComponentMapping sowohl auf einem Mikroprozessor als auch auf einer Recheneinheit (*ProcessUnit*) gemappt werden kann. PREEvision bietet bereits die Möglichkeit, mehrere Kerne unter einem Prozessor zu modellieren. Das oben beschriebene Mapping von SWC zu ECU-Komponente kann dann einem bestimmten Kern zugewiesen werden. Auf dem Bild ist das die *shall*-Beziehung. So ein Mapping erfüllt jedoch nicht den AUTOSAR-Standard. Eine der Aufgaben dieser Arbeit war es, dieses Mapping mit AUTOSAR-Standard abzubilden. Auf dem unteren Teil des Bildes ist eine abstrakte Repräsentation eines Mapping, das momentan von AUTOSAR unterstützt wird. Die drei Komponenten *SwcToApplicationPartionMapping*, *ApplicationPartition* und

ApplicationPartitionToEcuPartitionMapping, die für das AUTOSAR-kompatible Mapping verwendet werden, existieren bereits im AUTOSAR-Metamodell, da sie ursprünglich für andere Funktionen im AUTOSAR-Standard entwickelt wurden. Dies erfüllt den Zweck, dass in PREEvision die modellierten *Kerne* auf *ECUPartitions* abgebildet werden können. Durch dieses Mapping wird dann während der Synthese beim Import in das Timing Architects Tool die Generierung der in PREEvision fehlenden Betriebsinformationsschicht sowie die Zuweisung der Tasks zu konkreten Kernen durchführt.

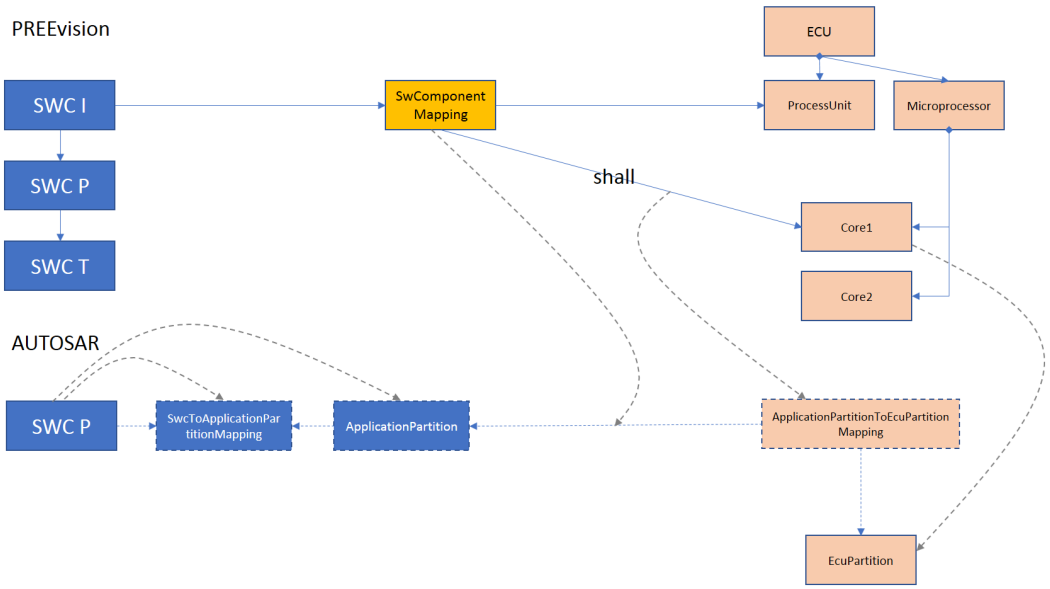

Abbildung 4.12.: Konzept für die Abbildung von SWCs auf Kerne

4.2.2. Transformation von Kernen und Taktfrequenzen der Kernen

Wie zuvor in Unterabschnitt 2.4.1 erläutert wurde, repräsentiert die *Left Hand Side (LHS)* die zu transformierende Elemente des Quellmodells und die *Right Hand Side (RHS)* die entsprechenden Elemente im Zielmodell.

Die hier behandelte Regel (Abbildung 4.13) zeigt die Transformation von einem *Core* in eine *ECUPartition* in AUTOSAR. Mittels der zweiten Regeln (Abbildung 4.14) werden die Taktfrequenzen, die für den bestimmten Kern festgelegt sind, in AUTOSAR-komformes Format mit Hilfe des AdminData-Konzeptes exportiert.

Wie Abbildung 4.14 zeigt, werden alle Objekte im Quellmodell vom Typ *Core* mit Komposition *MicroProcessor* und Assoziation *CPUType* im Zielmodell durch eine deutlich komplexere Struktur repräsentiert. Die *sdq*-Komposition ermöglicht es, spezielle Daten zu speichern, die nicht durch das Standardmodell repräsentiert werden. In diesem Fall werden der Name des Mikroprozessors, unter welchem der entsprechende Kern liegt (vgl. LHS-Teil der Regel) und die Taktfrequenz von *CPUType* übernommen und in die *ECUPartition* (vgl. RHS-Teil der Regel) transformiert.

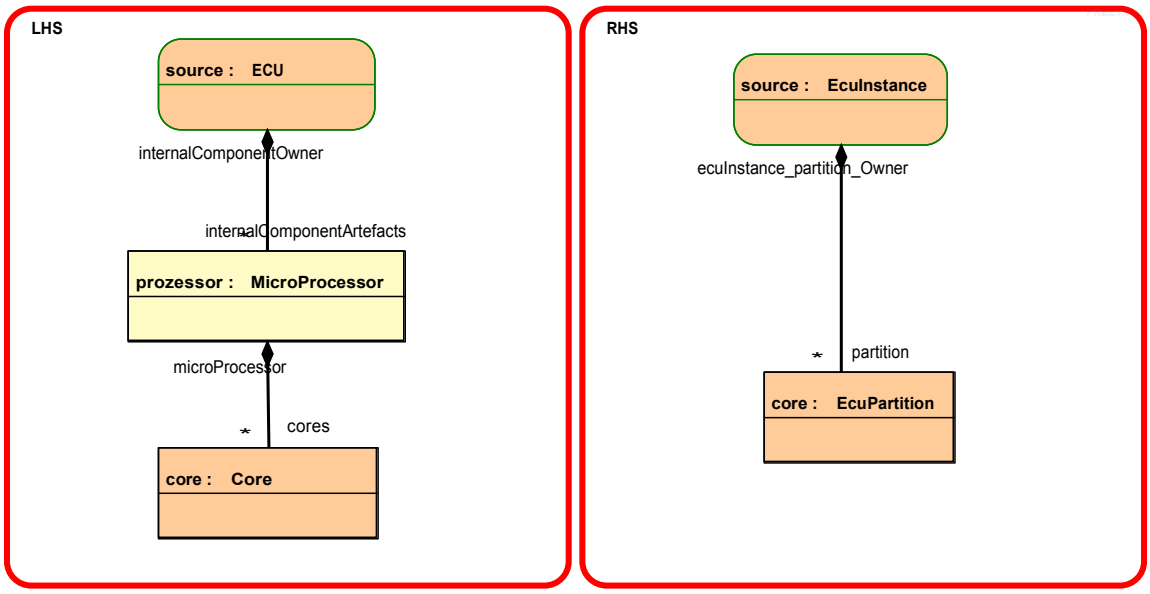

Abbildung 4.13.: Transformation der Kerne

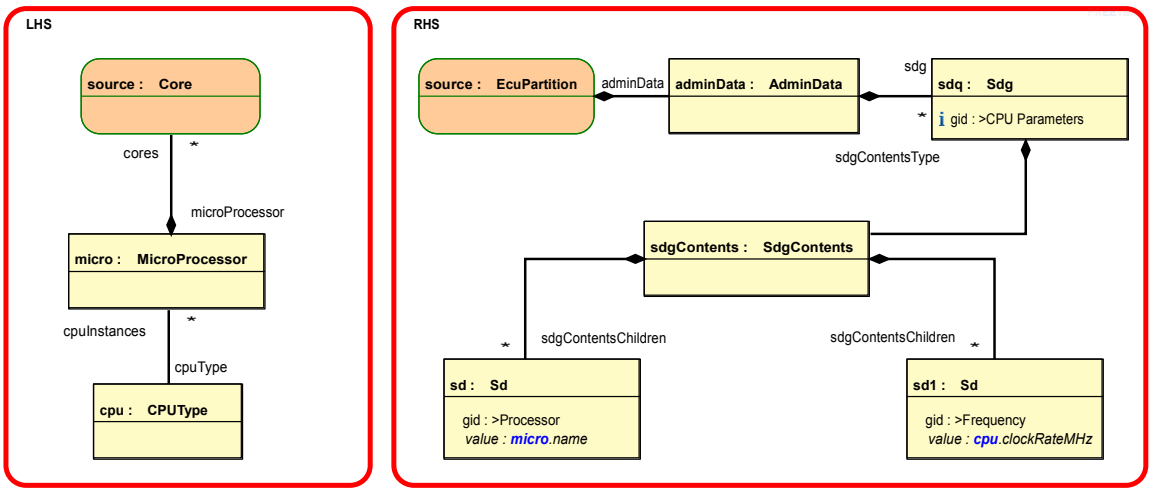

Abbildung 4.14.: Transformation der Taktfrequenzen

4.2.3. Transformation von Timing-Eventketten

Das in Unterabschnitt 4.1.1 erweiterte PREEVision-Metamodell für die Beschreibung von Eventketten und deren Beschreibungs ist dem von AUTOSAR definierten Timing-Metamodell sehr ähnlich. Dies ermöglicht eine einfachere EEA-AUTOSAR-Transformation.

Für die Transformation von Eventketten auf den in dieser Arbeit wurden immer zwei analoge Regeln umgesetzt. In Abbildung 4.15 ist die Transformationsregel für den Stimulus-Teil einer Eventkette dargestellt. Die Regel des Reaktionsteils ist analog definiert und ist im Anhang dieser Abschlussarbeit zu finden (vgl. Abbildung 4.15). Die hier abgebildete Regel zeigt die Transformation von einem Objekt vom Typ *TimingPath* in die Klasse *TimingDescriptionEventChain*. Der Hauptunterschied zwischen dem EEA- und dem

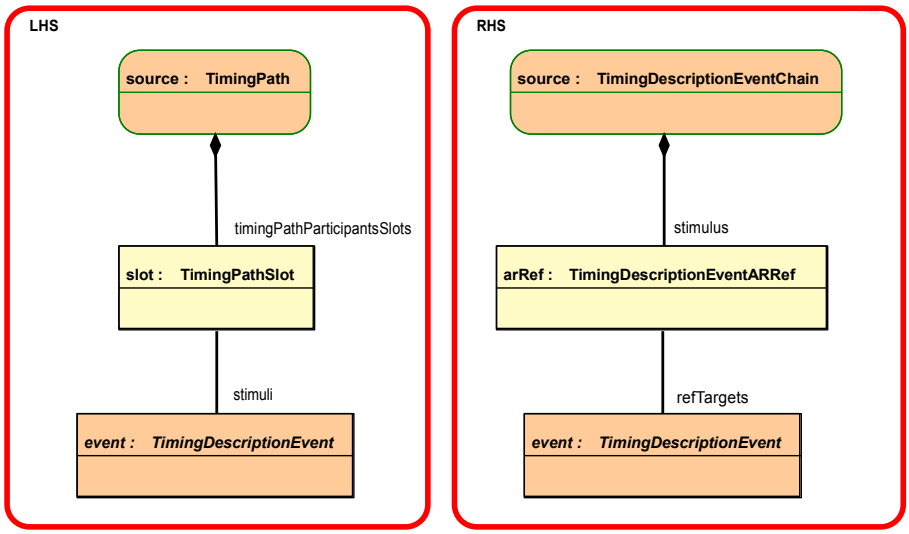

Abbildung 4.15.: Transformation einer Eventkette und des zugehörigen Stimulus

AUTOSAR-Metamodell ist, dass im erstgenannten ein *TimingPathSlot* auf der linken Seite vorhanden ist, der den *TimingPath* mit dem entsprechenden *Stimulus* verbindet. Sobald diese Struktur im Quellmodell zu finden ist, wird eine *TimingDescriptionEventChain* mit dem entsprechenden *Stimulus* exportiert. In dieser Regel ist das entsprechende Event durch die abstrakte *TimingDescriptionEvent*-Klasse dargestellt. Die konkrete Klasse wird dann im nachfolgenden Abschnitt ergänzt.

Segmentierung von Eventketten

Wie zuvor erwähnt, können Eventketten in weitere Eventketten unterteilt sein. Ob eine Eventkette in weitere unterteilt ist, wirkt sich auf das Transformationsregelwerk aus, was in diesem Unterabschnitt beschrieben wird.

Die erste Regel (vgl. Abbildung 4.16) findet die *TimingPaths*, die andere *TimingPaths* referenzieren durch die Assoziation *timingPathParticipants*.

Die zweite Regel (vgl. Abbildung 4.17) transformiert atomare *TimingPaths*, also solche, die nicht in weitere *TimingPaths* unterteilt sind. Nach dem Schema von AUTOSAR müssen solche *TimingPaths* bzw. die transformierte Eventketten auf sich selbst referenzieren. Dementsprechend unterscheiden sich Abbildung 4.16 und Abbildung 4.17 an dieser Stelle.

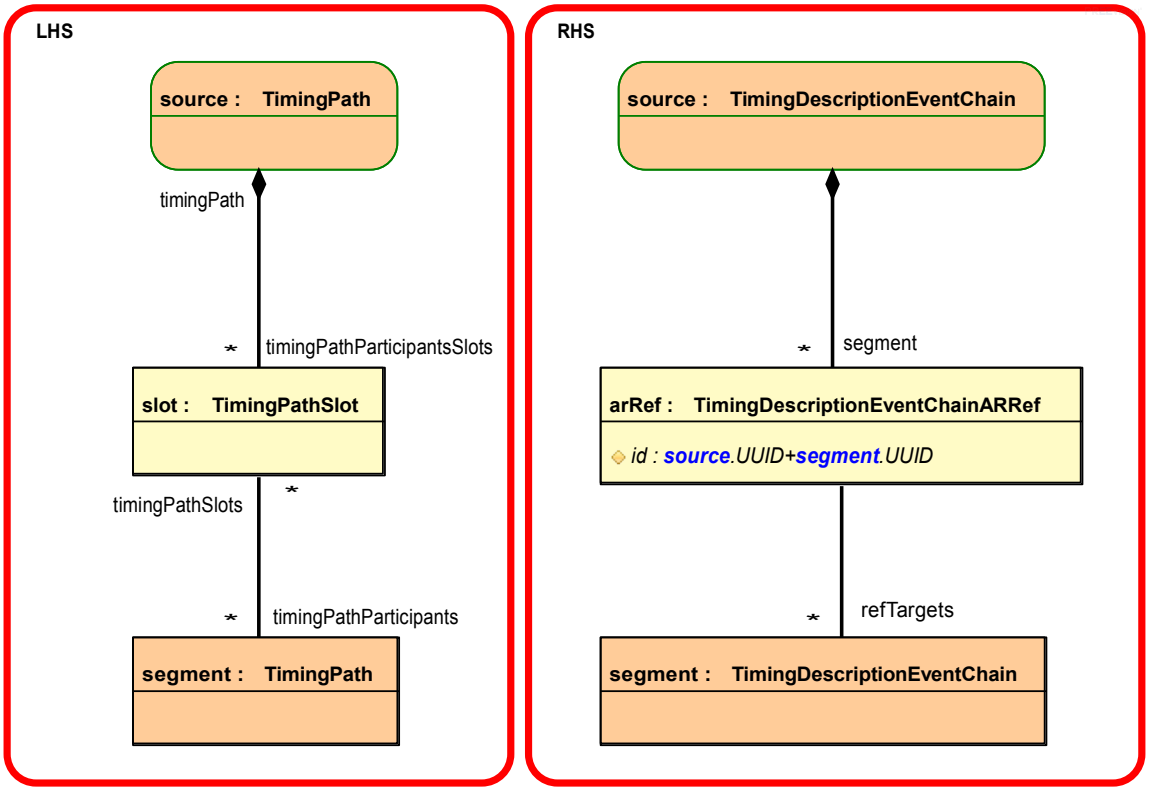

Abbildung 4.16.: Transformation von Segmenten - Anwendungsfall 1

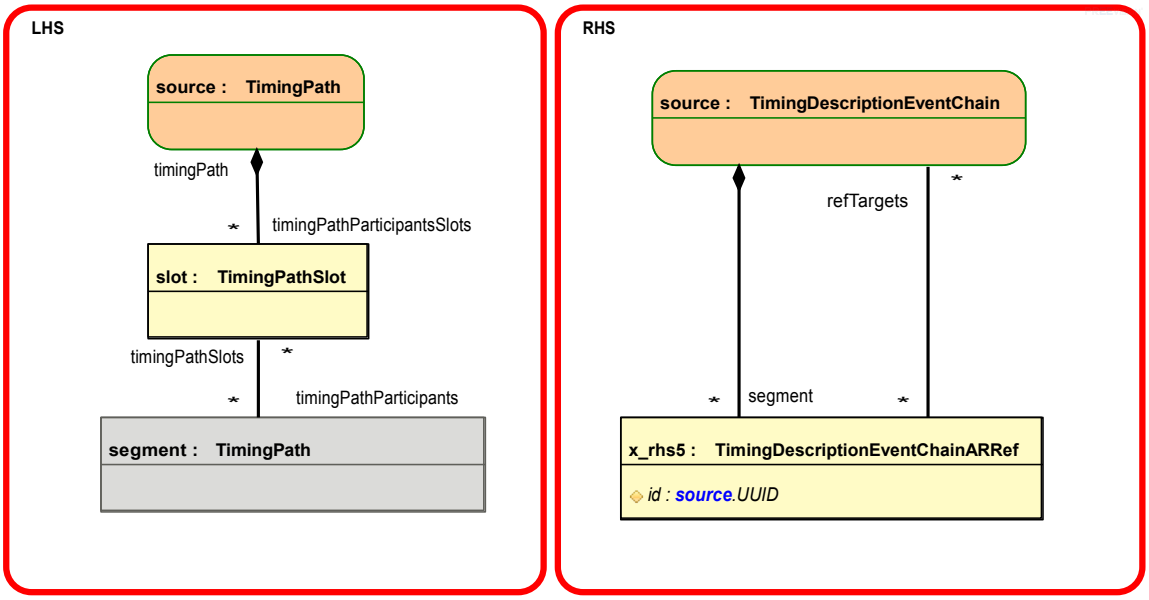

Abbildung 4.17.: Transformation von Segmenten - Anwendungsfall 2

Tranformation von Beschreibungsevents auf logischer Ebene

Der folgende Abschnitt stellt die Transformationsregeln dar, die den Export der Beschreibungsevents, die sich auf die Ports einer Softwarekomponenten beziehen, ermöglichen.

Die Regel in Abbildung 4.18 stellt im Prinzip eine Ergänzung der Regel aus Unterabschnitt 4.2.3 dar. Die Regel des Reaktionsteils ist analog definiert und ist im Anhang dieser Abschlussarbeit zu finden (vgl. Abbildung B.3)

Im linken Teil ganz unten ist das Objekt von Typ *TDEventVfb* gezeigt. Dieses liefert die Information, dass es sich um die in Abschnitt 4.1.1 definierten Beschreibungsevents handelt, nämlich alle die von der *TDEventVfb* erben. Das *CompositionType*-Objekt repräsentiert eine Komposition von Softwarekomponenten und hat die Rolle der Container von *TimingPath*. Sobald das Muster im Quellmodell vorhanden ist, wird die rechte Struktur generiert. Zunächst bildet das Transformationsregelwerk die Substruktur aus den Elementen von Typ *AUTOSAR*, *ARPackage* und *VfbTiming*. Durch die Unterstruktur aus den Elementen *CompositionSyComponent*, *SwComponentTypeARRef* und *VfbTiming* wird die Referenz zwischen der Komposition und der VfbTiming-Sicht ermöglichen. Das Objekt mit Name *source* und Typ *CompositionType* repräsentiert diese Komposition und muss ebenfalls im Quellmodell vorhanden sein.

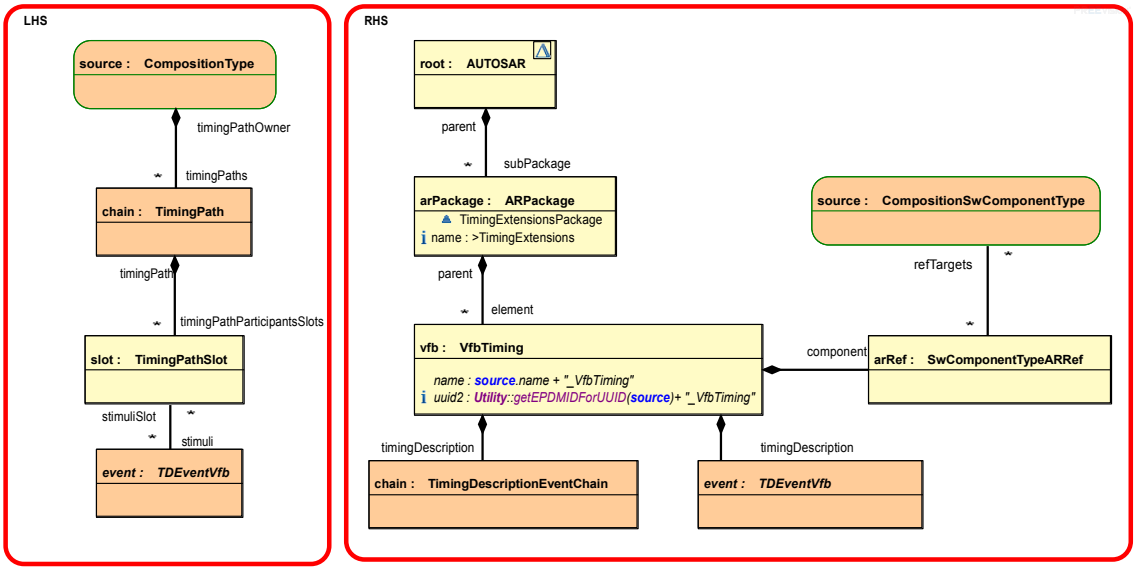

Abbildung 4.18.: Transformation des Stimulusteil einer Eventkette auf Ebene der Softwarekomponente - Teil 1

Um die Vollständigkeit der zu exportierenden Eventkette sicherzustellen, sollen die Softwarekomponente, die entsprechende Ports und die zu übertragende Datenelemente ebenfalls exportiert werden. Das ist die Aufgabe der nächsten zwei Regeln.

Der linke Teil der Abbildung 4.19 sucht die Ports auf den die mit PREEvision modellierte Slots zeigen. Zusätzlich sollen die Softwarekomponente der entsprechenden Ports abgebildet werden.

Diese in Abbildung 4.20 Regeln wird dann die Daten, die an dem entsprechenden Port gesendet oder empfangen werden dem Beschreibungevent zuordnen.

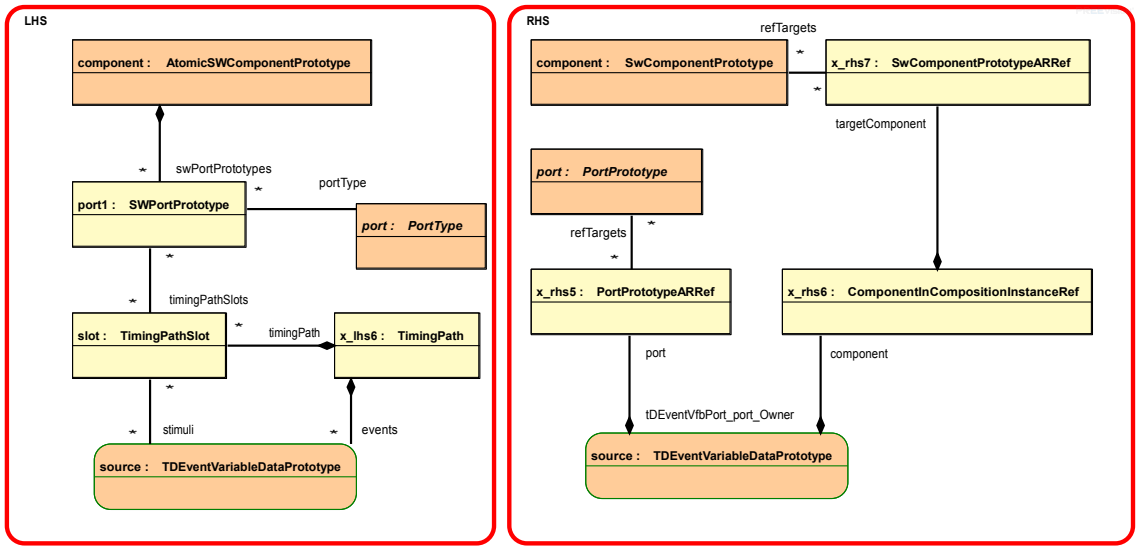

Abbildung 4.19.: Transformation des Stimulusteil einer Eventkette auf logischer Ebene - Teil 2

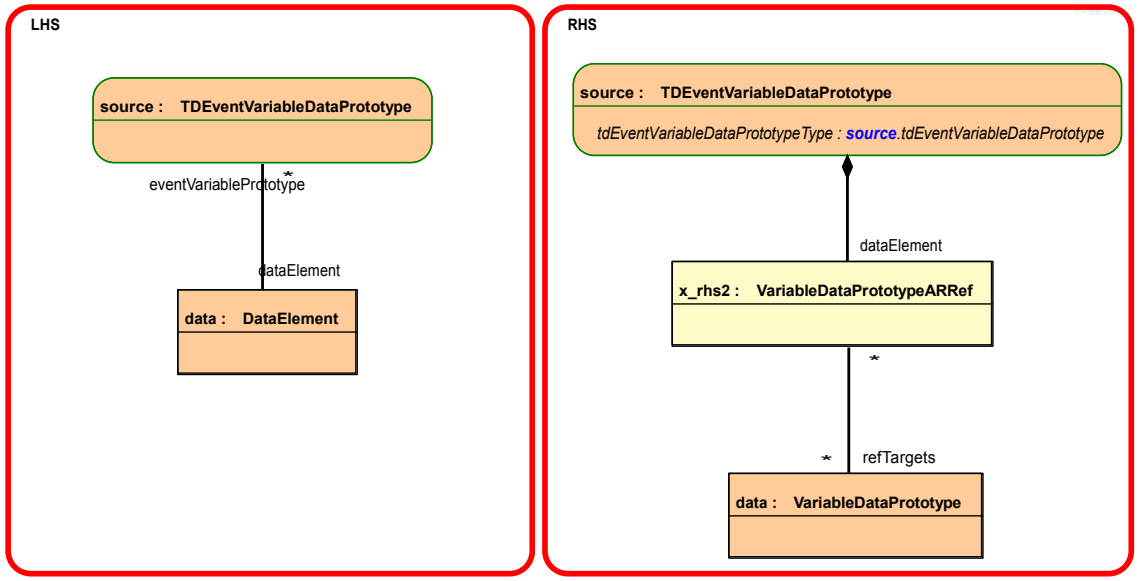

Abbildung 4.20.: Transformation des Stimulusteil einer Eventkette auf logischer Ebene - Teil 3

Die gleiche Vorgehensweise wie bei den letzten 3 Regeln wurde für die Implementierung der Transformationsregeln, die für die Abbildung der in Abschnitt 4.1.1 definierten Beschreibungsevents. Diese Transformationsregeln sind im Anhang dieser Arbeit zu finden.

Transformation von Latenzzeit

Sobald eine Eventkette und die entsprechende Beschreibungsevents exportiert sind, können die Beschränkungen auch exportiert werden. Die hier behandelte Regeln (vgl. Abbildung 4.21) zeigt die Tranformation eine *LatencyTimingConstraint*. Die andere Beschränkungen sind analog und werden im Anhand dieser Abschlussarbeit dargestellt. Der Container *CompositionType* liefert wieder die Information, wo sich der *TimingPath* befindet, nämlich unter der Komposition von Softwarekomponenten. Das bedeutet, dass diese Beschränkungen unter dem gleichen VfbTiming-Paket exportiert werden soll wie die Eventkette, die sie beschränkt. Zusätzlich wird der Typ von dem mit PREEvision modellierte *LatencyTimingConstraint* transformiert.

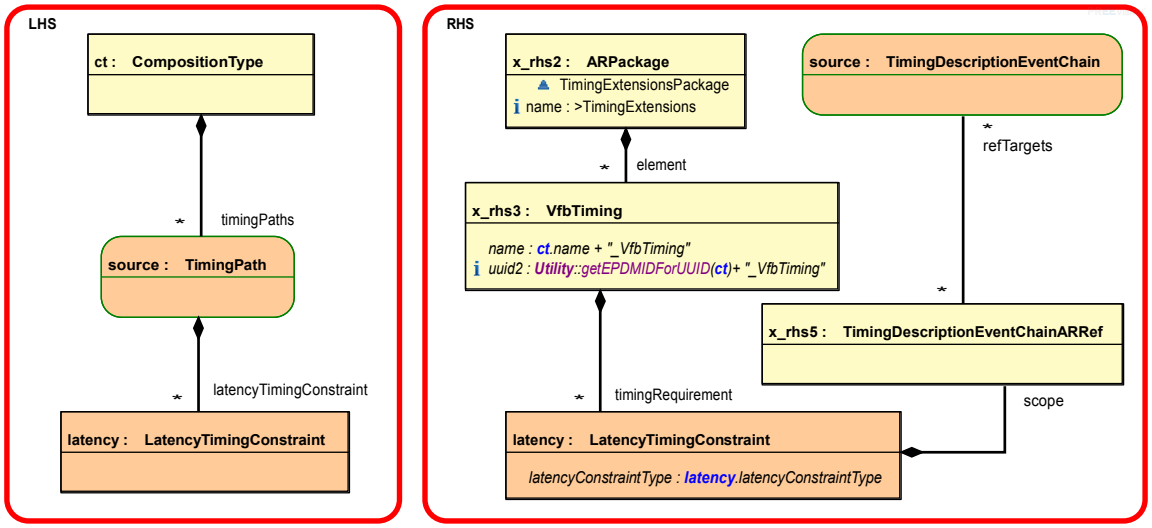

Abbildung 4.21.: Transformation vom *LatencyTimingConstraint* - Teil 1

Für den Export der entsprechenden Attributen wurde noch eine zusätzliche Regel implementiert. Je Attribut muss ein Objekt vom Typ *MultidimensionalTime* erzeugt werden, welches den jeweiligen Attributwert erhält. *MultidimensionalTime* hat zwei Attribute *cseCode* und *cseCodeFactor*. *cseCode* definiert die Basis der Zeit. Ein Skalierungsfaktor wird im *cseCodeFactor* erwähnt, um den Endwert zu bestimmen. Wenn beispielsweise cseCode 0 und cseCodeFactor 100 ist, bestimmt AUTOSAR den Wert als 100 usec; ebenso wenn cseCode 100 und cseCodeFactor 300 sind, dann stellt AUTOSAR einen Wert von 300 Winkelgraden dar.

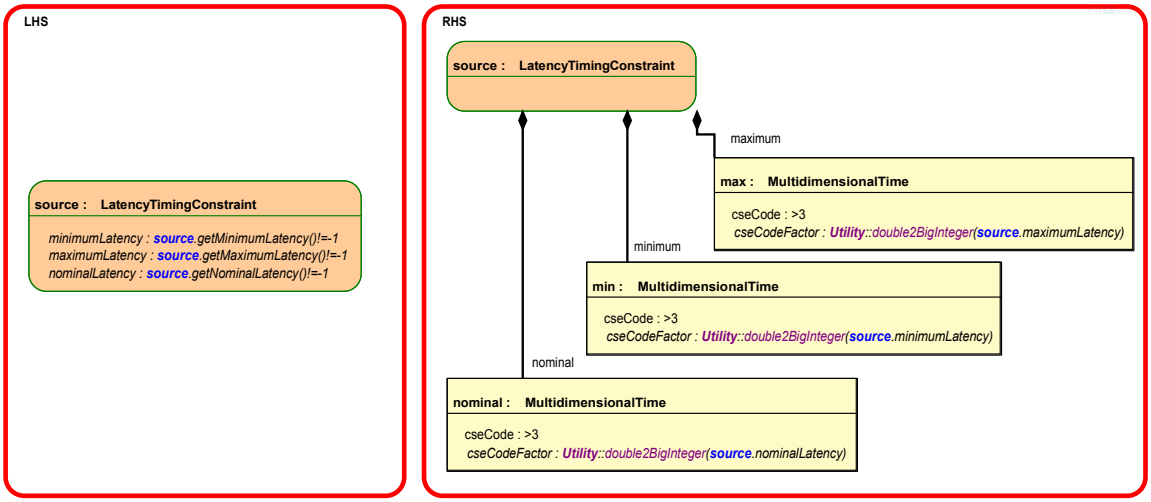

Abbildung 4.22.: Transformation vom *LatencyTimingConstraint* - Teil 2

5. Evaluation

Ziel dieser Evaluation ist es, zu untersuchen ob alle zur Simulation notwendigen Informationen mit PREEvision modelliert und im AUTOSAR-Format exportiert werden können. Dazu wird das erweiterte Metamodell bzw. die Modellierung von Echtzeitanforderungen in PREEvision evaluiert. Anschließend werden die generierten ARXML-Codeabschnitte untersucht, welche infolge der M2M-Implementierung erzeugt wurden. Zuletzt wurde das transformierte Modell im Timing Architects Simulationswerkzeug auf Simulierbarkeit überprüft.

5.1. Erwartetes Ergebnis des umgesetzten Konzept

Das in PREEvision modellierte Systemmodell und die von PREEvision produzierte ARXML sollten ein im Simulationswerkzeug Timing Architects simulierbares Modell liefern. Das erwartete Modell, das in das Simulationswerkzeug ankommen würde, lässt sich an der Stelle schwer beschreiben, da die Systemschritte in dem Timing Architects Simulationswerkzeug nicht dokumentiert sind. Die angenommene Erwartung, die den Erfolg dieser Arbeit anzeigt, beschränkt sich daher darauf, dass alles, was importiert wird, simulierbar ist.

5.2. Evaluation der Modellierung

In Abschnitt 3.4 ist das Fallbeispiel dargestellt worden. Es wurde darauf hingewiesen, dass ein Modell, das ein Multicore E/E-System für zukünftige Simulationen beschreibt, alle grundlegenden Aspekte abdecken muss. Genauer gesagt sind das die Software und ihres internes Verhalten, die Hardware, das Mapping einer Softwarekomponente auf einen Kern sowie die Timing Beschränkungen. Diese wurden mit dem Werkzeug PREEvision erfolgreich modelliert und werden in Abbildung 5.1 dargestellt. Dort sind von unten nach oben die Software- und Hardwarekomponenten sowie die entsprechenden Mappings zu sehen. Entscheidend ist hierbei, dass man sehen kann, dass unter *Prozessor1* mehrere Kerne angelegt werden konnten. Diese spielen eine wesentliche Rolle für die in der Abschnitt 4.2 umgesetzten Regeln. Diese werden anhand dieses Systemmodells ebenfalls evaluiert (vgl. Abschnitt 5.3).

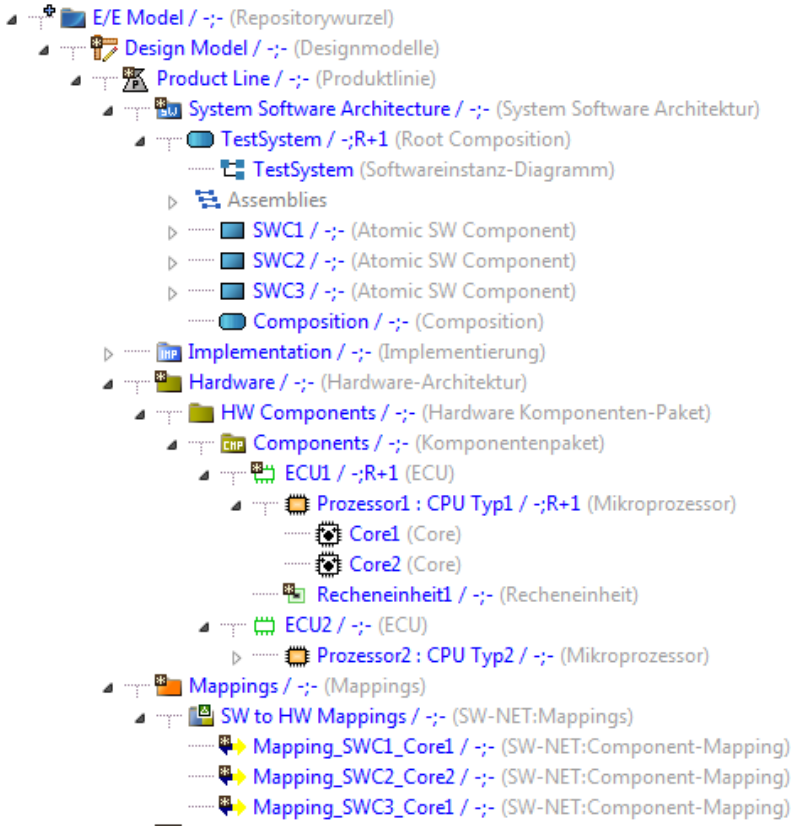

Abbildung 5.1.: Modellierung von Software-,Hardware-Architecktur und das Mapping

Das interne Verhalten einer atomaren Softwarekomponente sowie ihrer Runnables und die Events, auf die die Runnables reagieren werden, werden in einem anderen Teil des Modellbaums modelliert (siehe Abbildung 5.2). Abbildung 5.2 stellt dieselbe Softwarekomponente wie in Abbildung 5.1 dar, jedoch kann nur unter der Modellierungsebene in der erstgenannten Abbildung das interne Verhalten angelegt werden.

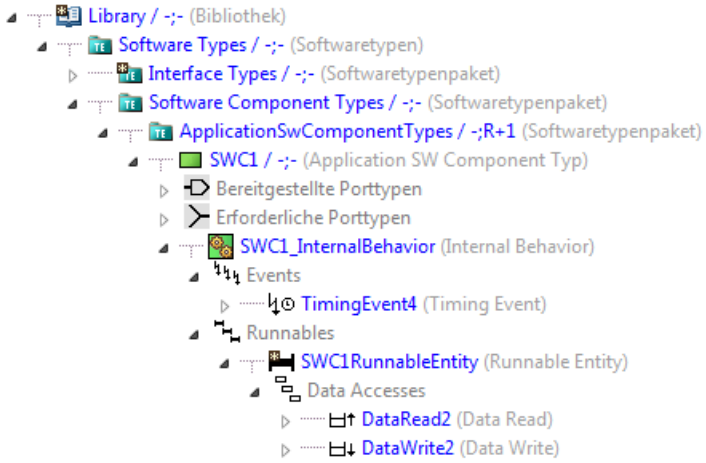

Abbildung 5.2.: Internes Verhalten einer Softwarekomponente

In folgenden wird ein Teil von den im Kapitel 4 festgelegten Metaklassen gezeigt:

Graphische Repräsentation der Events auf logischer Ebene

Abbildung 5.3 bezieht sich auf im Rahmen dieser Abschlussarbeit implementierten Metamodellerweiterungen(Abschnitt 4.1.1). Jetzt ist es möglich die *TDEventVariableDataPrototype*-Events unter dem entsprechenden *TimingPath* zu modellieren. Diese sind **EventOne** und **EventTwo** und werden jedoch den entsprechenden Slots zugewiesen um die Stimulus sowie die Reaktion-Rolle der Eventkette zu bestimmen. Die grafische Darstellung zeigt die Softwarekomponente, auf der die Eventkette(**Timing-Path2**) spezifiziert wurde. Die zwei Slots zeigen auf den entsprechenden Ports.

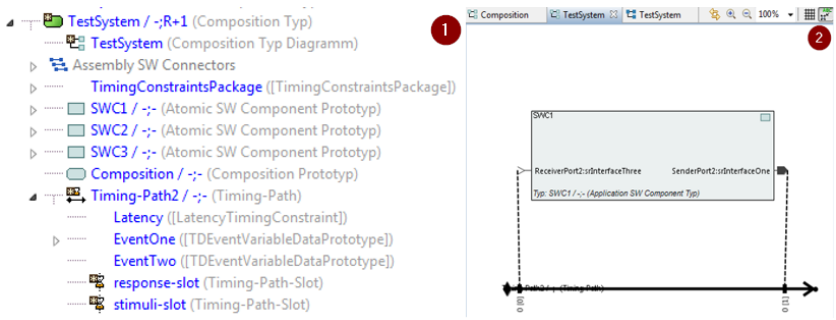

Abbildung 5.3.: Graphische Repräsentation der im Metamodell hinzugefügten *TDEventVariableDataPrototype*-Timing-Events

Graphische Repräsentation der Timing-Events des inneren Verhaltens einer Software Komponente

Durch die im Rahmen dieser Abschlussarbeit implementierten Metamodellerweiterungen (Abschnitt 4.1.1), ist es jetzt möglich die Timing-Beschreibungsevents, die sich auf dem inneren Verhalten einer Softwarekomponente beziehen, zu modellieren. Abbildung 5.4 zeigt wie das in der PREEvision-Umgebung aussieht. Für das interne Verhalten der Softwarekomponente gibt es in PREEvision momentan keine Diagramme. Die *TimingPath* und *TimingPathSlots* können auf dieser Ebene dadurch grafisch nicht dargestellt werden. In dem Modellanschicht sind die *TDEventSwcInternalBehavior*-Events unter dem entsprechenden *TimingPath* modelliert. Diese sind **RunnableEntityOneActivated** und **RunnableEntityOneTerminated** und werden jedoch den entsprechenden Slots zugewiesen um die Stimulus sowie die Response-Rolle der Eventkette zu bestimmen.

Abbildung 5.4.: Graphische Repräsentation der im Metamodell hinzugefügten Events

Graphische Repräsentation von Latenzzeit Beschränkung

Durch die im Rahmen dieser Abschlussarbeit implementierten Metamodellerweiterungen (Abschnitt 4.1.2), ist es jetzt möglich unter einem TimingPath eine Latenzzeit zu modellieren. Abbildung 5.5 zeigt wie das in der PREEvision-Umgebung aussieht:

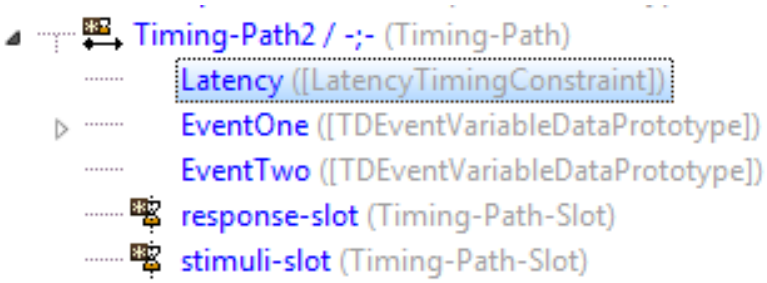

Abbildung 5.5.: Latenzzeit in der PREEvision-Modellierungsumgebung

Graphische Repräsentation von Ausführungszeit Beschränkung

Durch die im Rahmen dieser Abschlussarbeit implementierten Metamodellerweiterungen (Abschnitt 4.1.2), ist es jetzt möglich unter einem Runnable eine Ausführungszeit Beschränkung zu modellieren. Abbildung B.9 zeigt wie das in der PREEvision-Umgebung aussieht:

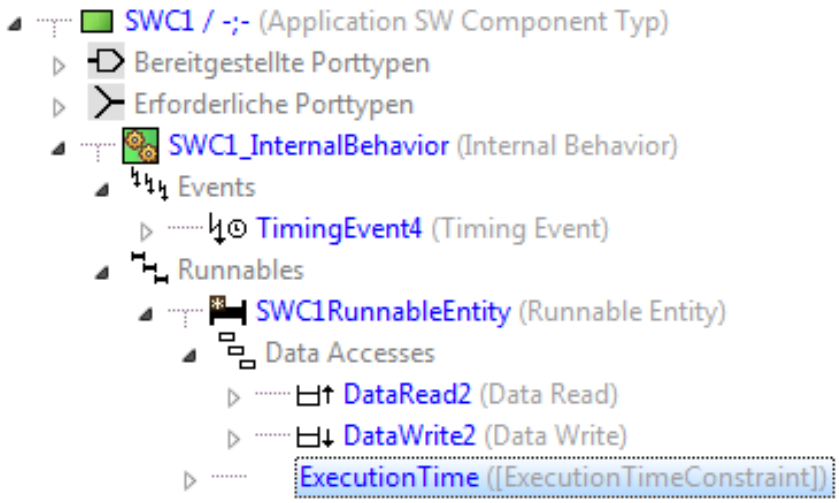

Abbildung 5.6.: Ausführungszeit Beschränkung in der PREEvision-Modellierungsumgebung

Graphische Repräsentation von Delay Beschränkung

Durch die im Rahmen dieser Abschlussarbeit implementierten Metamodellerweiterungen (Abschnitt 4.1.2), ist es jetzt möglich unter einem Runnable eine Delay Beschränkung zu modellieren. Abbildung 5.7 zeigt wie das in der PREEvision-Umgebung aussieht:

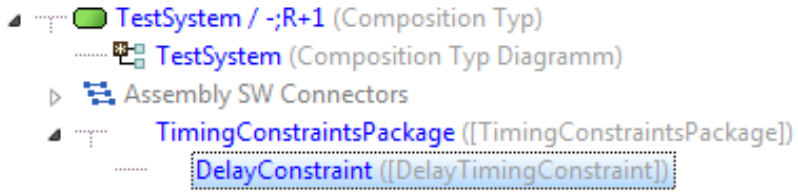

Abbildung 5.7.: Delay Beschränkung in der PREEvision-Modellierungsumgebung

5.3. Evaluation der Exportschnittstelle

In diesem Kapitel wird die Exportschnittstelle evaluiert.

5.3.1. Mapping einer Softwarekomponente auf einem Kern

Durch die im Rahmen dieser Abschlussarbeit implementierten Transformationsregeln (Unterabschnitt 4.2.1), ist es jetzt möglich, das mit PREEvision modellierte Mapping zwischen einem SWC und einem Kern zu exportieren. Dieser Unterabschnitt zeigt die entsprechenden Exportergebnisse.

Die folgenden zwei ARXML-Codeabschnitte bilden das in Unterabschnitt 4.2.1 dargestellte Konzept ab. Die Abschnitte wurden von der in Abbildung B.1 dargestellten Regel generiert. Die exportierten *SWC-TO-APPLICATION-PARTITION-MAPPING* und *APPLICATION-PARTITION-TO-ECU-PARTITION-MAPPING* stellen das Mapping einer Softwarekomponente auf einen Kern dar.

SWC-TO-APPLICATION-PARTITION-MAPPING liefert die Information, welche Komponente verwendet wurde. Das ist in diesem Fall die Komponente **SWC1**.

```
<SWC-TO-APPLICATION-PARTITION-MAPPING>
      <SHORT-NAME>SWC1ToApplicationPartitionMapping</SHORT-NAME>
      <APPLICATION-PARTITION-REF DEST="APPLICATION-PARTITION">
            SWC1_ApplicationPartition
      </APPLICATION-PARTITION-REF>
      <SW-COMPONENT-PROTOTYPE-IREF>
            <CONTEXT-COMPOSITION-REF>TestSystem</CONTEXT-COMPOSITION-REF>
         <TARGET-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE">SWC1
         </TARGET-COMPONENT-REF>
      </SW-COMPONENT-PROTOTYPE-IREF>
</SWC-TO-APPLICATION-PARTITION-MAPPING>
```

APPLICATION-PARTITION-TO-ECU-PARTITION-MAPPING liefert die Information, auf welchem Kern diese Komponente gemappt wird. Das ist in diesem Fall der Kern **Core1**.

```
 <APPLICATION-PARTITION-TO-ECU-PARTITION-MAPPING>
      <SHORT-NAME>ApplicationPartitionToCore2PartitionMapping</SHORT-NAME>
      <APPLICATION-PARTITION-REF DEST="APPLICATION-PARTITION"> SWC1_ApplicationPartition
      </APPLICATION-PARTITION-REF>
      <ECU-PARTITION-REF DEST="ECU-PARTITION">Core2
      </ECU-PARTITION-REF>
</APPLICATION-PARTITION-TO-ECU-PARTITION-MAPPING>
```

5.3.2. Multicore-ECUs und deren Taktfrequenzen

Der folgende Codeabschnitt wurde von der in Abbildung 4.14 dargestellten Regel generiert. Dem Steuergerät *ECU* hat zwei Kerne, die im AUTOSAR-Standard als zwei ECU-Partition exportiert wurden. Zwei Parameter sind unter den exportierten Kerne durch AdminData angelegt worden (vgl. Attribut *CPU-Parameters*). Zum einen liegt die Taktfrequenz vor, zum anderen wurde der Name des Prozessors auf *Microprocessor* festgelegt.

```
<ECU-INSTANCE>
      <SHORT-NAME>ECU</SHORT-NAME>
      <PARTITIONS>
            <ECU-PARTITION>
            <SHORT-NAME>Core1</SHORT-NAME>
                  <ADMIN-DATA>
                  <SDGS>
                        <SDG GID="CPU Parameters">
                        <SD GID="Frequency">101.0</SD>
                        <SD GID="Processor">Microprocessor</SD>
                        </SDG>
                  </SDGS>
                  </ADMIN-DATA>
            </ECU-PARTITION>
            <ECU-PARTITION>
            <SHORT-NAME>Core2</SHORT-NAME>
                  <ADMIN-DATA>
                  <SDGS>
                        <SDG GID="CPU Parameters">
                        <SD GID="Frequency">101.0</SD>
                        <SD GID="Processor">Microprocessor</SD>
                        </SDG>
                  </SDGS>
                  </ADMIN-DATA>
            </ECU-PARTITION>
      </PARTITIONS>
</ECU-INSTANCE>
```

Listing 5.1: Multicore-ECUs und deren Taktfrequenzen

5.3.3. Timing auf logischer Ebene

Die mit PREEvision modellierte Eventkette *Timing-Path2* und die entsprechende Latenzzeitbeschränkung (vgl. Abbildung 5.3) wurde durch die im Abschnitt 4.2.3 dargestellten Reihe von Regeln erfolgreich generiert.

Der hier dargestellte AUTOSAR-XML-Codeabschnitt stellt das Timing mit dem Namen *vfbTimingOFSWC* dar. Die exportierte Timing-Beschreibung beschreibt die Events, die sich auf die Ports der Softwarekomponente beziehen, insbesondere den Empfangs-

und bereitgestellten Port der Komponente. Die Daten, die an diesen Ports gesendet oder empfangen werden, werden ebenfalls exportiert (vgl. *DATA-ELEMENT-REF*). Zusätzlich wird eine Timing-Beschreibungs-Ereigniskette *TimingPath2* exportiert, die den kausalen Zusammenhang zwischen den Ereignissen *EventOne* und *EventTwo* beschreibt. Die in diesem Codebeispiel erfolgreich exportierte Latenzzeit legt fest, dass der Zeitpunkt, zu dem der Wert in von der Softwarekomponente namens *SWC1* empfangen wird, bis zu dem Zeitpunkt, zu dem der neu berechnete Wert gesendet wird, eine maximale Latenz von 2 ms aufweisen soll.

```
<AR-PACKAGE>
<SHORT-NAME>TimingExtensions</SHORT-NAME>
 <ELEMENTS>
 <VFB-TIMING>
 <SHORT-NAME>vfbTimingOFSWC</SHORT-NAME>
 <TIMING-DESCRIPTIONS>
 <TD-EVENT-VARIABLE-DATA-PROTOTYPE>
       <SHORT-NAME>EventOne</SHORT-NAME>
       <COMPONENT-IREF>
         <TARGET-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE"> SWC1
         </TARGET-COMPONENT-REF>
       </COMPONENT-IREF>
       <PORT-REF DEST="R-PORT-PROTOTYPE">ReceiverPort2</PORT-REF>
       <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE">Datenelement3
       </DATA-ELEMENT-REF>
 </TD-EVENT-VARIABLE-DATA-PROTOTYPE>
 <TD-EVENT-VARIABLE-DATA-PROTOTYPE>
       <SHORT-NAME>EventTwo</SHORT-NAME>
       <COMPONENT-IREF>
         <TARGET-COMPONENT-REF DEST="SW-COMPONENT-PROTOTYPE"> SWC1
         </TARGET-COMPONENT-REF>
       </COMPONENT-IREF>
       <PORT-REF DEST="P-PORT-PROTOTYPE">SenderPort2</PORT-REF>
       <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE">Datenelement1
       </DATA-ELEMENT-REF>
 </TD-EVENT-VARIABLE-DATA-PROTOTYPE>
 <TIMING-DESCRIPTION-EVENT-CHAIN>
       <SHORT-NAME>TimingPath2</SHORT-NAME>
       <STIMULUS-REF DEST="TD-EVENT-VARIABLE-DATA-PROTOTYPE">EventOne
       </STIMULUS-REF>
       <RESPONSE-REF DEST="TD-EVENT-VARIABLE-DATA-PROTOTYPE">EventTwo
       </RESPONSE-REF>
       <SEGMENT-REF DEST="TIMING-DESCRIPTION-EVENT-CHAIN">Timing_Path2
       </SEGMENT-REF>
 </TIMING-DESCRIPTION-EVENT-CHAIN>
 </TIMING-DESCRIPTIONS>
```

```
<TIMING-REQUIREMENTS>
 <LATENCY-TIMING-CONSTRAINT>
      <SHORT-NAME>Latency</SHORT-NAME>
      <LATENCY-CONSTRAINT-TYPE>REACTION</LATENCY-CONSTRAINT-TYPE>
      <SCOPE-REF DEST="TIMING-DESCRIPTION-EVENT-CHAIN">Timing_Path2
      </SCOPE-REF>
      <MAXIMUM>
        <CSE-CODE>3</CSE-CODE>
        <CSE-CODE-FACTOR>2</CSE-CODE-FACTOR>
      </MAXIMUM>
 </LATENCY-TIMING-CONSTRAINT>
</TIMING-REQUIREMENTS>
</VFB-TIMING>
</ELEMENTS>
</AR-PACKAGE>
```

Listing 5.2: Timing auf logischer Ebene

5.3.4. Timing des internen Verhalten einer Softwarekomponente

Die mit PREEvision modellierte Eventkette *CalculateValueOfDataElementTwo* (vgl. Abbildung 5.4) und die zwei Test-Beschränkungen wurden durch die im Anhang dargestellten Reihe von Regeln erfolgreich generiert.

Der hier dargestellte AUTOSAR-XML-Codeabschnitt stellt das Timing mit dem Namen *SWC2InternalBehaviorSwcTiming* dar. Das *SWC*-Timing bezieht sich auf das interne Verhalten der Softwarekomponente *SWC2*. Hier wird auf die RunnableEntity der Softwarekomponente *SWC2* verwiesen. Im Wesentlichen erfordert die angegebene Timing-Anforderung zum einen, dass die Verzögerung zwischen Aktivierung und Beendigung der RunnableEntity kleiner oder gleich 20 ms ist und zum anderen, dass die RunnableEntity bei einer Anforderung von 50 Hz ausgelöst wird, was bedeutet, dass die Runnable-Entität periodisch alle 20 ms aktiviert.

```
<AR-PACKAGE>
<SHORT-NAME>TimingExtensions</SHORT-NAME>
<ELEMENTS>
<SWC-TIMING>
 <SHORT-NAME>SWC2InternalBehaviorSwcTiming</SHORT-NAME>
 <TIMING-DESCRIPTIONS>
      <TD-EVENT-SWC-INTERNAL-BEHAVIOR>
            <SHORT-NAME>RunnableEntityOneActivated</SHORT-NAME>
            <RUNNABLE-REF DEST="RUNNABLE-ENTITY">Runnable_Entity_One
            </RUNNABLE-REF>
            <TD-EVENT-SWC-INTERNAL-BEHAVIOR-TYPE>
                  RUNNABLE-ENTITY-ACTIVATED
            </TD-EVENT-SWC-INTERNAL-BEHAVIOR-TYPE>
      </TD-EVENT-SWC-INTERNAL-BEHAVIOR>
```

```
      <TD-EVENT-SWC-INTERNAL-BEHAVIOR>
            <SHORT-NAME>RunnableEntityOneTerminated</SHORT-NAME>
            <RUNNABLE-REF DEST="RUNNABLE-ENTITY">Runnable_Entity_One
            </RUNNABLE-REF>
            <TD-EVENT-SWC-INTERNAL-BEHAVIOR-TYPE>RUNNABLE-ENTITY-TERMINATED
            </TD-EVENT-SWC-INTERNAL-BEHAVIOR-TYPE>
      </TD-EVENT-SWC-INTERNAL-BEHAVIOR>
      <TIMING-DESCRIPTION-EVENT-CHAIN>
            <SHORT-NAME>CalculateValueOfDataElementTwo</SHORT-NAME>
            <STIMULUS-REF DEST="TD-EVENT-SWC-INTERNAL-BEHAVIOR">
            RunnableEntityOneActivated </STIMULUS-REF>
            <RESPONSE-REF DEST="TD-EVENT-SWC-INTERNAL-BEHAVIOR">
            RunnableEntityOneTerminated </RESPONSE-REF>
            <SEGMENT-REF DEST="TIMING-DESCRIPTION-EVENT-CHAIN">
            CalculateValueOfDataElementTwo</SEGMENT-REF>
       </TIMING-DESCRIPTION-EVENT-CHAIN>
</TIMING-DESCRIPTIONS>
<TIMING-REQUIREMENTS>
      <PERIODIC-EVENT-TRIGGERING>
            <SHORT-NAME>PeriodicActivationRunnableEntityOne</SHORT-NAME>
            <EVENT-REF DEST="TD-EVENT-SWC-INTERNAL-BEHAVIOR">
                  RunnableEntityOneActivated
            </EVENT-REF>
            <JITTER>
              <CSE-CODE>3</CSE-CODE>
              <CSE-CODE-FACTOR>1</CSE-CODE-FACTOR>
            </JITTER>
            <PERIOD>
              <CSE-CODE>3</CSE-CODE>
              <CSE-CODE-FACTOR>10</CSE-CODE-FACTOR>
            </PERIOD>
      </PERIODIC-EVENT-TRIGGERING>
      <LATENCY-TIMING-CONSTRAINT>
            <SHORT-NAME>Latency</SHORT-NAME>
            <LATENCY-CONSTRAINT-TYPE>REACTION</LATENCY-CONSTRAINT-TYPE>
            <SCOPE-REF DEST="TIMING-DESCRIPTION-EVENT-CHAIN">
                  CalculateValueOfDataElementTwo
            </SCOPE-REF>
            <MAXIMUM>
                  <CSE-CODE>3</CSE-CODE>
                  <CSE-CODE-FACTOR>30</CSE-CODE-FACTOR>
            </MAXIMUM>
      </LATENCY-TIMING-CONSTRAINT>
</TIMING-REQUIREMENTS>
<BEHAVIOR-REF DEST="SWC-INTERNAL-BEHAVIOR">SWC2_InternalBehavior
```

```
 </BEHAVIOR-REF>
 </SWC-TIMING>
</ELEMENTS>
</AR-PACKAGE>
```

Listing 5.3: Timing des internen Verhalten einer Softwarekomponente

5.4. Evaluation der Simulierbarkeit

Die Simulierbarkeit der in PREEvision modellierten Multicore-Architektur wird evaluiert, indem es mittels AUTOSAR in TA Simulationswerkzeug importiert und die Simulation der Echtzeitanforderungen ausgeführt wird. Dabei werden sowohl der Import als auch die Simulierbarkeit auf Erfolg überprüft.

Im den folgenden Unterabschnitten wird der Importerfolg der einzelnen Modellteile demonstriert.

5.4.1. Software-Teilmodell

Abbildung 5.8 zeigt das Ergebnis des Imports des Software-Teilmodells. Wie man erkennen kann, wurden alle in PREEvision modellierten Softwarekomponenten (*SWC1*, *SWC2* und *SWC3*) sowie die entsprechende Runnables erfolgreich importiert.

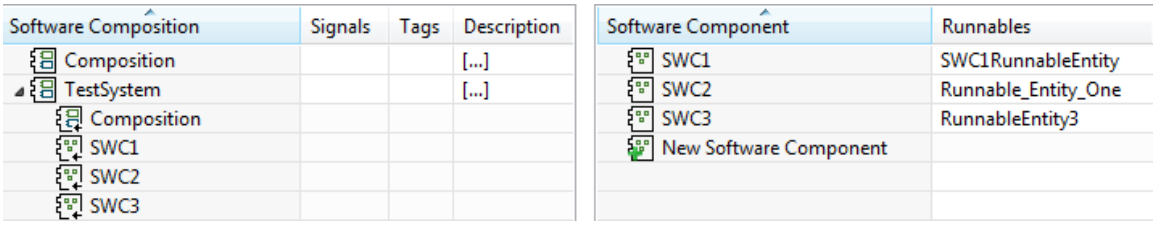

Abbildung 5.8.: Software-Teilmodell in Simulationswerkzeug Timing Architects

Hardware-Teilmodell

Abbildung 5.9 zeigt das Ergebnis des Imports des Hardware-Teilmodells. Wie man erkennen kann, wurden alle in PREEvision modellierten ECUs sowie die darunterliegenden Kerne mit ihren Taktfrequenzen erfolgreich importiert.

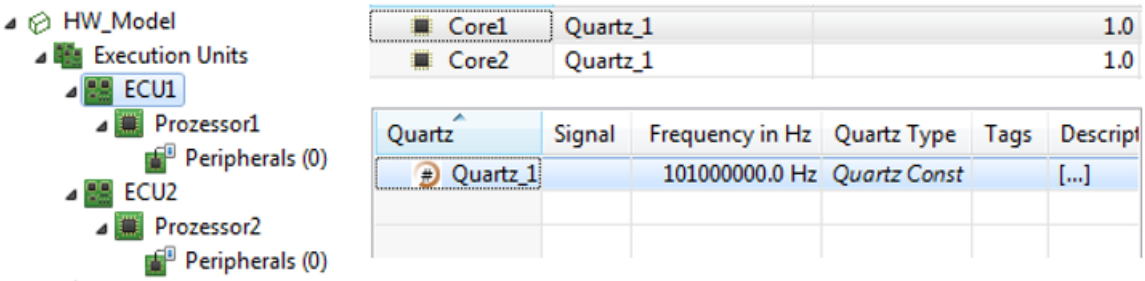

Abbildung 5.9.: Hardware-Teilmodell in Simulationswerkzeug Timing Architects

TIMEX-Teilmodell

Abbildung 5.10 zeigt das Ergebnis des Imports des TIMEX-Teilmodells. Wie man erkennen kann, wurden alle in PREEvision modellierten Eventketten und die darunterliegenden Beschreibungsevents erfolgreich importiert.

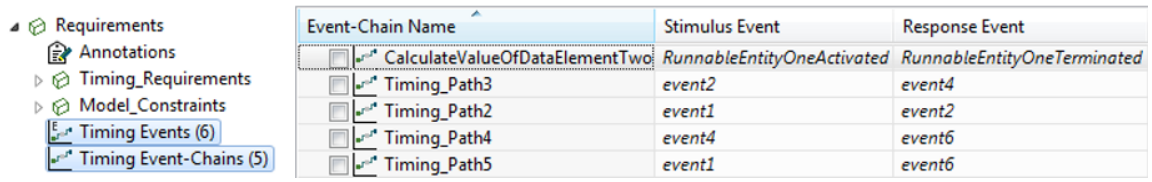

Abbildung 5.10.: Eventketten in Simulationswerkzeug

Abbildung 5.11 zeigt das Ergebnis des Imports des TIMEX-Teilmodells. Wie man erkennen kann, wurden alle in PREEvision modellierten Timing-Anforderungen erfolgreich importiert.

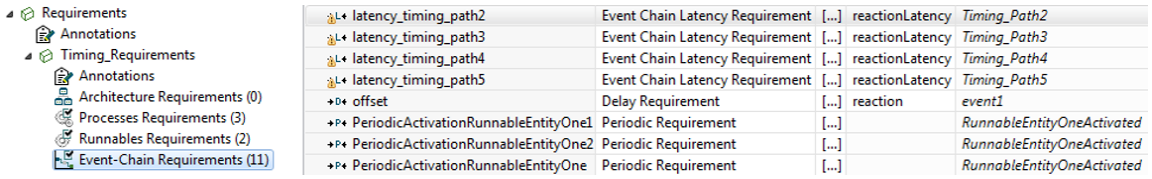

Abbildung 5.11.: Timing-Anforderungen in Simulationswerkzeug Timing Architects

Generierung der in PREEvision fehlenden Betriebssystem-Sicht

Abbildung 5.12 zeigt das Ergebnis von der in Hintergrund passierenden Synthese, bei der die fehlende in PREEvision Betriebssystem-Sicht generiert wird. Die Zuordnung von Runnables auf OS-Tasks, die Zuweisung dieser Tasks zu einer OS-Applikation sowie deren Zuordnung zu den Prozessorkernen mithilfe der in Unterabschnitt 4.2.1 dargestellten Mapping-Konzept ist erfolgreich abgeschlossen.

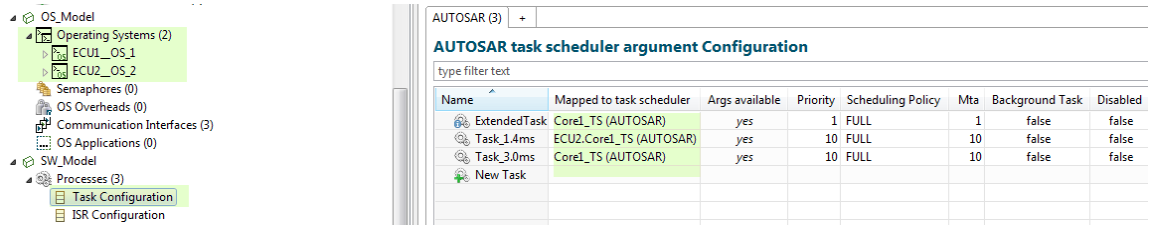

Abbildung 5.12.: Betriebssystem-Konfiguration in Simulationswerkzeug Timing Architects

5.4.2. Fehlendes Mapping

Darüber hinaus ist das mit PREEvision exportierte bzw. in das Simulationswerkzeug importierte Systemmodell simulierbar bis auf das in der Abbildung 5.13 dargestellten fehlenden Mapping, welche manuell gemacht werden sollte. Bei diesem Mapping handelt sich um die effiziente Verteilung von Daten auf mehrere Speichermodule, welche durch die Multicore-Architekturen hinzugefügt werden [Infa]. Daher ist das Ziel der Speicherzuordnung, die gesamte Speicherzugriffszeit zu minimieren, indem die Daten optimal auf verschiedene Speicher verteilt werden [Ama].

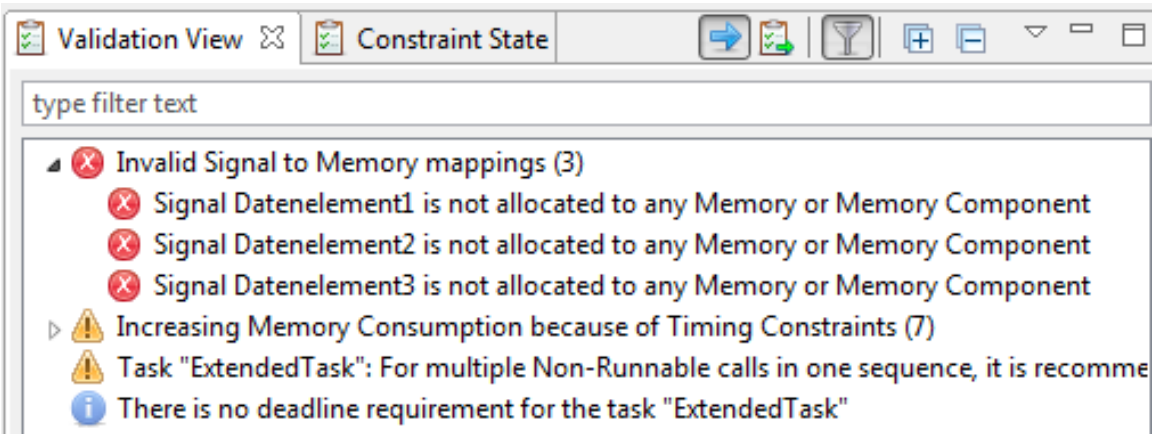

Abbildung 5.13.: Zuordnung von Daten auf Speicher in Simulationswerkzeug Timing Architects

5.4.3. Simulation des ergänzten Modells

Abbildung 5.14 zeigt, dass die Simulation erfolgreich durchgeführt werden konnte, nachdem das fehlende Mapping von Daten auf dem Speicher manuell ergänzt wurde.

Abbildung 5.15 zeigt die visuelle Repräsentation der Eventketten, die während der Simulationsberechnungen verwendet wurden.

Abbildung 5.16 zeigt quantitativ wie oft die festgelegten Echtzeitanforderungen verletzt wurden.

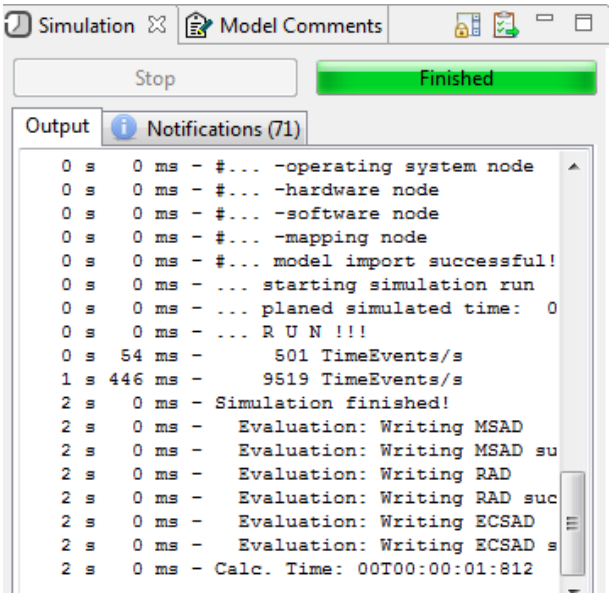

Abbildung 5.14.: Simulationssicht in Simulationswerkzeug Timing Architects

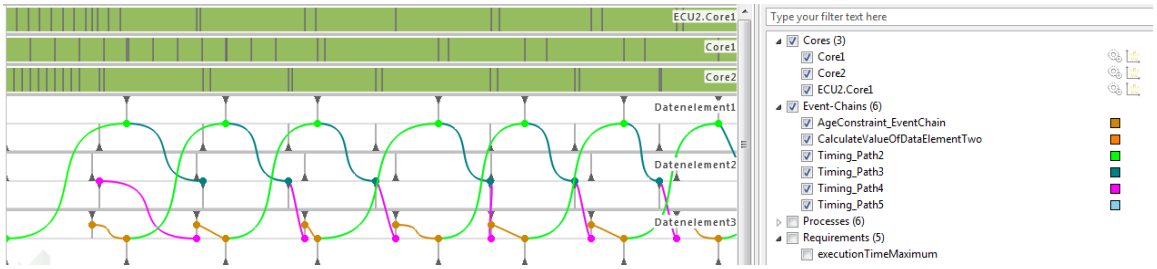

Abbildung 5.15.: Darstellung der importierten Eventketten in Simulationswerkzeug Timing Architects

5.5. Bewertung

Wie die Evaluation der Modellierung, des ARXML-Codes und der Darstellung des Simulationsergebnisses im Timing Architects Simulationswerkzeug zeigen, wurden im gewählten Fallbeispiel alle Informationen mit PREEvision erfolgreich modelliert und im AUTOSAR-Format exportiert. Außerdem war die im Hintergrund laufende Synthese ebenfalls erfolgreich. Dadurch wurde eine Betriebssystem-Konfiguration erzeugt, bei der das mit PREEvision modellierte Systemmodell bis um das fehlende Speicherzuordnung vollständig ist. Somit wurde gezeigt, dass mit der im Rahmen dieser Arbeit entwickelten Werkzeugkette Multicore-Architekturen mit Echtzeitbeschränkungen modelliert und simuliert werden können.

Process Table | Requirements Table | Event-Chain Requirement Table | Load T

Search: Filter settings

Event-Chain Requirement	DU min [ms]	DU max [ms]	DU avg [ms]	Violations count
Data Age Constraint				
AgeConstraint	0.03313	15.46980	6.35706	113
Delay Requirement				
offset	9.92283	17.84362	11.89047	168
Event Chain Latency R				
VpLtcTiming1	9.96532	17.96477	11.80921	169
VpLtcTiming2	9.96532	17.96477	11.80921	169
VpLtcTiming3	9.96532	17.96477	11.80921	169
latency_timing_patl	9.92283	17.84362	11.89047	168
latency_timing_patl	0.04937	13.72021	5.89847	93
latency_timing_patl	0.10190	12.88140	6.10794	88
latency_timing_patl	12.41509	35.42849	23.56470	133
Periodic Requirement				
	0.00000	0.00000	0.00000	170
Minimum	**0.00000**	**0.00000**	**0.00000**	**88**
Maximum	**12.41509**	**35.42849**	**23.56470**	**170**
Average	**5.19509**	**13.92345**	**8.42806**	**148**
Summation	**62.34110**	**167.08145**	**101.13674**	**1778**

Abbildung 5.16.: Eventketten Anforderungstabelle in Simulationswerkzeug Timing Architects

6. Zusammenfassung

Im Rahmen dieser Abschlussarbeit wurde eine durchgängige Modellierung und Simulation von Multicore-Eigenschaften umgesetzt. So können beispielsweise Probleme mit der E/E-Architektur für Steuergeräte mit Multicore-Prozessoren oder Verletzungen der Echtzeitanforderungen vor der Implementierung erkannt werden. Dies wurde durch die Kopplung eines Werkzeugs zur Modellierung von E/E-Architekturen mit einem Simulationswerkzeug erreicht. Als Austauschformat wurde AUTOSAR verwendet, da sich dieses als gängigstes Format in der Automobilindustrie etabliert hat. Dadurch ist die Kompatibilität zwischen den Werkzeugen und Produkten garantiert. Zur Modellierung der in dieser Arbeit identifizierten Multicore-Eigenschaften wurde das Werkzeug PREEvision verwendet. Für die frühzeitige Simulation bzw. für die Überprüfung der mit PREEvision modellierten Eigenschaften wurde das Simulationswerkzeug Timing Architects verwendet.

Zuerst wurden die ausgewählten Technologien vorgestellt. Zusätzlich wurden Ansätze zur Lösung anstehender Multicore-Probleme sowie Literaturrecherche dieser Arbeit erläutert.

Danach wurde das Systemmodell von AMALTHEA dargestellt und analysiert. Dieses weist die höchste Übereinstimmung mit dem Simulationswerkzeug auf und ist öffentlich zugänglich. Anhand dieser Analyse und der Literaturrecherche wurde eine Liste mit den notwendigsten Multicore-Aspekten herausgearbeitet. Außerdem gibt AUTOSAR mit seiner TIMEX-Spezifikation eine Reihe von Echtzeitbeschränkungen, die für die Beschreibung von Timings einer E/E-Architektur sowie für die nachfolgende Simulation enorm wichtig sind.

Es wurden zum einen die Teile des Systemmodells und zum anderen die Timings als Multicore-Eigenschaften berücksichtigt. Danach wurde auch das Systemmodell von PREEvision analysiert. PREEvision untersützt die Basissoftware-Ebene nicht in derselben Weise wie AMALTHEA bzw. das Simulationaswerkzeug Timing Architects, sondern beschränkt sich auf höhere Abstraktionsebenen. Aus diesem Grund wurde eine Synthese für die Erzeugung dieser fehlenden Sicht verwendet.

Als nächstes wurde darauf aufbauen ein Konzept für die Modellierung von Multicore-Eigenschaften mit Fokus auf Echtzeitbeschränkungen entwickelt. Bei dem Konzept wurde eine Metamodellerweiterung des zugrundeliegenden Metamodells des PREEvision-Metamodells implementiert. Danach wurde die AUTOSAR-Exportschnittstelle erweitert, um die vorgenommene Komponenten, die das Fallbeispiel vollständig beschreiben, in AUTOSAR-Format zu exportieren.

Als letztes wurden sowohl die Metamodellerweiterung als auch die Transformation der modellierten Komponenten evaluiert. Die notwendigen Multicore-Parameter wurden erfolgreich modelliert und exportiert. Zusätzlich hat die verwendete Synthese eine Konfiguration des Betriebssystems gefunden, bei dem das Systemmodell bis auf das fehlende Speichermapping simulierbar ist.

Darüber hinaus hat diese Arbeit gezeigt, dass es möglich ist, mit PREEvision, AUTOSAR und Timing Architects Simulationswerkzeug eine durchgängige Werkzeugkette für die Modellierung und Simulation von Multicore-Architekturen mit Fokus auf die Echtzeitbeschränkungen zu implementieren. Dies wurde durch die notwendigen Erweiterungen des Modellierungswerkzeugs PREEvision umgesetzt und das Ergebnis anschließend evaluiert. Somit wurde demonstriert, dass mit dieser Werkzeugkette Multicore-Eigenschaften von Fahrzeug-Steuergeräten modelliert, exportiert und so simuliert werden können.

Ausblick

Im Laufe der Arbeit wurden einige Erweiterungsmöglichkeiten sichtbar, welche die durchgängige Entwicklungskette der Multicore-Architektur weiter verbessern könnten. Diese sollten im Hinblick auf zukünftige Arbeiten in Betracht gezogen werden. Eine Erweiterungsmöglichkeit wäre die Vereinfachung der Abbildung von Softwarekomponenten auf Kerne in AUTOSAR. Eine andere Erweiterungsmöglichkeit wäre die Berücksichtigung von Taktfrequenzen als Eigenschaft der Kerne im Standard. Diese wurde im Rahmen dieser Arbeit durch das AdminData-Konzept exportiert, da AUTOSAR diese momentan nicht unterstützt. Die letzte herausgearbeitete Timing-Beschränkung, die Auslastung der einzelnen Kerne, ist ebenfalls von dem AUTOSAR-Standard nicht betrachtet. Ein anderer Punkt wäre, dass der Synthesealgorithmus so erweitert werden kann, dass eine optimale Zuordnung auf Speicher gefunden wird. Damit wäre das mit PREEvision modellierte Systemmodell vollständig simulierbar.

Literatur

[Aho08] Alfred V Aho. *Compiler: Prinzipien, Techniken und Werkzeuge*. Pearson Deutschland GmbH, 2008.

[AKa] AUTOSAR-Konsortium. *AUTOSAR Standard - Classic Plattform*. URL: https://www.autosar.org/standards/classic-platform/ (besucht am 08. 11. 2018).

[AKb] AUTOSAR-Konsortium. *Generic Structure Template*. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TPS_GenericStructureTemplate.pdf (besucht am 08. 11. 2018).

[AK08] AUTOSAR-Konsortium. *Technical Overview*. 2008. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/3-2/AUTOSAR_SWS_VFB.pdf.

[AK14] AUTOSAR-Konsortium. *Software Component Template*. 2014. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-1/AUTOSAR_TPS_SoftwareComponentTemplate.pdf (besucht am 10. 12. 2018).

[AK17] AUTOSAR-Konsortium. *AUTOSAR Timing Extensions*. 2017. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TPS_TimingExtensions.pdf (besucht am 28. 10. 2018).

[Ama] Rakshith Amarnath. *Techniques for Memory Mapping on Multi-Core Automotive Embedded Systems*. Masterarbeit.

[AMA14] AMALTHEA. *Data Model Overview*. 2014. URL: https://www.eclipse.org/app4mc/help/app4mc-0.9.1/index.html#section3 (besucht am 25. 10. 2018).

[AMA17] AMALTHEA. *AMALTHEA Projekthomepage*. 2017. URL: http://www.amalthea-project.org/ (besucht am 25. 10. 2018).

[AUT14] AUTOSAR. *Guide to Multi-Core Systems*. 2014. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-1/AUTOSAR_EXP_MultiCoreGuide.pdf (besucht am 08. 11. 2018).

[Bar+06] C Bartelt u. a. *V-Modell XT Das deutsche Referenzmodell für Systementwicklungsprojekte*. 2006.

[CS13] Daniel Calegari und Nora Szasz. „Verification of Model Transformations: A Survey of the State-of-the-Art“. In: *Electronic Notes in Theoretical Computer Science* 292 (2013). Proceedings of the XXXVIII Latin American Conference in Informatics (CLEI), S. 5 –25. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2013.02.002. URL: http://www.sciencedirect.com/science/article/pii/S1571066113000042.

[EJ09] C. Ebert und C. Jones. „Embedded Software: Facts, Figures, and Future“. In: *Computer* 42 (Apr. 2009), S. 42–52. ISSN: 0018-9162. DOI: 10.1109/MC.2009.118. URL: doi.ieeecomputersociety.org/10.1109/MC.2009.118.

[Fik16] Felix Fikke. „Electric/Electronic-Architectures-Automating and optimizing communication matrices“. In: (2016).

[Fri17] Patrick Friederich. *Multicore Technologie in der Automotive Domäne:Partitionierung und Deployment sicherheitskritischer Funktionen*. Masterarbeit. 2017.

[Gmb] Vector Informatik GmbH. *Ebenen einer E/E-Architektur in PREEvision*. URL: https://www.vector.com/de/de/produkte/produkte-a-z/software/preevision/#c19639 (besucht am 01. 10. 2019).

[Gro] Artop User Group. *ARText - Timing Language*. URL: https://www.artop.org/artext/ (besucht am 10. 07. 2018).

[Gro16] O.M. Group. *OMG Meta Object Facility (MOF) Core Specification 2.5.1*. 2016. URL: http://www.omg.org/spec/MOF/2.5.1. (besucht am 20. 04. 2018).

[Hil12] Robert Hilbrich. „How to Safely Integrate Multiple Applications on Embedded Many-core Systems by Applying the "Correctness by Construction"Principle“. In: *Adv. Soft. Eng.* 2012 (Jan. 2012), 3:3–3:3. ISSN: 1687-8655. DOI: 10.1155/2012/354274. URL: http://dx.doi.org/10.1155/2012/354274.

[Infa] Infineon. *Infineon Introduces Microcontroller Multicore Architecture for Automotive Applications*. URL: https://www.infineon.com/cms/en/about-infineon/press/market-news/2011/INFATV201110-003.html (besucht am 20. 01. 2019).

[Infb] Vector Informatik. *MICROSAR Produktinformation*. (Besucht am 26. 01. 2019).

[Inf12] Vector Informatik. *Vector Informatik - AUTOSAR 4.0 Basissoftware*. 2012. URL: https://www.elektroniknet.de/elektronik-automotive/autosar-4-0-basissoftware-86812.html (besucht am 16. 12. 2018).

[ITE] ITEA. *AMALTHEA Project description*. URL: https://itea3.org/project/amalthea.html (besucht am 01. 10. 2019).

[KM07] Tapio Kramer und Dr. Ralf Münzenberger. *Absicherung des Echtzeitverhaltens mittels virtueller Integration*. 2007. URL: https://inchron.com/fileadmin/INCHRON/3-PDFs/EN/Paper_IAV_Tagung_Sim_u_Test_B_Mai10_pdf.pdf (besucht am 29. 10. 2018).

[Kon11] ARAMIS Konsortium. *ARAMiS Projektseite*. 2011-2015. URL: https://www.projekt-aramis.de/ (besucht am 25. 04. 2018).

[Kon16] ARAMIS Konsortium. *ARAMiS II Projektseite*. 2016-2019. URL: https://www.aramis2.org/ (besucht am 25. 04. 2018).

[Mac+15] Georg Macher u. a. „Automotive embedded software: Migration challenges to multi-core computing platforms“. In: *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*. IEEE. 2015, S. 1386–1393.

[Mac15] Georg Macher. „Framework for the Integrated Model-Based Development of Dependable Automotive Systems and Software“. Diss. Graz University of Technology, 2015.

[Mey14] Jan Meyer. „Eine durchgängige modellbasierte Entwicklungsmethodik für die automobile Steuergeräteentwicklung unter Einbeziehung des AUTOSAR Standards“. Diss. Universitätsbibliothek, 2014.

[Nav+10] Nicolas Navet u. a. „Multi-source and multicore automotive ECUs-OS protection mechanisms and scheduling“. In: *Industrial Electronics (ISIE), 2010 IEEE International Symposium on*. IEEE. 2010, S. 3734–3741.

[Neg15] Mircea Florin Negrean. „Performance Analysis of Multi-Core Multi-Mode Systems with Shared Resources - Principles and Application to AUTOSAR -“. englisch. Diss. 2015.

[OR10] William L. Oberkampf und Christopher J. Roy. *Verification and Validation in Scientific Computing*. 1st. New York, NY, USA: Cambridge University Press, 2010. ISBN: 0521113601, 9780521113601.

[Pow18] Kalkin Powale. „Automotive Powertrain Software Evaluation Tool“. In: (2018).

[Pre+07] A. Pretschner u. a. „Software Engineering for Automotive Systems: A Roadmap“. In: *Future of Software Engineering (FOSE '07)*. 2007, S. 55–71. DOI: 10.1109/FOSE.2007.22.

[Rei05] Clemens Reichmann. „Grafisch notierte Modell-zu-Modell-Transformationen für den Entwurf eingebetteter elektronischer Systeme“. Diss. 2005. ISBN: 3-8322-4700-9.

[Roß14] André Christian Roßbach. *Evaluation of Software Architectures in the Automotive Domain for Multicore Targets in regard to Architectural Estimation Decisions at Design Time*. 2014.

[SAG12] Oliver Scheickl, Christoph Ainhauser und Peter Gliwa. „Tool support for seamless system development based on AUTOSAR timing extensions“. In: *Proceedings of Embedded Real-Time Software Congress (ERTS)*. 2012.

[Sch17] Martin Schend. *Entwurf und Erstellung einer VEC (Vehicle Electric Container)-Schnittstelle für ein Elektrik/Elektronik-Architekturwerkzeug*. 2017.

[Wal16] M. Mitchell Waldrop. *The chips are down for Moore's law*. 2016. URL: https://www.nature.com/news/the-chips-are-down-for-moore-s-law-1.19338f (besucht am 10. 07. 2018).

[Wan17] Wenhao Wang. „Design process for the optimization of embedded software architectures on to multi-core processors in automotive industry“. Diss. Université de Cergy Pontoise, 2017.

[Web09] Julian Weber. *Automotive Development Processes*. 1st. Springer, Berlin, Heidelberg, 2009. ISBN: 978-3-642-01253-2.

A. Abkürzungsverzeichnis

AUTOSAR Automotive Open System Architecture

ECUs Electronic control units

ECU Electronic control unit

TIMEX AUTOSAR Timing Extensions

B. Anhang

…

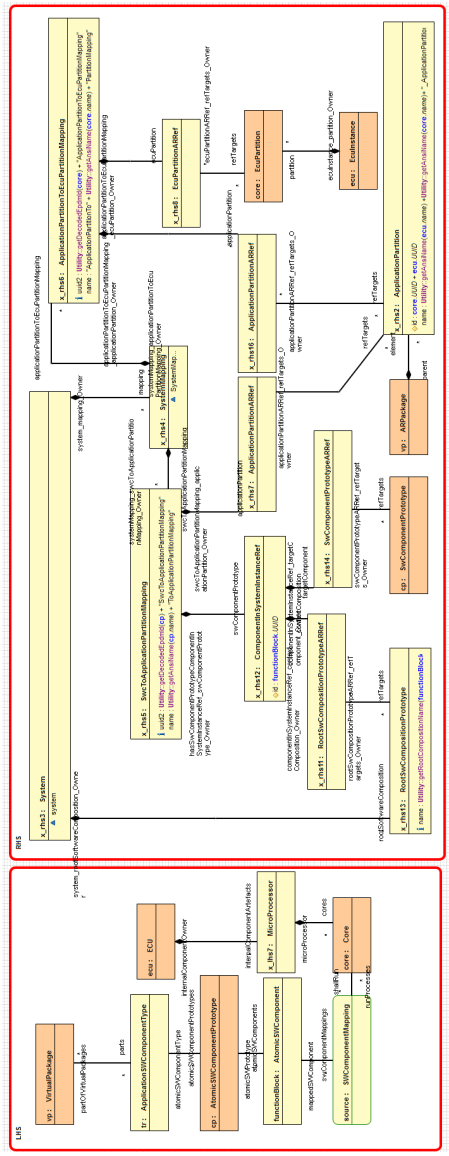

Abbildung B.1.: Atomic SWC auf Kern

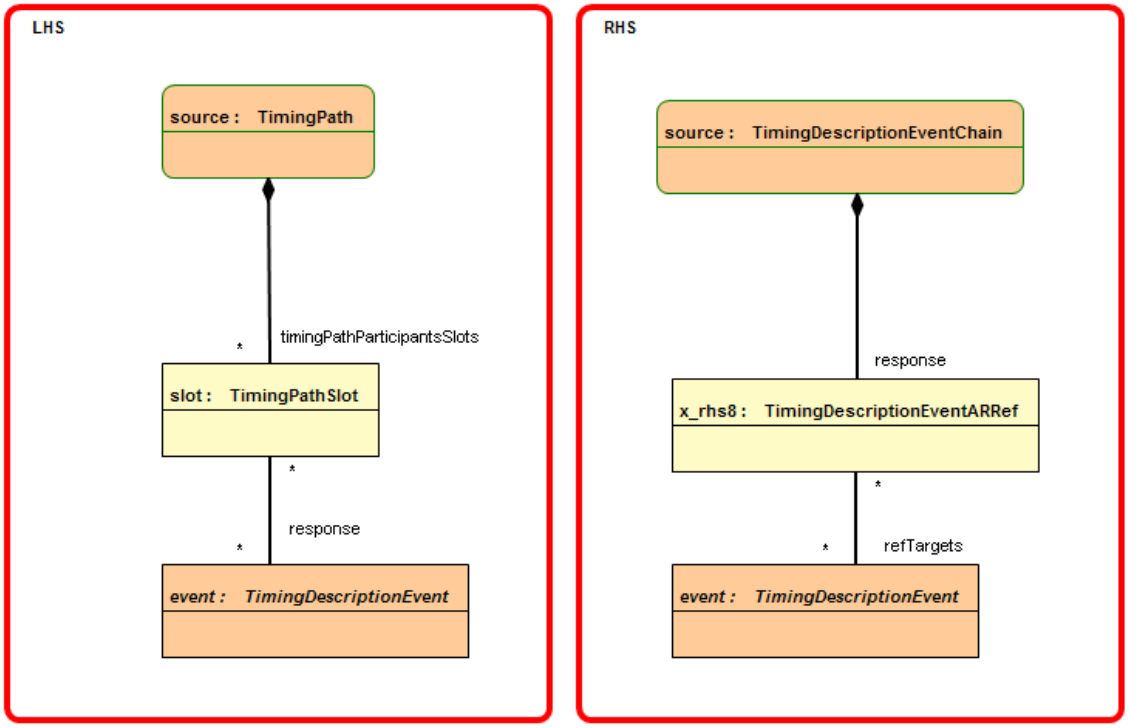

Abbildung B.2.: EventKette - Slot - Response

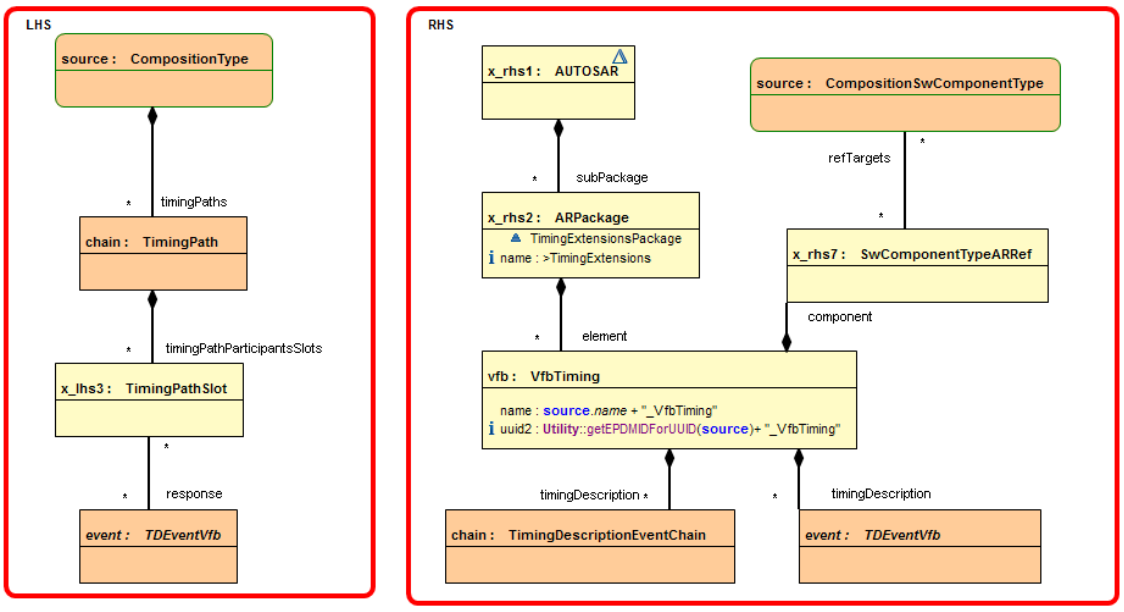

Abbildung B.3.: Transformation einer Reaktion auf logischer Ebene

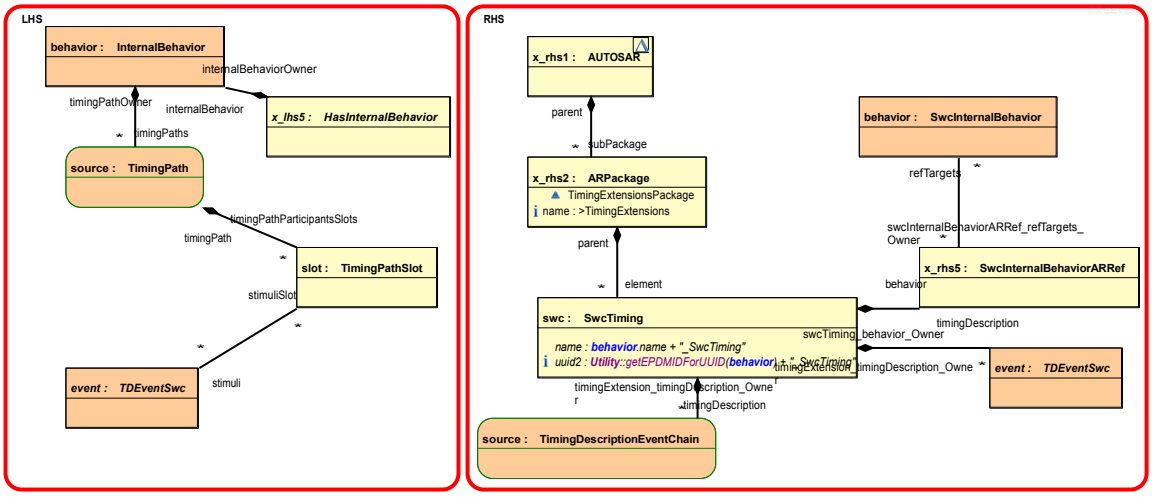

Abbildung B.4.: Transformation - Stimulus auf Ebene des internen Verhalten einer SWC - Teil 1

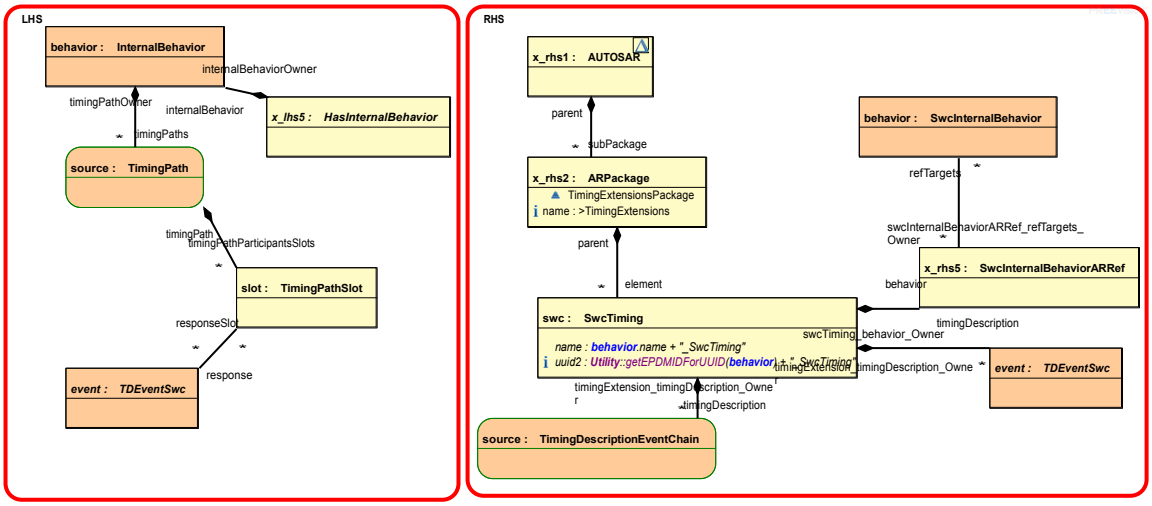

Abbildung B.5.: Transformation - Response auf Ebene des internen Verhalten einer SWC - Teil 1

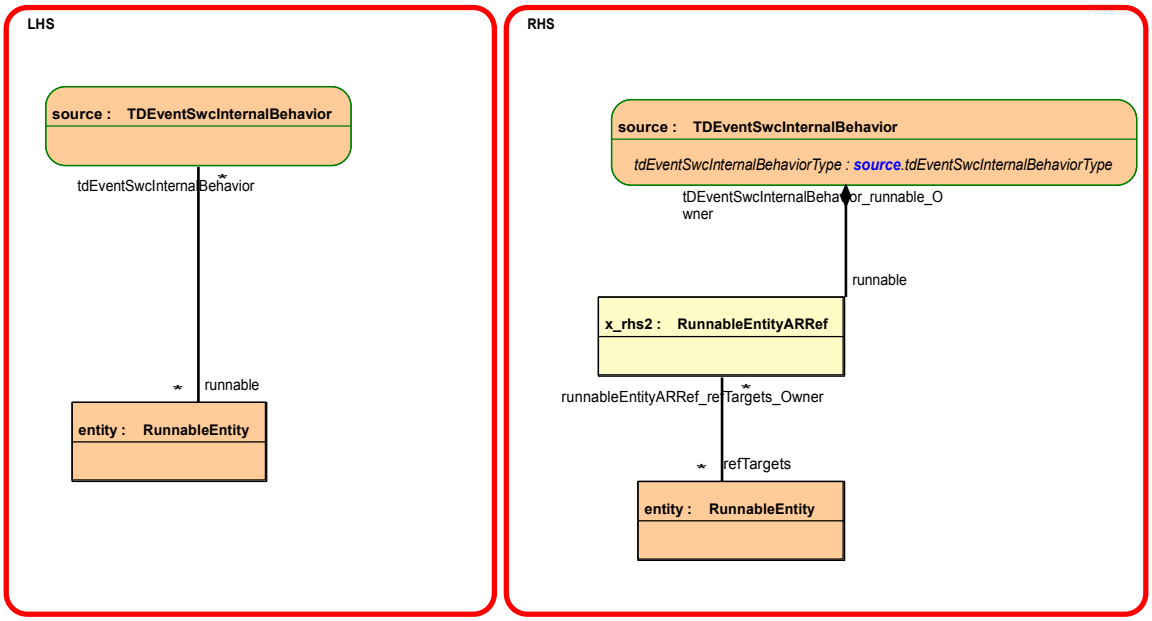

Abbildung B.6.: Transformation - Response auf Ebene des internen Verhalten einer SWC - Teil 2

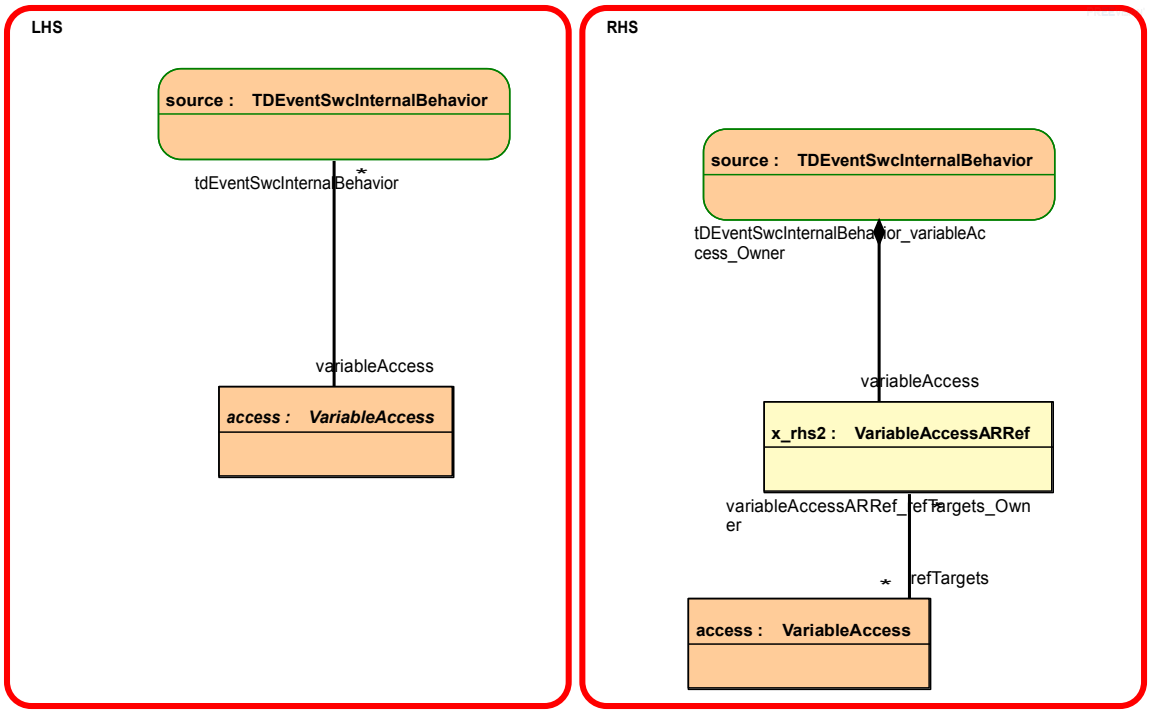

Abbildung B.7.: Transformation - Response auf Ebene des internen Verhalten einer SWC - Teil 3

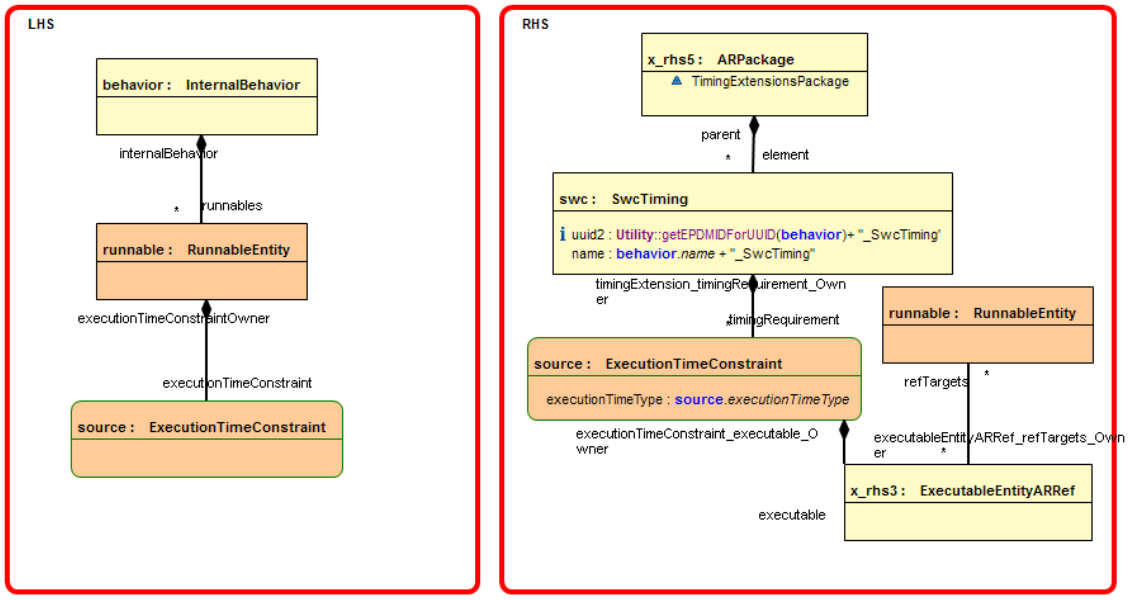

Abbildung B.8.: ExecutionTime - Teil 1

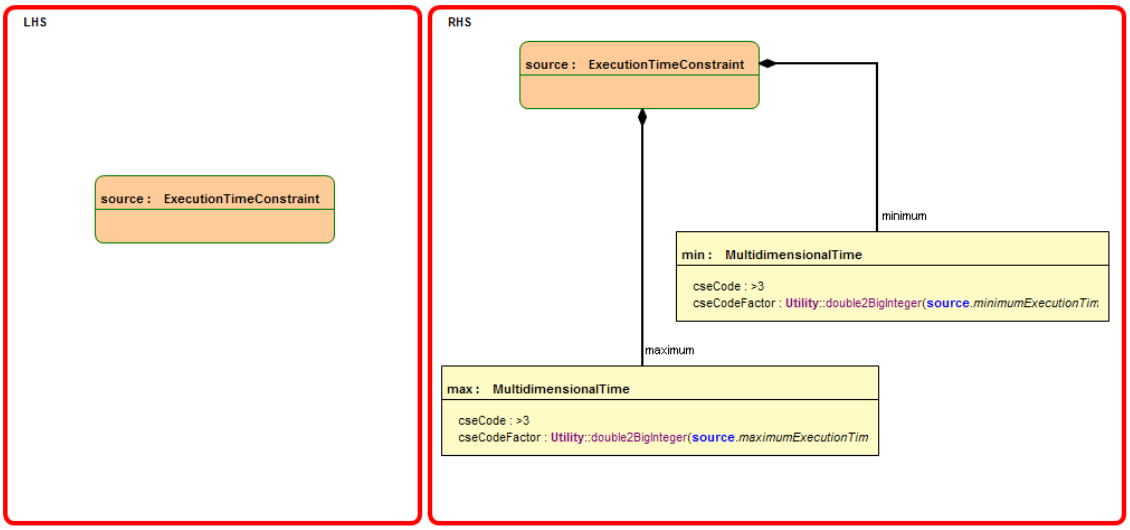

Abbildung B.9.: ExecutionTime - Teil 2

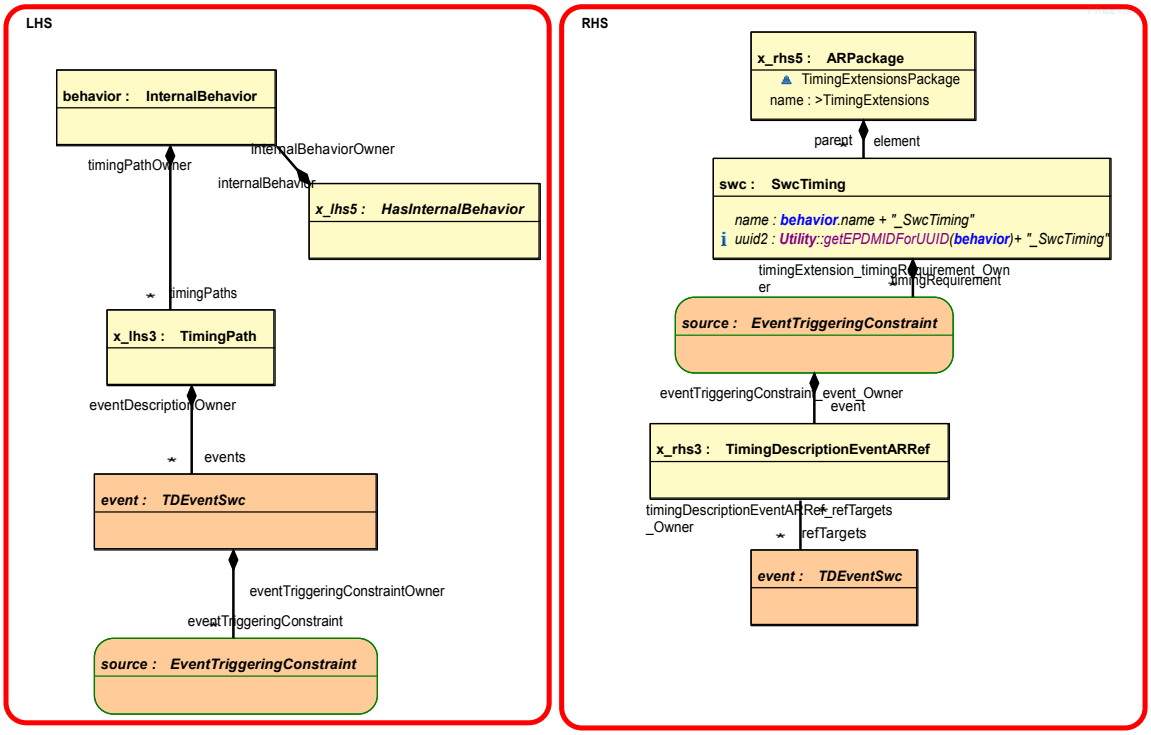

Abbildung B.10.: PeriodicEventTriggering Beschränkung - Teil 1

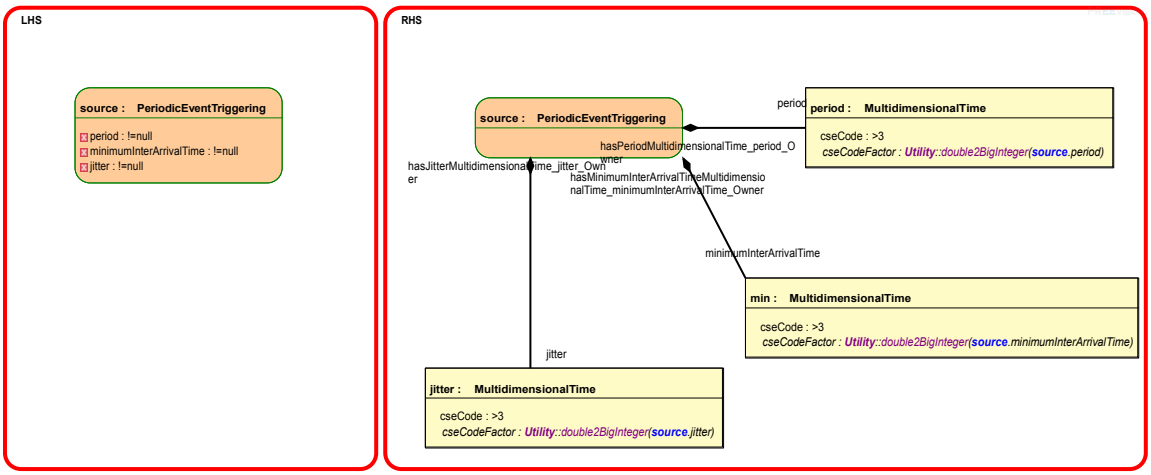

Abbildung B.11.: PeriodicEventTriggering Beschränkung - Teil 2

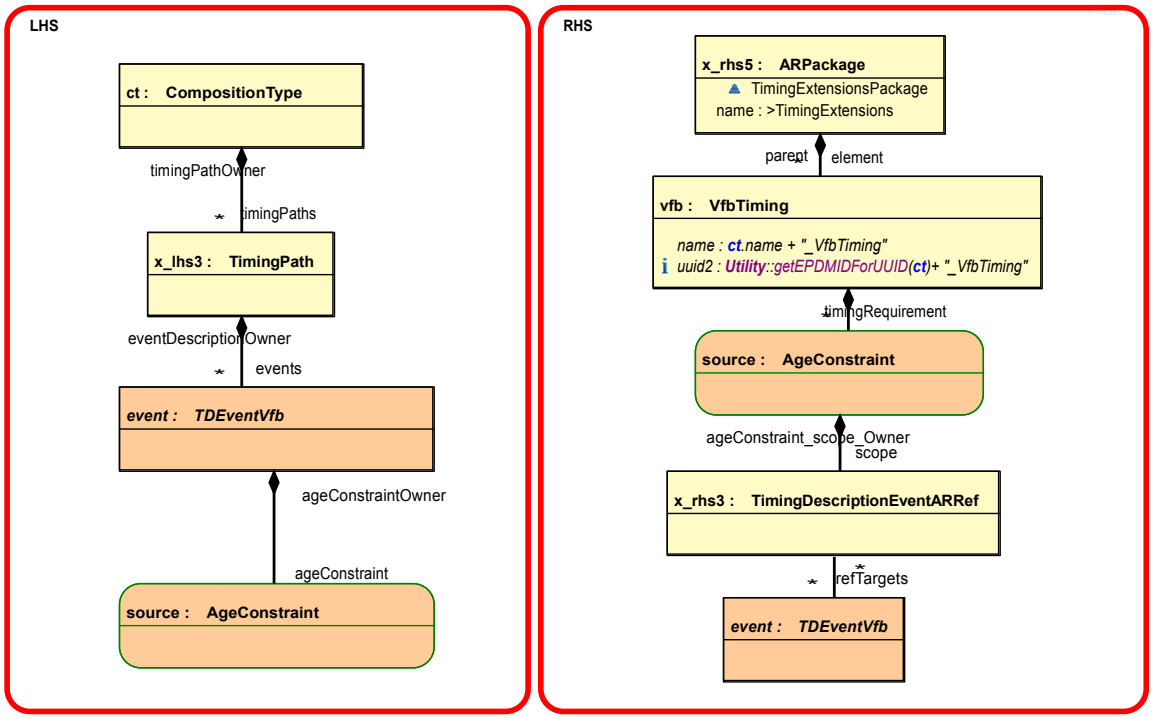

Abbildung B.12.: PeriodicEventTriggering Beschränkung - Teil 2

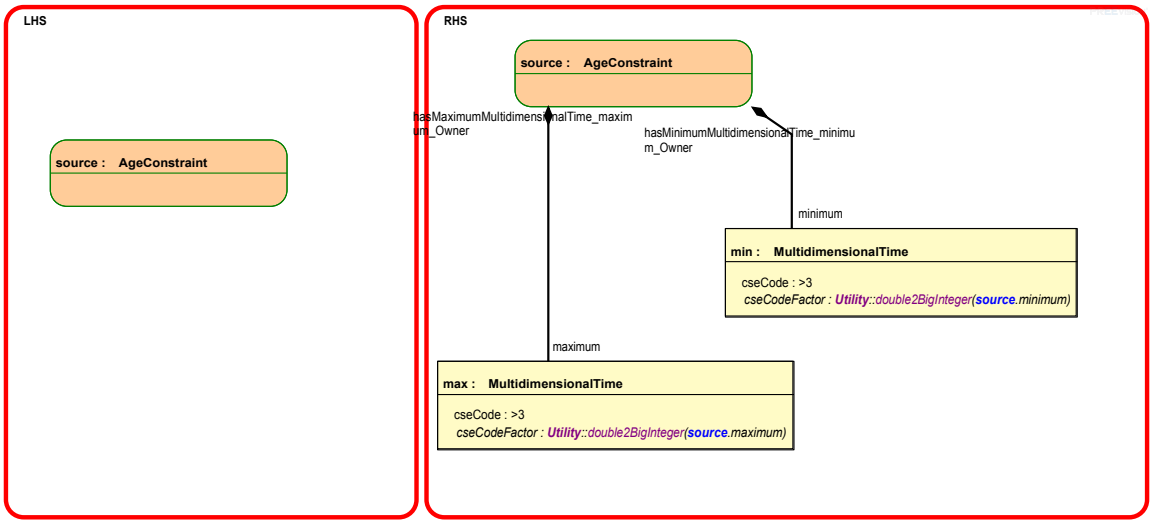

Abbildung B.13.: PeriodicEventTriggering Beschränkung - Teil 2