UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Fakultät für Elektrotechnik, Informatik und Mathematik
Heinz Nixdorf Institut und Institut für Informatik
Fachgebiet Softwaretechnik
Zukunftsmeile 1
33102 Paderborn

Eine durchgängige modellbasierte Entwicklungsmethodik für die automobile Steuergeräteentwicklung unter Einbeziehung des AUTOSAR Standards

Genehmigte Dissertation

zur Erlangung des akademischen Grades

„Doktor der Naturwissenschaften“ (Dr. rer. nat.)

vorgelegt von

Jan Meyer

Erstgutachter: Prof. Dr. Wilhelm Schäfer

Zweitgutachter: Prof. Dr. Holger Giese

Paderborn, den 19. Juli 2014

Kurzfassung

Die Automobilindustrie befindet sich derzeit in einem Wandel, ausgelöst durch die Verbreitung von internationalen Standards wie AutomotiveSPICE, ISO 26262, AUTOSAR und der Realisierung von innovativen und kooperierenden Funktionen im Fahrzeug. Hierdurch ergeben sich neue Problemstellungen und Herausforderungen, die das Zusammenspiel zwischen den Automobilherstellern und den Zulieferern nachhaltig verändern. Besonders für die Zulieferer ergeben sich bei der Entwicklung von Steuergeräten Änderungen, beispielsweise durch die Verwendung des AUTOSAR Standards. Dieser beinhaltet alle notwendigen Elemente, um die Strukturen einer Softwarearchitektur zu beschreiben. Dynamische Aspekte und die Anbindung einer Analysephase sind jedoch in dem Standard nicht vorgegeben. Somit besteht eine Lücke zwischen der Anforderungsanalyse und der Verwendung des AUTOSAR Standards. Zur Behebung dieses Defizits wird in dieser Arbeit der AUTOSAR Standard mit den internationalen Standards SysML und UML kombiniert, um einen durchgängigen und vollständigen Entwicklungsprozess von der Architektur bis zur Implementierung für die automobile Steuergeräteentwicklung zu etablieren (linke Seite des V-Modells). Hierzu wird die SysML/UML zunächst an die automobile Domäne angepasst, so dass eine domänenspezifische Sprache entsteht. Diese Sprache wird dazu verwendet ein Architekturmodell zu spezifizieren, welches sowohl eine Analyse als auch das dynamische Verhalten beinhaltet. Es werden darüber hinaus Echtzeitinformationen in das Modell integriert, da sie besonders für kooperierende Funktionen von großer Bedeutung sind. Die Daten aus dem SysML/UML Architekturmodell werden weiterhin in dem Konzept automatisch in ein Echtzeitanalyse-Werkzeug übertragen, um bereits in frühen Entwicklungsphasen eine Absicherung bezüglich der Echtzeit für die Architektur zu erreichen. Ferner wird in dieser Arbeit ein automatischer Übergang von UML nach AUTOSAR definiert, um die Informationen aus dem Architekturmodell in AUTOSAR weiterverwenden. Durch die Kombination von SysML/UML und der Transformation nach AUTOSAR werden die Schwachstellen des AUTOSAR Standards ausgeglichen. Hierdurch ergibt sich eine durchgängige Entwicklungsmethode und zwar von der Architektur bis zum Code.

Summary

The automotive industry is currently situated in a change, caused by the usage of international standards like AutomotiveSPICE, ISO 26262, AUTOSAR, and by the realization of innovative and cooperative functions in the vehicle. As a result of this new problems and challenges appear, that change sustainable the cooperation between the automotive original equipment manufacturer (OEM) and their suppliers. Especially for the suppliers arive some changes for the development of electronic control units, for example with the usage of the AUTOSAR standard. The standard contains all necessary elements to define the structure of a software architecture. Dynamically aspects and the integration of an analysis are not part of the standard. Therefore there is a gap in the development process between the requirements analysis and the software architecture with AUTOSAR. Accordingly this work combines the international standards SysML and UML with the AUTOSAR standard to establish a continouos and complete development process from the architecture to the implementation for the development of automotive electronic control units (left side of the V-model). For this purpose the SysML and UML is customized to the automotive domain, so that a domain specific language is defined. With this language an architecture model is specified, which includes both the analysis and the dynamic behavior. As well real-time information is integrated into the model, this is especially important for cooperating functions. In this work the data from the SysML/UML architecture model are automatically transformed to a real-time simulation tool, to validate the real-time for the architecture already in early development phases. Also an automatic transformation from UML to AUTOSAR is defined to reuse the information from the architecture model in AUTOSAR. With the combination of SysML/UML and the transformation to AUTOSAR the weakness of the AUTOSAR standard is compensated. Thus a continuous development method from the architecture to the implementation is defined.

Danksagung

Leider läßt sich eine wahrhafte Dankbarkeit mit Worten nicht ausdrücken.

Johann Wolfgang von Goethe
(1749 - 1932)

Diese Dissertation ist ohne Unterstützung nicht denkbar gewesen. Von daher möchte ich zu Beginn der Arbeit allen danken, die in irgendeiner Form zum Gelingen dieser Arbeit beigetragen und mich den langen Weg unterstützt haben.

Mein besonderer Dank gilt meinem Doktorvater, Herrn Professor Wilhelm Schäfer von der Universität Paderborn, für die Betreuung und Begutachtung dieser Arbeit. Während meiner Zeit als wissenschaftlicher Mitarbeiter in der Fachgruppe Softwaretechnik und am s-lab gab er mir immer den erforderlichen wissenschaftlichen Freiraum, um die Forschung an diesem Thema durchzuführen. Dies sorgte für ein gutes und innovatives Arbeitsklima. Hierdurch ergaben sich zahlreiche Diskussionen und Vorschläge, die die Arbeit verbessert und vorangebracht haben. In diesem Zusammenhang möchte ich allen damaligen Kolleginnen und Kollegen für die Zusammenarbeit danken. Insbesondere meinen damaligen Bürokollegen, die mir immer mit Rat und Tat zur Seite standen und dafür gesorgt haben, dass keine Phrasen Platz in dieser Arbeit gefunden haben. Weiterhin möchte ich diesem Zusammenhang Matthias Meyer für die sorgfältige Durchsicht der Arbeit und die konstruktiven Kommentare danken, die dazu beigetragen haben die Qualität der Arbeit zu verbessern und unnötige Wiederholungen zu entfernen („Ja doch - Effekt“).

Diese Arbeit ist in Zusammenarbeit mit dem international tätigen Automobilzulieferer HELLA KGaA Hueck & Co. aus Lippstadt entstanden. Ein gemeinsames Forschungsprojekt im Bezug auf die Softwareentwicklung bei einem Automobilzulieferer war die Grundlage für diese Arbeit, welche schließlich in der Mitarbeit am Forschungsprojekt *Software Plattform Embedded Systems 2020* (SPES 2020) geführt hat. Von daher möchte ich der Firma und deren Mitarbeiterinnen und Mitarbeitern – vor allem aus der Gruppe Prozesse, Methoden und Tools (PMT) – für die gute und konstruktive Zusammenarbeit herzlich danken. Sie haben mir wertvolle Hilfe geleistet beim Erkennen der Problemstellungen und Herausforderungen, die in der derzeitigen Entwicklung von automobilen Steuergeräten vorliegen. Des Weiteren hatten sie durch ihre aufbauenden und praxisbezogenen Anmerkungen ebenso Anteil daran, dass die hier erarbeitete Lösung immer einen sehr großen Praxisbezug hatte und direkt in der Serienentwicklung eingesetzt werden konnte.

Zum Schluss möchte ich mich bei meiner Familie und meinen Freunden bedanken. Sie hatten in den letzten Jahren immer viel Verständnis dafür, dass

ich an meiner Dissertation gearbeitet habe und daher nicht immer viel Zeit für sie hatte. Besonderer Dank gilt hierbei meiner Frau Karina, die mich immer unterstützt hat und auch viel Verständnis zeigte, wenn ich mal wieder meine Zeit am Schreibtisch beim Verfassen dieser Arbeit zugebracht habe. Ebenso möchte ich meinem Vater Friedhelm danken, der in fleißiger Weise immer darauf geachtet hat, dass die Arbeit lesbar und auch für Leute, die nicht Spezialisten in diesem Bereich sind, verständlich ist.

Paderborn,
den 19. Juli 2014

Jan Meyer

Inhaltsverzeichnis

Ich glaube an das Pferd. Das Automobil ist nur eine vorübergehende Erscheinung.

Kaiser Wilhelm II. (1859 - 1941)

1 Einleitung

Die Weltwirtschaft setzt sich aus einer Vielzahl unterschiedlichster Branchen und Industriezweigen zusammen. Eine von ihnen ist die Automobilindustrie. Diese wiederum besteht aus den Automobilherstellern – auch Original Equipment Manufacturer (OEM) genannt – und deren zahlreichen Zulieferern. Insgesamt sind mehrere Millionen Beschäftigte allein in diesem Industriezweig tätig [HKM+05]. Es werden weltweit jedes Jahr über 70 Millionen Fahrzeuge produziert, mit steigender Tendenz [oMVMO06], denn nur das Automobil ermöglicht eine individuelle Mobilität [BS13]. Auf Grund dieser Tatsache ist die Automobilindustrie ein lukrativer und wirtschaftlich bestimmender Sektor innerhalb der Weltwirtschaft. Daher ist es nur selbstverständlich, dass ein harter Konkurrenzkampf um jeden Kunden geführt wird. Dieser Kampf wird nicht nur durch den Preis, sondern in erster Linie durch neue Funktionalitäten im Automobil bestimmt. So überleben wirtschaftlich nur diejenigen, die in der Lage sind, Innovationen und neue Funktionalitäten rasch und kostengünstig umzusetzen. Es ist daher nicht verwunderlich, dass deshalb in der Automobilindustrie viele Innovationen und neue Schlüsseltechniken entwickelt werden [ku13]. Ein Schwerpunkt der Entwicklung sind derzeit neuartige Antriebssysteme, wie beispielsweise der Elektroantrieb, um möglichst schadstoffarm zu fahren [PSSZ12]. Demzufolge ist die Automobilindustrie ein stetiger Vorreiter in Sachen Technologie.

Die Innovation und die Qualität der Produkte und deren Teilsysteme kann aber nur dann hochklassig sein und überzeugen, wenn deren Entwicklung ebenso ideenreich und fehlerfrei ist. Kunden reagieren heutzutage schnell auf mangelnde Qualität. Dies hat vor nicht allzu langer Zeit der Automobilhersteller Toyota erfahren müssen, als mögliche Probleme mit dem Gaspedal bekannt wurden. Die Verunsicherung der Kunden und Nutzer führte letztendlich zu einem vorübergehenden Produktions- und Verkaufsstopp [FAZ10], der zwangsläufig auch wirtschaftliche Folgen hatte. Deswegen ist es zwingend erforderlich, dass bereits im Entwicklungsstadium die notwendigen Schritte eingeleitet werden,

um neben der reinen Funktionsentwicklung einen hohen und gleichbleibenden Qualitätsstandard sicherzustellen.

Die Automobilindustrie muss sich gegenwärtig vielen Herausforderungen stellen. Neben den steigenden Kundenwünschen und dem Erhalt des Qualitätsstandards müssen speziell die politischen Vorgaben, die sowohl national als auch international gesetzlich vorgegeben sind, umgesetzt werden. Exemplarisch wird auf die Themen Umweltschutz und Verkehrssicherheit verwiesen. Beim Umweltschutz steht besonders der Treibstoffverbrauch und die Verringerung der freigesetzten Abgaswerte im Fokus. Diese Ziele sind nur durch die Optimierung des Antriebs bzw. durch neuartige Antriebssysteme wie die Hybrid-, Elektro- oder Wasserstofftechnik zu erreichen. Beim Thema Verkehrssicherheit kommt es vor allem darauf an, die Zahl der Verkehrsunfälle insgesamt und hierbei insbesondere die Anzahl der verletzten bzw. getöteten Personen zu verringern. Zur Vermeidung von Unfällen und deren Schadenspotential werden immer neue Sicherheitssysteme entwickelt und in die Fahrzeuge integriert [KRL12].

Zur Erreichung einer erhöhten Verkehrssicherheit werden vermehrt sicherheitskritische Funktionen umgesetzt, z. B. bei der automatischen Einleitung eines Bremsvorgangs nach Erkennung einer Gefahrensituation. Die Automobilindustrie ist bekannt dafür, dass sie sicherheitskritische Funktionen realisieren kann. Dies ist anhand der Historie des Automobils abzulesen (vgl. Abbildung 1.1).

Die Entwicklung des Automobils begann im Jahre 1886 mit dem Automobilpatent von Carl Benz (Patentschrift DRP 37435) [AG13, Sei09b]. Dieser entwickelte ein von einem Motor angetriebenes Fahrzeug, was für damalige Verhältnisse einer Revolution gleichkam. In den folgenden Jahren wurden die einzelnen Komponenten immer weiter ausgebaut und verbessert. Das Automobil bestand aber weiterhin aus mechanisch arbeitenden Komponenten, ergänzt um kleinere Neuerungen.

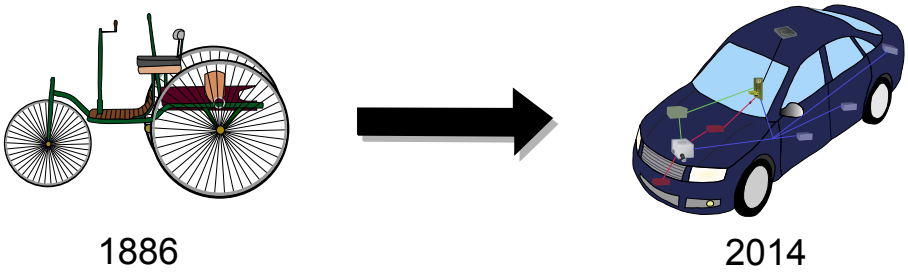


Abb. 1.1. Wandel vom ersten Automobil hin zum heutigen Fahrzeug mit mannigfachen Steuergeräten

Ein Quantensprung war der Einsatz von Software innerhalb eines Fahrzeuges. Erst ab den siebziger Jahren des vorherigen Jahrhunderts hielt die Software

Einzug in das Automobil. Die Software erlaubt von nun an eine präzisere Überwachung und Steuerung von einzelnen Teilfunktionen. Sie wird dabei lokal auf sogenannten Steuergeräten (Electronical Control Unit) (ECU)s eingesetzt. Diese haben das Ziel – wie der Name schon vermuten lässt – die mechanischen Elemente zu steuern.

Das Einsatzgebiet der ersten Software war die Überwachung und Steuerung der Motoreinspritzung. Sie diente dazu, die Einspritzung und den Verbrennungsprozess im Motor so abzustimmen, dass eine optimale Fahr- und Verbrauchsleistung erreicht wurde. Die Software wurde daher – in diesem Stadium ihres Einsatzes – für die Optimierung von bestehenden elektronischen Bauteilen verwendet. Vornehmlich wurden Bauteile für den Einsatz von Software ausgewählt, die zur Verbesserung des Fahrverhaltens beitrugen.

Durch die so erzielten positiven Resultate setzte ein grundlegender Technologiewandel ein, der nicht nur bis heute andauert, sondern das Zukunftsbild der Automobilindustrie prägen wird. So wurden und werden immer mehr Funktionen im Fahrzeug durch Software realisiert. Ausschlaggebend war, dass durch den Einsatz der Software zusätzliche Funktionen umgesetzt werden konnten, die mit reiner Mechanik bisher nicht zu realisieren waren. Ein Beispiel hierfür ist das Anti-Blockier System (ABS) (vgl. [Bos13]). Entwürfe für ein Anti-Blockier-System gab es bereits schon länger, jedoch konnten sie auf Grund der zur Verfügung stehenden Technik nicht umgesetzt werden. Erst durch den Einsatz von Software war eine so präzise Steuerung möglich, dass diese Sicherheitsfunktion in ein Auto integriert werden konnte.

Neben der Verbesserung der Fahrfunktion kommt die Software in weiteren Bereichen des Automobils wie bei den Komfort- und Sicherheitssystemen zum Einsatz. Seit den 80er Jahren wurden verstärkt Sicherheitssysteme wie – oben schon ausgeführt – das ABS oder das Elektronische Stabilitäts-Programm (ESP) entwickelt und in die Fahrzeuge eingebaut. Gegenwärtig liegt der Entwicklungsschwerpunkt vor allem bei den Fahrerassistenzsystemen (ab den 90er Jahren), die nicht nur zum überwiegenden Teil aus Software bestehen, sondern erstmals auf der kooperativen Zusammenarbeit von vernetzten Steuergeräten basieren. Sie unterstützen den Fahrer bei der rechtzeitigen Erkennung von Gefahrensituationen und dienen dazu, Unfälle zu vermeiden [Rei10b]. Letztendlich trägt die Software dazu bei, die Vorgaben und Ziele der Automobilindustrie umzusetzen. Als Beispiele für Fahrerassistenzsysteme sind der Abstandsregeltempomat Adaptive Cruise Control (ACC) und das Fahrstreifenerkennungssystem (Lane departure warning) (LDW) [WHW10] zu nennen.

Durch die immer neuen Einsatzgebiete ist es nicht verwunderlich, dass sowohl die Anzahl der Steuergeräte insgesamt als auch die Größe der Software in einem Fahrzeug stetig gestiegen ist und immer noch weiter steigen wird [ZHLK10, BKK+12]. Mittlerweile werden in einem Oberklassenfahrzeug bis zu 90 Steuergeräte mit mehr als 4000 Funktionen [BMT11] verbaut. Schätzungen gehen davon aus, dass in nicht allzu ferner Zukunft mehrere Gigabyte an

Software in einem Automobil zu finden sind [ZHLK10]. Hierdurch wird deutlich, dass die Softwareentwicklung bei der Neukonzipierung eines Automobils einen immer größer werdenden Anteil einnimmt [WFO09, BKK+12].

Ein Steuergerät erfüllt dabei eine zugedachte Funktion und wird vielfach von einem Zulieferer nach den Vorgaben des Automobilherstellers entwickelt. Das Steuergerät wird anschließend durch den Automobilhersteller in das Gesamtsystem des Fahrzeuges integriert. Bei der Integration kommt es häufig zu Komplikationen. Die Kombination von steigender Funktionalität und der damit einhergehenden Zunahme der Komplexität innerhalb der heterogenen Fahrzeugarchitektur führt zu Schwierigkeiten gerade bei der Entwicklung von neuen Produkten. So sind Entwicklungsfehler nicht auszuschließen, da die komplexe Kommunikationsstruktur und die Zusammenarbeit zwischen den einzelnen Steuergeräten nicht immer vollständig von den Entwicklern der Zulieferer nachvollzogen werden kann. Zwangsläufig entstehen hierdurch Qualitätsdefizite in der Gesamtfahrzeugarchitektur, aber genauso in den einzelnen Steuergeräten. Diese Defizite können zu teuren Rückrufaktionen führen [Krö13]. Die Entwicklungsfehler lassen sich durch verschiedene Statistiken belegen. So sind fast 40% aller Ausfälle von Personenkraftwagen (Pkw) auf Elektronik- bzw. Softwarefehler zurückzuführen [eV08].

Es sind derzeit nicht immer alle Anforderungen bezüglich der zeitlichen Abarbeitung von einzelnen Funktionen vorhanden [SVN07]. Insofern muss der OEM bei der Entwicklung der Steuergeräte stärker mit eingebunden werden, da nur hier die Kenntnisse über die Aufgaben der Steuergeräte und deren Signale vorliegen. Außerdem ist zu berücksichtigen, dass der OEM neuerdings vielfach eigene Softwarekomponenten bei der Entwicklung von Steuergeräten beisteuert, beispielsweise in Form von fusionierten Signalen (eine zentrale Quelle für mehrere Steuergeräte) [BRR11]. Durch diesen Umstand ist das Zusammenspiel der einzelnen Steuergeräte ein neuer und wesentlicher Aspekt, der bei der zukünftigen Entwicklung zu berücksichtigen ist. Die Veränderungen, hervorgerufen durch die Software, sind in der Übersichtsgrafik 1.2 gegenübergestellt. Hierbei ist nicht nur der Fortschritt der Softwareentwicklung zu erkennen, sondern auch der damit einhergehende Wandel in dem Beziehungsgeflecht zwischen Automobilhersteller und Zulieferer. Dies wird immer intensiver, da der Zulieferer zusätzliche Software des OEMs integrieren muss und somit der OEM als Käufer aber auch als Zulieferer auftritt.

Dieser Wandel, den die Software ausgelöst hat, führt nicht nur zu einem neuen Verhältnis zwischen OEM und Zulieferer, sondern erfordert auch eine Verbesserung der bei der Entwicklung eingesetzten Prozesse, Methoden und Werkzeuge. Sie sind kontinuierlich anzupassen. Nur so kann die Entwicklung unter optimalen Voraussetzungen stattfinden. Dadurch wird gewährleistet, dass die Entwicklungsschritte zeitnah und kostengünstig erfolgen und dies bei gleichbleibender Qualität der Produkte. Gerade bei sicherheitskritischen Funktionen ist dies von besonderer Relevanz, um einen Ausfall während der

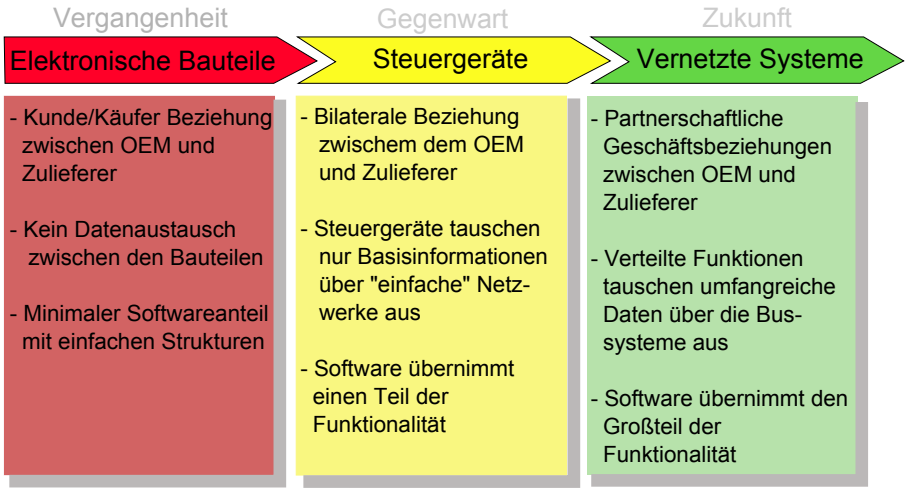


Abb. 1.2. Fortentwicklung der Software in einem Fahrzeug

Betriebszeit zu vermeiden. Von daher bedingt die Entwicklung solcher Funktionen ein standardisiertes Vorgehen unter Berücksichtigung von vorgegebenen Techniken nach dem neuen Sicherheitsstandard der ISO 26262 [fS10b].

Die Verwendung der neuen Sicherheitsrichtlinie und der Wandel hin zu einer vernetzten und kooperativen Fahrzeugarchitektur erfordert notwendige Anpassungen beim Entwicklungsprozess eines automobilen Zulieferers und den dort eingesetzten Methoden und Techniken. Um die notwendigen Verbesserungen durchführen zu können, sind zunächst die vorhandenen Defizite bei den heutigen Entwicklungsphasen zu identifizieren. Diese werden im nächsten Abschnitt herausgearbeitet.

Wie jede andere Branche hat auch die Automobilindustrie ihre ganz speziellen Besonderheiten und Merkmale. Die Automobilbranche ist vor allem durch das Zusammenwirken von Automobilherstellern und Zulieferern geprägt. Die Zulieferer selber sind noch weiter unterteilt in Tier-1, Tier-2 und Tier-3 (vgl. Abbildung 1.3). Der Tier-1 besitzt eine hohe Integrationskompetenz und fertigt komplexe Systeme während der Tier-2 Subsysteme und Module zuliefert und der Tier-3 Komponenten und Bauteile beisteuert [WFO09]. Zusammen übernehmen die Zulieferer nach jetzigem Stand bereits einen Entwicklungsanteil von über 65% [WFO09] an einem Fahrzeug. Somit befindet sich bereits ein wesentlicher Anteil des technischen Know-Hows bestehend aus Wissen und Erfahrung bei der Entwicklung von Steuergeräten und besonders bei der Integration von Hardware und Software bei den Zulieferern [ZHLK10]. Jedoch sind die Zulieferer nicht an einen Automobilhersteller gebunden, wie es beispielsweise in der Avionik zu finden ist. Vielmehr ist es gängige Praxis, dass ein Zulieferer immer mehrere OEMs als Kunden mit dem gleichen Produkt beliefert (vgl. Abbildung 1.3). Hierdurch ist es möglich, die Kosten für ein Steuergerät zu

minimieren. Jedoch bedeutet dies ebenso, dass der Automobilhersteller immer die Auswahl zwischen verschiedenen Zulieferern hat und damit der Konkurrenzkampf stetig zunimmt.

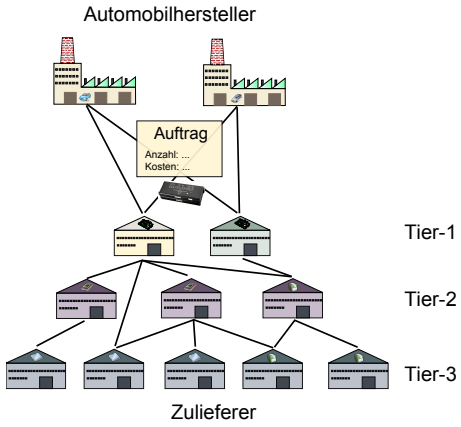


Abb. 1.3. Zusammenspiel zwischen Automobilhersteller und ihren Zulieferern

Der Faktor *Kosten* wird zukünftig angesichts des Konkurrenzdrucks eine noch größere Bedeutung erlangen [Mar07, Mal10]. Dieser Druck sorgt für einen Verdrängungseffekt bei den Automobilherstellern und innerhalb der Zuliefererindustrie. Er wird in den kommenden Jahren noch stetig wachsen [WFO09, Bec07]. Der Konkurrenz- und Kostendruck in der Automobilindustrie ist nichts Neues, sondern besteht schon seit Jahrzehnten. Wobei jedoch festzustellen ist, dass er immer verbissener geführt wird. Dieser Druck existiert aber nicht nur auf Seiten der Automobilhersteller, sondern wird vermehrt an die Zulieferer weitergegeben. Dies ist nicht verwunderlich, da der Zukauf der Teilsysteme (Steuergeräte) der größte Kostenfaktor beim OEM ist. Von daher wird versucht, einen Großteil der Kostenminimierung auf die Zulieferer abzuwälzen [Mal10, Bec07].

Die Zulieferer ihrerseits versuchen mit zwei Strategien dem Kostendruck zu begegnen (vgl. Abbildung 1.4). Zum einen werden durch verbesserte Techniken und Methoden die Entwicklungszeiten (Time-to-Market) verkürzt. Eine verkürzte Entwicklungszeit unter Verwendung gleichbleibender Ressourcen minimiert die entstehenden Kosten [USJ08]. Dies kann aber nur dann erfolgreich sein, wenn die Techniken und Methoden eine schnellere Entwicklung und somit eine Effizienzsteigerung der Entwickler zulassen. Zum anderen wird dem Kostendruck durch eine verbesserte Qualität der Produkte begegnet [PBK10, Bec07]. Denn neben dem Faktor *Preis* ist natürlich der Qualitätsstandard ein wesentliches Entscheidungskriterium für den Automobilhersteller bei der Auswahl des Zulieferers und dessen angebotener Produktpalette für die Entwicklung von neuen Funktionen [Hoy00]. Qualitätsprobleme verursachen zu-

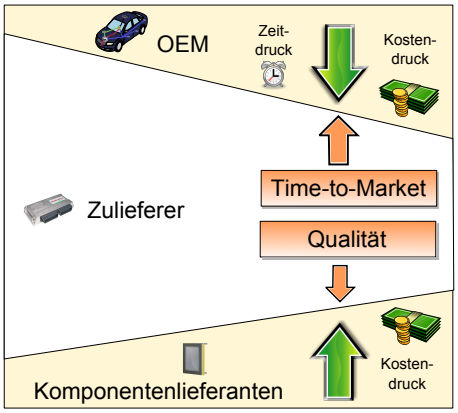


Abb. 1.4. Zunehmender Kostendruck für die Zulieferer

dem immer zusätzliche Kosten. Von daher lohnt es sich, in Qualitätssteigerung zu investieren. Die notwendige Optimierung der Entwicklung wird bei den Zulieferern zukünftig noch deutlicher im Vordergrund stehen [USJ08], da der Konkurrenzkampf sich noch ausweiten wird [WFO09].

1.1 Defizite bei der heutigen Steuergeräteentwicklung

Die Steuergeräteentwicklung wird durch verschiedene Standards bestimmt. Auf der einen Seite steht die ISO 26262 [fS10b], die für die Entwicklung von sicherheitskritischen Systemen Vorgaben in Bezug auf Methoden und Techniken macht. Auf der anderen Seite gibt es das Entwicklungsreferenzprozessmodell Automotive SPICE [dAV10]. Dieses definiert zehn verschiedene Entwicklungsprozesse, die nach dem V-Modell angeordnet sind. Neben der Einteilung in die Entwicklungsprozesse werden im Automotive SPICE aber ebenso verschiedene bewährte Techniken (Best Practices) für die einzelnen Entwicklungsprozesse vorgegeben. In der Abbildung 1.5 ist die linke Seite des V-Modells (Entwicklung) mit den vorgegebenen Methoden und Techniken wiedergegeben. Bei der Umsetzung einzelner Vorgaben besteht jedoch noch Optimierungspotential. So werden z. B. die Techniken der Verifikation der Systemarchitektur, die Definition des Ressourcenverbrauchs, die Nachverfolgbarkeit (Traceability) und die dynamische Verhaltensbeschreibung nur unter großem Aufwand und durch den Einsatz von viel Ressourcenkapazitäten umgesetzt.

Die Entwicklung der Steuergeräte als auch die Aufteilung in die verschiedenen Entwicklungsschritte nach Automotive SPICE [dAV10] hat sich bewährt. Die Methoden und Techniken innerhalb der einzelnen Entwicklungsschritte

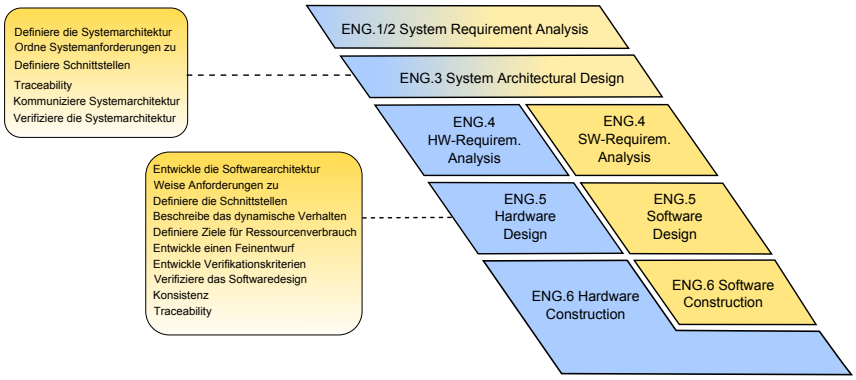


Abb. 1.5. Methoden und Technikvorgaben vom Automotive SPICE Referenzprozessmodell

funktionieren weitestgehend. So werden das Anforderungsmanagement (Requirements Engineering) oder die Funktionsentwicklung methodisch mit Erfolg umgesetzt. Es existieren aber noch Defizite bei der Architekturmodellierung mit dem neuen AUTOSAR Standard. Zusätzlich sind die Zusammenhänge bzw. die Übergange zwischen den jeweiligen Entwicklungsschritten oftmals nicht definiert, so dass bei den Übergängen zwischen den einzelnen Phasen – exemplarisch kann hier auf den Übergang von der System- zur Softwarearchitektur verwiesen werden – noch Verbesserungspotential vorhanden ist. Es bestehen bei den Übergängen immer noch Methoden- und Werkzeugbrüche, die zusätzlichen Aufwand verursachen und letztendlich Zeit und Geld kosten. Von daher fehlen systematische Übergange, die die einzelnen Methoden und Werkzeuge besser miteinander verbinden.

Durch die fehlenden Übergänge werden Informationen in mehreren nicht untereinander verknüpften Dokumenten abgelegt. Dies erschwert die Erstellung der Nachverfolgbarkeit (Traceability). Eine solche wird aber dann zwingend benötigt, wenn Anpassungen oder Änderungen vorzunehmen sind und die Auswirkungen mittels einer Auswirkungsanalyse (Impact Analyse) ermittelt werden müssen. Durch die Ablage in verschiedenen Dokumenten sind häufige Suchaktionen und durch Redundanzen herbeigeführte Inkonsistenzen gang und gäbe. In der Abbildung 2.33 sind die im Einsatz befindlichen Entwicklungsschritte und deren Übergänge dargestellt. Es ist erkennbar, dass die Übergänge vom Analysemodell zur Systemarchitektur und von der Systemzur Softwarearchitektur nicht hinreichend gelöst sind. Es sind Brüche in der Entwicklungskette vorhanden, die für eine durchgängige und optimale Entwicklung geschlossen werden müssen. Somit ist eine durchgängige Nachverfolgbarkeit nicht gegeben [BS05]. Hierdurch bedingt, ist ein zusätzlicher Entwicklungsaufwand erforderlich.

Die unvorteilhafte Dokumentenablage beginnt mit dem Erhalt der Entwicklungsartefakte vom OEM. Hierunter fallen sowohl die Anforderungen des Lastenheftes als auch die zusätzlichen Informationen und Vorgaben des OEM, wie beispielsweise die Kommunikationsbeziehungen oder aber mitgelieferte Softwarekomponenten (vgl. Abbildung 1.6). Die Anforderungen werden üblicherweise in entsprechenden Anforderungsmanagementwerkzeugen abgespeichert (wie beispielsweise das Werkzeug IBM Rational Dynamic Object Oriented Requirements System (DOORS) [IBM14a]). Somit ist eine Nachverfolgbarkeit gegeben. Die weiteren Artefakte, wie z. B. die Kommunikationsbeziehungen, werden auf Seiten der Zulieferer sodann häufig auf informale Art und Weise gespeichert. Hierdurch ist es zum einen schwierig, eine Nachverfolgbarkeit herzustellen und zum anderen, die Daten in den unterschiedlichen Artefakten konsistent zu halten.

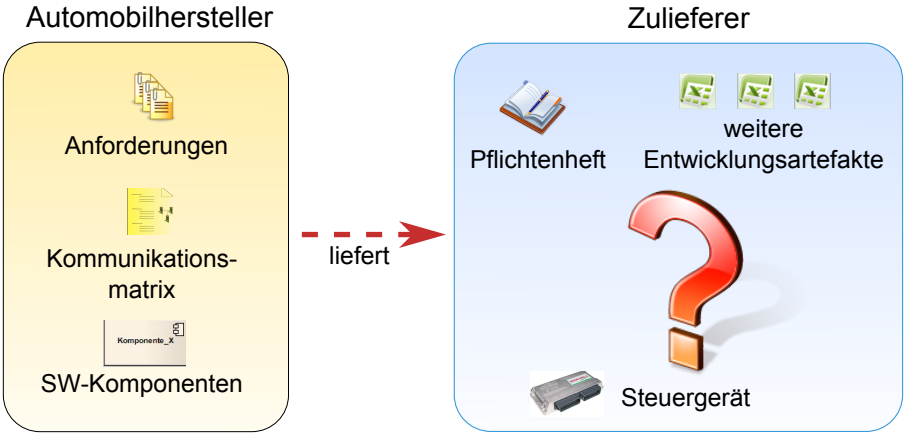


Abb. 1.6. Verschiedene Entwicklungsartefakte bei der dokumentbasierten Entwicklung und die Frage, wie daraus ein Steuergerät entsteht

Die informale Speicherung der Daten wäre noch akzeptabel, sofern das Datenmaterial dauerhaft konstant bleibt und nicht einem laufenden Veränderungsprozess unterliegen würde. Da aber die Entwicklungsphasen bei den Automobilherstellern genauso fließend sind wie bei den Zulieferern, ist es zwangsläufig, dass es zu Aktualisierungen und somit zu Änderungen und Anpassungen kommt. Ein manuelles Einpflegen der geänderten Daten in die Entwicklungsmodelle und ggf. in die entsprechenden Dokumente ist somit unabdingbar, damit Neuerungen bei der Steuergeräteentwicklung mit einfließen und es zu keinen Inkonsistenzen kommt. Eine solche Vorgehensweise erfordert einen enormen Arbeitsaufwand und kann darüber hinaus wegen der vorhandenen Redundanzen (Datei, Dokument, Modell) zu Inkonsistenzen führen. Die Gefahr von Inkonsistenzen ist besonders dann gegeben, wenn die vorzunehmenden Ak-

tualisierungen aus Zeitmangel bei der Entwicklung nicht sofort eingearbeitet werden.

Die Kommunikation zwischen den Steuergeräten erfolgt über Kommunikationsbusse. Vielfach werden Informationen über die Kommunikation innerhalb des Fahrzeugs in Tabellenform geliefert. Sie sind für jeden Bus unterschiedlich und spezifizieren die verschiedenen Buspakete, die enthaltenen Signale, den Sender, die Empfänger und die Qualität der Signale. Auf Seiten der Zulieferer werden die Informationen typischerweise in Form von Excel-Tabellen oder DOORS-Modulen neben den Anforderungen gespeichert. Durch eine entsprechende Verlinkung mit der Architektur und den Anforderungen wird die Nachverfolgbarkeit sichergestellt. Die Informationen müssen aber bei jeder Änderung manuell nachgepflegt werden. Dabei müssen nicht nur die einzelnen Signale gepflegt werden, sondern ebenso die Schnittstellen in der Architektur, da diese auf den Signalen beruhen. Somit müssen die Daten redundant nachgepflegt werden, was sehr arbeitsintensiv ist.

Neben dem zuvor beschriebenen Defizit ergibt sich aus der Einführung des AUTOSAR Standards für die Modellierung der Softwarearchitektur eine weitere Erschwernis bei der heutigen Steuergeräteentwicklung. Der AUTOSAR Standard beinhaltet eine Referenzarchitektur, eine Methodik und standardisierte Schnittstellen (vgl. Kapitel 2.7). Dagegen bietet der Standard keine Möglichkeiten zur Analyse der Anforderungen. Es wird nicht die Option eröffnet, Anforderungen zu analysieren oder das Verhalten der einzelnen Komponenten abzubilden. Zur Entwicklung von Steuergeräten gehört aber neben der Spezifikation der Architektur ebenso die Modellierung des Verhaltens. Dies wird im AUTOSAR Standard nicht abgedeckt [Gra11]. Von daher kann der Standard nur in den späteren Entwicklungsphasen *Design* und *Implementierung* für die Modellierung der Architektur eingesetzt werden. Für eine durchgängige und lückenlose Entwicklungsmethodik ist es jedoch unverzichtbar, dass vor dem Einsatz des AUTOSAR Standards eine Analyse der Anforderungen durchgeführt wird. Gerade in dieser Phase werden die Grundlagen für die Architekturentscheidungen geschaffen.

Bei der Integration der Analysephase mit dem AUTOSAR Standard ist es von Relevanz, dass ein wohldefinierter Übergang entwickelt wird. Ein solcher systematischer Übergang muss dabei automatisch eine Nachverfolgbarkeit herstellen [Gra11]. Dadurch wird sichergestellt, dass ein Großteil der Daten aus der Analysephase automatisiert übernommen wird. Redundanzen und Inkonsistenzen werden so ausgeschlossen. Aufbauend auf den übernommenen Daten muss die Option vorhanden sein, das Kommunikations- und Komponentenverhalten innerhalb der AUTOSAR Architektur zu spezifizieren. Hierzu fehlen aber entsprechende Möglichkeiten im Standard. Insofern ist der AUTOSAR Standard in diesem Bereich durch entsprechende Methoden zu ergänzen. Durch eine solche Erweiterung wird zum einen eine schnellere Entwicklung ermöglicht und zum anderen auf konstruktive Art und Weise die Qualität der Software

sichergestellt. Diese Vorgehensweise ist besonders bei der Entwicklung von sicherheitskritischen Komponenten essenziell.

Ein weiteres Problemfeld wird bei der Vernetzung der Steuergeräte und der Zunahme der Software offenkundig. Die zeit genaue Kommunikation zwischen den jeweiligen Komponenten ist die Basis für funktionierende Innovationen. Dabei liegt ein besonderer Stellenwert auf der Modellierung der Echtzeit und deren Anforderungen. Im gegenwärtigen Entwicklungsprozess werden die Ressourcen- und Zeitanforderungen oftmals noch ignoriert und finden erst bei der Integration Beachtung [BS05]. Das Vorgehen ist vergleichbar mit der Methode *Kopf in den Sand stecken*. In den frühen Entwicklungsphasen werden sie vernachlässigt und es wird darauf vertraut, dass sie schon eingehalten werden. Erst wenn dies nicht eintrifft, erfolgt eine genauere Analyse und entsprechende Gegenmaßnahmen werden ergriffen. Dies ist eine ressourcenverschwendende Methode eines *Trial-and-Error* Ansatzes [Bec07]. Entsprechend ist für den Systemarchitekten erst sehr spät im Entwicklungsprozess erkennbar, ob sich die von ihm gewählte Architektur für das Steuergerät als zutreffend erweist. Die Überprüfung erfolgt erst nachdem die Software auf die Hardware integriert wurde. Bei der sich dann anschließenden Überprüfung der oftmals als nicht funktional bezeichneten Anforderungen [CaPL09] – die für Echtzeitsysteme jedoch funktionale Anforderungen (Qualitätsanforderungen) darstellen – wie Laufzeit oder Speicherverbrauch [WTS08] werden häufig Performanceprobleme identifiziert.

Die Korrektur der Architektur zur Einhaltung der gestellten Qualitätsanforderungen ist in diesem Stadium mit enormen Kosten verbunden, da der Entwicklungsprozess schon recht weit fortgeschritten ist. Es kann dann nicht ausgeschlossen werden, dass Änderungen an der Architektur Anpassungen im Feindesign nach sich ziehen. Deswegen muss die komplette Entwicklung noch einmal durchlaufen werden. Dies bedingt einen großen Zeit- und Ressourcenaufwand [BS05]. Von daher ist es erstrebenswert, dass bereits zu einem möglichst frühen Zeitpunkt im Entwicklungsprozess eine Architekturbewertung gegen die Qualitätsanforderungen durchgeführt wird. Dies kann in Form einer virtuellen Ausführung z. B. durch eine Echtzeitsimulation erfolgen. Eine solche Bewertung gibt Aufschluss darüber, ob die gewählte Architektur und die darin getroffenen Designentscheidungen für das Steuergerät kompatibel sind. So wird bereits in einer frühen Entwicklungsphase eine Kostenreduktion und eine Verkürzung der Produktentwicklungszeit angestoßen [SHI+11].

Zusammenfassend sind folgende Problem- und Fragestellungen im derzeitigen Entwicklungsprozess eines Automobilzulieferers zu identifizieren:

1. Eine systematische Definition der Übergänge zwischen den jeweiligen Entwicklungsphasen ist bei den bisherigen Verfahren nicht gegeben [BFH+10]. Hieraus ergeben sich Brüche sowohl im Prozess an sich als auch bei den eingesetzten Methoden. Vor allem muss der Übergang zwischen den Anforderungen und der am Ende stehenden AUTOSAR Architektur

geschlossen werden, damit eine durchgängige Entwicklung gegeben ist. Die eingesetzten Techniken und Methoden müssen dabei auf (semi-)formale Weise spezifiziert werden, damit sie eindeutig sind und somit auch bei verteilten Entwicklungen (in verschiedenen Standorten national wie international) eingesetzt werden können.

2. Die Anzahl der Funktionen, ihre Vernetzung untereinander und damit einhergehend die Komplexität der Steuergeräte nimmt stetig zu [LT09]. Diese müssen so spezifiziert werden, dass sie durch eine geeignete Abstraktion noch überschaubar und damit beherrschbar sind. Notwendig ist die Nutzung von geeigneten Analysemodellen, die die Abhängigkeiten der einzelnen Funktionen beschreiben.

3. Durch die stetige Weiterentwicklung nehmen die Kosten für die Hardware ab. Gleichzeitig benötigt die Software aber immer mehr Rechenkraft. Deswegen werden immer leistungsfähigere Rechenkerne in die Steuergeräte bei einem gleichbleibenden Stückkostenpreis verbaut. Trotzdem reichen einzelne Rechenkerne oftmals nicht aus, so dass auf Mehrprozessorsysteme (Multicore oder aber mehrere per Bus verbundene Rechenkerne) zurückgegriffen wird. Die optimale Verteilung und die Kommunikation zwischen den jeweiligen Rechenkernen muss dann aber erst noch gefunden und abgestimmt werden [Nat07].

4. Viele Arbeitsschritte, wie beispielsweise die Informationen der Kommunikationssignale, werden derzeit noch mehrmals manuell, d. h. händisch vom Entwickler, in verschiedene Arbeitsartefakte eingepflegt. Es fehlt an Automatismen, die die Konzentration auf die wesentliche Arbeit erlauben. Die eigentliche Arbeit des Entwicklers ist die Umsetzung der geforderten Funktionalitäten für das Steuergerät (vgl. Kapitel 4.2). Für die Entwicklung von Automatismen wird eine durchgängige Entwicklungsmethode benötigt, um die Informationen automatisch von einem Arbeitsartefakt zum nächsten zu transportieren. Hierdurch wird eine Optimierung der Entwicklung sichergestellt, so dass die Entwicklungszeiten verkürzt werden.

5. Der AUTOSAR Standard kann ausschließlich für spätere Entwicklungsphasen genutzt werden. Es fehlt ihm aber eine Analysephase (vgl. Kapitel 2.7). Darüber hinaus ist eine Spezifikation des Verhaltens nicht in ausreichendem Maße vorhanden. Dies muss durch geeignete Methoden kompensiert werden, um eine durchgängige Entwicklungsmethodik mit Werkzeugunterstützung zu ermöglichen [BFH+10].

6. Zeitliche Informationen nehmen an Bedeutung zu und müssen im Zusammenhang mit der Steuergeräteentwicklung modelliert und verifiziert werden. Dies muss bereits zu Beginn der Entwicklung erfolgen, damit eine spätere kostenträchtige Architekturänderung vermieden wird. Um die zeitlichen Informationen entsprechend für Architekturentscheidungen zu nutzen, müssen sie zunächst formal spezifiziert werden. Aber allein die

Dokumentation der zeitlichen Informationen ist noch nicht ausreichend. Vielmehr sind sie noch zu analysieren. Dies kann in Form einer Echtzeitanalyse erfolgen.

1.2 Lösungsansatz AutoMoMe

Die zuvor benannten sechs Problemstellungen werden in dieser Arbeit aufgegriffen, und es wird ein Lösungskonzept in Form der *Automotive Modeling Methodology (AutoMoMe)*[1] vorgestellt. Zielsetzung der Methodik ist die Optimierung der Verfahrensabläufe durch die Erarbeitung einer durchgängigen Entwicklungsmethode [Sch05a] für automobile Steuergeräte, einhergehend mit geeigneten Analysemodellen, die eine frühzeitige Simulation und Optimierung des Echtzeitverhaltens erlauben, sowie die Kompensation des nicht vorhandenen Übergangs zum automobilspezifischen Standard AUTOSAR. Durch die Umsetzung dieser Vorgaben wird eine nachhaltige und signifikante Zeitminimierung [GGH+06] bei der automobilen Steuergeräteentwicklung erreicht. Es werden nicht nur die Arbeitsabläufe transparenter gestaltet, sondern dies führt – was gerade im Konkurrenzkampf der Zulieferer von besonderer Bedeutung ist – zu einer entscheidenden Kostenreduzierung im Entwicklungsprozess.

Um diese generellen Zielsetzungen zu erreichen, sind dabei zunächst konkrete Teilziele umzusetzen. Diese ergeben sich aus den eben beschriebenen Problemstellungen. Dabei ist ein wichtiges Teilziel die Erstellung eines durchgängigen modellbasierten Entwicklungsablaufs und zwar von den Anforderungen bis hin zur Architektur mit AUTOSAR (siehe hierzu Problemstellungen eins und drei) [GB10]. Wegen der Durchgängigkeit sind Daten nur einmal zu spezifizieren, um bisherige Doppelarbeiten überflüssig zu machen und die Entwicklungszeit zu reduzieren. Voraussetzung dafür ist der Einsatz von semi-formalen Sprachen und Modellen mit entsprechenden Transformationen, um einen automatisierten Informationsfluss zu gewährleisten.

Ein Bestandteil der semi-formalen Modelle ist das Analysemodell, das die Vernetzung und die Abhängigkeiten der Funktionen untereinander dokumentiert. Hierauf aufbauend wird die System- und daran anschließend die Softwarearchitektur entwickelt. Für alle Modelle wird mittels der Systems Modeling Language (SysML) bzw. der Unified Modeling Language (UML) das sogenannte Architekturmodell erstellt. Das Architekturmodell besteht aus verschiedenen Schichten und beinhaltet sowohl die funktionale, logische als auch die technische Architektur (vgl. Abbildung 4.4). Es wird auf diese beiden Sprachen zurückgegriffen, weil sie sich in der Praxis bewährt haben

[1] Auf Grund von Veröffentlichungen und der Weiterverwendung im internationalem Umfeld wird eine englische Abkürzung verwendet.

[TB03] und geeignete Modellierungsmittel bereitstellen, die ebenso das Verhalten einschließen [FGDT07] und eine domänen-spezifische Erweiterung erlauben [TB03]. Außerdem sind zahlreiche erprobte Werkzeuge vorhanden, die in einem industriellen Serienprojekt eingesetzt werden können. Ebenso ist die Basis-Software eines Automotive Open System Architecture (AUTOSAR) Systems vom AUTOSAR Konsortium mittels der UML definiert [AUT10i]. Diese Elemente müssen für eine vollständige Architekturübersicht in die Modellierung mit integriert werden. Ferner bestehen nicht alle Systeme nur aus reinen AUTOSAR Komponenten [SRT+05]. Durch die Verwendung von SysML und UML ist sichergestellt, dass die Gesamtarchitektur eines automobilen Steuergerätes (inklusive der Basis-Software) spezifiziert werden kann. Somit ist es sinnvoll, die eingesetzten Modellierungssprachen an die automobile Steuergeräteentwicklung anzupassen, um das Domänenwissen zu nutzen und eine geeignete Abstraktion zu erstellen. Dies erfolgt mit einer leichtgewichtigen Erweiterung in Form eines UML Profils [Wat07]. Da Steuergeräte besondere Eigenschaften besitzen und Limitierungen zu beachten sind, ist eine entsprechende auf eingebettete Systeme anwendbare Modellierungssprache erforderlich. Für eine angepasste Steuergeräteentwicklung ist die Modellierung der Kommunikationsdaten, die zum Größtenteil von den OEMs zugeliefert werden, notwendig. Darüber hinaus werden die Betriebssystemdaten und die Ressourceninformationen benötigt, um die Software optimal auf die Rechenkerne des jeweiligen Steuergerätes zu verteilen.

Es existieren schon Arbeiten, die mithilfe der SysML oder der UML eine Steuergeräteentwicklung beschreiben (vgl. [Mut05, Mar02, DAV09, dJ07]). Diese haben aber die UML nicht an die automobile Domäne angepasst, so dass das Domänenwissen nicht genutzt werden kann, beispielsweise die Nutzung der Kommunikationsdaten der Busse. Andere Arbeiten haben zwar eine Anpassung an die Automobildomäne vorgenommen (vgl. [PA12, RBvdBS02, ZBF+05, RLE02, STBW05, WFH+06]). Sie berücksichtigen dabei jedoch nicht den AUTOSAR Standard, der heutzutage zwingend erforderlich ist. Bei den Arbeiten (vgl. [GHN09, SWCD13]) wurde zwar ein automatisierter Übergang nach AUTOSAR definiert, aber es wurde nur die Modellierung der Applikationsbestandteile beschrieben und nicht die komplette Modellierung einer Steuergeräteentwicklung. Weiterhin gibt es Arbeiten, die die Modellierung der Echtzeit im Fokus haben (vgl. [GK02]). Sie berücksichtigen aber nicht die Transformation zu Echtzeitanalysewerkzeugen, um aus der Modellierung einen Vorteil zu ziehen. Von daher sind die Ansätze für eine Steuergeräteentwicklung nicht ausreichend.

Das hier vorgestellte Konzept AutoMoMe weist im Gegensatz zu den vorhandenen Arbeiten einen automatisierten Übergang zwischen dem Architekturmodell und dem AUTOSAR Modell für eine Steuergeräteentwicklung auf. Hierdurch ist eine durchgängige Entwicklung möglich. Vergleichbar hierzu ist der Ansatz der EAST-ADL [Ass13]. Jedoch ist bei diesem der Übergang nicht durch Transformationen realisiert, sondern durch eine parallele Modellierung.

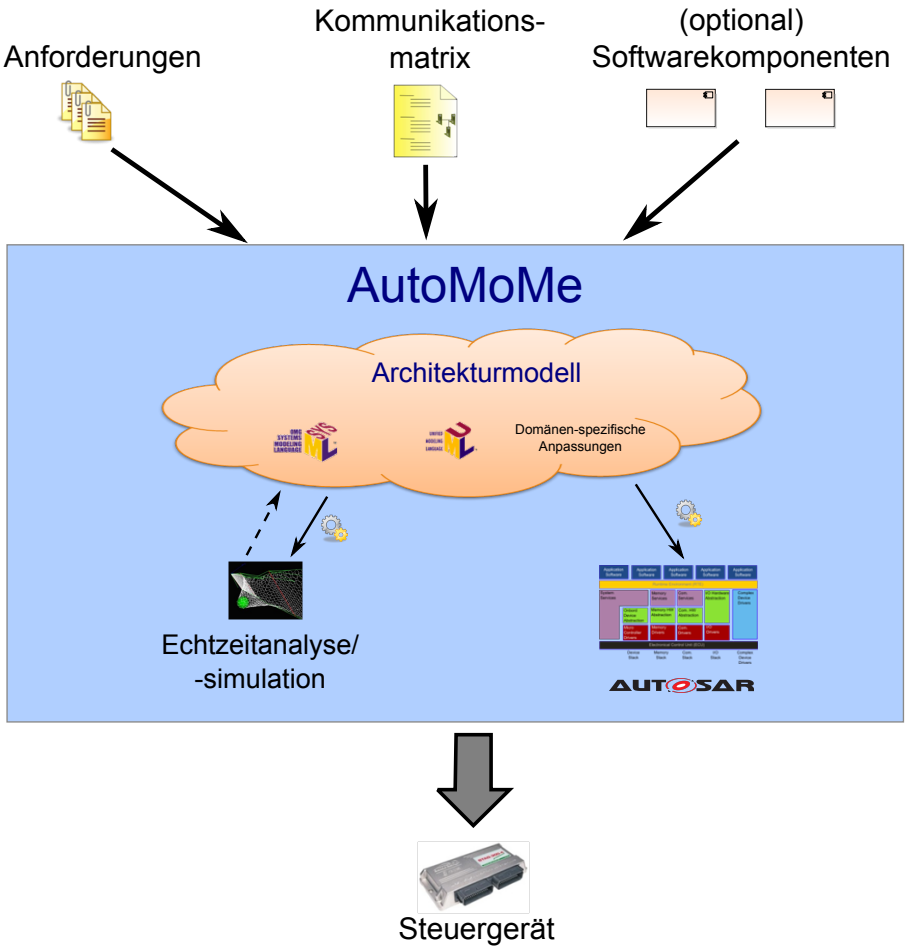


Abb. 1.7. AutoMoMe Lösungsansatz

Die Daten werden somit nicht automatisiert zwischen den Modellen ausgetauscht, so dass sie nochmals manuell eingegeben werden müssen, was nicht optimal ist. Das erarbeitete AutoMoMe Konzept weist hier Vorteile auf und ist für eine durchgängige und transparentere Entwicklung zweckmäßiger gestaltet.

Zusätzlich beinhaltet das Konzept die Verwendung von Echtzeitanalysen bzw. -simulationen bereits in frühen Entwicklungsphasen. Die bisherigen Arbeiten auf diesem Gebiet [Kra12b, KM10b, KM10a, PPE+08, RER06, RHER07] beschränken sich auf die Echtzeitanalyse bzw. -simulation. Jedoch beschreiben sie nicht, wie die Daten für die Analyse bzw. für die Simulation bereitgestellt werden. Die AutoMoMe bindet die Echtzeitanalyse/-simulation in die Entwicklung ein, so dass die Daten aus dem Architekturmodell wiederverwendet werden. Hierdurch ist es möglich, ohne größeren zusätzlichen Aufwand eine Anal-

yse bzw. Simulation durchzuführen und hierdurch Rückschlüsse auf die Güte der Architektur zu gewinnen und Architekturentscheidungen durch Fakten zu unterstützen. Diese sind zwingend erforderlich, um die vernetzten und aus mehreren Rechenkernen bestehenden Steuergeräte zu entwickeln.

Der Schwerpunkt der Arbeit ist, zum Teil bestehende allgemeine Ansätze und Methoden zur Architekturmodellierung [SFB09, SPE09] in eine Domäne, in diesem Fall die Automobildomäne, einzuführen und so miteinander zu integrieren, dass sich ein durchgängiger und lückenloser Entwicklungsprozess ergibt. Dabei geht es zunächst darum die Besonderheiten der Domäne zu ermitteln, um dann die Methodik entsprechend der Bedürfnisse anpassen zu können. Eine Anpassung ist oftmals erforderlich, damit die Methodik bestmöglich eingesetzt werden kann und auf Akzeptanz bei den Entwicklern stößt. Vor allem geht es bei der Integration in die Domäne darum, Methoden und Techniken soweit zu entwickeln, dass sie in industriellen Serienprojekten eingesetzt werden können. Dies erfordert den Einsatz von bewährten Werkzeugen und ein Konzept für die Einführung. Hierzu zählen die Planung von Schulungen, die Auswahl von Pilotprojekten und die Erstellung der notwendigen Dokumentation.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit AutoMoMe gliedert sich in acht Kapitel, die in drei Abschnitte aufgeteilt sind (vgl. Abbildung 1.8). Zunächst wird im ersten Abschnitt, der aus drei Kapiteln besteht, die Problembeschreibung der derzeitigen Steuergeräteentwicklung im Automobilbereich aufgezeigt. Ausgehend von den Problemstellungen wird im zweiten Abschnitt das Konzept der AutoMoMe einschließlich der Echtzeitsimulation und der Übergang zum AUTOSAR Standard erarbeitet. Im dritten Abschnitt wird die prototypische Umsetzung des Konzeptes in einem Werkzeug beschrieben und beinhaltet zusätzlich die praxisnahe Erprobung des Konzeptes. Die Arbeit schließt mit der Zusammenfassung und dem Ausblick einschließlich der Einordnung mit der praktischen Relevanz für die Automobilindustrie ab. Im Nachfolgenden werden die einzelnen Kapitel genauer beschrieben.

Im ersten Kapitel wird einleitend eine Motivation in die Thematik der Probleme bei der Steuergeräteentwicklung im Automobilbereich gegeben. Hieraus ergeben sich Fragen- und Problemstellungen wie auch Zielvorgaben, die anschließend aufgegriffen und für die Lösungen erarbeitet werden. Das zweite Kapitel vermittelt die augenblickliche Entwicklung der Software im Automobilbereich. Es werden die verschiedenen Kennzeichen herausgearbeitet, die in diesem Industriesektor anzutreffen sind und die für eine Unterscheidung zu anderen Branchen sorgen. Des Weiteren werden themenverwandte Arbeiten vorgestellt, die sich mit einer ähnlichen Zielsetzung auseinandersetzen

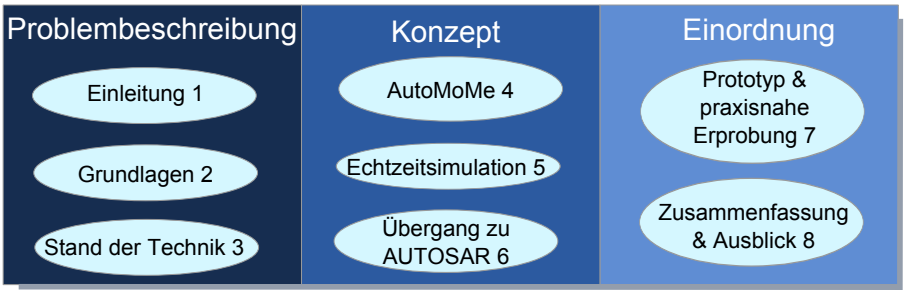


Abb. 1.8. Aufbau der Arbeit

bzw. beschäftigt haben. Im dritten Kapitel werden die Grundlagen der Arbeit vorgestellt, die den Stand der Technik wiedergeben. Dies beinhaltet die Entwicklungsmethodik des Sonderforschungsbereichs 614 „Selbstoptimierende Systeme des Maschinenbaus“. Ferner wird die System- und Softwareentwicklung mit UML und SysML erläutert, die den Kern der erarbeiteten Entwicklungsmethodik bildet. Das Kapitel endet mit der Darstellung des Standards AUTOSAR.

Das vierte Kapitel stellt die entwickelte durchgängige Entwicklungsmethodik für die Steuergeräteentwicklung in Form der AutoMoMe vor. Diese basiert auf der Entwicklung des Steuergerätes durch die Modellierungssprachen SysML und UML. Es werden die verschiedenen Anpassungen der Modellierungssprachen an den Automobilsektor vorgestellt, die für einen durchgängigen Entwicklungsprozess und den darin enthaltenen Methoden notwendig sind. Darin eingeschlossen ist die Einbeziehung einer Analysephase bis zur Modellierung der Architektur einschließlich dessen Verhalten. In diesem Kapitel wird ein (semi-) formaler und durchgängiger Entwicklungsprozess beschrieben, der die vorhandenen Defizite beseitigt. Aufbauend auf dem erarbeiteten Architekturmodell wird im fünften Kapitel ein Übergang zu einer Echtzeitsimulation definiert, damit die Architektur frühzeitig hinsichtlich der zeitlichen Anforderungen und der Ressourcenauslastung überprüft werden kann. Das sich daran anschließende sechste Kapitel befasst sich mit der Erweiterung der Entwicklungsmethodik, um einen Übergang zum AUTOSAR Standard herzustellen. Der Übergang beinhaltet sowohl Struktur- als auch Konfigurationsdaten und sorgt dafür, dass die bereits erarbeiteten Entwicklungsartefakte optimal weiterverwendet werden.

An die Beschreibung des Konzeptes AutoMoMe schließt sich die praxisnahe Erprobung des Ansatzes. Hierzu wird im siebten Kapitel die Umsetzung von AutoMoMe in einen Prototypen erläutert. Hierbei war es vorteilhaft, dass die Dissertation praxisnah und in enger Zusammenarbeit mit einem international tätigen automobilen Zulieferer entstanden ist, so dass die entwickelten Methoden und Techniken bereits in Serienprojekten eingesetzt und erprobt werden konnten. Das achte und letzte Kapitel fasst die erarbeitete durchgängige En-

twicklungsmethodik und die sich daraus ergebenden Vorteile im automobilen Entwicklungsprozess zusammen. Ferner wird die praktische Relevanz und die Umsetzung des Konzeptes betrachtet. Es wird dargelegt, dass die in dieser Dissertation erarbeitete AutoMoMe Methodik bereits in industriellen Serienprojekten mit Erfolg eingesetzt wird. Neben der Zusammenfassung des Ansatzes wird ein Ausblick auf zukünftige Arbeiten gegeben, um das Potential der Arbeit zu veranschaulichen.

Wenn man sagt, dass man einer Sache grundsätzlich zustimmt, so bedeutet es, dass man nicht die geringste Absicht hat, sie in der Praxis durchzuführen.

Otto von Bismarck (1815 - 1898)

2 Grundlagen

In diesem Kapitel werden die Grundlagen der Methoden und Techniken beschrieben, die in der Methodik AutoMoMe verwendet werden. Zusätzlich werden die Eigenschaften der Automobilindustrie und besonders der Entwicklung von automobilen Steuergeräten dargestellt, um die entwickelten Konzepte und Lösungen im Kontext zu sehen.

2.1 Steuergeräte

Definition (Steuergerät). Ein Steuergerät (eng. ECU) wird, wie der Name schon sagt, zum Steuern und Regeln von Sensoren und Aktoren verwendet. Dabei ist es an seinen Einsatzort angepasst und es verfügt über spezielle Schnittstellen, um die Aktorik und Sensorik anzusteuern [ST12]. Es ist somit dem Bereich der eingebetteten Systeme zuzuordnen. [SZ13])

Wie bereits ausgeführt, wird die Software in Automobilen auf Steuergeräten eingesetzt. Neben der Definition eines Steuergerätes (siehe Definition Steuergerät) findet sich ein allgemeiner Aufbau eines automobilen Steuergerätes in Abbildung 2.1 wieder. Hier sind die wesentlichen Bestandteile eines Steuergerätes zu erkennen. In blau ist der wichtigste Teil des Steuergerätes abgebildet. Dies ist der Mikroprozessor. In einem Steuergerät ist es aber durchaus zulässig, dass mehrere Prozessoren untergebracht sind. In lila sind die verschiedenen Speicher des Steuergerätes dargestellt. In einem Steuergerät sind drei Komponenten zu finden, die Dienste (Services) bereitstellen. In der Grafik sind sie in orange wiedergegeben. Den größten Bauraum in einem Steuergerät wird jedoch von den Ein- und Ausgabekomponenten eingenommen. Diese empfangen die Sensordaten und bereiten sie für den Prozessor auf bzw. geben die entsprechenden

Signale an die jeweiligen Aktoren weiter. Dabei sind die Sensorik und Aktorik stets an das Einsatzgebiet des Steuergerätes angepasst [ST12]. In der Grafik sind sie in gelb abgebildet.

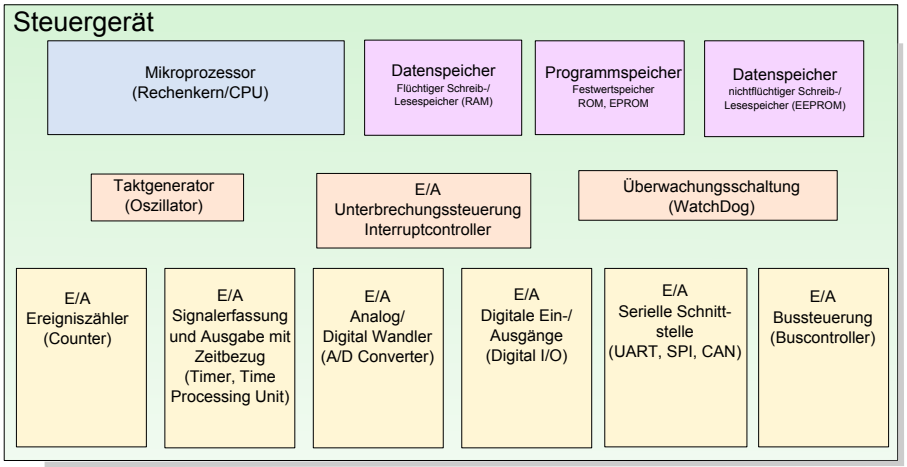


Abb. 2.1. Allgemeiner Aufbau eines Steuergerätes [Rei10a]

2.2 Die Unified Modeling Language (UML)

Die UML ist eine international standardisierte, objektorientierte Softwarebeschreibungssprache und zählt zur Klasse der semi-formalen Notationssprachen. Sie beinhaltet aber keine Entwicklungsmethodik und kein vordefiniertes Vorgehen [Gro11d]. Dies muss jeweils dem Einsatzgebiet und dem dabei genutzten Entwicklungsprozess angepasst und weiterentwickelt werden. Die Sprache kann somit in verschiedenen Entwicklungsprozessen [RQZ07] eingesetzt werden. Es muss jedoch stets eine an die Domäne angepasste Methodik für den Einsatz der UML genutzt bzw. entwickelt werden.

Die UML wird von der Object Management Group (OMG) stetig weiterentwickelt und ist derzeit in der Version 2.4 verfügbar (siehe Definition UML).

Definition (UML). Die UML ist eine grafische Modellierungssprache zur Spezifikation, Visualisierung, Konstruktion und Dokumentation von Softwaresystemen. Sie ist dabei unabhängig von dem Fach- und Realisierungsgebiet [Gro11d].

Ihre Entwicklungsgeschichte begann mit dem Vormarsch der objektorientierten Programmiersprachen in den letzten 20 Jahren. Für diese Form der Programmiersprachen wurde eine geeignete Notationsform gesucht, um die Analyse und das Design zu erstellen. Dabei sollten die Vorzüge der objektorientierten Ansätze wie Kapselung, Vererbung und Abstraktion berücksichtigt werden [Mut05]. Es entstanden eine Vielzahl von unterschiedlichen Notationsformen für die Analyse und für den Entwurf von Software. Anfang der 90er Jahre gab es bis zu 50 unterschiedliche Notationsformen und Sprachen, die sich in den gesetzten Zielen und in der Vorgehensweise mehr oder minder stark unterschieden [Mut05]. Von daher kam es zwischen den Softwareentwicklern immer wieder zu Widersprüchen und Inkompatibilitäten [RQZ07]. Aus diesem Grunde verstärkte die Industrie den Druck, die verschiedenen Techniken und Notationsformen zu vereinheitlichen.

Es wurden sodann verschiedene Sprachen miteinander integriert. Im Jahre 1997 entstand die erste Version der UML. Mit der Integration war jedoch noch kein internationaler Standard geschaffen. Gleichwohl fand die Sprache innerhalb der Industrie regen Zuspruch. Dieser wuchs umso mehr, da eine immer bessere Werkzeugunterstützung verfügbar war. In den darauffolgenden Jahren wurden Ergänzungen geschaffen, wie die Object Constraint Language (OCL) [Gro12] und der XML Metadata Exchange (XMI) [Gro11b], bevor die UML im Jahr 1999 in der Version 1.3 von der OMG übernommen wurde und zu einem internationalen Standard gelangte. Ein weiterer Meilenstein in der Entwicklung der UML war im Jahr 2005 die Verabschiedung der Version 2.0. In dieser Version waren die Erfahrungen der vergangenen Jahre eingeflossen. Darüber hinaus wurde vor allem die Semantik der Sprache präzisiert [RQZ07]. Mittlerweile sind viele Werkzeuge im Einsatz, die den UML Standard unterstützten. Die Bandbreite reicht von Opensource bis hin zu umfangreichen kommerziellen Werkzeugen.

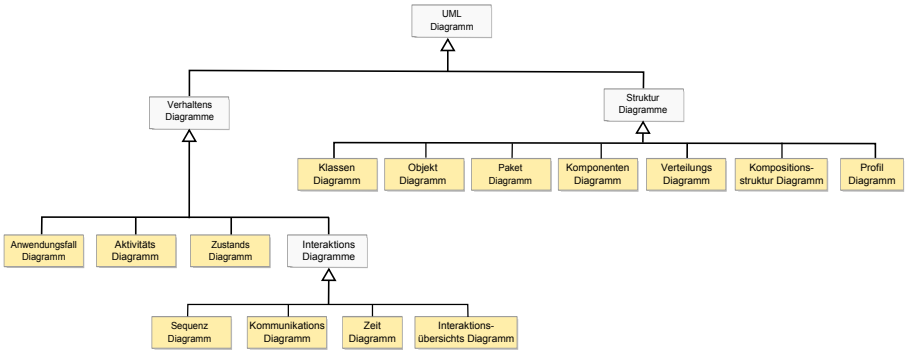


Abb. 2.2. Diagramme der UML in der Version 2.3

Die UML besteht aus 13 Diagrammen (siehe Abbildung 2.2), die in drei unterschiedliche Bereiche aufgeteilt sind. Dies sind Strukturdiagramme, die den Aufbau der Software beschreiben. Die Verhaltensdiagramme werden verwendet, um das dynamische Verhalten der Software zu beschreiben. Eine Untermenge hiervon sind die Interaktionsdiagramme, die die Kommunikation und das Zusammenspiel der Softwarebestandteile definieren.

Mithilfe der UML und ihrer Diagramme kann sowohl eine Softwareanalyse, eine Softwarearchitektur als auch ein Softwaredesign erstellt werden. Somit kann die UML durchgängig von der Analyse der Anforderungen bis hin zum Design eingesetzt werden. Sie ist somit für den gesamten Entwicklungsprozess nutzbar. Ein Vorteil ist die Erweiterungs- und Anpassungsfähigkeit der Sprache an unterschiedliche Domänen. Zu berücksichtigen ist jedoch, dass es keine Entwicklungsmethode gibt; insbesondere keine für die automobile Steuergeräteentwicklung.

2.3 Zusätzliche UML/SysML Profile

Die UML ist eine allgemeine Beschreibungssprache für Software. Logischerweise kann sie nicht alle speziellen Bereiche der Softwareentwicklung, wie die von eingebetteten Systemen, abdecken. Zur Schließung dieser Lücke sind Erweiterungen unumgänglich. Für die UML gibt es zwei Mechanismen, um die Sprache zu erweitern. Der eine ist die Erweiterung des Metamodells. Die andere Alternative ist die Erweiterung der UML in Form von Profilen [BH08]. Der Profilmechanismus ist der gängigere Weg, die UML zu erweitern und an spezielle Bereiche anzupassen. Er wird als leichtgewichtige Lösung bezeichnet [BH08]. Für den Bereich der eingebetteten Systeme sind besonders die nachfolgenden Profile von Bedeutung.

2.3.1 Die Systems Modeling Language (SysML)

Während die UML im Gebiet der Softwareentwicklung sich mittlerweile als Standard durchgesetzt hat, gab es bei der Anwendung von eingebetteten Systemen immer noch Vorbehalte. Vor allem in Bezug auf die Einbeziehung der Disziplinen Elektrotechnik und Mechanik, die bei der Entwicklung von eingebetteten Systemen eine zentrale Rolle spielen. Diese Disziplinen konnten sich mit der auf die Objektorientierung zugeschnittenen Notationsformen nicht identifizieren. Somit war eine disziplinenübergreifende Notationsform für das Design des Gesamtsystems nicht möglich [KSW13, Wei08]. Dies wurde erkannt, und es wurde an eine für die Systemmodellierung angepasste Form der UML gearbeitet. Das Ergebnis ist die SysML. Diese Sprache wurde im Jahr 2007 als offizieller Standard der OMG verabschiedet [Gro10] (siehe Definition SysML).

Definition (SysML). Die SysML ist eine grafische allgemeine Modellierungssprache zur Spezifikation, Analyse, Design und Verifikation von komplexen Gesamtsystemen. Hierunter fallen sowohl Hardware- als auch Softwarebestandteile [Gro10].

Die SysML ist ein Profil der UML. Sie übernimmt die Diagramme, die für die Spezifikation in einem disziplinenübergreifenden Ansatz nutzbar sind. Sie werden durch neue Notationsformen ergänzt. Diagramme, die alleine für die Softwareentwicklung benötigt werden, wie beispielsweise Objektdiagramme, wurden aus der SysML entfernt. Die Vorgehensweise der SysML ist in der Abbildung 2.4 beschrieben.

Da die SysML für die Entwicklung von eingebetteten Systemen und somit für die automobilen Steuergeräte von zentraler Bedeutung ist, werden im Folgenden die Diagramme vorgestellt, die neu in der SysML aufgenommen wurden. Für die aus der UML übernommenen Diagramme sei auf das Kapitel 2.2 hingewiesen.

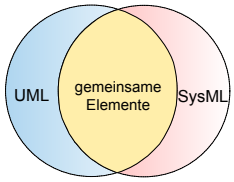


Abb. 2.3. Zusammenhänge zwischen UML und SysML

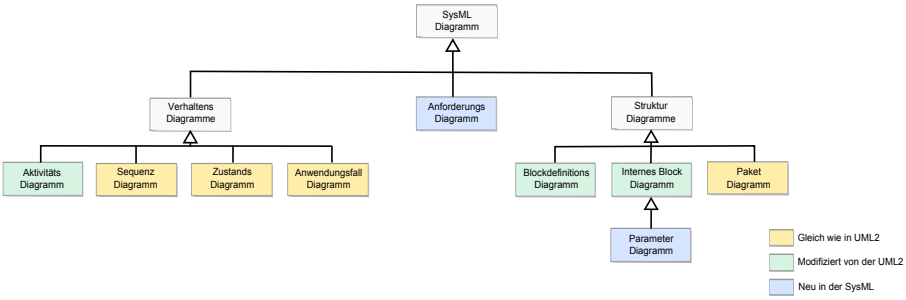


Abb. 2.4. Diagramme der SysML

Neu aufgenommen in der SysML ist das Anforderungsdiagramm. Mit dem Diagramm ist es möglich, Anforderungen grafisch abzubilden und Abhängigkeiten, wie abgeleitete Anforderungen, zu spezifizieren. Das Diagramm ermöglicht die Verbindung von Modell und Anforderung. Jedoch ist die Übersichtlichkeit in dem Diagramm bei der Vielzahl von Anforderungen, wie sie im Automobilbereich anzutreffen sind, nicht gegeben bzw. würde eine Vielzahl von Diagrammen entstehen. Von daher ist das Diagramm nur bedingt für die Anforderungserstellung bei der Entwicklung eines Steuergerätes einsetzbar. Ein Beispiel für

eine sinnvolle Nutzung des Diagramms ist die Darstellung von abgeleiteten Anforderungen.

Ferner sind in der SysML die Strukturmodellierungsdiagramme angepasst. An Stelle der Klassen- und Kompositionsdiagramme treten nunmehr das Blockdefinitions- bzw. das Interne Blockdiagramm. In diesen Diagrammen wird mittels Blöcken oder Parts die Struktur des zu entwickelnden Systems dargestellt. Die Blöcke übernehmen die Rolle der Komponenten in der UML [RQZ07]. Dabei ist der Block nicht einer Domäne zugeordnet. Die Realisierung in Form von Hardware, Software, Mechanik oder Elektrotechnik spielt zunächst keine Rolle. Dies wird erst innerhalb der Systemarchitektur entschieden. Das Zusammenspiel zwischen den Blöcken und Parts wird durch Ports und Flüsse beschrieben. Ein neuer Port in der SysML ist der Flowport. Dieser dient nicht nur zum Austausch von Informationen, wie es bei den Ports in der UML bekannt ist, sondern es können auch Materialflüsse, wie beispielsweise der Stromfluss, spezifiziert werden. Dabei erlaubt der Port zusätzlich die Modellierung von kontinuierlichen Flüssen.

Ferner wurde das Zusicherungsdiagramm (Parametric Diagram) neu in die SysML aufgenommen. Das Zusicherungsdiagramm ist eine Spezialisierung des Internen Blockdiagramms. Es beschreibt die Beziehungen zwischen Eigenschaften von verschiedenen Blöcken [RQZ07]. Das Diagramm wird hauptsächlich zur Modellierung von Formeln und Abhängigkeiten eingesetzt und dient zur Integration von Leistungs- und Zuverlässigkeitsmodellen [Wei08]. So kann die Berechnung von Werten (wie Geschwindigkeit abhängig von einem anderen Sensorwert) nachgebildet werden.

Es kann festgehalten werden, dass die SysML die Bedürfnisse des Systems Engineering erfüllt. Von daher kann sie als Modellierungsnotation bei der Entwicklung von automobilen Steuergeräten eingesetzt werden.

2.3.2 Das MARTE Profil

Das MARTE Profil bietet Erweiterungen an, die es erlauben, eingebettete Echtzeitsysteme auf Basis der UML zu spezifizieren. Das MARTE Profil liegt derzeit in der Version 1.1 vor [Gro11c] und ersetzt das alte und nicht mehr ausreichende Profil *UML for Schedulability, Performance and Time* (SPT) [Gro05]. Das Hauptziel von MARTE ist die Bereitstellung von Notationsformen und Techniken, die eine Modellierung des Schedulings ermöglicht und gleichzeitig eine modellbasierte Analyse innerhalb der Entwicklung unterstützt.

Das MARTE Profil besteht aus zwei Teilen. Zum einen sind dies Modellierungstechniken zur Spezifikation von Zeiten und Ressourcen und zum anderen aus der Bereitstellung von Analysemethoden und -techniken. Von daher ist das MARTE Profil in unterschiedliche Pakete aufgeteilt (vgl. Abbildung 2.5).

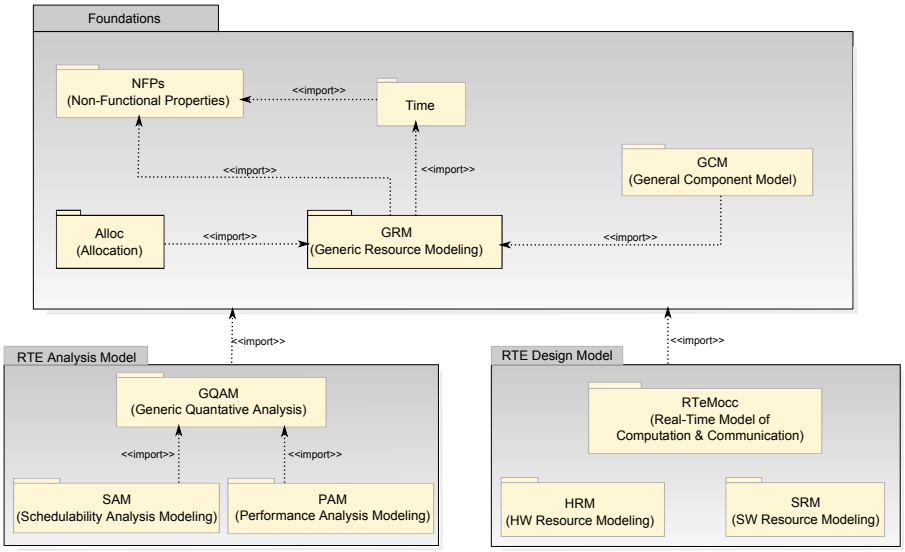


Abb. 2.5. Überblick über die verschiedenen Pakete des MARTE Profils [Gro11c]

Für AutoMoMe ist vor allem das Paket der Modellierungstechniken zur Spezifikation von Zeiten und Ressourcen von Bedeutung, da in diesem Bereich der Entwicklung von Steuergeräten noch Defizite vorhanden sind. Vor allem die Spezifikation von Zeiten ist hierbei gefragt. Das Paket für die Analyse bietet zwar Möglichkeiten, Daten für die Analyse abzulegen, jedoch geschieht dies über Referenzen zu neuen Elementen [GOO02]. Ferner wird im MARTE Profil das Augenmerk auf die Modellierung der physikalischen Elemente, z. B. von Uhren gelegt [AMdS07]. Es existieren erste Ansätze für die Kombination vom MARTE Profil mit Analysewerkzeugen [BMP08, MdS08a, PFS08]. Hierbei werden die Deadlines von Operationen spezifiziert. Dies Vorgehen ist jedoch nicht ausreichend für automobile Steuergeräte, da die Sensorik/Aktorik, sowie die eingesetzten Bussysteme nicht unterstützt werden.

Als Nachteil des MARTE Profils ist zu bewerten, dass alle Informationen durch neue Stereotypen und Eigenschaftswerte spezifiziert werden. Diese werden nur zu einem geringen Teil an bestehende Elemente angefügt. Ein anderer Teil wird durch eigene Elemente realisiert. Die insgesamt notwendigen Stereotypen sind dabei über das gesamte Modell verteilt und werden über Referenzen den Modellelementen zugewiesen [GOO02, DTA+08, ECSG09, EDMG05]. Wegen dieser Zuweisung werden die bestehenden Diagramme durch die zusätzlichen Notationselemente unübersichtlich. Von daher muss eine Möglichkeit eröffnet werden, die Informationen direkt in den vorhandenen Elementen im UML Modell zu speichern. Gleichzeitig muss eine Strukturierung der Diagramme vorgenommen werden.

2.4 CONSENS und MechatronicUML

In AutoMoMe werden Methoden und Techniken für die Modellierung und den Entwurf von Software in der Automobilindustrie entwickelt. Hierzu kann auf einige Vorarbeiten aus der Methodik CONSENS und der Modellierungstechnik MechatronicUML, die im Rahmen des Sonderforschungsbereiches 614 „Selbstoptimierende Systeme des Maschinenbaus“ (SFB 614) [SFB09] für die Modellierung und Entwicklung von Software für mechatronische Systeme entwickelt wurden, zurückgegriffen werden. Diese für die generelle Entwicklung von eingebetteten Systeme entwickelten Methoden und Techniken werden in AutoMoMe aufgegriffen und an die Erfordernisse der Automobilindustrie angepasst.

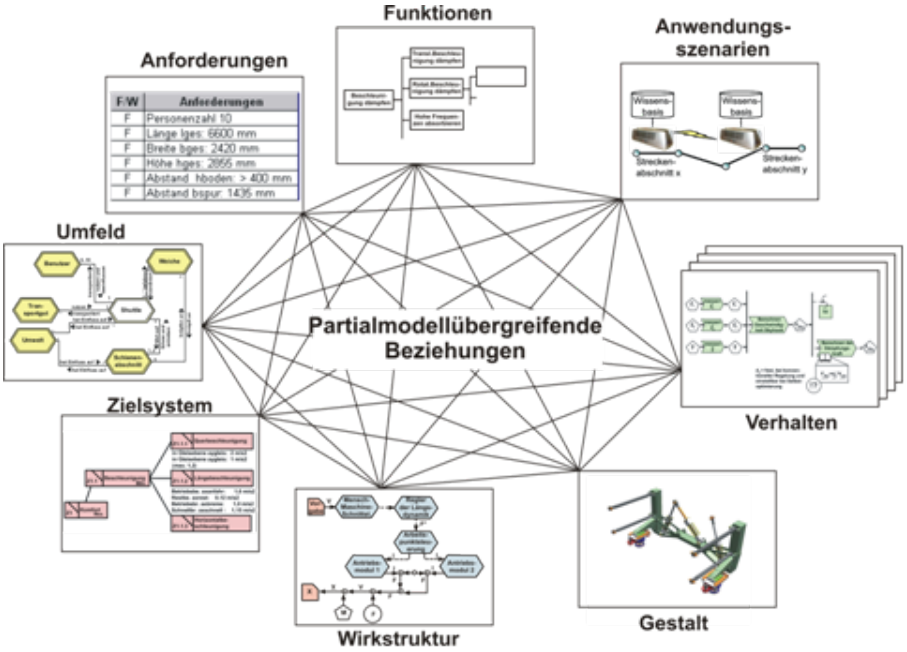


Abb. 2.6. Systemmodellierung mit Partialmodelle im SFB614 [SFB09]

Als Grundlage für die Erstellung eingebetteter Systeme dient im SFB 614 eine durchgängige modellbasierte Entwicklung, wie sie in der VDI-Richtlinie 2206 [dI04] für den Entwurf eines mechatronischen Systems gefordert wird [GRSS11]. Sie besteht aus vielen unterschiedlichen Modellen, den sogenannten Partialmodellen (vgl. Abbildung 2.6). Diese acht Partialmodelle umfassen das gesamte zu entwickelnde System und zwar von der Anforderungsbeschreibung bis hin zur Implementierung. Ein jedes Partialmodell bildet dabei eine ganz bestimmte Sicht auf das System ab. Dabei ergänzen sich die verschiedenen Sichten, so dass sämtliche Gesichtspunkte des Systems spezifiziert werden

[GSA+11]. Im Folgenden werden die einzelnen Sichtweisen kurz vorgestellt [GSA+11]:

- **Umfeld:** In diesem Partialmodell wird der Kontext des Systems beschrieben. Es werden alle (Fremd-)Systeme, die mit dem zu entwickelnden System in Kontakt treten, dargestellt. Hierbei werden die Signale und Einflüsse modelliert, die auf das System einwirken und zu einer Kommunikation führen.
- **Anforderungen:** In dieser Sicht werden die Anforderungen an das System zusammengefasst. Die Anforderungen liegen dabei in textueller Form vor, werden jedoch durch quantifizierbare Anforderungen mit Attributen unterstützt.
- **Anwendungsszenarien:** In dem Partialmodell Anwendungsszenarien werden verschiedene Szenarien beschrieben, die eine situationsspezifische Sicht auf das Gesamtsystem geben. Die wichtigsten Verhaltenseigenschaften des Systems werden spezifiziert und in einer formalen Weise verständlich gemacht.
- **Funktionen:** Innerhalb der Funktionsbeschreibung des Systems werden die verschiedenen Funktionalitäten definiert, die das System zu erfüllen hat. Hierbei ist eine Hierarchisierung der Funktionen möglich und erwünscht. Eine Funktion wird soweit verfeinert, bis eine konkrete Lösung gefunden werden kann. Dies kann beispielsweise ein bestehendes Produkt wie eine Busanbindung sein.
- **Wirkstruktur:** In der Wirkstruktur wird die prinzipielle Wirkungsweise des Systems abgebildet. Hierzu werden die Flüsse zwischen den Systemelementen in Form von Stoff-, Energie- und Informationsflüssen spezifiziert. Außerdem ist Zielvorgabe die Festlegung, welche Elemente in welcher Domäne realisiert werden. Diese werden dann im domänenspezifischen Entwurf weiterentwickelt, beispielsweise in der Softwarearchitektur. Die Wirkstruktur entspricht somit größtenteils einer Systemarchitektur.
- **Verhalten:** Das Verhalten hat bei einem eingebetteten System immer eine besondere Bedeutung. In diesem Partialmodell wird das Verhalten des Systems definiert. Dies erfolgt durch Verhaltensdiagramme, die Aktivitäten, Zustände und deren Übergänge spezifizieren.
- **Gestalt:** Bei eingebetteten Systemen sind sehr oft Einschränkungen bzw. Vorgaben für die Gestalt des Systems zu treffen. Dadurch kann es zu weiterführenden Anforderungen an einzelne Teilsysteme oder auch Domänen kommen. So können z. B. Anforderungen durch die Gestalt oder den Einbauort entstehen, die das System erfüllen muss. Die Anforderungen können beispielsweise die Temperaturverträglichkeit von Hardwarebestandteilen betreffen. Die Modellierung der Gestalt des Systems erfolgt hauptsächlich mit gängigen 3D Systemen.

Neben der domänenübergreifenden Systembeschreibung in den unterschiedlichen Partialmodellen beschäftigt sich der SFB 614 ebenfalls mit der domänenspezifischen Modellierung. Hierzu kommt bei der Softwaretechnik die MechatronicUML zur Anwendung, um die diskreten und kontinuierlichen Softwareanteile zu beschreiben [GSA+11, GH06]. Die MechatronicUML basiert dabei auf der etablierten Softwaremodellierungssprache UML (vgl. Kapitel 2.2). Die UML wurde hierzu angepasst und ergänzt, um Zeiten und die Umschaltung von Reglern zu spezifizieren. Vor allem die Modellierung der Zeiten ist für die Entwicklung eines eingebetteten Systems von Bedeutung.

2.4.1 Real-Time Statecharts (RTSCs)

Für die Modellierung der Struktur der Software werden die aus der UML bereits bekannten Komponenten- und Klassendiagramme genutzt. Das Verhalten wird in Form von erweiterten Zustandsdiagrammen, den sogenannten Real-Time Statecharts (Real-Time Statecharts (RTSC)) spezifiziert. Die RTSCs wurden um Zeitbedingungen wie Deadlines, Worst Case Execution Time (WCET)s und Perioden erweitert, um das Verhalten von eingebetteten Systemen spezifizieren zu können [GSA+11].

Die RTSCs definieren zeitbehaftete UML Zustandsdiagramme auf der Basis von Timed Automata. Im Gegensatz zu diesen beinhalten die Real-Time Statecharts eine weitere Abstraktion und kombinieren somit die High-Level-Konstrukte der Statecharts mit der Semantik der Timed Automata. Dies erlaubt einen besseren Überblick über das Verhalten [GB03]. Sie sind formal definiert und eignen sich deshalb, die gestellten Anforderungen zu erfüllen.

Die Funktionsweise der RTSCs soll am Beispiel des Startup-Verhaltens des Komfortsteuergerätes (vgl. Abbildung 2.7) verdeutlicht werden. Dabei werden anhand des Beispiels die verschiedenen Modellierungsmöglichkeiten veranschaulicht, die mit einem Real-Time Statechart eingesetzt werden können. Der Zustand *Initialisierung* besitzt die Zeitinvariante $t_0 \leq 5ms$. Bei Eintritt in den Zustand (entry-Methode) wird die Uhr t_0 auf 0 zurückgesetzt und die Operation *InitComfort()* mit der WCET w = 5 ausgeführt. Der Übergang zum Zustand *Aktiv* erfolgt dabei automatisch, nachdem die Operation InitComfort beendet wurde. Dabei muss die Zeitinvariante $t_0 \leq 5ms$ eingehalten werden. Gleichzeitig wird die Uhr t_0 wieder zurückgesetzt.

Während des Aufenthaltes in *Aktiv* wird die Methode *computeSignals()* mit einer Periode, die zwischen 2ms und 3ms liegt, aufgerufen. Die dargestellte Transition schaltet, wenn das Ereignis *sleep* anliegt und der Zeitguard $3ms \leq t_1$ wahr ist. Beim Verlassen des Zustandes *Schlafmodus* wird die Uhr t_1 wieder zurückgesetzt. Der Zustand wird dann verlassen, wenn das Ereignis *activate* vorliegt. Dabei wird als Seiteneffekt die Operation *activateComponents()* ausgeführt.

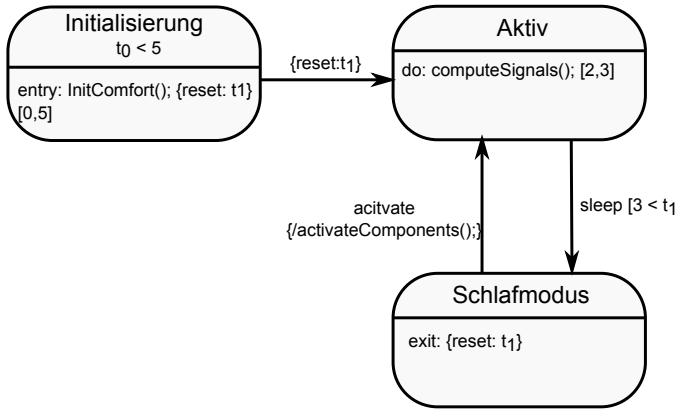


Abb. 2.7. Real-Time Statechart zur Beschreibung des Startup-Verhaltens

Aufbauend auf den RTSC werden noch Koordinationsprotokolle definiert, die die Kommunikation zwischen den einzelnen Komponenten spezifizieren. Die RTSCs und Koordinationsprotokolle können hinsichtlich verschiedener Sicherheitseigenschaften verifiziert werden [GTB+03]. Die Semantik der Real-Time Statechart ist durch die Semantik von Extended Hierarchical Timed Automata (ExHTA) [GB03] definiert. Die ExHTA ist eine Erweiterung der hierarchischen Timed Automata. Daher ist es machbar, die Real-Time Statecharts auf Modelle der Timed Automata abzubilden [BGS05]. Der sich hierdurch ergebende Vorteil besteht darin, dass Timed Automata als Eingabe für den Model Checker Uppaal dient. Somit ist eine Analyse der zeitlichen Informationen gesichert. Der Model Checker erhält als Eingabe die in Timed Automata abgebildeten Real-Time Statecharts und die zu erfüllenden Sicherheitsanforderungen (siehe hierzu auch Abbildung 2.8). Dies kann beispielsweise die Deadlock Freiheit sein. Mit diesen Informationen kann sodann eine Überprüfung erfolgen. Somit besteht die Chance, ein Analyseverfahren in Form einer formalen Verifikation an die erweiterten SysML Modelle anzubinden [SW07].

Die Abbildung 2.8 veranschaulicht die Vorgehensweise zur Anbindung des Model Checkers Uppaal zur Überprüfung der Real-Time Statecharts noch einmal grafisch. Der Modelchecker erhält als Eingabe sowohl das Real-Time Statechart als auch die zu überprüfenden Anforderungen. Dabei muss das RTSC zunächst in ein Modell der Timed Automata überführt werden. Dies dient anschließend als Eingabe für Uppaal. Falls das Modell einen Fehlerfall aufweist, beispielsweise ein zeitliches Fehlverhalten, erzeugt der Model Checker ein Gegenbeispiel. In diesem ist die Abfolge der Zustände beschrieben, die zu dem Fehler führten. Falls kein Fehler vorliegt, gibt der Model Checker ein Ok zurück.

Die Real-Time Statecharts können aber nicht nur zur Modellierung von diskreten ereignisgesteuerten Systemen verwendet werden, sondern ebenso für

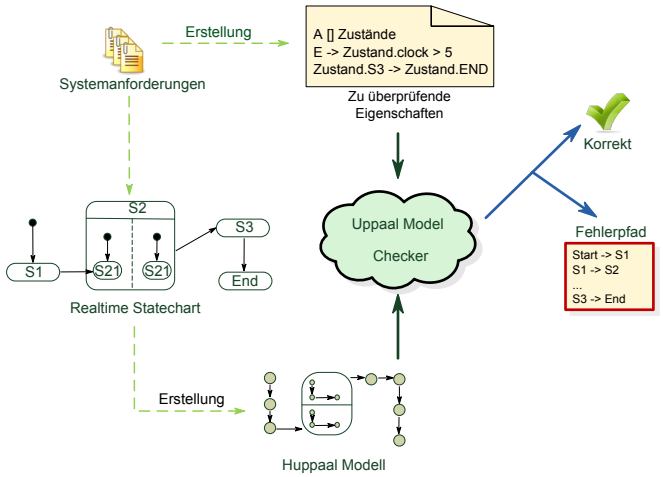


Abb. 2.8. Anbindung an den Uppaal Model Checker nach [GB03]

die Modellierung von hybriden Systemen. Die Modellierung erfolgt dabei in sogenannten hybriden Synchronisationsstatecharts [GSA+11]. Ein hybrides System besteht dabei sowohl aus diskreten Modell- als auch aus kontinuierlichen Modellelementen. Diese können beispielsweise Regler sein. Die Regler werden dabei in Form von Komponenten mit kontinuierlichen Ports modelliert. Die Modellierung erfolgt innerhalb eines Zustandes. Dies bedeutet, falls der Zustand aktiv ist, wird der entsprechende Regler genutzt. Es erfolgt somit eine Reglerumschaltung. Mittels der hybriden Modellierung ist eine vollständige Verhaltensbeschreibung möglich [SW07].

Die Methoden und Techniken aus dem SFB 614 können für die Modellierung und Spezifikation der Software für eingebettete Systeme genutzt werden. Jedoch sind einige Techniken, wie die Verwendung der Wirkstruktur, nur prototypisch umgesetzt. Ebenso fehlt die Betriebssystemmodellierung und der Übergang nach AUTOSAR. Von daher ist eine Anpassung der Technik an die Bedürfnisse eines Zulieferers in der Automobilbranche erforderlich. Diese Anpassungen und Erweiterungen werden in AutoMoMe erarbeitet und umgesetzt.

2.5 Echtzeitanalyse

Es sind verschiedene Ansätze vorhanden, um das vollständige Systemverhalten und besonders das zeitliche Verhalten zu analysieren und zu bewerten (vgl. Abbildung 2.9). Ein manuelles Ausrechnen der Performance ist auf Grund der Komplexität bei den heutigen eingebetteten Systemen nur noch eingeschränkt

möglich. Deswegen sind weitergehende Methoden und Techniken anzustreben. Die Ansätze sind in zwei Bereiche zu unterteilen. Der erste Komplex umfasst die experimentellen Verfahren. Bei diesen Verfahren werden durch Messungen bzw. durch Simulationsmodelle die Zeiten ermittelt oder simuliert. Sie können daraufhin analysiert werden. Der zweite Bereich sind die analytischen Verfahren. Hierbei wird das zeitliche Verhalten anhand von mathematischen Modellen berechnet [Tra10].

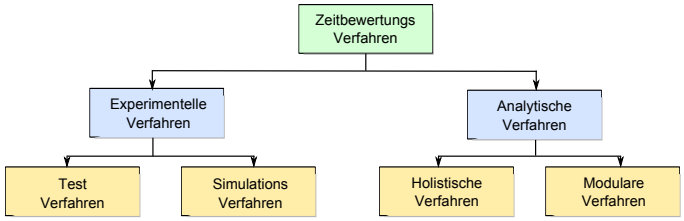


Abb. 2.9. Unterschiedliche Zeitbewertungsverfahren nach [Tra10]

Beide Techniken zur Performanceanalyse beinhalten sowohl Vor- als auch Nachteile. Diese werden in diesem Abschnitt herausgearbeitet, um eine Grundlage zur Beurteilung der beiden Ansätzen zu schaffen. Die analytischen Werkzeuge können dabei wie folgt charakterisiert werden. Für Analysezwecke kommen die schlimmsten anzunehmenden Antwortzeiten Worst Case Response Time (WCRT) zum Einsatz, um Aussagen über die Architektur zu erhalten [KPK+10]. Es kann nicht ausgeschlossen werden, dass unrealistische bzw. kaum sich stellende Fälle auftreten können, auf Grund dessen eine Architektur verworfen werden muss. Dies ist nur dann gegeben, wenn mehrere Ereignisse eintreffen, obwohl dies real gar nicht vorkommen kann. Dieser theoretische Fall zeigt, dass in manchen Situationen eine gewisse Überapproximation bei dem Einsatz von analytischen Werkzeugen durchaus eintreten kann. Ferner gilt es zu berücksichtigen, dass die Laufzeit der analytischen Werkzeuge mit der Größe der Architektur deutlich zunimmt, da der mögliche Zustandsraum mit jeder weiteren Komponente wächst. Es gibt mehrere Werkzeuge, die einen analytischen Ansatz verfolgen [KPK+10]. Das im Automobilbereich bekannteste Werkzeug ist das Tool SymTA/S [Sym14]. Dies erlaubt die Analyse einer Architektur. Hierbei werden bereits sehr detaillierte Informationen verlangt.

Eine andere Art von Werkzeugen, die eine Analyse des zeitlichen Verhaltens und der zu erwartenden Ressourcennutzung zulässt, sind die experimentellen Werkzeuge. Diese werden in dieser Arbeit in Form der Simulationswerkzeuge betrachtet. Diese Werkzeuge simulieren anhand der vorgegebenen Daten das zu erwartende Verhalten des Systems. Durch die Simulation werden die Abläufe und das Zusammenspiel der verschiedenen Systemkomponenten betrachtet [KPK+10]. Hierdurch erkennt der Entwickler wie das Gesamtsystem funktioniert und kann dadurch ein mögliches Fehlverhalten identifizieren. Dies ist

nicht nur bei der Verletzung von Zeitanforderungen zu sehen. Darüber hinaus ist es auch möglich, eine Mehrfachnutzung von Daten und zusätzliches Optimierungspotential in der Architektur zu erkennen. Als Beispiel kann angeführt werden, dass eine bestimmte Funktion immer einen sehr langen Zeitraum warten muss, obwohl sie abgearbeitet werden kann. Dies Verhalten entspricht dem Performance Anti-Pattern *Unbalanced Processing* [SW00] und kann durch geeignete Umstrukturierungsmaßnahmen behoben werden. Die Identifikation des Anti-Patterns erfolgt anhand der mitprotokollierten Ergebnisse der Simulation.

Der Unterschied zwischen den beiden Performanceanalysetechniken lässt sich an einem Beispiel, dargestellt in der Abbildung 2.10, verdeutlichen. In dieser sind die zeitlichen Werte für eine Funktion des Systems wiedergegeben. Die beste Best Case Response Time (BCRT) und die schlechteste Antwortzeit WCRT werden in den Analysewerkzeugen verwendet. Die sich aus der Simulation ergebenden Daten befinden sich zwischen diesen beiden Zeiten und sind in der Abbildung in blau gekennzeichnet. Es ist zu erkennen, dass es einen deutlichen Abstand zwischen der besten und schlechtesten Zeit und den in der Simulation tatsächlich auftretenden Zeiten gibt. Diese Zeitspannen zeigen die mögliche Überapproximation an.

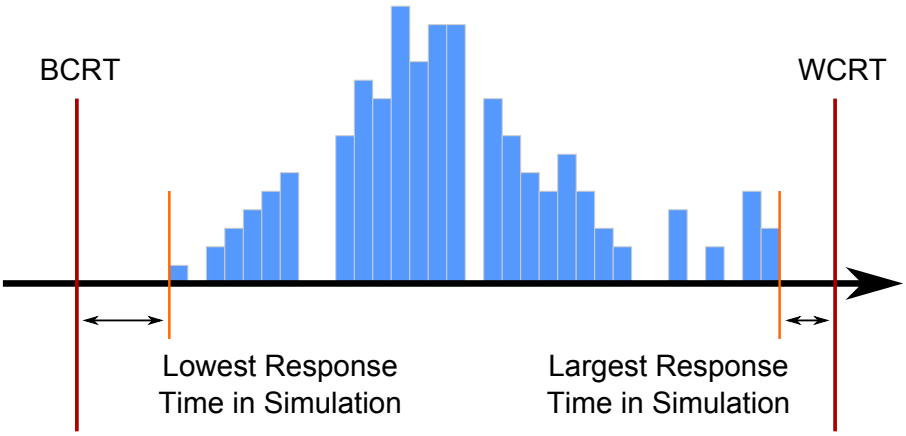


Abb. 2.10. Unterschied zwischen einer Analyse und einer Simulation [Inc14]

Um eine möglichst exakte Vorstellung von dem Zusammenspiel der unterschiedlichen Systemfunktionen zu erhalten und gleichzeitig genauere Aussagen über die Performance des Systems treffen zu können, ist eine Kombination der vorhin vorgestellten Techniken sinnvoll. Die Werkzeuge chronVAL und chronSIM der Inchron Toolsuite [Inc14] erlauben sowohl eine formale Analyse als auch die Durchführung einer Simulation. Die beiden Werkzeuge bieten den Vorteil, dass die Analyse als auch die Simulation auf ein und demselben Sim-

ulationsmodell ausgeführt werden kann. Es reicht somit aus, nur ein Simulationsmodell zu erstellen, um ein besseres Verständnis für das System zu erlangen. Anschließend besteht die Möglichkeit, die Simulationsergebnisse in einer formalen Analyse abzusichern. In der Systemsimulation wird in den frühen Phasen der Entwicklung die Systemarchitektur simuliert, um Entscheidungen und Ergebnisse frühzeitig zu gewinnen. Bei einem eingebetteten System betrifft dies vor allem das Zusammenspiel zwischen Hard- und Software. Es gilt zu überprüfen, inwieweit die zeitlichen und performancetechnischen Anforderungen durch die Architektur abgedeckt sind. Die Simulation ist dabei so früh wie möglich vorzunehmen. Denn je früher ein Fehler gefunden wird, desto geringer ist der Aufwand ihn zu beheben [Bür12]. Als Nebeneffekt werden dadurch auch die Kosten minimiert.

2.6 Automobilspezifische Eigenschaften

In den folgenden Abschnitten werden die besonderen Charakteristika von automobilen Steuergeräten vorgestellt. Die Erläuterungen dienen dazu, die späteren Lösungen verständlicher zu machen und die besonderen Eigenschaften im Gegensatz zu anderen Domänen herauszustellen. Es wird ebenso erarbeitet, warum die einzelnen Problemstellungen existent sind. Deswegen werden besonders die Bereiche fokussiert, die in dem Konzept AutoMoMe behandelt werden. Dies sind die Kommunikationsbusse, die die Kommunikation zwischen den einzelnen Steuergeräten definieren, die Gesamtarchitektur eines Steuergerätes, das globale Verhalten inklusive der Echtzeit und ein automobiles Betriebssystem.

2.6.1 Vergleich mit anderen Domänen

Eine vergleichbare Entwicklung findet ebenso in anderen Domänen statt. Damit einhergehend sind ähnliche Probleme – aber auch Lösungen – in anderen Domänen zu finden. Insbesondere bei der Luft- und Raumfahrtindustrie sind solche Gemeinsamkeiten anzutreffen [MDD+10]. Es wird analysiert, ob in diesen Domänen bereits Lösungen vorhanden sind, die für die Steuergeräteentwicklung übernommen werden können. Zum Schutz von Leben werden in beiden Branchen hohe Anforderungen an Zuverlässigkeit und Verfügbarkeit [SZ13, MDD+10] gepaart mit hohen Sicherheitsanforderungen an die jeweiligen Systeme gestellt. Während im Avionikbereich die Sicherheit oftmals bereits durch autonome Reaktionen realisiert ist [MDD+10], steht beim Automobil noch größtenteils die Verlässlichkeit des Fahrers und dessen Reaktionen im Vordergrund.

Gemeinsam ist beiden Domänen, dass die Systeme extremen und wechselnden Umgebungsbedingungen unterliegen. Dies sind sowohl schwankende Temperaturen als auch Feuchtigkeit und Erschütterungen [SZ13]. Diese Vorgaben müssen bei der Steuergeräteentwicklung berücksichtigt werden. So muss nicht nur die Hardware dementsprechend konzipiert, sondern gleichfalls muss die Software auf diese ausgelegt sein. Exemplarisch wird auf die redundante Speicherung und Überprüfung von Signalwerten verwiesen, die eine Störung durch elektromagnetische Impulse nahezu ausschließt.

Die Entwicklung eines Automobils und dessen Steuergeräte erfolgt über mehrere Jahre. Nach Abschluss der jeweiligen Entwicklungsphasen und der Auslieferung ist ein Automobil mehrere Jahre auf den Straßen unterwegs. Ähnlich verhält es sich bei einem Flugzeug. Der Produktlebenszyklus der Systeme ist in beiden Branchen somit auf mehrere Jahre auszulegen (vgl. Abbildung 2.11) [ST12]. Von daher sind spätere Aktualisierungen der Software bei der Entwicklung mit einzuplanen. Ebenso muss die Spezifikation wartbar sein. Es ergeben sich daher Anforderungen an eine zukünftige Entwicklungsumgebung.

Abb. 2.11. Produktlebenszyklus eines Fahrzeugs nach [SZ13]

Gleichwohl sind in der Domäne des Automobilbaus einige charakteristische Eigenschaften zu beachten, die diese Branche von den anderen grundlegend unterscheidet. Es ist daher nicht ohne weiteres möglich, Techniken und Erfahrungen aus den anderen Branchen eins zu eins zu übernehmen. Vor allem das Zusammenspiel und die Abhängigkeit zwischen Automobilherstellern und Zulieferern hat einen besonderen Stellenwert und ist in anderen Domänen nicht so ausgeprägt. Während in der Luftfahrtindustrie die Zulieferer eng mit den Flugzeugherstellern verzahnt sind, ist in der Automobilindustrie nur eine lose Verknüpfung zwischen Zulieferer und OEM vorherrschend [Bec06]. In der Automobilindustrie ist es üblich, dass ein Zulieferer mehrere OEMs gleichzeitig beliefert und der OEM von unterschiedlichen Zulieferern Steuergeräte bezieht. Diese Konstellation kann speziell bei der Integration der Steuergeräte in einem Fahrzeug von Nachteil sein, da die einzelnen Steuergeräte von verschiedenen Hersteller kommen und nur durch die Anforderungen passend zueinander sind. Durch die bestehenden Auslagerungen ist der Zuliefererenteil in der Automobilherstellung extrem hoch. Der Anteil beträgt inzwischen bis zu 65% [WFO09], so dass die Erfahrung und das Know-How zur Erstellung der jeweiligen Komponenten zum größten Teil nicht beim Hersteller, sondern bei den Zulieferern liegt. Von daher übernehmen die Zulieferer bei der Entwicklung eines Automobils einen wesentlichen Part.

Im Verhältnis zu einem Flugzeug ist das Automobil aber ein Massenprodukt. Mit Ausnahme von Sondermodellen werden von jedem Fahrzeugtyp ho-

he Stückzahlen produziert. Um dauerhaft konkurrenzfähig zu sein und den Fahrzeugtyp aussichtsreich und gewinnbringend am Markt platzieren zu können, ist es für den Automobilhersteller essenziell, den Kostenaufwand für alle Komponenten auf ein vertretbares Minimum zu reduzieren. Von daher kommen möglichst kostengünstige und ressourcensparende Hardwarebauteile zum Einsatz. Bei der Kostenberechnung sollten jedoch vorrangig nicht nur die Stückkosten ausschlaggebend sein. Vielmehr stellen auch die nicht unerheblichen Entwicklungskosten einen nicht zu unterschätzenden Kostenfaktor dar [BS05]. Im Vergleich zu anderen Branchen tritt der Automobilhersteller nicht nur als Auftraggeber auf, sondern liefert ebenso Entwicklungsartefakte zu. Solche Zulieferungen können sehr unterschiedlich sein und reichen von Schnittstellenbeschreibungen über spezifizierte Modelle bis hin zu fertigen Softwarekomponenten, die durch die Zulieferer zu integrieren sind.

2.6.2 Kommunikationsbusse

Für neuartige Innovationen ist die Zusammenarbeit der Steuergeräte und der Funktionen eine zwingende Voraussetzung. Denn nur so können die notwendigen Daten für die jeweilige Funktion bereitgestellt werden. Ferner erlaubt die Vernetzung der Steuergeräte die Verteilung der Funktionen auf unterschiedliche Steuergeräte. Hierdurch besteht die Möglichkeit, mehrere Funktionen auf leistungsfähigen Steuergeräten unterzubringen und somit die Anzahl der Steuergeräte im Automobil zu verringern [AUT10b]. Trotzdem wird in Zukunft die Zahl der Kommunikationspartner in Form der Funktionen innerhalb der internen Fahrzeugkommunikation weiter steigen. Von daher muss immer die Alternative für einen weiteren Ausbau der Kommunikationsteilnehmer gegeben sein, um neue Steuergeräte unterzubringen. Dabei gilt es, die Skalierbarkeit zu gewährleisteten. Um dieser Vorgabe nachzukommen, sind die OEMs schnell zu dem Entschluss gelangt, die Kommunikationsverbindungen auf ein gemeinsames Kommunikationsmedium abzubilden [SZ13]. Dies war die Geburtsstunde der Busse.

Definition (Kommunikationsbus). Unter einem Kommunikationsbus – kurz Bus genannt – versteht man ein gemeinsames Kommunikationsmedium, das unterschiedlichste Sensoren und Aktoren über wenige Leitungen miteinander verbindet. Dabei werden digitale Signale ausgetauscht, die in Steuerungs- und Nutzdaten unterteilt sind [ZS08].

Der Einsatz der Busse erfolgte bedarfsorientiert ohne eine grundlegende und systematische Planung. Eine Gesamtarchitektur wurde folglich nicht entwickelt, in der sich die Steuergeräte und Busse einbringen konnten [ZS08]. Vielmehr wurden die unterschiedlichen Busse nach und nach im Fahrzeug integriert.

Dabei waren oftmals ökonomische Gründe ausschlaggebend, so dass vielfach ein kostengünstiger Bus neu entwickelt und eingebaut wurde. Eine komplizierte und von verschiedenen Bussen geprägte heterogene Architektur ist somit die Folge (vgl. Abbildung 2.12). Dies ist auch der Grund, dass schnell die Einsicht reifte, dass standardisierte Busse, sogenannte Bussysteme, vorteilhafter sind [ZS08]. Die Busse können dabei in unterschiedlichen Netzwerk-Topologien vorkommen. Es gibt die Unterscheidung in sogenannte Ring-, Stern- und Linien-Topologien [ZS08]. Die Auswahl der Netzwerk-Topologie erfolgt je nach Anwendungsgebiet. So wird beispielsweise im Infotainmentbereich die Ring-Topologie eingesetzt.

Für die zukünftige Entwicklung der Elektronik im Fahrzeug ist die Verwendung der Bussysteme vorteilhaft. Funktionen können so räumlich verteilt werden. Ein Vorteil, der besonders bei der Komfortelektronik von Nutzen ist. Ein zentrales Komfortsteuergerät steuert und überwacht das Innenlicht, die Schließ- und ggf. auch noch die Alarmanlage (vgl. Kapitel 4.2). Eine solche Steuerung kann nur deshalb gewählt werden, weil die Sensoren und die kleineren Steuergeräte, z. B. für die Ansteuerung des Innenlichts, über einen Bus mit dem zentralen Steuergerät verbunden sind. Diese Architektur ermöglicht eine einfache Erweiterung und somit eine gute Skalierbarkeit. Insbesondere bei der heutzutage vorherrschenden großen Variantenvielzahl mit diversen optionalen Sonderwünschen, die ein Hinzunehmen bzw. ein Weglassen von Steuergeräten vorgeben, ist dies unverzichtbar. Der Aufbau und die Vernetzung der Steuergeräte ist immer identisch. Bei zusätzlichen Steuergeräten werden diese an den Bus angeschlossen. Durch eine dementsprechende Parametrisierung wird das Steuergerät im Bus erkannt. Eine Änderung der Netzwerktopologie kann so vermieden werden.

Eine weitere positive Eigenschaft der standardisierten Busse besteht darin, dass Funktionen und Steuergeräte redundant und verteilt ausgelegt werden können. Derzeit wird diese Möglichkeit aus Kostengründen oftmals nicht eingesetzt. Für zukünftige Innovationen, wie beispielsweise die X-by-Wire Technologien [NHB05], ist eine redundante Auslegung auf Grund der sicherheitskritischen Einstufung der Systeme jedoch unumgänglich. So fordert die funktionale Sicherheit eine redundante Auslegung, damit das System jederzeit betriebsbereit ist [fS10b, Sto96]. Die Nutzung von Bussen für die Kommunikation legt somit den Grundstein für die zukünftige Entwicklung.

Es kommen verschiedene Bus-Standards zur Anwendung. Sie weisen unterschiedliche Charakteristiken auf und werden dementsprechend eingesetzt. Im Folgenden werden die einzelnen Bussysteme und ihre Eigenschaften wiedergegeben, um die spätere Modellierung der Kommunikation verständlicher darzustellen.

In einem Automobil werden die folgenden Bussysteme unterschieden:

- **Controller Area Network (CAN)**
  Der CAN Bus und das dazugehörige Protokoll wurde Ende der 80er Jahre von der Firma Bosch entwickelt. Es wird seit 1991 als Klasse C Bus (Bussystemklassifizierung der SAE) in Fahrzeugen eingesetzt [ZS08, NHB05]. Der CAN Bus hat sich durch seine kostengünstige und kompakte Bauweise in der Automobilindustrie und teilweise ebenso in der Automatisierungsdomäne etabliert, so dass er zwischenzeitlich international standardisiert wurde. Er ist in den Spezifikationen ISO 11898 und SAE J2284 und J1939 definiert.

  Der CAN Bus ist ein prioritätsgesteuerter bitstrom-orientierter Bus, der mittels der *Carriere Sense Multiple Access* (CSMA) den Buszugriff prüft, um Kollisionen bei der Buskommunikation auszuschließen [NHB05]. Dabei ist der Bus ein Broadcast System. Dies bedeutet, dass die Nachricht nicht nur bis zum Empfänger weitergeleitet, sondern dass sie vielmehr an alle Teilnehmer, die an dem CAN Bus angeschlossen sind, versandt wird. Jede Nachricht erhält eine eindeutige ID, den sogenannten *Message Identifier*. Jedes Steuergerät empfängt die Nachricht und kann anhand der Identifizierung überprüfen, ob sie gelesen und weiterverarbeitet werden muss oder ob sie ignoriert werden kann [ZS08].

  Dabei übernimmt der CAN-Controller automatisch das Schreiben und Lesen auf dem Bus. Gesteuert wird er über Register von der darüber liegenden Software [ZS08]. Dies bedeutet, dass die Identifizierungsnummer (Message Identifier), die Anzahl der Datenbytes und schließlich die Nachricht selber von der Software in die Speicher des CAN-Controllers geschrieben werden. Dieser versendet sie dann bei der nächsten zyklischen Aktivierung des Controllers durch das Betriebssystem und bei einem freien CAN-Bus. Der Empfang der Daten erfolgt entsprechend. Hierzu schreibt der CAN Controller die Daten in den Speicher und benachrichtigt den entsprechenden Mikrocontroller durch einen Statusbit oder einen Interrupt [ZS08].

  Mittlerweile sind drei verschiedene Arten des CAN-Busses im Einsatz. Dies sind der sogenannte Low und High Speed CAN sowie der Time-Triggered CAN (TTCAN) Bus [ZS08]. Während der Low und der High Speed CAN Bus sich nur durch ihre unterschiedlichen Datenraten unterscheiden, garantiert der Time-Triggered CAN zusätzlich ein deterministisches Übertragungsverfahren. Hierzu sind verschiedene Zeitfenster definiert, in denen die jeweiligen Steuergeräte senden dürfen. Diese CAN Bus Version läßt nunmehr eine genaue Vorhersage, wann eine Nachricht gesendet wird, zu. Der anfängliche Nachteil der CAN Busse, nämlich die nicht deterministische Kommunikationsübertragung, konnte so behoben werden. Der CAN Bus hat sich wegen seiner zahlreichen Vorteile durchgesetzt und ist mittlerweile in jedem Fahrzeug vorzufinden [ZS08].

- **Local Interconnect Network (LIN)**
  Der LIN Bus wurde Ende der 90er Jahre vom LIN Konsortium als

kostengünstige Alternative zum Low-Speed-CAN Bus für einfache Sensor-Aktor-Anwendungen (SAE Klasse A), wie sie beispielsweise in der Tür- oder Schiebedachelektronik zu finden ist, entwickelt [GvdW05]. Wie beim CAN Bus ist der LIN Bus verbindungslos und arbeitet über inhaltsbezogene Addressierung (LIN Identifier) [GvdW05]. Dabei werden die Botschaften streng zeitsynchron gesendet, so dass im Gegensatz zum Low oder High Speed CAN Bus ein streng deterministisches Verhalten vorliegt [GvdW05].

Die Reihenfolge und Wiederholperiode der einzelnen Botschaften werden während der Entwicklungsphase des Netzes festgelegt. Damit die Nachrichten von allen Steuergeräten erkannt und ggf. beantwortet werden können, ist eine zeitliche Synchronisation erforderlich. Diese erfolgt durch das sogenannte Mastersteuergerät. Das Mastersteuergerät ist oftmals ein Steuergerät, das an den CAN-Bus angebunden ist [GvdW05]. Es gibt die Zeitbasis für den LIN Bus vor. Als weitere Aufgabe übernimmt es die Funktion eines Gateways zum CAN-Bus, d.h. in dem Steuergerät werden die Botschaften in das jeweilige andere Busformat übertragen und entsprechend versandt. Der LIN Bus wird vielfach mit einem anderen Bussystem kombiniert.

- **FlexRay**
  Der FlexRay Bus wurde für zukünftige X-by-Wire Anwendungen in den letzten Jahren entwickelt [Rau07] und ist zwischenzeitlich in der ISO 10681 standardisiert [fS10a]. Die X-by-Wire Anwendungen, also durch Software realisierte Funktionen, sind besonders sicherheitsrelevant und erfordern neben einer hohen Durchsatzrate eine exakte zeitliche Kommunikation. Für eine solche Anwendung stößt der größtenteils verwendete CAN-Bus an seine Grenzen. Zum einen sind Nachteile bei der Topologie des CAN-Busses vorhanden. Ein CAN-Bus ist ein Linienbussystem mit kurzen Stichleitungen. Hierdurch ist oftmals eine suboptimale Kabelführung in den Fahrzeugen erforderlich, um alle Steuergeräte zu vernetzen. Des Weiteren besitzt er nur eine maximale Datenrate von 1 MBit/s [ZS08]. Zum anderen ist nachteilig, dass der CAN-Bus nicht redundant ausgelegt und nicht deterministisch ist [ZS08].

  Hier schafft der FlexRay Bus Abhilfe. Der FlexRay Bus wurde zur Kommunikation von Steuer- und Regelaufgaben für sicherheitsrelevante Echtzeitsysteme entwickelt. Er unterstützt ein- und zweibandige Systeme sowohl in Linien- als auch in der Sterntopologie. Hierdurch ist eine redundante Auslegung der Kommunikation möglich. Ferner ist er ein zeitsynchroner Bus, d.h. sein Verhalten ist deterministisch und der Zeitpunkt des Versendens bzw. Empfangens von Nachrichten kann vorhergesagt werden. Dies wird durch vorgegebene Zeitschlitze (Slots) mit gleicher Länge erreicht [ZS08]. Während der Entwicklungsphase wird festgelegt, welches Steuergerät in welchem Zeitschlitz senden darf. Somit ist die Zeitspanne für das Versenden einer Nachricht bekannt. Eine besondere Bedeutung obliegt dabei der Synchronisation. Denn nur wenn alle Steuergeräte am Bus synchronisiert sind,

können sie in ihrem vorgegebenen Zeitschlitz senden. Von daher wird in der Aufweckphase des Busses eine Synchronisation mittels Nachrichten durchgeführt. Hierzu werden von sogenannten Kaltstart-Knoten (Coldstart Nodes) Nachrichten versendet, die den anderen Steuergeräten eine Synchronisation erlauben [ZS08]. Die Definition der Aufweckphase wird während der Entwicklung vorgenommen. Der FlexRay Bus wird derzeit vorwiegend in der Motorsteuerung eingesetzt [Rau07].

- **Media Oriented System Transport (MOST)**
  Der MOST Bus wurde speziell als Infotainment Bus für Telematik- und Multimediaaufgaben konzipiert [Grz07]. Die Anforderungen in diesem Bereich an Echtzeitverhalten und Übertragungssicherheit sind geringer als in anderen Bereichen des Automobils. Dafür erfordern sie jedoch eine höhere Datenrate. Dies wird beim MOST Bus beispielsweise durch die Verwendung von Lichtwellenleitern gewährleistet [ZS08]. Der MOST Bus nutzt dabei eine bitstrom-orientierte Übertragung in einer logischen Ring-Struktur. Dabei dient ein Steuergerät als Master (Timing Master) [ZS08]. Beim Startvorgang leitet der Master zunächst eine Abfrage nach den entsprechenden Teilnehmern ein. Die an den Bus angeschlossenen Steuergeräte senden eine Antwort an den Master. Die hieraus gewonnenen Informationen für die angeschlossenen Geräte werden in einer Tabelle (Central Registry) gespeichert [ZS08]. Die Daten beinhalten die Verteilung der Softwarefunktionen, d. h. es wird gespeichert, welche Steuergeräte welche Funktionen beinhalten. Die Nachrichten werden dabei stets von Gerät zu Gerät weitergeleitet. Hierzu ist ggf. das Aufwecken der jeweiligen Steuergeräte erforderlich, da sie zwischenzeitlich aus Gründen der Energieeffizienz in den Ruhezustand versetzt wurden.

Die divergierenden Einsatzgebiete der Busse ergeben sich nach deren individuellen Eigenschaften. So werden die schnellen Busse (High Speed Systeme) wie CAN (High Speed) und FlexRay zur Kommunikation zwischen echtzeitkritischen Steuergeräten eingesetzt [ZS08]. Anwendungsbeispiele sind die Motor-, Brems- oder Fahrwerksteuerung. Das Hauptmerkmal dieser Systeme besteht darin, dass Daten möglichst schnell zwischen den Steuergeräten übertragen werden, wobei zusätzliche Maßnahmen, wie die Kollisionsdetektion, eingesetzt werden, die die Übertragung der Daten garantieren. Aus Kostengründen sind die schnelleren Busse mit hohen Datenraten den echtzeitkritischen Systemen vorbehalten.

Für weniger kritische Steuergeräte kommen Busse mit geringerer Datenrate als kostengünstigere Alternative zum Einsatz. Dies sind die CAN (Low Speed) und LIN Busse. Sie übertragen größtenteils Ansteuerungssignale für Lampen, Fenstermotoren oder andere Aktoren [ZS08]. Die Datengröße der Signale in diesem Segment ist gering. Von daher ist die niedrige Datenrate der Busse ausreichend. Für den Infotainmentbereich wird ein Bus mit hohen Durchsatzraten benötigt, der dagegen keinen „harten“ Echtzeitanforderungen genügen muss

[ZS08]. Höhere Latenzzeiten sind somit durchaus zulässig. Aus diesem Grund wird hier der MOST Bus eingesetzt.

Zum Datenaustausch zwischen den einzelnen Bussystemen wird ein zentrales System, das sogenannte Gateway eingesetzt [ZS08]. Es ist ein zentraler Bestandteil der Netzwerktopologie in einem Fahrzeug (siehe Abbildung 2.12). Die Aufgabe des Gateways ist die Kopplung und der Informationsaustausch zwischen den jeweiligen Bussen. So können CAN-Nachrichten zum Beispiel in LIN-Nachrichten transformiert und dann weiter übertragen werden. Unter Zugrundelegung dieser Merkmale ergibt sich die folgende Netzwerktopologie eines Kraftfahrzeugs (vgl. Abbildung 2.12).

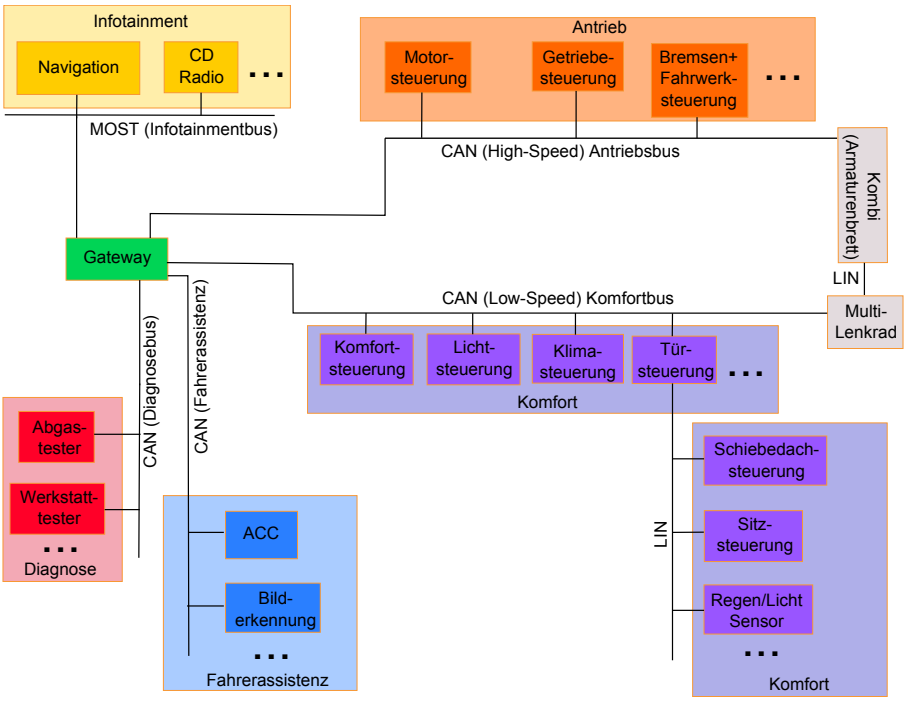


Abb. 2.12. Verschiedene Bussysteme in einem Automobil

Ein Hauptmerkmal aller Bussysteme besteht darin, dass es keine Punkt-zu-Punkt Kommunikation gibt, sondern dass alle Nachrichten im Broadcast Verfahren gesendet werden [ZS08]. Trotzdem können die Kommunikationsbeziehungen in einem Fahrzeug als Sender-Empfänger Beziehungen dargestellt werden. Es ist bekannt, wer die Nachricht sendet und welche Steuergeräte die Nachricht lesen und weiterverarbeiten. Die Sender-Empfänger Beziehungen werden auf Grund dessen in einer so genannten Kommunikationsmatrix (K-Matrix) tabellarisch zusammengefasst. Die Matrix enthält alle kommunika-

tionsrelevanten Informationen des Netzwerks [SZ13]. Neben den eigentlichen Kommunikationsbeziehungen beinhaltet die Matrix weitere Informationen bezüglich der übertragenden Daten. In den Kommunikationsdaten ist die physikalische Auflösung, der Offset und die jeweilige Einheit des Signals spezifiziert. Mit diesen Daten sind alle für die Kommunikation notwendigen Daten definiert.

2.6.3 Software im Steuergerät

Die Software in einem automobilen Steuergerät kann in die drei Bestandteile Applikations-, Diagnose- und Basis-Software unterteilt werden (vgl. Abbildung 2.13). Dabei werden in der Applikations-Software die Algorithmen für die jeweilige Funktion umgesetzt. Wie aber zu erkennen ist, nimmt diese Software nur einen kleinen Anteil an der Gesamtsoftware in einem Steuergerät ein. Der Großteil der Software besteht aus der Diagnose- und der Basis-Software. In diesen Bereichen werden die Zugriffe auf die Vielzahl der Ein- und Ausgänge des Steuergerätes umgesetzt (vgl. Kapitel 2.1). Die Größe der Software nimmt beständig zu, so dass vor der Erstellung der Software noch eine Analysephase benötigt wird, um die Komplexität der Software in den Griff zu bekommen. Für die erfolgreiche Realisierung des Steuergerätes bedarf es nicht nur der Modellierung der Applikations-Software, sondern vielmehr die Modellierung der gesamten Software mit einem besonderen Fokus auf die Basis-Software.

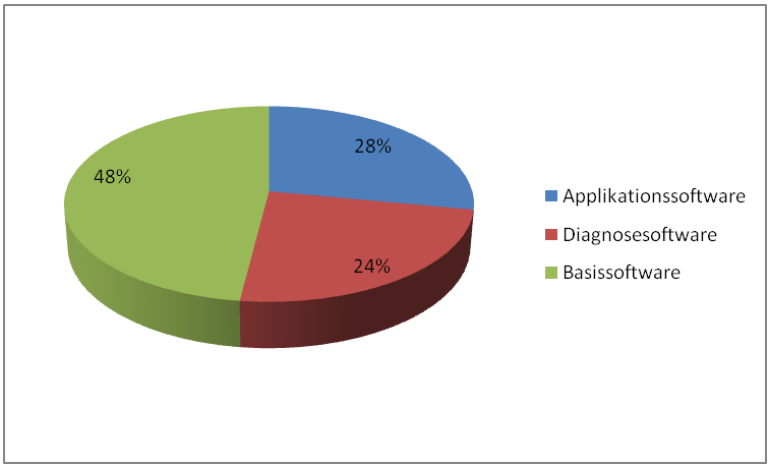


Abb. 2.13. Exemplarische Aufteilung der Gesamtsoftware eines Steuergerätes anhand des Beispiels des Komfortsteuergerätes

2.6.4 Globales Verhalten inklusive Echtzeit

Ein Steuergerät ist ein echtzeitkritisches System. Deswegen spielt das Verhalten und die Ausführungszeiten eine wichtige Rolle bei der Entwicklung. Ein Beispiel, die die besondere Bedeutung unterstreicht, ist beim Startverhalten des Steuergerätes auszumachen. Es sind fast immer Fristen vorgegeben, zu denen das Steuergerät hochgefahren sein muss, um bereits einkommende Nachrichten über den Kommunikationsbus empfangen und verarbeiten zu können. So muss beispielsweise die Alarmanlage beim Öffnen eines Fahrzeugs sofort deaktiviert werden. Es darf kein Alarm ausgelöst werden, wenn eine Fahrzeugtür geöffnet wird. Ein vergleichbares Verhalten ist beim Runterfahren der Systeme zu erkennen. Die Systeme müssen die Nachricht zum Herunterfahren erhalten und in einer vorgegebenen Zeit verarbeiten. Ansonsten besteht die Gefahr, dass das Steuergerät weiterhin aktiv geschaltet bleibt. Infolgedessen wird die Energieversorgung belastet. Letztendlich kann es sogar dazu führen, dass die Batterie hierdurch entleert wird. Daher muss das Verhalten und die entsprechenden Ausführungszeiten detailliert spezifiziert und umgesetzt werden.

2.6.5 Automobiles Betriebssystem

Das Herzstück eines jeden Rechners und somit auch eines Steuergerätes ist der Prozessor (engl. Central Processing Unit (CPU)). Neben dem Prozessor gibt es weitere Ressourcen wie Speicher etc. Für die Kontrolle und Steuerung der CPU und der weiteren Ressourcen ist das Betriebssystem verantwortlich. Es dient auf der einen Seite zur Ressourcenverwaltung und auf der anderen Seite erlaubt es der übrigen Software einen besseren Zugriff auf die darunter liegende Hardware [Tan02]. In Abbildung 2.14 ist die Einordnung des OSEK Betriebssystems in einem Steuergerät veranschaulicht. Die überliegende Applikations-Software wird vom Betriebssystem gesteuert. Das Betriebssystem besitzt somit einen großen Einfluss auf die Performance des Systems. Dies muss bei der Architekturmodellierung, vor allem bei größeren Steuergeräten, wenn mehrere Prozessoren vorhanden sind, berücksichtigt werden.

Das konzeptionelle Modell, das hinter dem Betriebssystem steht, ist eine Vielzahl von sequentiellen Prozessen (eng. Task). Ein Prozess befindet sich immer in einem Zustand. Grundsätzlich sind dies die Zustände *Rechnend*, *Blockiert* und *Rechenbereit* (siehe Abbildung 2.15). Bei erweiterten Tasks ist ein zusätzlicher Zustand vorhanden [Tan02]. Dies ist der Zustand *Wartend* (siehe Abbildung 2.15). Da er eine Erweiterung darstellt, ist er grün gekennzeichnet. Die Prozesse werden den einzelnen Rechenkernen zugeordnet. Dabei sind einem Prozessor fast immer mehrere Prozesse zugeordnet, die um die Laufzeit konkurrieren. Die Festlegung der Reihenfolge, in welcher die Prozesse

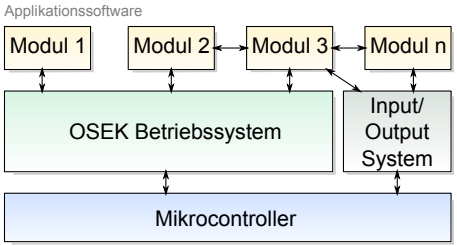


Abb. 2.14. Einordnung des OSEK Betriebssystems in die Architektur eines Steuergerätes

ausgeführt werden, ist Aufgabe des Betriebssystem. Sie wird im Scheduling-Algorithmus festgelegt. Ein Scheduling-Algorithmus für Echtzeitsysteme muss dafür sorgen, dass die Fristen (Deadlines) eingehalten werden. Hierbei kann in einem Echtzeitsystem zwischen periodischen (treten in regelmäßigen Zeitabständen) und aperiodischen (treten unvorhersagbar auf) Ereignissen unterschieden werden [Tan02]. Aperiodisch auftretende Ereignisse werden durch Unterbrechungen (Interrupts) gestartet, die von der Hardware ausgelöst werden. Exemplarisch kann hier der Nachrichteneingang von Bussignalen genannt werden.

Damit beide Arten von Ereignissen auf einer CPU ausgeführt und gleichzeitig die Fristen eingehalten werden, sind den Prozessen Prioritäten zugeordnet. Die einzelnen Task werden nach der Priorität entsprechend nacheinander aufgerufen. Damit die Prozesse, die durch Interrupts ausgelöst werden, möglichst zeitnah auf dem Prozessor rechnen können, werden ihnen hohe Prioritäten zugeordnet. Innerhalb einer Task können Funktionen zusammengefasst werden, die die gleiche Zykluszeit aufweisen. Es ist ebenso möglich, dass Funktionen mit einer Vielzahl der Zykluszeit der Task zugeordnet werden. Dies bedeutet, dass während der Laufzeit des Tasks die zugeordneten Funktionen in der spezifizierten Reihenfolge aufgerufen werden. Die Festlegung der Reihenfolge wird durch den Entwickler bestimmt. Die gewählten Einstellungen, beispielsweise die Erlaubnis einen Prozess zu unterbrechen, um einen Prozess mit höherer Priorität zu starten (Full Preemption) oder die Aufteilung der Programmfunktionen auf die Betriebssystemtasks und deren zyklische Aktivierung, bestimmen die Laufzeiten [OSE05]. Die Informationen bezüglich des Betriebssystems und insbesondere des Schedulings sind ein wichtiger Eingabewert für die Systemsimulation, die zu einem frühen Entwicklungszeitpunkt überprüft, ob bei den getroffenen Einstellungen mit Problemen beim Ressourcenzugriff oder mit zeitlichen Problemen zu rechnen ist.

Gewöhnlich wird als Betriebssystem in einem automobilen Steuergerät ein „Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug“ – kurz OSEK Betriebssystem – eingesetzt [OSE05]. Dieser internationale Standard wird vom OSEK/VDX Konsortium spezifiziert. Die nach dem Standard

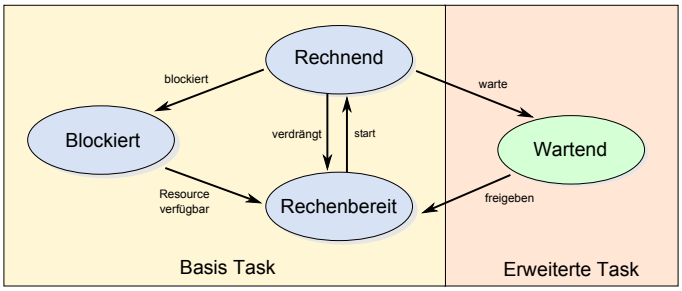


Abb. 2.15. Zustände im OSEK Betriebssystem [OSE05]

entwickelten Betriebssysteme können auf einer breiten Palette von Steuergeräten (ressourcenarme 8-bit bis hin zu leistungsstarken 32-bit Systemen) eingesetzt werden [OSE05]. Von daher kommt das Betriebssystem auf vielen Steuergeräten zum Einsatz. Eine Anforderung an das OSEK Betriebssystem war ein hoher Grad an Modularität und Konfigurierbarkeit. Hierdurch wird ermöglicht, das Betriebssystem auf die Bedürfnisse des jeweiligen Steuergerätes zuzuschneiden und an die jeweiligen Bedingungen anzupassen.

Die Modularität wird durch die verschiedenen Komponenten sichergestellt, die je nach Bedarf genutzt werden können. Die Konfigurierbarkeit der einzelnen Module erfolgt durch Parameter. Die Parameter werden mittels der *OSEK Implementation Language (OIL)* spezifiziert [OSE05]. Die standardisierte Sprache erlaubt die vollständige Spezifikation der Konfiguration des Betriebssystems. Sie kann als textuelle domänenspezifische Entwicklungssprache angesehen werden. Die Sprache stellt aber nur die Konfiguration des Betriebssystems zur Verfügung und muss folglich mit der restlichen Systementwicklung kombiniert werden. Sie kann jedoch dazu verwendet werden, um die Betriebssystemeigenschaften mit anderen Werkzeugen auszutauschen, z. B. mit speziellen Betriebssystemkonfigurationswerkzeugen. Durch einen Import- und Exportmechanismus ist sichergestellt, dass die Daten konsistent bleiben und durchgängig in allen Werkzeugen genutzt werden können.

2.7 Automotive Open System Architecture (AUTOSAR)

Um die Ziele und die Vorgehensweise der Transformation bzw. Synthese der Daten aus dem System-/Softwaremodell verständlicher zu machen (siehe hierzu Kapitel 6), wird in diesem Kapitel der AUTOSAR Standard beschrieben. Das Akronym AUTOSAR steht für eine offene Systemarchitektur im Automobilbereich (*AUTOSAR*) und ist ein internationaler Standard in der Automobilindustrie. Der Standard beinhaltet sowohl eine Referenzarchitektur als auch eine anzuwendende Methodik [AUT10b]. Beide werden am Ende dieses Abschnitts noch näher erläutert.

Durch die hohe Beteiligung und Einbeziehung sowohl der Automobilhersteller als auch der Zulieferer und der Werkzeughersteller nimmt die Verbreitung des Standards stetig zu. Inzwischen gibt es die ersten Steuergeräte, die komplett nach dem AUTOSAR Standard gefertigt werden [SF11] [FKS10]. Derzeit liegt der Standard in der Version 4.0 vor. An neueren Versionen wird aktuell gearbeitet [AUT10b]. Innerhalb der AUTOSAR Partnerschaft gibt es verschiedene Stufen der Zugehörigkeit (vgl. Abbildung 2.16), die mit mehr Mitspracherechten aber auch mit mehr Ressourcenbereitstellung gekoppelt sind.

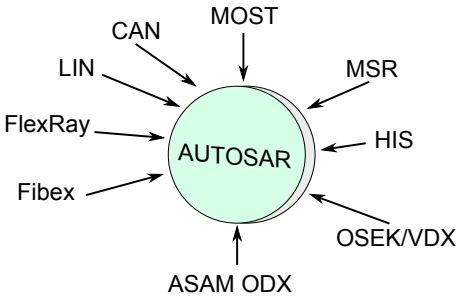


Abb. 2.17. Von AUTOSAR weiter verwendete Standards [AUT10b]

Unter einer offenen Systemarchitektur für ein eingebettetes System, zu denen die automobilspezifischen Systeme gehören, werden die Bereiche Hardware, Mechanik und Software subsummiert [KF09]. Im AUTOSAR Standard werden viele bereits anerkannte Standards aufgegriffen und weiterverwendet (vgl. Abbildung 2.17). Für die Hardware werden die Informationen im AUTOSAR Standard behandelt, die für die Erstellung der Software notwendig sind. Dies sind Informationen bezüglich der Rechenkerne, der Hardware/Software Schnittstellen und der verwendeten Busse. Weitergehende Konzepte bezüglich der Hardwareentwicklung werden nicht behandelt, da hierfür bereits auf bestehende internationale Standards zurückgegriffen werden kann. Exemplarisch kann hier auf die diversen internationalen Busstandards verwiesen werden [ZS08]. Die gleiche Aussage trifft ebenso auf die Mechanik zu. Von daher werden im Standard keine neue Spezifikationen für die Mechanik vorgenommen [KF09].

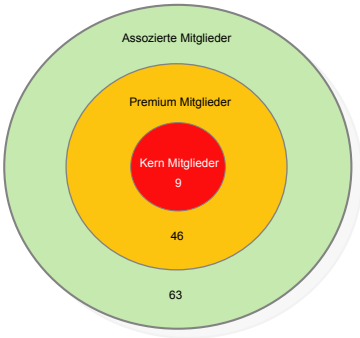


Abb. 2.16. Verschiedene Klassen von AUTOSAR Partnern [KF09]

Für die Softwareentwicklung im Automobilbereich existierte bislang kein internationaler Standard. Deshalb wurde mit dem AUTOSAR Standard Abhilfe geschaffen [KF09]. Ein solcher Standard für die Entwicklung der Software ist unverzichtbar, da nicht nur der Umfang der Software zunimmt, sondern ebenso der Vernetzungsgrad zwischen den einzelnen Softwarekomponenten steigt. Dies wird noch dadurch erschwert, dass bestehende oder zugelieferte Komponenten integriert werden müssen, so dass viele heterogene Komponenten in der Architektur zusammenarbeiten (vgl. Abbildung 2.18). Von daher ist es das Ziel des AUTOSAR Standards, die Integration und damit die Wiederverwendung zu erhöhen [POFG05]. Dies erfolgt durch mehr Hardwareunabhängigkeit, die durch die Einführung einer Mittelschicht erreicht wird. Hierdurch sollen u. a. die steigenden Entwicklungskosten beim Automobilhersteller und Zulieferer minimiert werden (vgl. Abbildung 2.19).

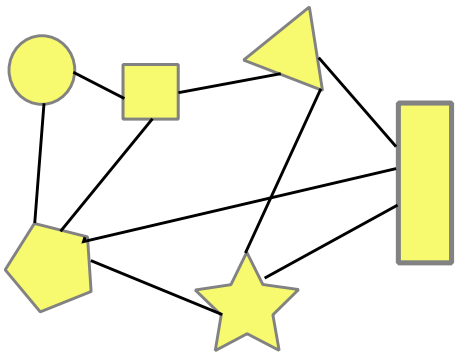


Abb. 2.18. Was macht eine komplexe Architektur aus? nach [KF09]

Im Automobilbereich wurde bereits früh eine Aufteilung der Software in hardwareabhängige (Basis-Software) und hardwareunabhängige Bereiche (Applikations-Software) vorgenommen [SZ13]. In der Basis-Software wird die Infrastruktur für die Applikations-Software wie das Betriebssystem, die Kommunikationsmechanismen, Treibern und weitere benötigte Komponenten bereitgestellt.

Ein großes Problem bei einem nicht AUTOSAR Steuergerät ist die Modellierung der Schnittstellen zwischen der Applikations- und der Basis-Software. Es bestehen keine Absprachen. Jeder Automobilhersteller und Zulieferer hat seine eigenen Schnittstellen definiert. Dies sind die sogenannten Basis-Cores. Dies führt zwangsläufig dazu, dass die Applikations-Software fortlaufend angepasst werden muss, wenn sie auf einer anderen Hardware oder aber für einen anderen Anwender verwendet wird. Damit ist der Vorteil, der durch die Aufteilung in die beiden Bereiche erreicht werden sollte, nicht mehr so groß wie zunächst angestrebt.

Hier schafft der AUTOSAR Standard Abhilfe. Durch die Verwendung einer Zwischenschicht (Middleware) – der sogenannten Runtime Environment (Runtime Environment (RTE)) – und standardisierte Schnittstellen ist es gelungen, dass die Applikations-Software nicht mehr verändert werden muss, wenn sie auf einer anderen Hardware oder für einen anderen Kunden wiederverwendet werden soll (vgl. Abbildung 2.19) [KF09]. Eine genauere Erläuterung der Funktionsweise der RTE wird im Kapitel Die AUTOSAR Architektur (siehe Kapitel 2.7.1) gegeben.

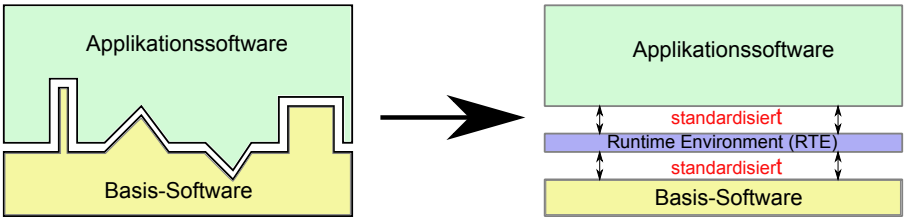


Abb. 2.19. Unterschied zwischen klassischem und AUTOSAR Steuergerät [KF09]

Um die Beherrschung der Komplexität und die Steigerung der Wiederverwendung zu erreichen, gibt AUTOSAR einen standardisierten Rahmen (die Schnittstellen) und eine allgemeine Referenzarchitektur vor. Die konkrete Implementierung wird den einzelnen Entwicklern überlassen. Dies wird durch das AUTOSAR-Motto „Cooperate on standards, compete on implementation." belegt [AUT10b]. Die Rahmenrichtlinien werden von AUTOSAR vorgegeben, insbesondere der Aufbau der Schnittstellen. Das Verhalten der Komponenten ist dabei vom Entwickler frei wählbar.

Der AUTOSAR Standard kann in drei Bereiche, die sich teilweise überschneiden, aufgeteilt werden (vgl. Abbildung 2.20). So gibt es zunächst den Bereich der **Architektur**. In diesem Bereich des AUTOSAR Standards wird die Referenzarchitektur einschließlich der darin spezifizierten Komponenten dargestellt.

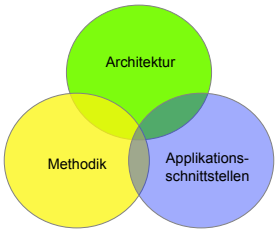


Abb. 2.20. AUTOSAR Bestandteile

Der zweite Bestandteil von AUTOSAR ist die explizit hierfür entwickelte **Methodik**. In der Methodik wird beschrieben wie AUTOSAR einzusetzen ist, um ein Steuergerät im Automobilbereich zu entwickeln. Sie beschreibt die verschiedenen Artefakte und deren Zusammenhang innerhalb der AUTOSAR Entwicklung. Die Methodik ist jedoch nicht mit einem eigenständigen Entwicklungsprozess zu verwechseln, da wesentliche Eigenschaften fehlen. Hierunter fallen die fehlenden Verantwortlichkeiten für die einzelnen Tasks. Dahinter steht die Idee, dass die AUTOSAR Methodik auf die einzelnen Phasen eines bereits existierenden Entwicklungsprozesses zugeordnet wird. Die Methodik beschreibt die Abhängigkeiten der Entwicklungsartefakte untereinander und gibt somit vor, *wann* (in Form der Reihenfolge) und *wie* (in Form der zu erstellenden Artefakte) etwas zu entwickeln ist [KF09].

Der dritte und letzte Bereich des AUTOSAR Standards betrifft die **Applikationsschnittstellen**. Innerhalb des Standards werden für einige Bereiche des Automobilbereichs (Komfort, Sicherheit, Antrieb etc.) typische Schnittstellen spezifiziert [AUT10a]. Diese dienen wiederum als Standard für die zu erstellende Applikations-Software. Genauere Informationen sind in den entsprechenden AUTOSAR Dokumenten nachzulesen [AUT10a]. Im Folgenden werden somit zunächst die Spezifikationen der AUTOSAR Architektur beschrieben und anschließend die Methode erläutert.

2.7.1 Die AUTOSAR Architektur

Ein wichtiger Aspekt des AUTOSAR Standards ist die Bereitstellung einer Rahmenarchitektur, die den typischen Aufbau einer AUTOSAR Architektur für ein Steuergerät bestimmt (vgl. Abbildung 2.21) [AUT10c, AUT10j]. Wesentliche Besonderheiten der Referenzarchitektur ist die Verwendung des sogenannten Virtual Functional Bus (VFB) Prinzips und der Standardisierung der Schnittstellen zur Anbindung der Komponenten an die Runtime Environment (RTE). Auf der untersten Ebene befindet sich die Hardware des Steuergerätes. Die Hardware besteht aus Speicher, Prozessor, Sensoren, Aktoren und weiteren Peripheriegeräten sowie ggf. die Anbindung an einen Kommunikationsbus. Hierauf aufbauend befindet sich die Basis-Software. Die Basis-Software besteht aus unterschiedlichen hardwareabhängigen Komponenten und stellt verschiedene Dienste bereit. Die Dienste umfassen dabei das Betriebssystem, den Speicherzugriff, den Zugriff auf die Kommunikation und weitere Systemdienste.

Oberhalb der Basis-Software liegt die Runtime Environment. Die Runtime Environment ist eine Implementierung des Virtual Functional Bus Prinzips (vgl. Kapitel 2.7.1). Sie ist durch standardisierte Schnittstellen an die Basis-Software angebunden. Diese Schnittstellen sorgen dafür, dass die Kenntnis der konkreten Komponenten, die sich in der Basis-Software befinden, nicht erforderlich ist. Von daher können Teile der Basis-Software ausgetauscht werden, ohne dass die

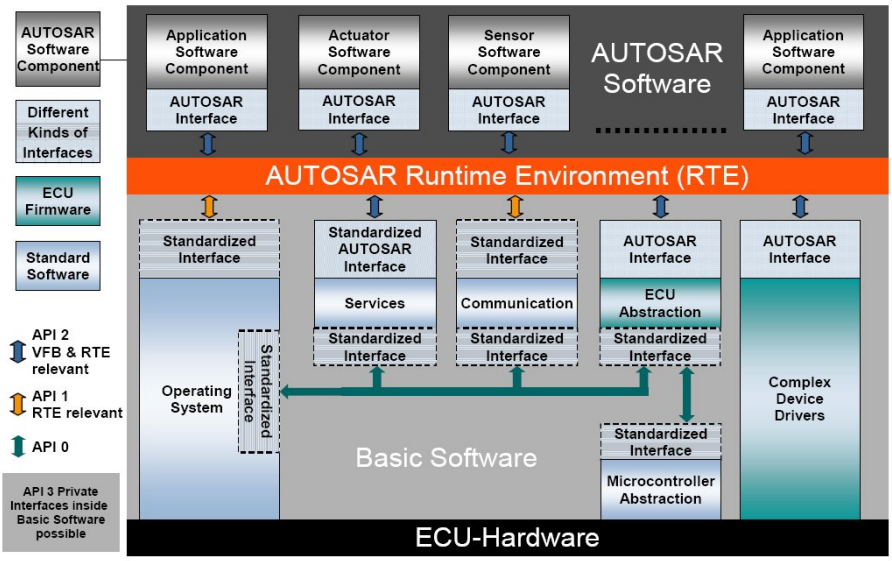


Abb. 2.21. Aufbau eines AUTOSAR Steuergerätes [AUT10b]

Applikations-Software erneuert werden muss [KF09]. Die RTE ist vergleichbar mit einer Middleware und verbindet die Basis-Software mit der Applikations-Software. Ihre Hauptaufgabe ist die Herstellung der Kommunikation zwischen den einzelnen Komponenten [KF09]. Die Kommunikation besteht nicht nur allein zwischen Applikations- und Basis-Software einer Softwarekomponente, sondern es kann ebenso eine Verbindung zu einem anderen Steuergerät aufgebaut werden, sofern eine Komponente auf einem anderen Steuergerät untergebracht ist (vgl. Kapitel 2.7.1).

Oberhalb der Runtime Environment befindet sich die Applikations-Software. In dieser sind die Logik und das Verhalten untergebracht. Durch die standardisierten Schnittstellen zur RTE ist es möglich, die Applikations-Software auf unterschiedlichen Steuergeräten unterzubringen oder zu verlagern, solange die Basis-Software die notwendigen Dienste und Daten bereitstellt. Hierdurch ist eine Wiederverwendung sowohl bei neu entwickelten Produkten als auch für die Nutzung in verschiedenen Varianten, beispielsweise für verschiedene Automobilhersteller, gewährleistet.

Das Prinzip des Virtual Functional Bus

Das Prinzip dient dazu, auf Softwareebene die logischen Kommunikationsbeziehungen zu modellieren. Dies wird in der Abbildung 2.22 im oberen Abschnitt veranschaulicht. Es werden dort die verschiedenen Kommunikationsbeziehungen zwischen den einzelnen Softwarekomponenten dargestellt. Die Art

der Kommunikationsverbindung (Client/Sender oder Sender/Receiver) wird in diesem Modellierungsschritt mit festgelegt [KF09]. Durch die Modellierung der logischen Kommunikationsverbindung wird genau spezifiziert, welche Softwarekomponenten miteinander in Verbindung stehen und welche nicht.

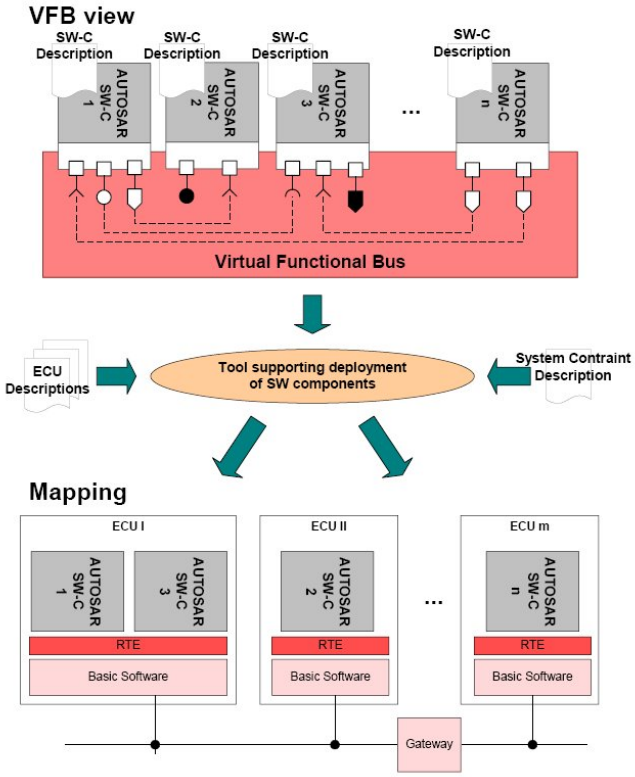


Abb. 2.22. AUTOSAR Virtual Functional Bus Prinzip [AUT10b]

Die Verteilung der Software und die genaue Spezifikation der Hardware- und Kommunikationsarchitektur werden durch die ECU und die System Description zu einem späteren Zeitpunkt hinzugefügt. Aus diesen zusätzlichen Informationen wird dann mithilfe einer Werkzeugunterstützung eine konkrete Softwarearchitektur mit der entsprechenden Verteilung generiert (vgl. Abbildung 2.22). Dies bedeutet, dass die einzelnen Softwarekomponenten auf der entsprechenden Hardware allokiert werden. Außerdem wird für das jeweilige Steuergerät die entsprechende Runtime Environment generiert. Diese stellt eine Implementierung des Virtual Functional Bus dar.

Die Hauptaufgabe der RTE ist die Bereitstellung der notwendigen Kommunikationskanäle, so dass die in der logischen Ebene spezifizierten Kommunikationsverbindungen realisiert werden können. Hierzu muss die Verteilung der Software auf die Hardware definiert sein. Dem RTE-Generator muss bekannt sein, ob eine interne (realisiert über Speicherzugriffe) oder aber eine externe (realisiert über einen Bus) Kommunikation vorliegt.

Die AUTOSAR Basis-Software

Die AUTOSAR Basis-Software wird in sechs vertikale und drei horizontale Bereiche weiter unterteilt, wobei die jeweiligen Komponenten sowohl einem horizontalen als auch einem vertikalen Bereich zugeordnet werden (vgl. Abbildung 2.23). Die horizontalen Bereiche abstrahieren von der Hardware. Es sind die folgenden Bereiche:

- **Serviceschicht (Services Layer)**: Die Serviceschicht umfasst die anwendungsunabhängigen Dienste wie das Betriebssystem, verschiedene Kommunikationsprotokolle und die Speicherverwaltung. Die Serviceschicht ist eine große Schicht und umfasst sowohl hardwareunabhängige als auch -abhängige Module [KF09].
- **ECU Abstraktionsschicht (ECU Abstraction Layer)**: Die ECU Abstraktionsschicht ist die erste der hardwareabhängigen Schichten innerhalb der Basis-Software. Sie befindet sich oberhalb der Microcontroller Abstraktionsschicht und bietet Schnittstellen zu dieser Schicht an. Sie sorgt dafür, dass die höher liegenden Schichten unabhängig von der eingesetzten Hardware sind und bietet zudem noch Zugang zu den einzelnen Ein- und Ausgabesignalen.
- **Mikrocontroller Abstraktionsschicht (Microcontroller Abstraction Layer MCAL)**: Die Mikrocontroller Abstraktionsschicht ist die zweite der hardwareabhängigen Abstraktionsschichten. Sie abstrahiert von der eigentlichen Hardware also den Microcontrollern und der weiteren integrierten Peripherie. So befinden sich beispielsweise die verschiedenen Treiber in dieser Schicht.

Die vertikalen Bereiche werden entsprechend den einzelnen Komponenten und ihren Funktionen gegliedert. Es sind die folgenden Bereiche vorhanden:

- Systemdienste
- Geräte-Stack
- Speicher-Stack
- Kommunikations-Stack
- I/O Stack

- Komplexe Gerätetreiber (Complex DeviceDrivers)

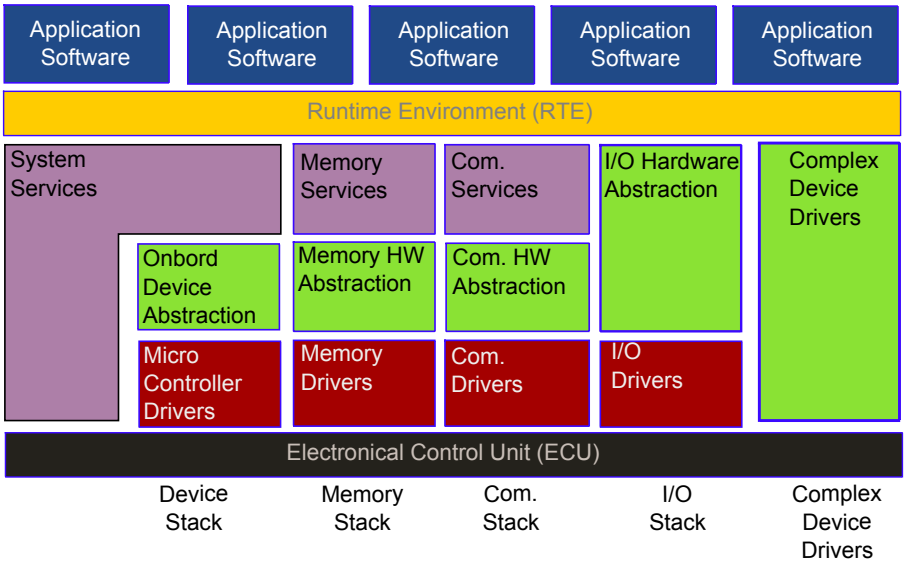


Abb. 2.23. Schichten der AUTOSAR Architektur [AUT10b]

Aufgabe des Service Stacks ist die Versorgung der Applikations- und der übrigen Basis-Software mit grundlegenden Diensten (Services) [KF09]. Diese beinhalten zahlreiche Dienste, wie das Betriebssystem oder aber die Diagnose. In Abbildung 2.23 befinden sich die Systemdienste in der Basis-Software ganz links. Wie deutlich zu erkennen ist, liegen die Dienste zum Teil oberhalb der anderen Stacks und bilden somit den Übergang zur RTE. Andere Teile setzen aber auch direkt auf der ECU auf. Im Gegensatz zu den anderen Stacks erstrecken sich die Systemdienste ausschließlich über eine einzige Schicht, der Serviceschicht und sind nicht in weitere Schichten aufgeteilt.

Der sich daran anschließende Bereich ist der sogenannte Geräte-Stack (Device Stack). Dieser Stack erstreckt sich neben und unterhalb der Systemdienste. Er beinhaltet zwei funktionale Gruppen von Modulen. Dies sind die Onboard Device Abstraction und die Microcontroller Abstraction [KF09]. Sie sorgen für den Zugang zum Watchdog und zum Microcontroller.

Daran schließt sich der Speicher-Stack an. Dieser setzt sich aus drei Schichten zusammen. Er stellt Funktionen zum Speichern und Lesen von Daten in den flüchtigen (RAM) und den nicht-flüchtigen Speicher (Flash oder EEPROM) zur Verfügung. Ebenso kann über den Stack der NVRAM-Manager angesprochen werden.

Der vierte Bereich der AUTOSAR Basis-Software ist die Gruppe der Kommunikationsdienste im gleichnamigen Stack. In ihm befinden sich zwölf Module zur Abstraktion und zum Management der Bus-Kommunikation [KF09]. Sie ermöglichen das Senden und Empfangen von Kommunikationsnachrichten. Der I/O Stack abstrahiert von den Signalpfaden auf der Hardware. Er ist zuständig für das Setzen und Auslesen von digitalen und analogen Werten, der Pulsweitenmodulation (PWM) bzw. von I/O-Werten [KF09]. Der I/O Stack besteht aus den beiden Modulen I/O Hardware Abstraction und den zugehörigen I/O Treibern.

Der letzte Bereich in der AUTOSAR Basis-Software sind die sogenannten Complex Device Treiber. Sie gehören keiner Schicht an und liegen quasi neben den anderen Modulen (vgl. Abbildung 2.23). Für sie ist in AUTOSAR keine genauere Spezifikation gegeben. Dies ist sinnvoll, um ggf. bereits vorhandene Software in einem AUTOSAR Ansatz unterzubringen. So wird eine schrittweise Migration hin zu einem kompletten AUTOSAR System ermöglicht. Hierdurch kann bereits bestehende Software weiter genutzt werden, obwohl insgesamt eine AUTOSAR Architektur zur Anwendung kommt. Die Complex Device Treiber werden weiterhin eingesetzt, wenn Zeitanforderungen an Komponenten bestehen, die durch eine AUTOSAR Architektur nicht zu realisieren sind. Dazu werden die Complex Device Treiber genutzt, um durch eine hardwareabhängige Implementierung die Zeitvorgaben umzusetzen.

2.7.2 AUTOSAR Methodik

In der AUTOSAR Methodik gibt es erst ab der Version 4.0 Rollenbeschreibungen [AUT10b]. Dies ist ein weiterer Schritt hin zu einem vollständigen Entwicklungsprozess. Es fehlen aber noch Verantwortlichkeiten. Vor allem beschreibt die AUTOSAR Methodik nur die Entwicklung von AUTOSAR Artefakten. Die zuvor beschriebenen Entwicklungsschritte, wie Anforderungsanalyse und die Erstellung einer logischen Architektur, sind dabei nicht beschrieben [DJM+08]. Darüber hinaus sind keine Beschreibungen für die Testphasen vorhanden. Von daher ist die AUTOSAR Methodik nicht mit einem Entwicklungsprozess zu vergleichen. Vielmehr entspricht die Methodik eher einem Arbeitsproduktfluss (Work Product Flow).

Die Methodik behandelt dabei drei verschiedenen Sichten. Dies sind:

- die Systemsicht
- die Steuergerätesicht (ECU)
- die Komponentensicht(Component)

In der Methodik wird u. a. beschrieben, wie der Übergang von der Systemsicht (Fahrzeugsicht) zur Steuergerätesicht (ECU Sicht) erfolgt. Hieraus resultiert die *System Configuration Description*. In ihr sind alle Informationen,

die das gesamte Fahrzeug betreffen, gespeichert. Aus diesem Artefakt werden anschließend die Informationen extrahiert, die für ein einzelnes Steuergerät notwendig sind. Dies sind größtenteils die Kommunikationsinformationen und teilweise – wenn die *System Configuration Description* sehr detailliert ist – weitere Informationen bezüglich der Softwarearchitektur.

Darauf aufbauend schließt sich der Übergang vom OEM zum Zulieferer an. Der OEM gibt die erforderlichen Informationen an den Zulieferer weiter. Nach der Analyse der Daten und dem weiteren Design wird das Steuergerät kontinuierlich weiter konfiguriert. Die notwendigen Artefakte, die für die Konfiguration herangezogen werden, werden in dem Paket *AUTOSAR Model* nachgewiesen. Aus allen Informationen ergibt sich sodann die *ECU Configuration Description* [AUT10d]. In ihr befinden sich alle Konfigurations- und Modellierungsdaten, die für das entsprechende Steuergerät relevant sind. Aus diesen Informationen werden durch entsprechende Generatoren die jeweiligen Basis-Softwarekomponenten (wie Betriebssystem (Operating System) (OS), MCAL, COM, etc.) erzeugt. Die Komponenten können dann in einem Kompilierungsschritt zu einer ausführbaren Datei zusammengefasst werden. Die Basis-Software wird anschließend mit der RTE und der Applikations-Software zu einer Gesamtsoftware integriert. Die Software wird danach auf die Hardware übertragen und das Steuergerät kann an den OEM ausgeliefert werden. In dieser Arbeit steht vor allem die Arbeit des Zulieferers im Fokus. Von daher wird ein besonderes Augenmerk auf die unteren Arbeitsschritte und Artefakte gelegt.

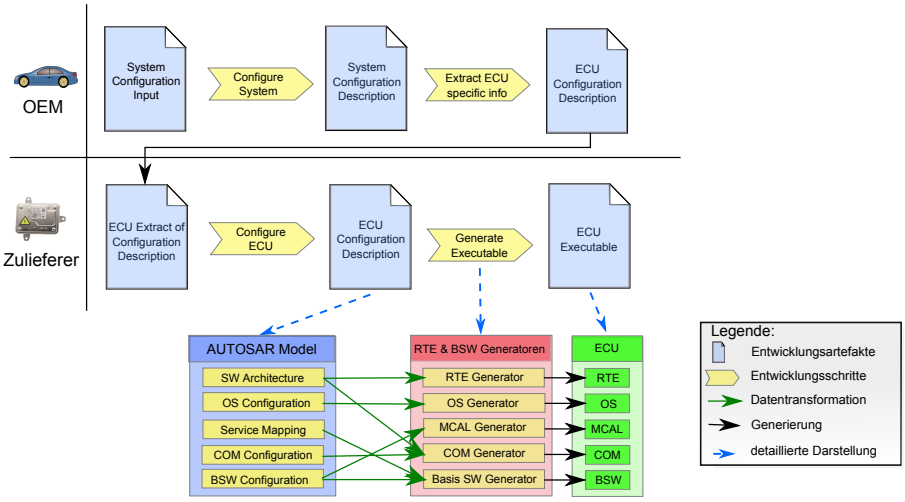


Abb. 2.24. Auszug aus der AUTOSAR Methode [AUT10b]

2.8 Entwicklungsprozesse im Automobilbereich

Die Erstellung von komplexen Systemen ist ohne einen dementsprechenden Entwicklungsprozess inklusive Methoden und Techniken, die diesen unterstützen, nicht mehr vorstellbar. Ein solcher Prozess regelt die einzelnen Arbeitsabläufe und -ergebnisse innerhalb der Entwicklung und ermöglicht außerdem die Projektplanung [SHI+11]. Ein Prozess ist eine Sammlung von Tätigkeiten, die für die Erstellung eines Produktes notwendig sind [Som07]. Dabei kann der Prozess als Sammlung von erfolgreichen Techniken und Arbeitsweisen (Best Practices) gesehen werden [SDW+10]. Ein Entwicklungsprozess definiert immer die notwendigen Aufgaben und Arbeitsschritte, also das was zu tun ist [Ost87]. Dies ist jedoch für die Entwicklung nicht ausreichend, so dass ein Prozess immer durch Methoden unterstützt wird. Die Methoden beschreiben wie die Aufgaben und Schritte des Prozesses erledigt werden können. Aufbauend auf den Methoden werden Werkzeuge ausgewählt, die auf der einen Seite eine Unterstützung und Automatisierung der Methoden zu lassen. Auf der anderen Seite können sie aber gleichzeitig für eine Einschränkung sorgen, wenn sie die Methode nicht vollständig umsetzen. Von daher ist immer das Zusammenspiel zwischen Prozessen, Methoden und Tools für die Entwicklung von Bedeutung (vgl. Abbildung 2.25).

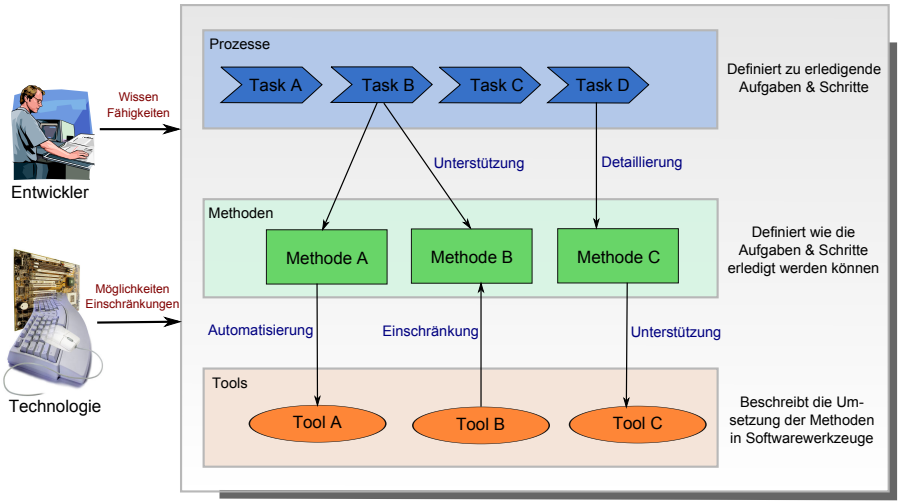


Abb. 2.25. Übersicht Prozesse, Methoden und Tools [Est08]

Studien belegen, dass nachhaltige und positive Änderungen und damit einhergehend eine Steigerung der Effektivität in der Produktivität, Qualität und Projektplanung sich nur dann erzielen lassen, wenn die Systementwicklung durch Prozesse und Methoden begleitet werden, die ständig angepasst und

optimiert werden [LR05]. Von Belang ist hierbei, dass in einem Entwicklungsprozess die beteiligten Rollen und Verantwortlichkeiten klar definiert sind. Durch eine solche Spezifikation erhält jeder Entwickler bei der Betrachtung des Entwicklungsprozesses die Möglichkeit, seine Aufgaben und Arbeitsartefakte zu identifizieren.

Definition (Entwicklungsprozess). Der Entwicklungsprozess ist das Vorgehensmodell bei der Softwareentwicklung. Er beschreibt auf abstrakte Weise die erforderlichen Aktivitäten, beginnend mit den Anforderungen, bis hin zum Betrieb der Software [IEE90]. Ein Entwicklungsprozess kann in unterschiedliche Stufen (Entwicklungsphasen) unterteilt werden. Im Prozess müssen ferner die Rollen spezifiziert sein, die für die Erstellung der Entwicklungsartefakte zuständig sind.

Einen allgemein verwendbaren Entwicklungsprozess, der für jedes Projekt oder jedes Unternehmen optimal einsetzbar ist, kann und wird es nicht geben [Som07]. Durch zahlreiche unterschiedliche Rand- und Nebenbedingungen wird es immer unumgänglich sein, stets einen maßgeschneiderten Entwicklungsprozess zu definieren. Konsequenterweise wird dieser von einem allgemeinen und anerkannten Entwicklungsprozess abgeleitet. Hierdurch ist sichergestellt, dass der Prozess und die darin definierten Methoden dem Stand der Technik entsprechen. Er deckt dabei alle Aufgaben und somit nahezu die gesamte Bandbreite einer Entwicklung ab [LR05]. Da jedes Projekt einzigartig in Bezug auf die Rahmenbedingungen und Projektziele ist, muss der Prozess für einzelne Projekte innerhalb eines Unternehmens weiter angepasst werden (Tailoring). Letztendlich entsteht somit für jedes Projekt ein individueller angepasster Entwicklungsprozess.

Als Grundlage zur Erarbeitung eines Entwicklungsprozesses sind verschiedene allgemein gültige Entwicklungsprozesse existent, die als Vorlage herangezogen werden können. Der klassische Entwicklungsprozess orientiert sich an dem sogenannten Wasserfallmodell [Roy70]. Dieser Prozess teilt sich in verschiedene Phasen auf und ist geprägt von Start- und Endpunkten. Das Charakteristikum liegt darin begründet, dass die nächste Phase erst dann gestartet werden kann, wenn die vorherige abgeschlossen ist. Von daher ist es zwingend erforderlich, dass alle Anforderungen von vornherein bekannt sein müssen. Im Nachhinein können sie nicht geändert oder angepasst werden. Somit ist dies ein sehr starrer Prozess. Dies führt dazu, dass das Wasserfallmodell kaum noch eingesetzt wird.

Zur Behebung der Defizite wurde das Wasserfallmodell weiterentwickelt. Es entstand das sogenannte V-Modell [Boe76]. Dieses Prozessmodell ist in der Industrie, und insbesondere bei der Entwicklung von eingebetteten und/oder sicherheitskritischen Systemen, weit verbreitet. Es besitzt, wie das Wasser-

fallmodell, verschiedene Phasen. Diese sind in Form eines Vs angeordnet. Daher entstand der Name dieses Prozesses. Dabei werden den Entwicklungsphasen auf der linken Seite des V-Modells die entsprechenden Teststufen zur Prüfung auf der rechten Seite gegenübergestellt (vgl. Abbildung 2.28 bzw. 2.30). So wird sichergestellt, dass die Ergebnisse der Entwicklungsphasen den gewünschten Zielen entsprechen. Dies erhöht die Qualität der zu entwickelnden Produkte.

Während der Steuergeräteentstehung wird die Entwicklung in einzelne Muster unterteilt. Dies sind vier Musterphasen (A bis D Muster), die bis zum Start of production (SOP) durchlaufen werden müssen (siehe Abbildung 2.26). Je näher das Muster am SOP liegt, desto mehr Funktionalität ist in dem Muster umgesetzt. Dies bedeutet, dass im A-Muster nur ein erster Entwurf der Funktionalität vorhanden ist, während bei einem D-Muster bereits die volle Funktionalität zur Verfügung steht.

Da für jedes Muster neue und zum Teil geänderte Anforderungen vom OEM erstellt werden, muss mindestens einmal das gesamte V-Modell durchlaufen werden (vgl. Abbildung 2.26), damit die erwartete Funktionalität im Muster korrekt umgesetzt wird. Hierfür ist es notwendig, die Entwicklungsphasen Analyse, Design, Implementierung und nicht zu vergessen die Testphase jeweils komplett zu durchlaufen, auch wenn dies mit einem großen Arbeitsaufwand verbunden ist. Um den Aufwand noch beherrschbar zu gestalten, ist vor allem eine durchgängige Entwicklungsmethodik notwendig, die tunlichst viele Automatismen beinhaltet. Gerade die Automatismen ermöglichen es, neue oder geänderte Anforderungen gezielt zu ergänzen und die Iterationen des Entwicklungsprozesses möglichst schnell zu durchlaufen. Die Entwicklungszeit und somit auch die Kosten können hierdurch beibehalten bzw. verringert werden.

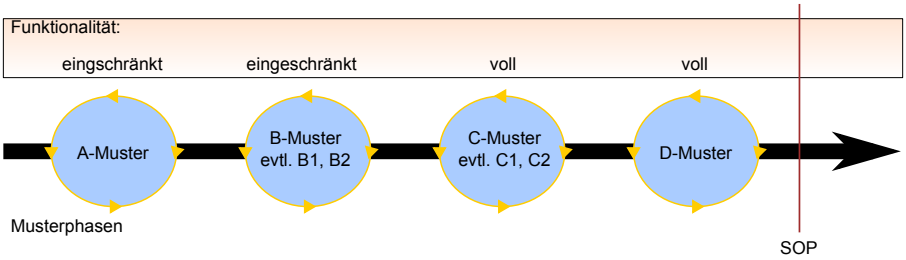


Abb. 2.26. Unterschiedliche Musterphasen in der Automobilentwicklung nach [Rei10a]

Ein weiterer Entwicklungsprozess ist die prototypische Entwicklung (Rapid Prototyping). Hierbei werden jeweils kleine Teile von Anforderungen möglichst früh in zahlreichen Prototypen umgesetzt und in Zusammenarbeit mit dem Kunden validiert [Bor08]. Ergeben sich neue Anforderungen, werden diese in den darauf folgenden Prototypen umgesetzt. Diese Vorgehensweise ist ver-

gleichbar mit den neuerdings eingesetzten sogenannten Agilen Verfahren, wie beispielsweise SCRUM [SI07, SDW+10] oder Extreme Programming [Bec00]. Diese Vorgehensweise setzt aber voraus, dass einerseits der Auftraggeber aktiv bei der Entwicklung mitwirkt und das Projekt andererseits überschaubar sein muss, um jederzeit den Überblick zu wahren. Bei der Entwicklung von eingebetteten Systemen im Automobilbereich sind diese Voraussetzungen jedoch nicht gegeben.

2.8.1 Konkreter Entwicklungsprozess eines automobilen Zulieferers

Bei der bisherigen Entwicklung wird vielfach nach dem „Throw it over the wall" Prinzip (vgl. Abbildung 2.27) [Ehr06] vorgegangen. Bei diesem Prinzip fängt eine Disziplin, meistens die Hardware an und die anderen Disziplinen folgen sequentiell. Auf Grund dieser Vorgehensweise werden bereits viele Vorgaben durch die erste Disziplin gesetzt, die dann unter Umständen ein optimales Zusammenwirken aller Disziplinen verhindert.

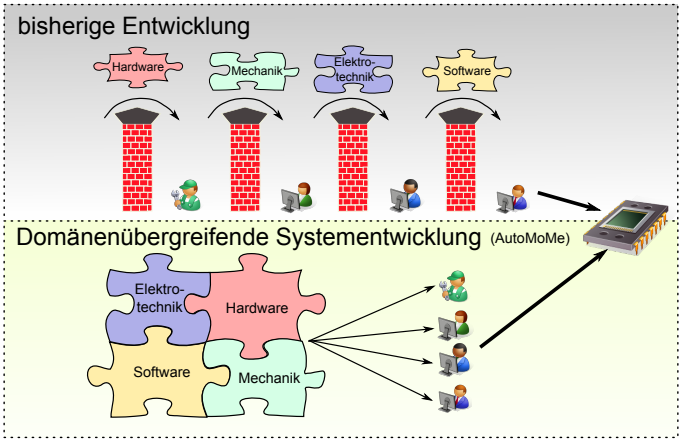


Abb. 2.27. Gegenüberstellung der derzeitigen und der zukünftigen Systementwicklung

Die gemeinsame Entwicklung der Hard- und Software muss aufeinander abgestimmt sein. Nur so kann das Gesamtsystem optimal entwickelt werden. Dies muss durch den Entwicklungsprozess gewährleistet sein. Von daher muss eine Entwicklungsphase vorgeschaltet werden, in der zunächst die einzelnen Disziplinen (Mechanik, Hard- und Software) gemeinsam den Systementwurf gestalten [KSW13, PBFG06]. Darauf aufbauend wird in die domänenspezifische Entwicklung übergegangen. Für den Automobilbereich ist danach eine multidisziplinäre Entwicklung erforderlich, wie sie beispielsweise in [SZ13] vorgestellt wird. Jedoch wird dabei offen gelassen, welche Methoden und Techniken dort

eingesetzt werden. Als Entwicklungsprozess ist noch immer das V-Modell erkennbar. Es wurde aber durch eine gemeinsame Systementwicklung und in verschiedenen Disziplinen unterteilte Prozesse ergänzt (vgl. Abbildung 2.28).

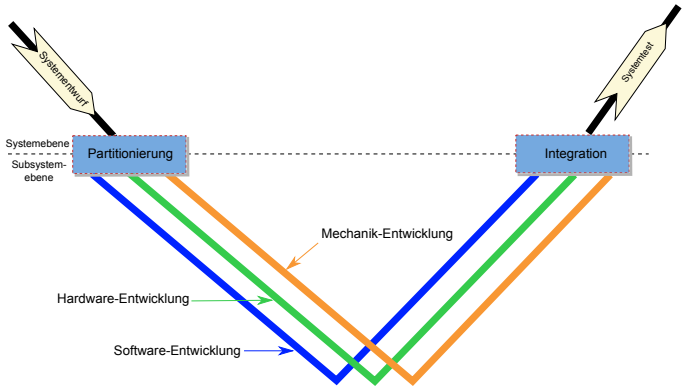


Abb. 2.28. Entwicklungsprozess im Automobilbereich [LR05]

Bei eingebetteten und sicherheitskritischen Systemen ist die Einhaltung von Normen und Standards unverzichtbar, um zukünftige Gefahrensituationen auszuschließen. Durch die Normen und Standards wird versucht, die Qualität der Systeme inklusive der Software hoch zu halten. Beispiele für genutzte Sicherheitsstandards sind die International Electrotechnical Comission (IEC) 61508 [IEC99] oder die für den Automobilsektor geltende International Organization for Standardization (ISO) 26262 [fS10b]. In den Standards werden zusätzliche Anforderungen und Vorgehensweisen für die Entwicklung definiert, beispielsweise welche Analysen und Testmethoden für besonders kritische Komponenten unabdingbar sind. Für die Architekturentwicklung wird z. B. gefordert, dass mindestens eine semi-formale Notation für sicherheitskritische Systeme eingesetzt werden muss. Diese Vorgabe ist für eine zukünftige Entwicklungsmethode zu beachten und wird im Lösungskonzept AutoMoMe entsprechend umgesetzt. Durch diese zusätzlichen Vorgaben wird die Qualität des Systems und der darin befindlichen Software optimiert.

In der Industrie ist es weithin anerkannt, dass systematische Entwicklungsprozesse und ein anerkanntes Prozesswissen für die Entwicklung von eingebetteten Systemen relevant sind [LR05]. Von daher wird immer mehr dazu übergegangen, die jeweiligen bei der Entwicklung zum Einsatz gelangten Prozesse zu bewerten, um die Qualität der Produkte beurteilen zu können. Die Bewertung eines Entwicklungsprozesses erfolgt durch sogenannte Reifegradmodelle. Diese Reifegradmodelle werden bei Begutachtungen, den sogenannten Assessments, genutzt. Es existieren verschiedene Reifegradmodelle. Die bekanntesten sind CMMI [SEI10] und SPICE [fS12]. In der Automobilindustrie hat sich eine abgewandelte Form des Software Process Improvement

and Capability Determination (SPICE) Modells, das sogenannte Automotive SPICE [dAV10], durchgesetzt. Dieses Reifegradmodell besteht aus den beiden Bestandteilen *Prozessreferenzmodell* (PRM) und *Prozessassessmentmodell* (PAM) [MHDZ07, fS12]. Für den Entwicklungsprozess ist die Kenntnis des Prozessassessmentmodells durchaus ausreichend [MHDZ07].

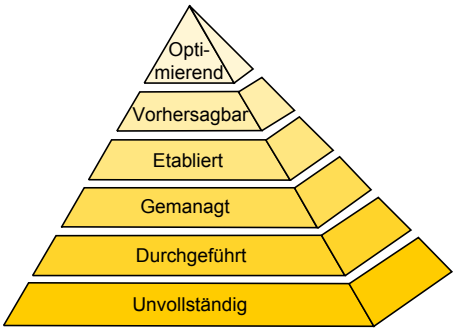


Abb. 2.29. Sechs Stufen des Automotive SPICE Reifegradmodells nach [MHDZ07]

Die Modelle besitzen zur Bewertung verschiedene Reifegradstufen. Beim Automotive SPICE sind dies sechs Stufen (vgl. Abbildung 2.29). Die Skala erstreckt sich von unvollständig (0) bis optimierend (5). Mittels der Modelle und ihrer Reifegradstufen ist es möglich, Fehler oder Mängel innerhalb des Entwicklungsprozesses zu identifizieren und zu beheben. Die Anwendung der Reifegradmodelle hat insbesondere bei der Zusammenarbeit der OEMs mit dem jeweiligen Zulieferer eine große Bedeutung. Ein Automobilhersteller setzt grundsätzlich voraus, dass sein beim Zulieferer in Auftrag gegebenes eingebettetes System mit größter Sorgfalt und Qualität entwickelt wird. Damit dies erfolgen kann, muss ein geeigneter Entwicklungsprozess mit unterstützenden Methoden und Techniken vorhanden sein. Von daher führen OEMs regelmäßig Beurteilungen und Bewertungen (Assessments) beim Zulieferer durch, um sich von der Qualität der Prozesse und der Methoden zu überzeugen. Von daher müssen die Entwicklungsprozesse des Zulieferers konform zum Automotive SPICE sein.

In der Abbildung 2.30 ist der Entwicklungsprozess eines namhaften deutschen Automobilzulieferers abgebildet. Dieser Entwicklungsprozess ist abgeleitet vom Automotive SPICE Bewertungsmodell, wobei die unterstützenden Prozesse (Zulieferer-, Qualitätsprozess etc.) in der Abbildung nicht dargestellt sind. Erkennbar ist die Ableitung von Automotive SPICE durch die Einteilung des Entwicklungsprozesses in verschiedene Entwicklungsphasen, den sogenannten Engineering Process Groups (Eng.1 - Eng.10) [MHDZ07]. Eine Besonderheit ist die Zusammenfassung der Entwicklungsphasen Eng.1 und Eng.2. Sie stellt eine Anpassung des Zulieferers dar, da die Anforderungserhebung (Eng.1) auf Seiten des Automobilherstellers erfolgt. Von daher ist es nicht erforderlich,

dass beim Zulieferer diese Phase erneut durchgeführt wird. In diesen beiden Phasen stehen die Anforderungen im Vordergrund. Sie werden erfasst und analysiert. Häufig werden dabei spezielle Anforderungsmanagementwerkzeuge (wie beispielsweise IBM Rational DOORS [IBM14a]) eingesetzt.

Bei diesem Entwicklungsprozess ist zu erkennen, dass in den frühen Phasen (Eng.1 - Eng.3) keine Aufteilung in Hard- und Softwareprozesse vorgenommen wird, sondern eine disziplinübergreifende Entwicklung stattfindet. Die Teilung wird erst in den späteren Entwicklungsphasen vorgenommen, in denen die beiden Disziplinen, eventuell unterstützt durch einen mechanischen Entwicklungsprozess, parallel arbeiten. Eine solche Aufteilung ist wichtig, damit die Disziplinen einen gemeinsamen Systementwurf als Basis haben, um ein möglichst optimales Produkt zu entwickeln.

Derzeit ist bei der Entwicklung von eingebetteten Systemen und demzufolge in der Automobilindustrie vielfach noch festzustellen, dass die Systeme, oftmals noch traditionell bedingt, durch die Hardware geprägt werden [BMT11] [GLT03], wobei die Entwicklung eine deutliche Tendenz zu einem Gesamtsystemansatz erkennen lässt. In der Eng.3 Phase wird die Systemarchitektur erstellt. Diese Architektur ist die Grundlage für die folgenden darauf aufbauenden Entwicklungsphasen. Von daher muss die Systemarchitektur verständlich und vollständig spezifiziert und dokumentiert werden.

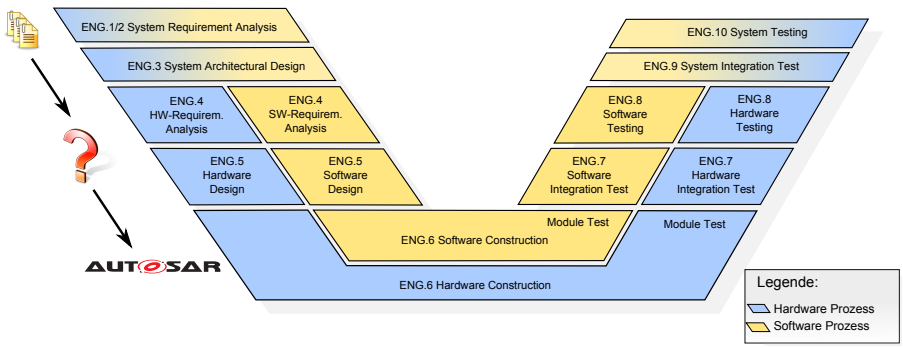


Abb. 2.30. Entwicklungsmodell nach Automotive SPICE [dAV10]

In der vierten Phase des Entwicklungsprozesses werden aus den Leistungsvorgaben und der Systemarchitektur die Anforderungen an die einzelnen Disziplinen abgeleitet. Es werden Anforderungen definiert, die durch die Hardware und andere, die von der Software erfüllt werden müssen. Diese werden aus den bereits existierenden Systemanforderungen abgeleitet, analysiert und ggf. noch erweitert. Daran anschließend werden die jeweiligen Architekturen und Designs in den unterschiedlichen Disziplinen erstellt. Für die Disziplin Software bedeutet dies, dass die Softwarearchitektur und das Feindesign für die darin befindlichen Komponenten erstellt werden. Das Feindesign bezieht sich

dabei auf die Struktur und das Verhalten der Komponenten. Im Anschluss wird der Code für die Komponenten erarbeitet. Hierzu kann der Code sowohl aus dem Modell heraus generiert oder aber manuell erstellt werden. Für die letzten beiden Phasen kann der AUTOSAR Standard eingesetzt werden. Die Verwendung des Standards wird von den Automobilherstellern immer öfter als Bedingung vorgegeben.

Zwischen dem Prozess der Anforderungsanalyse und der Verwendung des AUTOSAR Standards während des Designs- bzw. der Implementierungsprozesses entsteht aber ein großes Vakuum, das nicht methodisch abgedeckt ist (vgl. Abbildung 2.30). Der AUTOSAR Standard beinhaltet keine Techniken für eine Anforderungsanalyse [Gra11] und setzt gewisse Designentscheidungen bereits voraus. So beginnt AUTOSAR bereits mit der Erstellung der Softwarekomponenten. Die Aufteilung der Funktionalitäten und deren Dekomposition muss aber zunächst in der Analysephase erfolgt sein, um mit der Komponentenerstellung beginnen zu können.

Ein weiteres negatives Beispiel ist bei der funktionalen Analyse zu finden. So können im AUTOSAR Standard Daten im Speicher durch geeignete Schutzmaßnahmen abgesichert werden. Jedoch muss vorher bestimmt werden, welche Daten überhaupt sicherheitsrelevant sind und von daher geschützt werden müssen. Die Einteilung ergibt sich wiederum aus der Analysephase. Von daher ist es nicht möglich, diese oder ähnliche Designentscheidungen direkt aus den Anforderungen abzuleiten. Deswegen ist eine umfangreiche Analysephase mit unterstützenden Techniken und Methoden erforderlich.

Ferner muss neben der Architektur auch das Verhalten spezifiziert werden. Die Verhaltensmodellierung wird beim AUTOSAR Standard aber weitestgehend ausgeklammert. So sind weitere Techniken zu entwickeln und zu integrieren, um sowohl eine Analyse als auch eine Verhaltensspezifikation durchzuführen. Eine Lösung dieser Problemstellung wird in Form der AutoMoMe vorgestellt (siehe Kapitel 4).

2.9 Softwareentwicklung in der Steuergeräteentwicklung

Die Software in einem Automobil ist gekennzeichnet durch die Steuergeräteentwicklung. Bei der Steuergeräteentwicklung steht das Zusammenspiel zwischen den OEMs und den unterschiedlichen Automobilzulieferern maßgeblich im Vordergrund. Der OEM definiert die Gesamtfahrzeugarchitektur [BS05]. Die verschiedenen Teilsysteme der Gesamtarchitektur – in Form von Steuergeräten – werden von den unterschiedlichsten Zulieferern entwickelt. Diese können wiederum weitere Zulieferer beauftragen, Teilkomponenten zu entwickeln und zu liefern. Die Erstellung eines Steuergerätes ist Aufgabe eines Automobilzulieferers. Dieser muss die Anforderungen des OEMs sowohl funktional

als auch qualitätsbezogen realisieren. Zur Sicherstellung der Qualität eines Steuergerätes ist ein durchgängiger Entwicklungsprozess unverzichtbar. Gerade bei sicherheitskritischen Systemen, wie beispielsweise bei einem Airbag oder aber beim Anti-Blockier System (ABS), ist diese Vorgehensweise unter allen Umständen einzuhalten. Eine Beschreibung des Entwicklungsprozesses für Software im Automobilbereich ist in Kapitel 2.8 zu finden. Durch die Auslagerung der Komponentenentwicklung sind somit zwei Integrationsschritte bei der Steuergeräteentwicklung vorzufinden. Im ersten Integrationsschritt fügt der jeweilige Zulieferer die zugelieferten bzw. entwickelten Software- und Hardwarekomponenten zu einem Steuergerät zusammen. Anschließend integriert der OEM die Steuergeräte zu einer gesamten Fahrzeugarchitektur (siehe Abbildung 2.31).

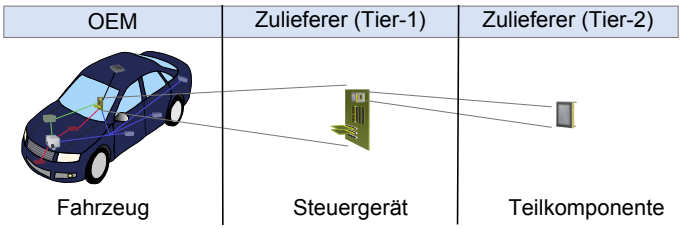


Abb. 2.31. Prozesskette in der Automobilindustrie

Derzeitig ist die Integration, sei es die Integration der Komponenten zu einem Steuergerät oder aber die Integration der unterschiedlichen Steuergeräte, oftmals diffizil, da das Kommunikationsverhalten oder die zeitlichen Anforderungen nicht immer korrekt umgesetzt werden. Bei einem fehlerhaften Verhalten kann die Integration sehr zeitaufwendig werden, da die Ursache nur schwerlich zu isolieren und somit die Fehlerquelle kaum zu identifizieren ist [BS05]. Erschwert wird die Fehlersuche zusätzlich, wenn mehrere Unternehmen an der Entwicklung des Steuergerätes beteiligt sind. Dies führt dann zu einer langen und kostspieligen Suche, die es möglichst zu vermeiden gilt. Von daher wird immer größerer Wert auf die Nutzung von Standardisierungen und modellbasierten Ansätzen, wie z. B. AUTOSAR, gelegt.

Die modellbasierten Ansätze mit ihren Modellierungssprachen sorgen durch eine höhere Abstraktion dafür, dass sie verständlicher sind [BMR12]. Hierdurch kann effizienter mit der Komplexität der Steuergeräte umgegangen, Eigenschaften besser dargestellt und verstanden werden. In der Abbildung 2.32 ist dargestellt, dass durch die Verwendung von höheren Programmiersprachen die Abstraktion bereits gestiegen ist [BST10]. Dabei sind die objektorientierten deutlich abstrakter als die funktionalen Sprachen anzusehen. Eine weitere Stufe der Abstraktion ergibt sich durch den Einsatz von Modellierungssprachen, wie beispielsweise der SysML oder der UML. Wegen der hohen Abstraktion ist es möglich, die Komplexität der heutigen Systeme in den Griff zu bekommen.

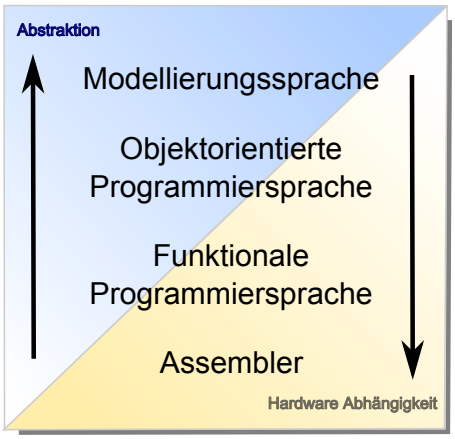


Abb. 2.32. Steigende Abstraktion der verschiedenen Notationsformen nach [Wei08]

Gleichwohl muss auf die exakte Einhaltung der Anforderungen, vor allem der qualitativen, bei der Entwicklung der Steuergeräte geachtet werden. Denn der Einsatz der standardisierten Schnittstellen ist nur dann vorteilhaft, wenn die qualitativen Anforderungen nicht verletzt werden. Die Umsetzung der qualitativen Anforderungen ist ein Aufgabenbereich der Architektur. Zur Sicherstellung, dass die qualitativen Anforderungen eingehalten werden, kann eine frühe Überprüfung der Architektur in Form einer Systemsimulation beitragen. Sie weist die Einhaltung der Anforderungen nach und fördert das Vertrauen in die Architektur [Sta09b].

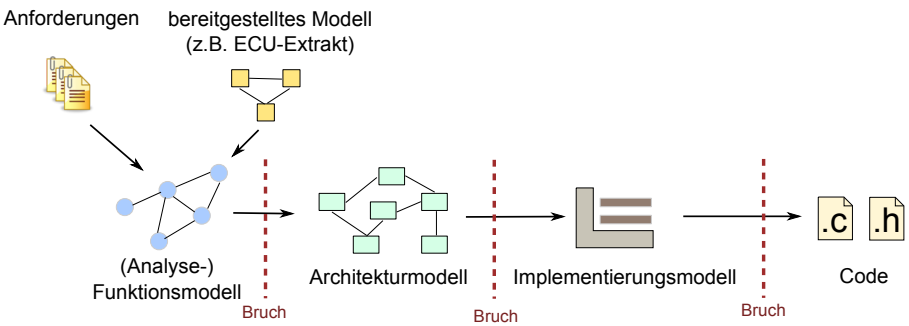


Abb. 2.33. Verschieden genutzte Modelle und fehlende Übergänge

Im Automobilbereich wird schon seit mehreren Jahren der Eindruck erweckt, dass eine durchgängige und komplett modellbasierte Softwareentwicklung stattfindet. Diese Aussage trifft jedoch nur zum Teil zu. Während innerhalb der Funktionsentwicklung zum überwiegenden Teil modellbasiert entwickelt und das Prinzip des Rapid Prototyping genutzt wird [BS05], findet vor allem auf

der Ebene der Architekturerstellung kein durchgängiger modellbasierter Entwicklungsprozess statt. Bei der Funktionsentwicklung werden Modelle für die Analyse, das Design und ebenso für die Verifikation in Form von Simulationen genutzt. Beispielhaft stehen für die modellbasierte Funktionsentwicklung die Werkzeuge Matlab/Simulink [Mat14] und TargetLink [dSP14]. Mit diesen lässt sich aber keine vollständige Architektur eines Steuergerätes modellieren. So fehlt diesen Werkzeugen eine Betriebssystemmodellierung bzw. die Verteilung von einzelnen Komponenten ist nur über Umwege oder gar nicht möglich. Von daher ist an dieser Stelle eine andere Modellierungsform zu wählen.

Bei der Funktionsentwicklung werden diverse Algorithmen entwickelt. Die einzelnen Algorithmen haben durch den eben beschriebenen modellbasierten Ansatz ein korrektes Verhalten. Jedoch ist ihr Zusammenspiel noch nicht verifiziert und es treten oft zeitliche Unstimmigkeiten auf [KM10a, WMK+09]. Eine ähnliche Vorgehensweise bezüglich der Modellierung und Absicherung auf Architekturebene ist jedoch nicht Stand der Technik und macht das derzeitige Defizit bei der Steuergeräteentwicklung offenkundig. Ein durchgängiger modellbasierter Prozess ist somit nicht vorhanden [BFH+10]. Vielmehr werden die Modelle nur in einzelnen Entwicklungsphasen angewandt (vgl. 2.33). Dementsprechend ist es nicht verwunderlich, dass für die einzelnen Modelle unterschiedliche Spezifikationssprachen und Techniken zum Einsatz kommen. Im Folgenden werden die bei der Softwareentwicklung von automobilen Steuergeräten verwendeten Ansätze und die eingesetzten Modellierungssprachen näher vorgestellt.

2.9.1 Der informale Ansatz

Der informale Ansatz ist heutzutage oft bei der Anforderungsspezifikation und teilweise bei der Systemarchitektur zu finden. Er erfolgt mittels natürlicher textueller Sprache und in Form von Blockschaltbildern, die jedoch keine einheitliche Syntax oder Semantik aufweisen. Die textuelle Sprache wird vornehmlich beim Anforderungsmanagement (Requirements Engineering (RE)) eingesetzt. Der Vorteil besteht darin, dass die Sprache für jedermann auch ohne Vorkenntnisse verständlich ist [PR09]. Darüber hinaus kann der Anforderungsentwickler seiner Kreativität freien Lauf lassen und die Anforderungen so beschreiben, wie es ihm beliebt. Hieraus ergibt sich eine große Vielzahl von Möglichkeiten, die Anforderungen zu formulieren. Die textuellen Anforderungen sind einfach zu erstellen, da sie von jedermann gut beherrscht werden [PR09]. Deshalb sind schnell erste Ergebnisse zu erzielen.

Neben den Vorteilen überwiegen jedoch die Schwachstellen. Mit der textuellen Sprache können zwar beliebige Anforderungen definiert werden. Zunächst einmal können durch die textuelle Beschreibung Mehrdeutigkeiten der Anforderungen entstehen [PR09]. Bei der Vielzahl der Anforderungen kann

die Konsistenz und Vollständigkeit nicht mehr gewährleistet werden. Bedingt durch die hohe Anzahl der Anforderungen ist nicht auszuschließen, dass sich Redundanzen einstellen, die schließlich zu Inkonsistenzen führen können [WW03]. Derzeit wird die Konsistenz und Vollständigkeit durch manuelle Reviews sichergestellt [MRZ+05]. Bei der Zunahme der Anforderungen und der Komplexität der zu entwickelnden Funktionen ist dies zukünftig nicht mehr realisierbar. Die textuellen Anforderungen können nicht automatisch überprüft oder weiterverarbeitet werden. Dies ist aber unerlässlich.

Es bestehen Arbeiten, die versuchen, die Nachteile des informalen Ansatzes zu beheben [Mut05, NT09]. Eine Überlegung ist die Verwendung von Satzmustern (Schablonen) für textuelle Anforderungen [KK06, KC02, Hol10, HMvD11, PMP11, Kap06]. Satzmuster ermöglichen eine automatisierte Überprüfung und zum Teil einen Übergang in eine erste initiale Systemarchitektur. Jedoch lässt sich der textuelle Ansatz in den anschließenden Entwicklungsphasen nicht weiter verwenden, da die verschiedenen Detaillierungsgrade nur schwer zu realisieren sind [Mut05]. Darüber hinaus sind keine Möglichkeiten vorhanden, das Verhalten der Anforderungen zu simulieren bzw. formal zu verifizieren. Deshalb ist der Einsatz von (semi-)formalen grafischen Notationen in den späteren Entwicklungsphasen erforderlich.

Falls bei der Systemarchitektur der informale Ansatz zum Einsatz gelangt, werden vielfach Blockschaltbilder mit zusätzlichen textuellen Erläuterungen genutzt. Die Erstellung der Blockschaltbilder erfolgt häufig mit den Werkzeugen Powerpoint [Mic14a] oder Visio [Mic14b]. Der Vorteil dieser Methode besteht in der schnellen, übersichtlichen und einfachen Entwicklung von Systemübersichten. Die erstellten Bilder können bei einer Entscheidungsfindung auf Managementebene noch nützlich sein.

Als nachteilig muss aber herausgestellt werden, dass mit den Blockschaltbildern keine vollständige Spezifikation bzw. keine koordinierte Entwicklung realisiert werden kann. Hauptnachteil ist die fehlende Semantik der Blockschaltbilder. So können die Diagramme häufig anders interpretiert werden. Es fehlt an Informationen, wie beispielsweise einer formalen Schnittstellenbeschreibung und an Aussagen zur Vorgabe des Kommunikationsverhaltens. Diese Informationen sind aber für eine Architektur unverzichtbar. Besonders bei der Systemarchitektur wird die fehlende Semantik zu einem Problem, da an dieser Stelle mehrere Disziplinen zusammenarbeiten und von daher ist eine eindeutige und vollständige Spezifikation erforderlich [KSW13].

Durch die Zunahme der Anforderungen und Funktionen und der damit verbundenen Komplexität wird es immer schwieriger und aufwendiger, die informalen Ansätze weiter zu verwenden. So ist die Vermeidung von Inkonsistenzen und Missverständnissen bei der Entwicklung nur schwerlich und wenn nur mit verhältnismäßig großem Aufwand möglich. Von daher wird nach Alternativen gesucht. Eine Alternative ist der modellbasierte Ansatz. Dieser beinhaltet

mehrere Vorteile und setzt sich auch in der Domäne der eingebetteten Systeme immer mehr durch.

2.9.2 Der modellbasierte Ansatz

Neben dem informalen existieren noch weitere Ansätze. Einer davon ist der modellbasierte Ansatz. Der Kern dieses Ansatzes bildet das Modell, das zusätzlich als Kommunikationsmittel innerhalb des Entwicklungsteams genutzt wird [CR11]. Für ein besseres Verständnis ist zunächst ist der Begriff eines Modells zu definieren.

Definition (Modell). Ein Modell ist ein Abbild von einem Gegenstand bzw. eines Systems, bei dem von unwichtigen Details abstrahiert wird, damit auf die wesentlichen Details für die Entwicklung fokussiert werden kann [CS06]. Ein Modell kann dabei aus verschiedenen Modellen zusammengesetzt sein. Hierdurch besteht die Möglichkeit, aus verschiedenen Sichten das Modell zu betrachten [Sta73].

Die modellbasierten Notationsformen werden dabei in funktions-, verhaltens- und objektorientierte Ansätze aufgeteilt. Diese werden nachfolgend vorgestellt.

Funktions- und verhaltensorientierter Ansatz

Bei der Modellierung kamen in der Vergangenheit häufig funktionsorientierte Notationsformen bei der Steuergeräteentwicklung zum Einsatz. Kennzeichen dieser Sprachen ist die Aufteilung des Systems in einzelne, klar definierte Funktionen, die untereinander interagieren [Mut05]. Sowohl Zustand als auch Daten des Systems werden an zentraler Stelle beispielsweise in Form eines Wörterbuchs (Data Dictonary) abgelegt. Dieser Ansatz findet sich in *Strukturierte Analyse/Strukturiertes Design* (Strukturierte Analyse (SA)/Strukturiertes Design (SD)) [DeM79] und Structured Analysis and Design Technique (Structured Analysis and Design Technique (SADT)) [Ros77] wieder. Durch diese Spezifikationssprache sind die Informationsflüsse zwischen den Funktionen, die aus den Anforderungen abgeleitet werden, darstellbar. Es hat sich aber gezeigt, dass diese Modellierungssprachen sehr gut für die Analyse eines Systems geeignet sind. Jedoch haben sie Nachteile bei der Beschreibung oder Spezifikation einer Architektur. So sind beispielsweise keine Schnittstellen modellierbar. Dies schränkt die Sprachen ein und erschwert die Modellierung der komplexen Steuergeräte mit ihren Funktionen bzw. die Verbindung mit dem AUTOSAR Ansatz ist nicht ohne weiteres zu realisieren. Es existieren aber Arbeiten, die eine Überführung dieser Modelle in objektorientierte Ansätze erlauben [Fri06], damit die bisherigen Modelle weiter verwendet werden können.

Der objektorientierte Ansatz

Der objektorientierte Ansatz erstellt ein Modell basierend auf Objekten mit genau definierten Schnittstellen, die miteinander kommunizieren und gegenseitig ihre Dienste aufrufen [Som07]. Ein Objekt wird dabei wie folgt definiert:

> **Definition (Objekt).** Ein Objekt ist ein individuelles Exemplar von Dingen, Personen oder Begriffen der realen Welt. Es besitzt ein eindeutiges Verhalten, bestehend aus Operationen und Zuständen und hat eine eindeutige Identität [Mut05, CS06, Som07].

Ein Objekt ist die Instanz einer Klasse. Eine Klasse beschreibt die Struktur und das Verhalten einer Menge gleichartiger Objekte [Som07]. Dies erfolgt durch Attribute und Operationen. Die wichtigsten Prinzipien bei den objektorientierten Sprachen, und damit auch die Abgrenzung zu den anderen Ansätzen, sind die Kapselung, die Vererbung und das Überladen (Polymorphie). Unter dem Prinzip der Kapselung ist zu verstehen, dass die innere Struktur einer Klasse bzw. eines Objektes nach außen hin verborgen bleibt. Der Zugriff erfolgt über vorweg definierte Schnittstellen. Das Prinzip der Vererbung bedeutet, dass Daten und Verhalten in den vererbenden Klassen weiterverwendet werden können. Die vererbenden Klassen sind dabei spezielle Klassen der übergeordneten. Sie überschreiben die Daten bzw. das Verhalten, das sich geändert hat. Die nicht überschriebenen Daten bzw. das Verhalten wird übernommen. Eng verwandt hiermit ist das Prinzip des Überladens. Dieses Prinzip ermöglicht das Ausführen von gleichnamigen Operationen mit unterschiedlichen Daten.

Es ist eine Vielzahl von objektorientierten Sprachen auf dem Markt vorhanden [RH94]. Vor allem zu Beginn der 90iger Jahre des letzten Jahrhunderts gab es sehr viele unterschiedliche objekt-orientierte Ansätze [Mut05]. Zum Teil wurden sie in der UML bzw. in deren Abbild in der Systemwelt (SysML) zusammengefasst. Bei der Entwicklung von informationsbasierten Systemen hat sich die objekt-orientierte Spezifikation und Programmierung bereits als Standard etabliert.

Bei der Entwicklung von eingebetteten Systemen wird fast ausschließlich die Programmiersprache C genutzt [BST10, Bor08]. Hieran wird sich auf Grund der Ressourcenknappheit bei den Steuergeräten auch so schnell nichts ändern. Gleichwohl ist ein neuer Trend bei der Modellierung von eingebetteten Systemen zu erkennen. Auf die objektorientierten Spezifikationssprachen wird immer mehr zurückgegriffen und zwar vornehmlich in Form der SysML und der UML. Diese Modellierung wird bei der Codegenerierung dann auf die Programmiersprache C abgebildet. Von daher ist die Nutzung der Programmiersprache C kein Ausschluss für eine objektorientierte Modellierung. Vorteilhaft bei der objektorientierten Modellierung ist die Fähigkeit, die Komplexität des

zu entwickelnden Systems beherrschbar zu machen. Die Vorzüge kommen aber erst dann voll zur Geltung, wenn ein durchgängiger und nicht unterbrochener Entwicklungsprozess vorliegt, d. h. von den Anforderungen bis hin zur Implementierung.

Geh nicht immer auf dem vorgezeichneten Weg, der nur dahin führt, wo andere bereits gegangen sind.

Alexander Graham Bell (1847 - 1922)

3 Stand der Technik

Zum besseren Verständnis der verwendeten Methoden, Techniken und der Lösungsansätze in AutoMoMe ist ein Überblick über den gegenwärtigen Stand der Technik bei der Softwareentwicklung für automobile Steuergeräte bzw. von eingebetteten Systemen angezeigt. Der Fokus liegt auf vergleichbare Arbeiten und Forschungsprojekte, die ebenfalls ihren Schwerpunkt auf die Verbesserung der automobilen Steuergeräteentwicklung gelegt haben. Somit ist ein Vergleich mit dem hier vorgestellten Ansatz AutoMoMe möglich. Von daher werden in diesem Kapitel nur generelle Ansätze vorgestellt. Der Stand der Technik für einige spezielle Methoden (beispielsweise Modelltransformationen mit dem AUTOSAR Standard) wird direkt in den jeweiligen Kapiteln beschrieben, um sie unmittelbar mit der entwickelten Lösung von AutoMoMe zu vergleichen.

3.1 Das Projekt STrukturierter EntwicklungsProzess (Step-X)

Das Projekt Step-X wurde von der Volkswagen AG und der Technischen Universität Braunschweig im Jahr 2004 gestartet. Projektziel war die Erstellung eines durchgängig modellbasierten Entwicklungsprozesses für den Automobilbereich. Dabei wurden alle Entwicklungsphasen von der Anforderungs- bis hin zur Testphase einbezogen. Schwerpunkte waren die grafische Spezifikation vor allem der Anforderungen und der Testfälle. Dazu wurde ein sogenanntes digitales Lastenheft entworfen. In diesem Lastenheft wurden die Anforderungen durch die Notationsform UML auf (semi-)formale Art und Weise spezifiziert. Die Anforderungen wurden dann in der Testphase verifiziert. Da für die Codegenerierung für Mikrocontroller die UML Werkzeuge noch nicht ausreichend optimiert waren, wurde für die Implementierung auf die Werkzeuge ASCED SD [Gro14] und Matlab/Simulink [Mat14] zurückgegriffen [EHV+03]. Zwischen den beiden Arbeitsschwerpunkten Anforderungs- und Testmodellierung

wurde die Diagnose und die Toolkopplung der eingesetzten Werkzeuge schwerpunktmäßig im Projekt bearbeitet (vgl. Abbildung 3.1) [EHV+03, VHG+04]. Ebenso wurden erste Maßnahmen zur Sicherstellung der Software-Qualität integriert [Mut05].

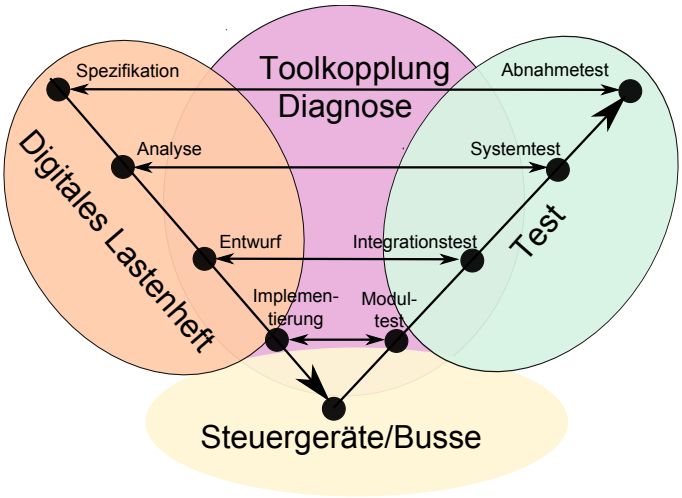


Abb. 3.1. Verschiedene Schwerpunkte im Step-X Projekt nach [Mut05]

Ein Ergebnis dieses Projektes bestand in der Feststellung, dass der durchgängig modellbasierte Entwicklungsprozess für die Automobilentwicklung einsetzbar ist. Darüber hinaus hat sich gezeigt, dass die Projektergebnisse vielversprechender gegenüber dem bisherigen informalen Ansatz sind [VHG+04]. Vor allem die formale Spezifikation der Anforderungen und die damit einhergehende Konsistenzgewährleistung, sowie die enge Integration der Diagnose und des Testens im Entwicklungsprozess, wurden als Vorteile herausgestellt [VHG+04].

Das Projekt hat gezeigt, dass ein durchgängiger Entwicklungsprozess mittels der UML durchführbar ist. Zu berücksichtigen ist jedoch, dass die Schwerpunkte bei der Formalisierung der Anforderungen und im Testbereich lagen. Die Architektur wurde nur am Rande mit einbezogen (siehe Kapitel 2.6.3). Der Modellierung von Echtzeit und der frühen zeitlichen Analyse der Architektur wurden in dem Projekt keine Beachtung geschenkt (siehe Kapitel 2.6.4). Auch konnte der neue AUTOSAR Standard nicht einbezogen werden. Von daher können die Erkenntnisse des Projektes nur als Startpunkt für die weiterführende Modellierung in AutoMoMe genutzt werden.

3.2 Weitere SysML und UML Ansätze

In der Literatur existieren verschiedene SysML und UML-basierte Ansätze für die Modellierung innerhalb der Automobilindustrie. Einer dieser Ansätze ist die Automotive Modeling Language (AML) [RBvdBS02]. Sie ist eine Modellierungssprache basierend auf der UML, die im Rahmen des Forschungsprojektes FORSOFT entstanden ist. Die Modellierungssprache definiert dabei verschiedene Abstraktionsstufen, um sowohl funktionale als auch technische Architekturen abbilden zu können. Hierfür wurde ein eigenes Metamodell entwickelt [RBvdBS02]. Mithilfe des Modells lassen sich insbesondere Funktionsnetzwerke beschreiben und darstellen. Jedoch werden keine Anpassungen vorgenommen, um die Modellierung von Echtzeit und der technischen Architektur zu verbessern (siehe Kapitel 2.6.4). Ebenso spielt der AUTOSAR Standard bei der Automotive Modeling Language keine Rolle, da diese entwickelt wurde, bevor der AUTOSAR Standard in der Entwicklung Einzug hielt (siehe Kapitel 2.7). Daher ist dieser Ansatz für die heutige Steuergeräteentwicklung nicht mehr ausreichend. Es ist also festzuhalten, dass die AML nur einen Teil der notwendigen Anforderungen an eine heutige automobile Entwicklungsmethodik umsetzt. Von daher sind weitergehende Ansätze notwendig.

Ein anderer auf SysML basierter Ansatz ist in der *SysCARS* Methodik [PA12] beschrieben. In dem Ansatz wird die SysML durch ein Profil und Stereotypen erweitert. Dies ist vergleichbar mit dem Ansatz AutoMoMe. Jedoch liegt in der *SysCARS* Methodik ein Schwerpunkt auf der Dokumentation des Systems. So wurde die Zielsetzung auf die Generierung von Reports und die Anbindung an Matlab/Simulink [Mat14] gelegt. Es wird jedoch keine Integration einer Echtzeitanalyse oder die Anbindung an den AUTOSAR Standard vorgenommen (vgl. Kapitel 2.6.4 und 2.7). Von daher ist der *SysCARS* Ansatz eine Methodik zur Erfüllung der Sicherheitsnorm. Er kann bei der Erstellung eines AUTOSAR Systems nur bedingt hilfreich sein.

3.3 EAST-ADL2

Das Projekt EAST-ADL2 ist ein europäisches Forschungsprojekt, in dem eine eigene Architektursprache entwickelt wurde, die innerhalb der automobilen Entwicklung genutzt werden kann. Sie wird mittlerweile von einem Konsortium weiter entwickelt [Ass13]. Die entstandene Architektursprache (ADL) vereint Konzepte der generellen Sprachen UML und SysML und passt diese dem Automobilbereich an [CFJ+09, CCG+07]. Ferner wird der AUTOSAR Standard in der Implementierungsphase eingesetzt. Somit werden auch Konzepte dieses Standards genutzt. Das EAST-ADL2 Projekt ist die Fortführung des Projektes

EAST-ADL und ergänzt die entwickelte Sprache um Techniken und Methoden für die Varianten- und Sicherheitsmodellierung. Außerdem wurde mit dem Timmo-Projekt (siehe Kapitel 3.10) eine Erweiterung der Architektursprache für die Spezifikation von Zeiten während der Entwicklung erarbeitet [Sei09a].

Im Fokus des Projektes steht die Modellierung eines Systems in der Automobilbranche. Dabei wird besonders auf die Einteilung in verschiedene Schichten und auf die Variantenmodellierung Wert gelegt (vgl. Abbildung 3.2) [LKM+12]. Diese erlauben die Modellierung von den Anforderungen bis hin zur Implementierung. Die einzelnen Entwicklungsartefakte werden den jeweiligen Schichten zugeordnet. Es ist anzumerken, dass in der Implementierungsphase auf den AUTOSAR Standard zurückgegriffen wird. Der Übergang zwischen der Design- und der Implementierungsphase wird in der EAST-ADL2 nur knapp beschrieben [Ass13]. So werden die Artefakte größtenteils auf Komponenten bzw. Runnables abgebildet. In [QCLT11] sind weitergehende Transformationsregeln beschrieben, jedoch wird darauf hingewiesen, dass beispielsweise die Zuweisung zu Betriebssystemeigenschaften nicht transformiert werden kann (siehe Kapitel 2.6.5). Von daher existieren für die automobile Steuergeräteentwicklung noch Defizite beim Übergang zwischen dem Design und der Implementierung.

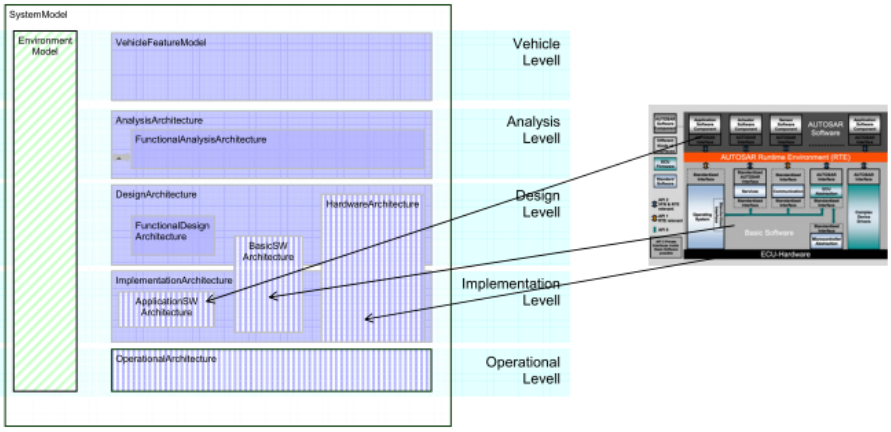


Abb. 3.2. EAST-ADL2 Methodik [Ass13]

Die EAST-ADL2 ist eine an die Automobilbranche angepasste Sprache. Sie beinhaltet bereits Konzepte und Methoden, die für die Modellierung genutzt werden. Nachteile sind aber existent. So ist derzeit nur eine Werkzeugunterstützung in Form eines Prototyps [EtSTA10, QTP+11] vorhanden und es wird gerade an einer Umsetzung in Eclipse gearbeitet [EAT14]. Deswegen ist es schwierig, sie in der industriellen Praxis einzusetzen. Ferner ist der Übergang zwischen dem Design und der Implementierung in Form von AUTOSAR nur

zum Teil definiert. Dort sind noch Defizite vorhanden, die geschlossen werden müssen. Wesentlich ist jedoch, dass die Sprache hauptsächlich für die Modellierung eines Automobils und nicht für ein Steuergerät konzipiert wurde. So wurde der Bereich der Verhaltensmodellierung nur am Rande betrachtet und es wird auf andere Modellierungsformen referenziert [Ass13, FCLT11] (vgl. Kapitel 2.6.4). Mit der Folge, dass die EAST-ADL vornehmlich von OEMs zur Strukturmodellierung eingesetzt werden kann und weniger von Zulieferern bei der Entwicklung von vollständigen Steuergeräten. Ein Beispiel für die Defizite bei der Steuergeräteentwicklung ist die Verwendung der vom AUTOSAR Standard definierten Schnittstellen für die Basis-Software (siehe Kapitel 2.7). Sie können nicht ohne weiteres genutzt werden. Von daher bedarf es noch weiterer Methoden und Techniken, um diese Implementierung zu unterstützen und zu verbessern.

3.4 EDONA

In Frankreich wurde das Projekt *Environnements de Dèveloppement ouverts aux Normes de l' Automobile*, kurz EDONA genannt [Pro11], durchgeführt. Das Projekt beschäftigt sich ebenso wie diese Arbeit mit dem Ziel einer durchgängigen Entwicklungsmethodik für die Automobilentwicklung. Dabei baut das Projekt auf die beiden Standards EAST-ADL2 (siehe Kapitel 3.3) und AUTOSAR (siehe Kapitel 2.7) auf. Der Übergang zwischen diesen beiden Standards wird im Projekt EDONA durch Modelltransformationen durchgeführt. Dabei wird die Analyse und die Modellierung der logischen Funktionen durch die EAST-ADL2 modelliert, während die technische Sicht und die Implementierung in AUTOSAR erfolgt [AQST10]. Somit werden, wie bereits beim Projekt EAST-ADL2 vorgeschlagen, Funktionen den Runnables oder Atomic Softwarekomponenten (Software component (SWC)) zugeordnet. Die Basis-Software oder weitere Zuordnungen werden nicht weiter betrachtet. Wie hier ein Übergang erfolgen soll, wird nicht näher spezifiziert.

Bei dem Design wird im EDONA Projekt die Modellierung des Betriebssystems einbezogen. Ein Standard Betriebssystem, wie z. B. das OSEK Betriebssystem [OSE05], wird jedoch nicht genutzt. Vielmehr wird auf ein eigenes Betriebssystem zurückgegriffen und zwar das sogenannte PharOS [OT10]. Die Eigenschaften und Fähigkeiten sind jedoch eingeschränkt, damit die Aussage „Correct by Construction“ zutrifft [OT10]. Von daher können die Ergebnisse und Erkenntnisse des Projektes nicht ohne weiteres auf andere Steuergeräte übertragen werden.

Des Weiteren sind in dem Projekt keine Erweiterungen zur Modellierung beim zeitlichen Verhalten vorgesehen (siehe Kapitel 2.6.4). Diese Vorgabe ist aber für die Modellierung und Beherrschung von komplexen Steuergeräten erforderlich. Für die Spezifizierung von komplexen und verteilten Systemen ist dieses

Projekt daher nur bedingt geeignet. Der Ansatz AutoMoMe stellt einen vergleichbaren, aber weitergehenden Ansatz dar.

3.5 CAMoS

Das Projekt *Code Generation for Automotive Real-Time Applications based on modeldriven Simulation* (CAMoS) ist ein Teilprojekt der Fraunhofer-Einrichtung für Systeme der Kommunikationstechnik (ESK). Es beschäftigt sich mit der Erstellung eines durchgängigen Entwicklungsprozesses für die Steuergeräteentwicklung im Automobilbereich. Von daher besitzt das CAMoS Projekt eine vergleichbare Zielsetzung wie der Ansatz AutoMoMe. Als Standardmethode wird in dem Projekt die UML eingesetzt. Ein herausragendes Merkmal des Projektes ist die Modellierung der Kommunikationsmechanismen. Sie werden in den einzelnen Schritten des Entwicklungsprozesses in geeigneter Abstraktionsform modelliert und getestet [LB08]. So wird der Kommunikationsfluss, exemplarisch kann hier der CAN-Bus genannt werden, zunächst mittels der UML in Form einer Kommunikationsverbindung spezifiziert. Dies wird in den nachgelagerten Entwicklungsschritten durch Schnittstellen weiter konkretisiert. Anschließend kann die Kommunikation simuliert oder sogar auf entsprechender Hardware ausgeführt und getestet werden.

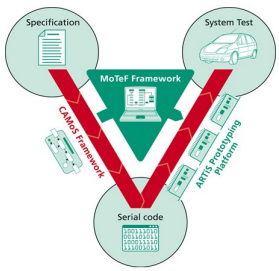


Abb. 3.3. Übersicht zu den Projekten CAMoS, ARTiS und MoTeF [LB08]

Eng verwandt mit dem CAMoS Projekt sind die Projekte *Automotive Real Time Prototyping System* (ARTiS) und *Modellbasierte Testfallerstellung* (MoTeF). Die Projekte zusammen ergeben einen durchgängigen Entwicklungsprozess (vgl. Abbildung 3.3). Ziel des ARTiS Projektes ist die Prototypenentwicklung von Steuergerätefunktionen. Dazu wurde im Projekt eine Zielplattform entwickelt, auf der eine zielsystemnahe Evaluierung neuer Fahrzeugfunktionen möglich ist [LB08]. Ein Aspekt hierbei ist wiederum die Kommunikation zu anderen Steuergeräten.

Das dritte Projekt, das zu den beiden vorgenannten Projekten zu zählen ist, ist das MoTeF Projekt. Dies Projekt ist für die automatische Testfallgenerierung aus dem Modell heraus zuständig. So können aus dem UML Modell des CAMoS Projektes Testfälle für den System- bzw. Integrationstest automatisch generiert werden. Dies ist eine entscheidende Hilfe, den Aufwand für die qualitätssichernden Prozesse zu reduzieren.

Die drei Projekte zusammen ergeben einen durchgängigen Entwicklungsprozess. Der Schwerpunkt liegt in den drei Bereichen: Prototyperstel-

lung, Kommunikation und Test. Die Bereiche der Modellierung von Betriebssystemeigenschaften und des zeitlichen Verhaltens, sowie der Übergang zur Softwarearchitektur in AUTOSAR, bleiben in diesen Projekten jedoch außer Betracht (vgl. Kapitel 2.6.5, 2.6.4, 2.7). In diesen Bereichen besitzt der Ansatz AutoMoMe Vorteile. Die Bereiche Test und Kommunikationssimulation werden dagegen beim Ansatz AutoMoMe nicht betrachtet. Von daher sind einige Unterschiede zu erkennen. Da die drei Projekte auf UML basieren, ist es durchaus realistisch, dass der Ansatz AutoMoMe um diese weiterführenden Aspekte ergänzt werden kann.

3.6 AxBench

Die *AUTOSAR extensible Workbench* (AxBench) [AxB11] ist ein Werkzeug für AUTOSAR-konforme Software. Das Projekt wurde am Fraunhofer Institut ISST für eine verbesserte AUTOSAR Entwicklung entwickelt. Durch Nutzung der AxBench können der zeitliche Aufwand und die Entwicklungskosten gesenkt werden. Dies wird durch die vier Funktionen Modellierung von Systemfamilien, Simulation von Anwendungskomponenten, Systemgenerierung zur Erstellung von unterschiedlichen Konfigurationen und zum Schluss die Systemoptimierung durch eine Systembewertung erreicht, die in der AxBench adressiert werden [Har11].

Die Modellierung erfolgt mittels der eigens entwickelte textbasierte Architekturbeschreibungssprache AxLang [GR11]. In dieser Sprache wird die Architektur mit der Verteilung der Softwarekomponenten auf die Ressourcen des Systems, sowie mögliche Varianten, spezifiziert. Bei der Simulation werden frühzeitig die Softwarekomponenten anhand ihrer Funktionsflüsse kontrolliert, so dass die Kompatibilität sichergestellt werden kann. Hieraus werden unterschiedliche Verteilungen der Applikations-Software auf die Hardware erstellt. Diese können mittels Bewertungskomponenten beurteilt werden. Die Ressourcenauslastung kann als Beispiel angeführt werden.

Der Ansatz zeigt wie die AUTOSAR Entwicklung mit einer Simulation verknüpft werden kann, um ein System zu optimieren. Jedoch wird nicht der Gesamtansatz in den Fokus gestellt, d.h. für die Analysephasen sind keine Entwicklungstechniken vorhanden. Mögliche Brüche im Entwicklungsprozess werden somit nicht behoben. Ferner adressiert das Projekt AxBench nicht die Modellierung von zeitlichen Informationen (siehe Kapitel 2.6.4). Hier bietet AutoMoMe einen Ansatz für zusätzliche Entwicklungsmethoden, um eine durchgängige Entwicklung zu gewährleisten.

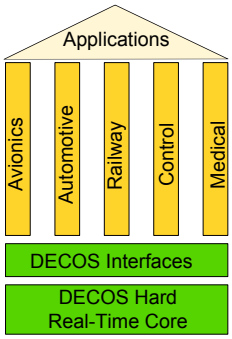


Abb. 3.4. Übersicht über das DECOS Projekt [DEC11]

3.7 DECOS

Das Projekt Dependable Embedded Component and Systems (DECOS) [DEC11] war ein europäisches Projekt (2004 - 2007) und verfolgte die Zielsetzung eine grundlegende Middleware Technologie für integrierte und verteilte sicherheitskritische Architekturen zu realisieren [Buh07]. Das Projekt sollte Hindernisse bei der Entwicklung von hochzuverlässigen eingebetteten Systemen ausräumen. Dabei wurden unterschiedliche Domänen betrachtet, in denen sicherheitskritische Systeme zum Einsatz kommen (vgl. Abbildung 3.4). Es wurde vor allem untersucht, mehrere Funktionen auf einem Steuergerät zu integrieren. Gleichzeitig wurde eine Verbesserung der Diagnose und Wartung vorgenommen. Ziel des Projektes war es, die Kosten zu minimieren und die Zuverlässigkeit zu maximieren.

Kern des Projektes war die Verwendung einer Time-Triggered Architektur (TTA) und die Nutzung von komponenten-basiertem Design. In der Abbildung 3.5 wird die verwendete Werkzeugkette inklusive der Aufteilung in Struktur- und Verhaltensdiagrammen sichtbar. Ebenso ist in der DECOS Werkzeugkette die Aufteilung in plattformunabhängige und -abhängige Modelle zu erkennen. Aus ihnen wird durch Konfiguration anschließend der Code generiert und auf die Hardware verteilt.

Das DECOS Projekt hat einige Resultate für die zukünftige Entwicklung von eingebetteten Systemen erarbeitet. Jedoch geht es von einigen Voraussetzungen aus, die derzeit nicht erfüllbar sind. Innerhalb des DECOS Projektes wurde eine Time-Triggered Architektur eingesetzt. Die AUTOSAR Architektur ist aber keine TTA. Außerdem entspricht die RTE nicht der im DECOS Projekt erarbeiteten Middleware. Diese bietet mehrere Dienste an, die für die Verteilung genutzt werden. Von daher kann das Vorgehen im DECOS Projekt nicht ohne weiteres auf ein derzeitiges Steuergeräteentwicklungsprojekt übertragen werden.

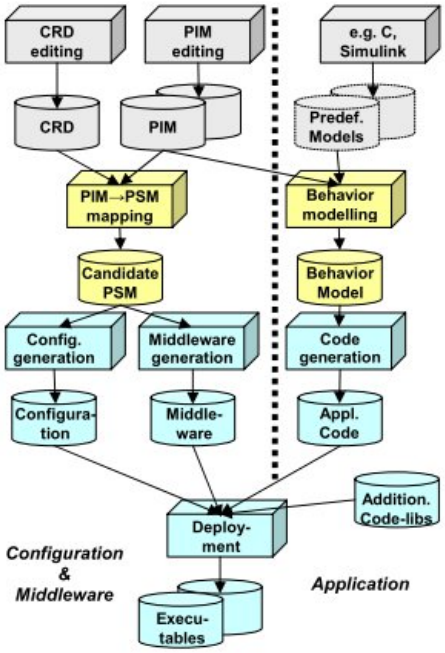


Abb. 3.5. DECOS Werkzeugkette nach [HSS+07]

3.8 AutoFOCUS

Das Projekt AutoFocus ist ein Forschungsprojekt der Technischen Universität München (TUM) [HSF+09, HLP+10, FHP+05]. Es befindet sich derzeit in der 3. Version (AutoFocus3) und bildete auch den Grundstein für das AutoMoDe Forschungsprojekt [BBRS05, ZBF+05]. Ziel des Projektes ist die Modellierung und Analyse der Struktur und des Verhaltens von verteilten, reaktiven und zeitlich gesteuerten Computersystemen [HSF+09]. Für die Analyse ist eine Simulation des Verhaltens als Technik im Projekt eingebunden. Für einzelne kleine besonders kritische Teile ist außerdem eine formale Verifikation vorgesehen. Diese kann auf Grund der Komplexität jedoch nicht auf das gesamte System ausgedehnt werden. Aus dieser Zielsetzung ergibt sich, dass die in diesem Projekt angewandten Techniken und Methoden ebenfalls für die Steuergeräteentwicklung im Automobilbereich verwendet werden können.

Im AutoFocus Projekt werden drei verschiedene Sichten unterschieden (vgl. Abbildung 3.6). Dies sind die *Functional Architecture*, die *Logical Architecture*, sowie die *Technical Architecture* [HLP+10]. Die Functional Architecture konzentriert sich auf das für den Benutzer sichtbare Verhalten des Systems. Dies kann bei einem Fahrzeug beispielsweise die Steuerung des Lichtes sein. Die Logical Architecture spezifiziert die für das Verhalten notwendigen Funktionen

mittels Komponenten. Hierdurch wird die Software des Systems beschrieben. In der Technical Architecture wird die Hardware des Systems abgebildet. Dies bedeutet, dass die Rechenkerne, die Ein- und Ausgabegeräte (Aktoren und Sensoren) sowie die sie verbindenen Netzwerke (Busse) modelliert werden. Innerhalb der Technical Architecture wird die Verteilung der Komponenten auf die einzelnen Rechenkerne vorgenommen. Die Einteilung in verschiedene Sichten ist im Bereich der Entwicklung von eingebetteten Systemen Standard.

Ein Nachteil im AutoFocus Projekt ist darin zu sehen, dass nicht auf bestehende Standards zurückgegriffen wird, sondern es wird ein eigenes Vorgehen und eine eigene Semantik definiert. Es besteht ein hoher Einarbeitungsaufwand für Außenstehende [Wat07, HLP+10]. Ferner ist nur die werkzeugseitige Unterstützung durch einen Prototypen realisiert. Hierdurch ist ein Einsatz im industriellen Bereich nur bedingt möglich. Beim Einsatz im Automobilbereich ist ferner die Beziehung zwischen Zulieferer und OEM noch zusätzlich in Betracht zu ziehen. Zwischen beiden werden Entwicklungsartefakte ausgetauscht und müssen von beiden Seiten gleich interpretiert werden. Aus diesem Grund ist die Nutzung von international anerkannten Standards sinnvoller und ratsamer.

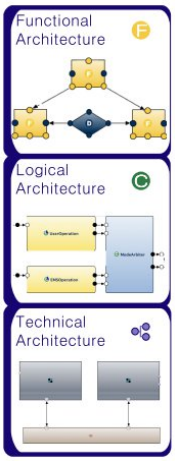


Abb. 3.6. Verschiedenen Sichten im Projekt AutoFOCUS [HLP+10]

Ein weiterer Punkt, der in AutoMoMe adressiert wird, ist der Übergang nach AUTOSAR und damit die Herstellung eines durchgängigen Entwicklungsprozesses (siehe Kapitel 2.7). Diese Thematik bleibt in AutoFocus außer Betracht [HLP+10]. Von daher gibt es zwar eine ähnliche Ausgangslage und Zielsetzung zwischen den beiden Ansätzen. Jedoch gibt es bei AutoFocus Nachteile, die die Erarbeitung einer Entwicklungsmethodik basierend auf UML notwendig und ratsam erscheinen lassen.

3.9 PreeVision

Eine weitere domänenspezifische Methode im Automobilbereich ist die im Werkzeug PreeVision implementierte Technik. Das Werkzeug PreeVision ist ursprünglich ein Architekturwerkzeug der Firma Aquintos[1] und wird nun von

[1] http://www.aquintos.de/

der Firma Vector[2] vertrieben. Die Methode dient zur modellbasierten Entwicklung und Bewertung von Elektrik/Elektronik Architekturen im Automobilbereich [Gmb14]. Dabei besitzt sie ein eigenes domänenspezifsches Metamodell. Aus dem Domänenmodell werden Modelle in Form einer grafischen Sprache und Tabellen erzeugt. In dem Modell werden die Topologie, das Bordnetz, die Stromversorgung und die Vernetzungs- und Funktionsverteilung eines Fahrzeugs spezifiziert [KR07]. Besonders werden mögliche Varianten und die Buskommunikation herausgearbeitet.

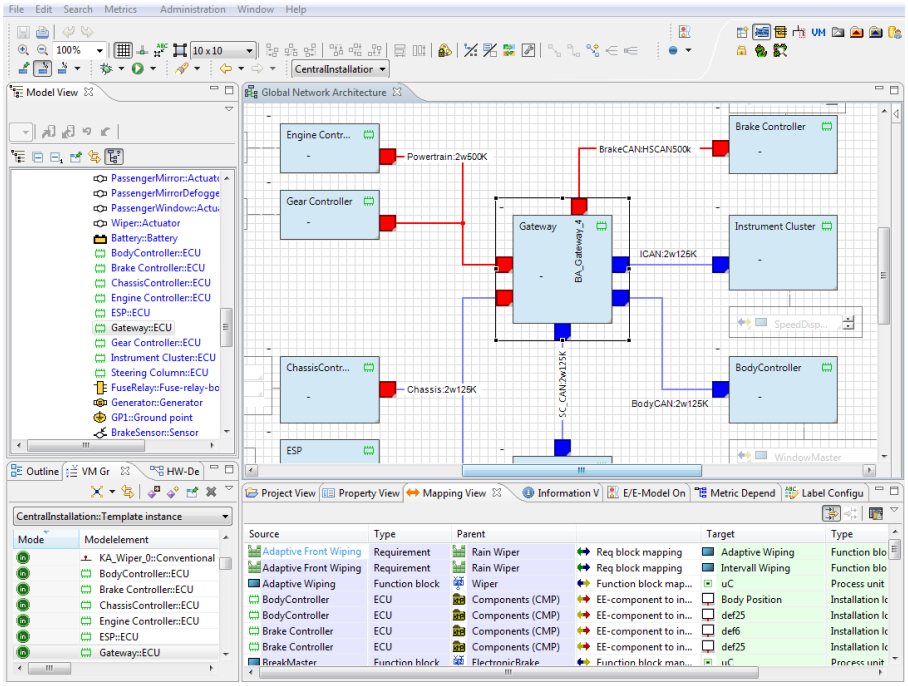


Abb. 3.7. Beispielarchitektur mit der Methode PreeVision [Gmb14]

Durch das domänenspezifische Metamodell ist die Modellierung einer Fahrzeugarchitektur mit PreeVision möglich (vgl. Abbildung 3.7). Jedoch kann die Methode hauptsächlich nur für die Gesamtfahrzeugarchitektur angewandt werden, da die Kabelbäume und der Ort der Verbauung der Steuergeräte für einen Zulieferer keine Bedeutung hat. Daher wird sie vielfach nur von den OEMs genutzt. Bei der Entwicklung von Steuergeräten können die Vorteile der Methode bei der Verteilung der Funktionen und der Entwicklung von Kommunikationsstrukturen nur begrenzt eingesetzt werden. Der größte Nachteil dieser Methode besteht darin, dass das Verhalten nicht spezifiziert werden kann.

[2] www.vector.de

Dies ist aber bei der Entwicklung von Steuergeräten ein äußerst wichtiger Bestandteil. Ferner kann in der Methode nur die Applikations-Software modelliert werden. Somit ist eine Darstellung einer Gesamtarchitektur für ein automobiles Steuergerät nicht möglich. Daher ist diese Methodik für einen Zulieferer nicht vorteilhaft.

3.10 Timmo

Bei den bislang untersuchten Arbeiten ging es es hauptsächlich um die Entwicklung von eingebetteten Systemen bzw. von automobilen Steuergeräten. Im hier vorgestellten TIMMO Projekt wird die Modellierung von Echtzeit untersucht [Pro09, ERG08]. Das TIMMO Projekt ist ein europäisches Projekt mit der Zielsetzung der Modellierung und Formalisierung von Echtzeit innerhalb der Entwicklung. Das Projekt lief in der ersten Phase bis 2009 und das Nachfolgeprojekt TIMMO2 von 2010 bis 2012.

Ein Ergebnis aus dem ersten Projekt war die Erarbeitung einer Entwicklungssprache zur Spezifikation von Echtzeit. Dies ist die sogenannte *Timing Augmented Description Language (TADL)* [Pro09]. Sie bietet die Möglichkeit, das zeitliche Verhalten eines eingebetteten Systems genauer zu spezifizieren. Dabei wurde ein besonderes Augenmerk auf die Spezifikation von Ereignissen, die sich über mehrere Komponenten erstrecken, gelegt. Die TADL ist dabei in allen Entwicklungsphasen einsetzbar. Sie ist in der Implementierungsphase auf den AUTSOAR Standard abgestimmt. In den vorherigen Entwicklungsphasen (Analysephasen) wird die TADL zusammen mit der EAST-ADL2 eingesetzt. Dies veranschaulicht die Abbildung 3.8. Die TADL ist dabei orthogonal zu den Entwicklungsphasen angeordnet.

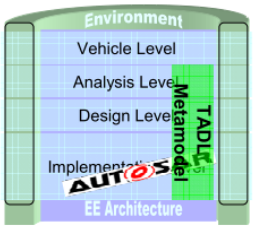


Abb. 3.8. Verwendung der TADL innerhalb der EAST-ADL2 [Pro09] Deliverable D6

Vor allem wurde in der TADL eine Möglichkeit geschaffen, um die Zeiten von Wirkketten zu modellieren. Dies bedeutet, dass es Erweiterungen von Ereignissen (Events) gibt. Zwischen den Ereignissen, die dabei auf mehrere Komponenten verteilt sein können, ist die Wirkkette mit einer Zeit spezifiziert. So ist

in Abbildung 3.9 das Beispiel einer ACC dargestellt, bei dem die Zeit zwischen dem Sensor und dem Aktor 85 ms beträgt.

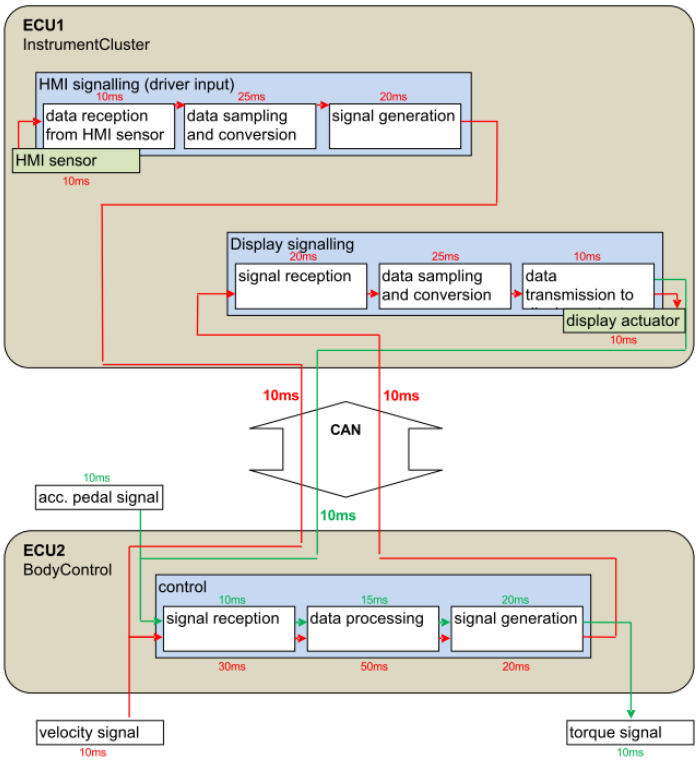


Abb. 3.9. Einsatz der TADL bei der Modellierung einer Adaptive Cruise Control (ACC) [Pro09] Deliverable D7

Die Ergebnisse des TIMMO Projektes finden zum Teil bereits in anderen Projekten Anwendung. So ist im AUTOSAR Standard ab der Version 4.0 eine Beschreibungssprache für Zeitinformationen, in diesem Fall von WCRTs, enthalten [Jer07]. Es wurden Techniken des TIMMO Projektes aufgegriffen und weiter entwickelt.

Der Nachteil der TADL ist, dass sie hauptsächlich konzipiert wurde, um in einer Architektur zu Dokumentationszwecken eingesetzt zu werden. Sie wird dabei überwiegend für die Strukturelemente verwendet. So ist beispielsweise die Start- oder die Schlafphase eines Steuergerätes nur schwerlich mit der TADL darstellbar, da in diesen Phasen besonders das dynamische Verhalten mit zahlreichen Ereignissen überwiegt (siehe Kapitel 2.6.4). Von daher muss für diese Arbeit eine andere Modellierungstechnik gesucht werden.

3.11 Software Plattform Embedded Systems (SPES) 2020

Im Rahmen der Innovationsallianz Software Plattform Embedded Systems (SPES) 2020 wird an einer domänenübergreifenden und modellbasierten Entwicklung für eingebettete Systeme geforscht [SPE09]. Das Ziel ist die Erarbeitung von Methoden und Techniken, die eine effiziente Entwicklung von eingebetteten Systemen im Jahre 2020 ermöglichen. Um dieses Ziel zu verwirklichen, arbeiten und forschen 21 deutsche Partner aus Wissenschaft und Wirtschaft. In der Innovationsallianz werden dabei unterschiedliche Anwendungsgebiete betrachtet, um so die Allgemeingültigkeit der erarbeiteten Ergebnisse zu gewährleisten.

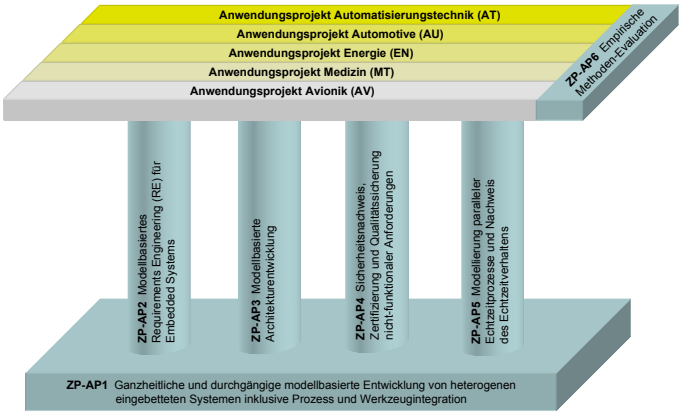


Abb. 3.10. Aufbau des SPES 2020 Projektes [SPE09]

Das Projekt ist in sechs sogenannte Zentralprojekte und in fünf Anwendungsgebiete unterteilt. Die Zentralprojekte haben die Aufgabe, domänenübergreifende und somit allgemeine Ziele abzudecken. In den Anwendungsgebieten wird dediziert auf Problemstellungen der einzelnen Domänen eingegangen. Der Aufbau des Projektes ist in der Abbildung 3.10 dargestellt. Teile dieser Arbeit sind in das Anwendungsgebiet Automotive eingeflossen.

Ein zentrales Ergebnis des Projektes ist die Definition von Perspektiven, Abstraktionsschichten und eines Metamodells zur Spezifikation von allgemeinen eingebetteten Systemen [Lee09]. In den einzelnen Entwicklungsstufen innerhalb der Perspektiven kommt es immer weiter zu Detaillierungen (vgl. Abbildung 3.11). Die Einteilung in die verschiedenen Perspektiven werden in AutoMoMe zum großen Teil übernommen und an die automobile Steuergeräteentwicklung angepasst. Die in SPES 2020 eingeflossenen Arbeiten von AutoMoMe werden in dieser Arbeit zu einem durchgängigen modellbasierten Ansatz für die automobile Steuergeräteentwicklung zusammengefasst.

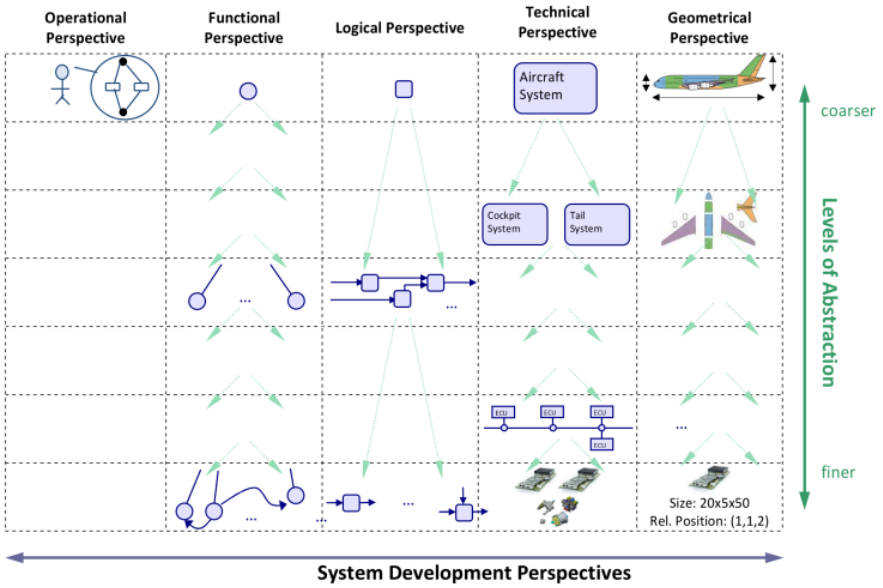


Abb. 3.11. Abstraktionsschichten und Detaillierungen der SPES Architektur [PHAB12]

3.12 Fazit zum Stand der Technik

Als Fazit zum Stand der Technik kann festgehalten werden, dass in der Automobilindustrie und insbesondere bei der automobilen Steuergeräteentwicklung besondere Eigenschaften vorherrschen. Zum einen ist diese geprägt von der anforderungsbasierten Entwicklung (Zusammenspiel OEM und Zulieferer), d. h. es müssen bei der Steuergeräteentwicklung eine Vielzahl von textuellen Anforderungen analysiert und später realisiert werden. Zum anderen spielt die Architektur eine große Rolle. Hierbei kommt es nicht nur auf die Applikations-Software an, sondern auf die gesamte Architektur, die bei einem Steuergerät ebenso die Basis-Software mit einschließt. Bei der Architektur spielt ebenfalls die Verhaltensmodellierung eine gewichtige Rolle, um die Echtzeitfähigkeit des Steuergerätes sicherzustellen [PBKS07, NSV10]. Die hier vorgestellten Ansätze und Methoden erbringen durchaus eine punktuelle Verbesserung bei der automobilen Steuergeräteentwicklung. Für eine Übersicht werden die Ansätze und die Erfordernisse der automobilen Steuergeräteentwicklung in der Tabelle 3.1 dargestellt. Dabei existieren vier Abstufungen, die die Erfüllungen der Erfordernisse angeben. Das + besagt, die Bedingung ist vollständig umgesetzt. Während das (+) bedeutet, die Bedingung ist zum größten und bei (-) zu einem geringen Teil umgesetzt. Ein - symbolisiert, dass der Ansatz die Bedingung der automobilen Steuergeräteentwicklung gar nicht adressiert bzw. umsetzt. Wie

aus der Tabelle zu erkennen ist, werden durch keinen der eben vorgestellten Ansätze alle Anforderungen der automobilen Steuergeräteentwicklung erfüllt. Daher wurde die Methodik AutoMoMe entwickelt und wird im Folgenden vorgestellt.

Ansatz	Kommunikation	Funktion	Architektur	Verhalten (inklusive Echtzeit)	Betriebssystem	Übergang zu AUTOSAR	Methodik
Step-X	-	(+)	+	(+)	-	-	+
AML	-	+	+	(-)	-	-	(-)
SysCARS	(-)	(-)	(+)	(+)	-	-	+
EAST-ADL2	(+)	+	(+)	-	-	(+)	+
EDONA	(+)	(+)	+	(-)	(+)	(+)	+
Camos	+	(-)	(+)	(-)	-	-	+
AxBench	(+)	-	(-)	-	(+)	(+)	-
Decos	(+)	+	(+)	(-)	(+)	-	(+)
AutoFOCUS	(+)	+	(+)	(+)	-	-	+
PreeVision	+	+	(+)	-	-	+	(+)
TIMMO	-	-	-	(+)	-	-	-
AutoMoMe	+	+	+	+	+	+	+

Tabelle 3.1. Zusammenfassung der vergleichbaren Ansätze und Arbeiten

Nicht mit Erfindungen, sondern mit Verbesserungen macht man Vermögen.

Henry Ford (1863 - 1947)

4

Lösungsansatz: AutoMoMe

In den folgenden Abschnitten wird der Lösungsansatz AutoMoMe vorgestellt. Der Ansatz beschreibt eine Methodik wie der Entwicklungsprozess für ein automobiles Steuergerät mit (semi-) formalen Methoden und Techniken von der funktionalen Systemarchitektur bis hin zur Codegenerierung unterstützt werden kann. Er greift dabei die Besonderheiten der automobilen Steuergeräteentwicklung auf (siehe Kapitel 2.6) und stellt eine Verwendung der generellen Ansätze aus [SFB09] und [SPE09], angewandt auf die automobile Steuergeräteentwicklung, dar. Dabei wird auf die bekannte UML Notation [TB03, FR07] zurückgegriffen, um eine lückenlose Modellierung einschließlich der Verwendung des AUTOSAR Standards zu ermöglichen. Für diese Anforderungen sind die bestehenden Möglichkeiten der UML jedoch nicht ausreichend, um die notwendigen domänenspezifischen Informationen abbilden zu können. Deswegen erweitert der Ansatz die Modellierungssprache, um eine vollständige Beschreibung des Steuergerätes (Applikations- und Basis-Softwarebestandteile) zu ermöglichen. Dadurch werden zudem die notwendigen Entwurfsentscheidungen dokumentiert. Auf Grund der eindeutigen (semi-) formalen Beschreibungsart und die Anpassung an die automobile Steuergeräteentwicklung werden spätere Fehlinterpretationen ausgeschlossen, da die Alternativen und Entscheidungen bekannt sind [HOT11]. Aufbauend auf der (semi-) formalen Spezifikation werden weitergehende Modelltransformationen bzw. Syntheseschritte vorgestellt, die es ermöglichen, eine durchgängige automobile Steuergeräteentwicklung inklusive Echtzeitsimulation und Modellierung in AUTOSAR zu verwirklichen. Der Ansatz AutoMoMe setzt einen Schwerpunkt besonders in den frühen Entwicklungsphasen, um hierdurch die Qualität und die Effizienz der Entwicklung so zu erhöhen, dass insgesamt Zeit und Kosten bei der Produktentwicklung eingespart werden. Das Vorgehen durch verstärkte, zielgerichtete und qualitativ hochwertige Arbeiten in den frühen Entwicklungsphasen den Grundstein für die weitere Entwicklung zu legen, ist unter dem Begriff *Frontloading* bekannt und wird im Lösungskonzept AutoMoMe umgesetzt [SHI+11].

4.1 Eigenschaften der Steuergeräteentwicklung mit AutoMoMe

In den Kapiteln *Grundlagen* und *Stand der Technik* (siehe Kapitel 2 und Kapitel 3) wurde herausgearbeitet, dass die Modellierungslösungen und Techniken der jetzigen Verfahren nicht immer optimal und ausreichend sind, um mit der Komplexität neuerer Produkte und Entwicklungen in der Automobilindustrie Schritt halten zu können. Es hat sich deutlich gezeigt, dass bei der Modellierung noch erhebliche Anpassungen und Neuerungen erforderlich sind (siehe hierzu Kapitel 1.1) [Bro06, BS05]. Um den gestellten Anforderungen zu entsprechen, wird für die Steuergeräteentwicklung als Verbesserung die modellbasierte Systementwicklung vorgeschlagen [BKK+12]. Durch den Einsatz der Modelle und der damit verbundenen Abstraktionen können neuere und komplexere Produkte entwickelt werden.

Bislang gibt es einige Bereiche in der Steuergeräteentwicklung in denen bereits intensiv die modellbasierte Entwicklung umgesetzt wird [NS08]. Dies ist z. B. bei der Funktionsentwicklung mit Matlab/Simulink [Mat14] bzw. Targetlink [dSP14] der Fall. Mit diesen Werkzeugen wird der (Reglungs-) Algorithmus für einzelne Applikationen (beispielsweise die Blinkersteuerung) modellbasiert realisiert. Eine Beschränkung der modellbasierten Entwicklung nur auf diesen Teilbereich ist nicht optimal, da die Architektur und somit das Zusammenspiel der einzelnen Algorithmen außer Acht gelassen wird, zumal ein Großteil der schweren Fehler dort entstehen [Jon97]. Es wird das Gesamtverhalten nicht modelliert. Somit kann nicht ausgeschlossen werden, dass es bei der Integration der verschiedenen Algorithmen zu Problemen kommen kann.

Um dem vorzubeugen und die Qualität der Entwicklung insgesamt zu verbessern, wird bei dem Ansatz AutoMoMe eine ganzheitliche modellbasierte und (semi-) formale Spezifikation umgesetzt. Hierdurch kann die Architektur und ebenso das Gesamtverhalten eindeutig modelliert werden. Als nicht zu vernachlässigenden Vorteil der modellbasierten Architekturmodellierung ist die weitere Verwendung der Informationen – möglichst automatisiert – in späteren Entwicklungsphasen bei Analyse-, Simulations- oder Testwerkzeugen einzustufen. Erst hiermit entsteht eine durchgängige modellbasierte Entwicklung. Für eine modellbasierte Architekturmodellierung in der Steuergeräteentwicklung sind aber Anpassungen bzw. Ergänzungen bei den bisherigen Entwicklungsarbeiten bzw. der eingesetzten Methodik unumgänglich, die sich im Ansatz von AutoMoMe wiederfinden. Für die Modellierung von Steuergeräten im Automobilbereich weist der Ansatz AutoMoMe die folgenden Eigenschaften auf:

1. **Anpassung an die automobile Steuergeräteentwicklung**
   Eine Entwicklung ist besonders dann effizient, wenn die Entwickler zur Spezifikation des Systems ihre Sprache und die Eigenschaften der Domäne einsetzen. Von daher hat eine domänen-spezifische Sprache (DSL) Vorteile

gegenüber einer allgemeinen Sprache [Tol08]. Damit die Eigenschaften und Besonderheiten der automobilen Steuergeräteentwicklung (siehe Kapitel 2.6.1) in AutoMoMe Berücksichtigung finden, wird die im Architekturmodell verwendete Sprache UML an die Domäne angepasst. So finden sich dort beispielsweise die verwendeten Kommunikationsbusse, aber ebenso eine Notation für Steuergeräte wieder. Es wurden die Erkenntnisse aus der Entwicklung von eingebetteten Systemen [SFB09, SPE09] berücksichtigt.

2. **Durchgängige Spezifikation von der System- bis zur Softwareentwicklung**
   Immer mehr Funktionen mit zusätzlichen Anforderungen werden auf einem Steuergerät realisiert. Dieser Anstieg der Anforderungen führt zwangsläufig zu einer zunehmenden Komplexität. Vor allem bei Änderungen von Anforderungen muss frühzeitig erkannt werden, welche Auswirkungen sich hieraus ergeben. Um dies zu gewährleisten, ist eine durchgängige Entwicklungsmethodik und Nachverfolgbarkeit (Traceability) von der funktionalen Architektur bis hin zum AUTOSAR Modell erforderlich. Diese Forderung wird bereits in dem Referenzprozess Automotive SPICE [dAV10] aufgestellt. Für die lückenlose und durchgängige Modellierung zwischen den verschiedenen Artefakten im Entwicklungsprozess werden im Ansatz AutoMoMe entweder geeignete automatisierte Transformationen oder aber Referenzen eingesetzt, wobei letztere an den Stellen genutzt werden, wo keine automatische Transformation möglich ist. Ein Schwerpunkt der automatisierten Transformationen ist dabei der Übergang von der Architekturmodellierung in UML hin zum AUTOSAR Modell.

3. **Formal definierte Spezifikation von zeitlichen Verhalten**
   Eine Anforderung an die zu erarbeitende Entwicklungsmethodik besteht in einer formal definierten Spezifikation von zeitlichen Informationen. Bei einer Analyse der zeitlichen Anforderungen ist festzustellen, dass unterschiedliche Zeitinformationen in der Automobilbranche vorhanden sind und spezifiziert werden müssen. Zum einen gibt es Zeitgrenzen (max. Zeitdauer), die eine maximale Verarbeitungszeit vorschreiben. Hier kann als Beispiel die Anforderung angeführt werden, die Startphase in maximal 100 ms abzuarbeiten. Dabei gilt es zu berücksichtigen, dass die vorgeschriebene Zeitspanne sich unter Umständen auf mehrere Komponenten beziehen kann, die nacheinander für die Funktionalität verantwortlich sind. Bei einer solchen Konstellation wird von sogenannten Wirkketten (Event-Chains) gesprochen.

   Zum anderen gibt es Zeitangaben, die vorgeben, dass eine Funktion oder eine Reaktion eine Mindestzeit benötigt (min. Zeitdauer). Eine Mindestzeit muss dann vorgegeben werden, wenn eine Nachricht übermittelt wird und der Entwickler Kenntnis davon hat, dass die Übertragung über einen Kommunikationskanal (z.B. einen Bus) eine gewisse Zeit in Anspruch nimmt. Eine weitere in der Automobilbranche häufig verwendete zeitliche Informa-

tion ist die zyklische Zeitangabe. In dieser wird die Zeit spezifiziert, in der der Zustand oder die Funktion zyklisch zu aktivieren ist. Beispielsweise, dass die Funktion Running alle 5 Millisekunden aufzurufen ist.

Der Ansatz AutoMoMe erfüllt diese Anforderung durch eine (semi-) formale Spezifikation der Betriebssystemeigenschaften und der Funktionalität, die Ausführungszeiten der jeweiligen Elemente zu spezifizieren. Hierdurch werden alle notwendigen Daten im Modell gespeichert und sind jederzeit abrufbar.

4. **Entwicklung von verteilt laufenden Echtzeitsystemen**
   Bei neuartigen Entwicklungen, als solche sind zum Beispiel die Fahrerassistenzsysteme zu nennen, ist festzustellen, dass vielfach verteilt laufende Software zusammen eine neue Funktionalität ergeben [Rei10b]. Exemplarisch sei hier das ACC genannt. Das ACC erfüllt zusammen mit dem Motor- und dem Bremssteuergerät die neue Funktionalität [WHW10]. Bei der Modellierung ist darauf zu achten, dass die zeitliche Zusammenarbeit bzw. Kommunikation der verschiedenen Softwarekomponenten spezifiziert wird. Das hat zur Folge, dass eine einheitliche und ebenso für verteilte Komponenten nutzbare Modellierung erforderlich wird. Der Ansatz AutoMoMe unterstützt diese Eigenschaft durch die Modellierung der Architektur mittels Komponenten und Schnittstellen, die auf verschiedene Rechenkerne verteilt werden. Ebenfalls bietet der Ansatz die Funktionalität Kommunikationsbusdaten zu importieren, um somit die Kommunikation zwischen den verteilten Systemen zu spezifizieren.

5. **Anbindung an Echtzeitanalyseverfahren**
   Eine weitere Anforderung an zukünftige Modellierungstechniken ergibt sich aus der zunehmenden Komplexität der Systeme. Die Systeme beinhalten immer mehr Funktionen, die um Ressourcen konkurrieren. Dabei müssen sehr häufig harte Echtzeitanforderungen eingehalten werden. Von daher kommt der Architektur eine gewichtige Rolle im Entwicklungsprozess zu. Die Absicherung der Architektur muss bereits in den frühen Entwicklungsphasen erfolgen, um einerseits die Kosten bei Änderungen zu beschränken und andererseits nicht den kompletten Entwicklungsprozess noch einmal durchlaufen zu müssen. Ergebnisse aus dem Systemtest – beispielsweise durch Hardware-in-the-loop (HIL) Tests –, die ähnliche Erkenntnisse erbringen, liegen erst am Ende des Entwicklungsprozesses vor. Um einen vergleichbaren und zum Teil noch besseren Erkenntnisstand bereits frühzeitig zur Verfügung zu stellen, wird im Konzept AutoMoMe in frühen Entwicklungsphasen eine Methode inklusive eines entsprechenden Werkzeugs für die Echtzeitanalyse vorgehalten [Sch05b]. Die Echtzeitsimulation bringt zusätzlich einen verbesserten Kenntnisstand über das dynamische Verhalten des Systems zu bestimmten Zeitpunkten mit sich. Die Echtzeitanalyse ist vergleichbar mit der Absicherung des funktionalen Verhaltens, welches durch Model-in-the-loop (MIL) Simulationen bere-

its während der Entwicklungsphase überprüft wird. Für die Architektur ist eine Kontrolle der Echtzeitfähigkeit eine vergleichbare Sicherheitsmaßnahme in diesem Entwicklungsstadium. Der Ansatz AutoMoMe berücksichtigt diese Eigenschaft durch die Integration einer Echtzeitsimulation, so dass die im Architekturmodell hinterlegten Werte zeitnah überprüft werden.

6. **Integration von Softwarekomponenten**
   Eine neue Methodik bei der automobilen Steuergeräteentwicklung muss berücksichtigen, dass eine Integration von Softwarekomponenten notwendig ist. Im Zuge der Steuergeräteentwicklung werden Teile der Basis-Software von einem Tier-2 zugeliefert. Dies wird durch die Standardisierung mittels AUTOSAR noch weiter forciert. Ferner ist zu beachten, dass neuerdings auch die Automobilhersteller eigene Softwarekomponenten den Zulieferern zur Verfügung stellen und diese die Komponenten in das Steuergerät integrieren müssen. In AutoMoMe wird mittels der UML eine Möglichkeit geschaffen, die Spezifikation der Basis-Software aus der AUTOSAR Definition zu importieren und somit zur Verfügung zu stellen (vgl. BSW Modell im AUTOSAR Standard [AUT10i]). Ferner werden die Komponenten der Automobilhersteller anhand der Schnittstellen und ihres Echtzeitverhaltens in dem Architekturmodell berücksichtigt, so dass eine problemlose Integration in das Gesamtsystem erfolgt.

7. **Integration in die bestehende Entwicklungsumgebung**
   Für eine zukünftige Entwicklungsmethodik ist die Einbindung in die bestehende Entwicklungsumgebung unverzichtbar, da bei der Funktionsentwicklung verschiedene Methoden und Werkzeuge eingesetzt werden. Besonders häufig kommt hier das Werkzeug Matlab/Simulink [Mat14] zum Einsatz, um reglerbasierte Algorithmen modellbasiert zu spezifizieren. Damit eine einheitliche Überleitung zwischen der Architekturmodellierung aus AutoMoMe und der Funktionsentwicklung in Matlab/Simulink entsteht, wird ein wohldefinierter Übergang zwischen beiden Bereichen definiert. Dabei wird schwerpunktmäßig die Synchronisation der Schnittstellen betrachtet.

4.2 Komfortsteuergerät als durchgängiges Beispiel

Als durchgängiges Beispiel für die Steuergeräteentwicklung wird ein Komfortsteuergerät (KSG) verwendet. Ein KSG ist im Bereich Karosserie bzw. Komfort vorzufinden. Neben diesem Bereich sind noch fünf weitere Bereiche in einem Automobil anzutreffen (vgl. Abbildung 4.1). Die Bereiche teilen sich dabei entsprechend ihrer Funktionalität auf. Während manche Bereiche den Fahrzeuginsassen direkt als Komfortleistung angeboten werden, werden andere für die Hauptfunktionalität, also zum Fahren eines Kraftfahrzeuges, benötigt. Dem

ersten Bereich sind neben Karosserie-/Komfort ebenso die Bereiche Multimedia und Mensch-Maschine-Schnittstelle zuzuordnen. Die für das Fahren eines Fahrzeuges verantwortlichen Bereiche werden in Fahrwerk, Antrieb und Sicherheit aufgeteilt. Jeder dieser Bereiche hat seine individuellen Eigenschaften und Merkmale. Da das als Beispiel dienende Komfortsteuergerät aus dem Karosserie-/Komfortbereich stammt, ist dieser Teilbereich in der Abbildung 4.1 besonders hervorgehoben.

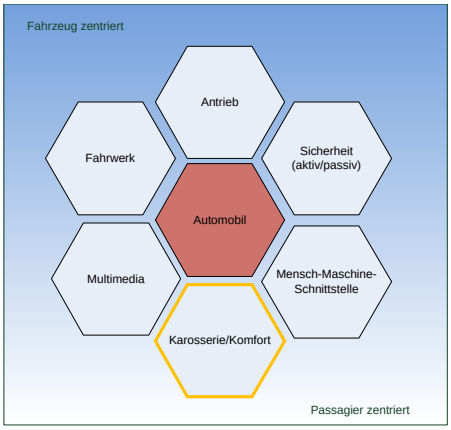


Abb. 4.1. Verschiedene Bereiche in einem Automobil

Der Bereich Karosserie-/Komfort ist, wie der Bereichsname schon erkennen lässt, für den Komfort der Fahrzeuginsassen verantwortlich und nicht für die eigentliche Fahrfunktion eines Automobils. Hierunter fallen viele Funktionen, die die Nutzung des Automobils angenehmer gestalten. Die folgenden Funktionen sind in einem Komfortsteuergerät zu finden, wobei es aber auch optionale Bestandteile gibt. Diese sind je nach Konfiguration bzw. Auswahl von Zusatzelementen des Fahrzeugs aktiviert.

- Zentralverriegelung
- Ansteuerung Fahrzeugbeleuchtung (intern und extern) inklusive der Blinkersteuerung
- elektronisches Zündschloss
- Komfort öffnen/schließen der Seitenfenster und des Schiebedaches
- Wegfahrsperre
- (optional) Diebstahlwarnanlage
- (optional) Reifendruckkontrolle

Auf Grund der Vielzahl von Funktionalitäten des Komfortsteuergerätes werden in AutoMoMe nicht alle Funktionalitäten betrachtet. Vielmehr wird der Fokus auf die Funktionalitäten Zentralverriegelung, Diebstahlwarnanlage und Fahrzeugbeleuchtung gelegt. Diese Funktionalitäten weisen die Charakteristika auf, die für die anderen Funktionalitäten ebenso gelten. Von daher sind sie als repräsentativ anzusehen.

Die Funktionalität der Fahrzeugbeleuchtung umfasst dabei nicht nur die externe Beleuchtung mit den verschiedenen Modi (Standlicht, Abblendlicht, Nebellicht, Fernlicht, Bremslicht etc.), sondern ebenfalls die interne Beleuchtung. Hierunter fällt die Türausstiegs- und Türwarnbeleuchtung, die Fußraumbeleuchtung, die Gepäckraumbeleuchtung sowie das Innenlicht. Die Charakteristika dieser Funktionen sind, dass sie größtenteils ereignisgesteuert sind. Sie weisen zwar zeitliche Anforderungen auf wie z. B., dass das Innenlicht spätestens 50 ms nach Öffnen der Tür aktiviert sein muss. Jedoch ist dies keine harte echtzeitkritische Anforderung, da kein Risiko davon ausgeht, wenn die Anforderung überschritten wird.

Die Funktionalität der Zentralverriegelung ist zuständig für das Öffnen und Schließen eines Fahrzeugs und kann um die optionale Funktion einer Diebstahlwarnanlage ergänzt werden. Die Zentralverriegelung beinhaltet sowohl die Türen als auch den Kofferraum und das Tankschloss. Es gilt, dass nach Identifizierung des Fahrzeuginhabers das Fahrzeug innerhalb von 150 ms geöffnet sein muss. Die Identifizierung erfolgt dabei anhand des Autoschlüssels, der Fernbedienung oder aber mittels eines Chips (Keyless Access). Des Weiteren wird durch die Zentralverriegelung gesteuert, dass die Fahrzeugtüren ab einer Geschwindigkeit größer als 15 km/h automatisch geschlossen werden. Dies soll Diebstähle und Überfälle verhindern. Jedoch muss diese Funktion bei einem Unfall deaktiviert und somit die Türen automatisch geöffnet werden, damit die Insassen nicht eingeschlossen werden. Hierdurch wird sie zu einer sicherheitskritischen Funktion.

Somit ergeben sich auch Echtzeitanforderungen an das Komfortsteuergerät. Das Komfortsteuergerät steht ferner in enger Verbindung zu anderen zeit- oder sicherheitskritischen Funktionen, wie beispielsweise die Airbagsteuerung oder aber der Unfalldetektion zur Notöffnung der Türen nach einem Unfall. Von daher gibt es Echtzeitanforderungen, die von anderen Systemen an ein Komfortsteuergerät gestellt werden, beispielsweise dass nach Erkennen eines Unfalls alle Türen innerhalb von 5 ms geöffnet sein müssen. Durch die enge Vernetzung zu den anderen Systemen muss ein Komfortsteuergerät darüber hinaus eine vielfältige Kommunikationsfähigkeit aufweisen. Dies führt zu einer vielschichtigen und komplexen Architektur auf dem Steuergerät [NSL08].

Dieses Merkmal ist bei allen Steuergeräten im Karosseriebereich vorzufinden. Ein weiteres Merkmal dieses Bereichs ist die Existenz von verteilten Substrukturen. Dies bedeutet, dass die Funktionen zum Teil auf mehreren kleineren Steuergeräten (Tür- und Sitzsteuergeräte) verteilt und durch LIN-

Busse miteinander verbunden sind. So existiert in jeder Tür ein Türsteuergerät, dass das Öffnen und Schließen der Tür übernimmt. Es steuert somit die in den Türen untergebrachten Aktoren. Die Steuerung der Türsteuergeräte erfolgt aber über das zentrale Komfortsteuergerät, welches an den Komfort-CAN Kommunikationsbus angeschlossen ist und darüber die Kommunikation zu anderen Steuergeräten durchführt (vgl. Abbildung 4.2). Da die kleineren Steuergeräte über einen LIN-Bus mit dem Komfortsteuergerät kommunizieren, muss eine Umwandlung zwischen den verschiedenen Busprotokollen erfolgen.

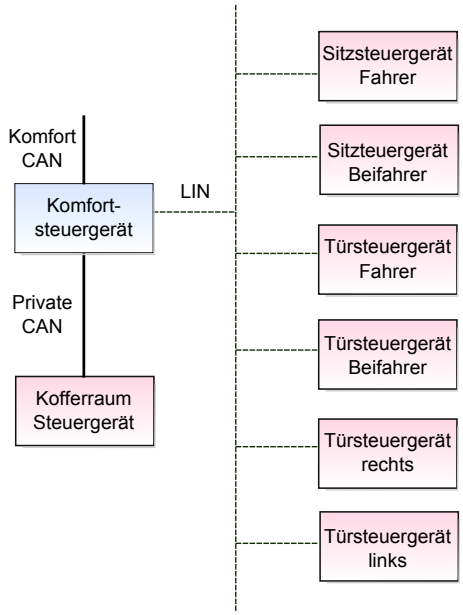


Abb. 4.2. Anbindung des Komfortsteuergerätes an die Kommunikationsbusse

Somit fungiert das Komfortsteuergerät auch als Gateway, da es die Kommunikationsdaten entsprechend transformiert und weitersendet. Dies kann für die zeitliche Performance des gesamten Steuergerätes wichtig sein, da die Datenintegrität der zu übermittelnden Daten berücksichtigt werden muss. Die Anordnung der verschiedenen Türsteuergeräte und Bestandteile des Komfortsteuergerätes, die im gesamten Fahrzeug verteilt sind, werden in der Abbildung 4.3 veranschaulicht. In der Abbildung ist das zentrale Steuergerät in dunkelblau dargestellt, während die anderen Steuergeräte, die zum Gesamtsystem Komfortsteuergerät gehören, in hellblau abgebildet sind.

Beim Komfortsteuergerät wird, wie bei anderen Steuergeräten ebenfalls zu beobachten ist, immer mehr Software zur Realisierung der Funktionen eingesetzt. Von daher ist es ein typisches Beispiel für ein Steuergerät in einem

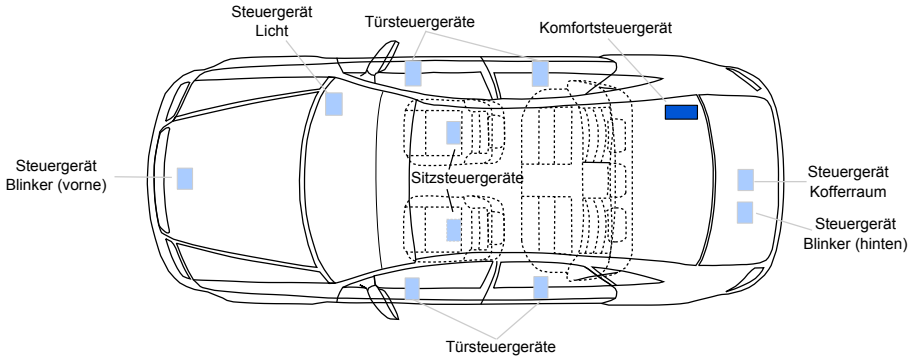


Abb. 4.3. Verschiedene Bestandteile des Komfortsteuergerätes

Automobil. Die im Ansatz AutoMoMe entwickelten Methoden und Techniken werden anhand dieses Beispiels erläutert.

4.3 Architekturmodellierung

Erforderlich ist somit ein durchgängiger Entwicklungsprozess [Sch05a], der nicht nur die Charakteristiken der Steuergeräteentwicklung in der Automobildomäne berücksichtigt, sondern gleichfalls möglichst viele Automatismen und Optimierungen vereinigt, um den Anforderungen bezüglich schnellerer Entwicklungszeiten und überschaubarer Kosten gerecht zu werden. In diesem Zusammenhang muss ebenso die Wiederverwendung von Softwarekomponenten bedacht werden [BS05], da im Automobilbereich, vor allem in der Zuliefererindustrie, die Systeme nicht immer neu („auf der grünen Wiese“) entwickelt werden. Vielmehr ist von einer stetigen Weiterentwicklung und Weiterverwendung in anderen Projekten auszugehen.

Daher ist die Zielsetzung der vorliegenden Arbeit, die einzelnen bestehenden aber oftmals isolierten Methoden zu vereinen und durch gezielte Ergänzungen und Anpassungen so zu erweitern, dass ein einheitlicher, (semi-) formalisierter und durchgängiger Entwicklungsprozess für einen Zulieferer in der Automobilindustrie entsteht.

Als Ausgangspunkt dient dabei das Systemmodell des Sonderforschungsbereiches 614 (siehe Abbildung 2.6) [SFB09]. In diesem werden für ein eingebettetes System die einzelnen Partialmodelle zu einem Gesamtmodell zusammengefasst. Jedoch werden im Systemmodell nicht die Besonderheiten der automobilen Steuergeräteentwicklung wie die Aufteilung in Applikations- und Basis-Software oder aber die Verwendung des AUTOSAR Standards berücksichtigt. Demzufolge wurde in AutoMoMe ein verwandtes Konzept für die Konstruktion von automobilen Steuergeräte entwickelt, das ebenfalls mehrere Mod-

elle im sogenannten Architekturmodell (siehe Definition Architekturmodell) miteinander verknüpft, um so die Entwicklung einer effizienten und vor allem durchgängigen Entwicklung bis hin zur Implementierung zu ermöglichen. Die entwickelte Methode AutoMoMe entspricht demzufolge einer domänenspezifischen Instanziierung des Systemmodells.

Basis für einen solchen Ansatz ist eine (semi-) formale Modellierung des zu entwickelnden Steuergerätes und das bereits in einer frühen Entwicklungsphase. Dies wurde bei der Auswahl der einzusetzenden Beschreibungssprachen berücksichtigt. Aus diesem Grund ist es nicht möglich, mit dem AUTOSAR Standard zu beginnen, da dieser keine Analyse von Anforderungen erlaubt noch die Spezifikation des dynamischen Verhaltens zulässt [KF09]. In AutoMoMe werden die verschiedenen Modelle im sogenannten Architekturmodell zusammengefasst [TKL11, MGRK07]. Das Architekturmodell umfasst, wie die Definition bereits zum Ausdruck bringt, mehrere integrierte Modelle [BKS10, NS08], die von der Analyse bis hin zur Implementierung das System und vor allem die darin befindliche Software in einer gemeinsamen Sprache [PS12] spezifizieren. Das Architekturmodell stellt dabei die Grundlage [Noe05] für die weitere zukunftssichere Modellierung eines Steuergerätes [EGG+09, GGS12] und sorgt darüber hinaus für eine bessere Kommunikation im Entwicklungsteam [CR11].

Definition (Architekturmodell). Das Architekturmodell beschreibt die Strukturen des Systems und dessen Bestandteile. Des Weiteren werden die Schnittstellen genauso wie deren Zusammenspiel definiert. Hierdurch wird das globale Verhalten des Systems spezifiziert [SH09]. Das Architekturmodell besteht aus drei verschiedenen Sichten: funktionale-, logische und technische Architektur.

Um die zentrale Bedeutung des Architekturmodells und die Bestandteile des Konzeptes von AutoMoMe zu veranschaulichen, wird in der Abbildung 4.4 das Architekturmodell – das zentrale Konzept von AutoMoMe – skizziert. In dem Architekturmodell wird die funktionale wie auch die technische Architektur spezifiziert [vdB06], die die entsprechenden Sichtweisen auf das Steuergerät geben [Nat08, NSV10, PBKS07]. Dies erfolgt durch die Modellierungssprachen SysML auf Systemebene und UML auf Softwareebene. Die beiden Sprachen werden durch Erweiterungen [FR07] für die automobile Steuergeräteentwicklung ergänzt [Win05], damit alle Informationen (semi-) formal dargestellt werden. Dies sind beispielsweise Modellierungsformen für Betriebssystemdaten, Zeit- und Ressourceninformationen sowie die Modellierung von Kommunikationssignalen.

Bei dem Umfang der Steuergeräte ist es nicht verwunderlich, dass das Architekturmodell verschiedene Sichtweisen aufweist (siehe Abbildung 4.5)

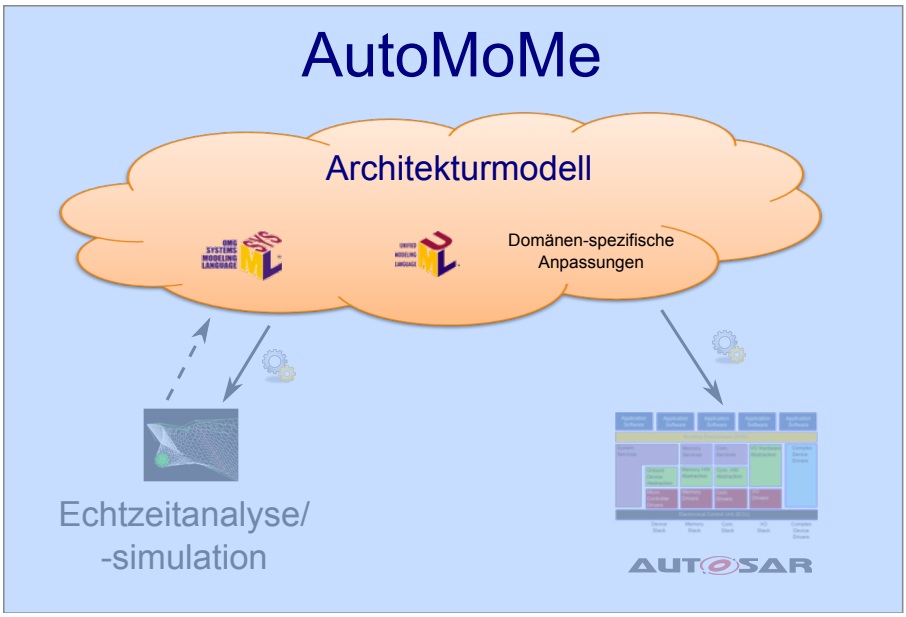


Abb. 4.4. Architekturmodellierung im Ansatz AutoMoMe

[BKS10, Nat08, FFKM10]. Die einzelnen Sichtweisen verfügen dabei selber noch über verschiedene Abstraktionsstufen und Schichten. In automobilen Steuergeräten wird sehr häufig das Architekturmittel von Schichten verwendet [KF09, AUT10b]. Hierdurch wird die steigende Komplexität beherrschbar gemacht, zumal bei der Entwicklung die einzelnen Elemente immer weiter verfeinert werden. Dies erfolgt in Abstraktionsstufen [SPE09, PHAB12]. Die Bestandteile des Architekturmodells und die Gliederung in die drei Sichtweisen sind in der Abbildung 4.5 wiedergegeben. Die einzelnen Modelle des Architekturmodells sind den drei Sichtweisen zugeordnet. Sie beschreiben dabei die unterschiedlichen Bereiche des Systems und repräsentieren außerdem verschiedene Phasen bei der Entwicklung. Hierdurch wird gewährleistet, dass die Nachverfolgbarkeit und Konsistenz im Modell umgesetzt wird. Die Auswahl der Sichtweisen ist denen aus Forschungsprojekten für eingebettete Systeme, wie Software Plattform Embedded Systems (SPES) und Cost-efficient methods and processes for safety relevant embedded systems (CESAR) [SPE09, Pro14], entnommen.

Damit die Einteilung der einzelnen Bestandteile des Steuergerätes in Sichtweisen und Schichten erfolgen kann, wurde das Architekturmodell und zugehörige Daten strukturiert. Hierfür wurde das Modellierungsmittel der Pakete verwendet, das sowohl Bestandteil der SysML als auch der UML ist. Der Vorteil der vorgegebenen Paketstruktur besteht darin, dass in jedem Projekt die Informationen an der gleichen Stelle wiederzufinden sind [RP10]. Ebenso

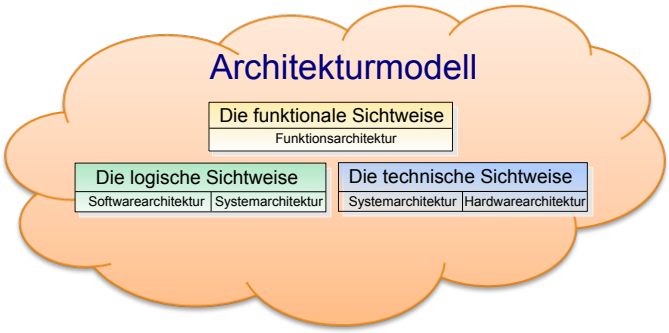


Abb. 4.5. Sichtweisen des Architekturmodells

wird hierdurch die Möglichkeit eröffnet, einzelne Pakete in anderen Projekten wiederzuverwenden, da eine identische Projektstruktur vorliegt. Die Strukturierung unterteilt auf oberster Ebene die beiden Bereiche *System* und *Software* und beinhaltet zusätzlich die beiden notwendigen Profile *SysML* und *AutoMoMe* (vgl. Abbildung 4.6). Die beiden Bereiche *System* und *Software* sind identisch aufgebaut, so dass an dieser Stelle stellvertretend nur der Bereich *System* beschrieben wird.

Wie der Abbildung 4.6 zu entnehmen ist, besitzt der Bereich *System* für die Analyse zunächst eine Unterstruktur. In dem Analysebereich werden sowohl die Hauptfunktionalitäten des Systems (*Ziele*) als auch die Funktionen des Systems (*funktionale Architektur*) beschrieben. Daher ergibt sich die hier aufgezeigte Unterstruktur. Daneben existiert ein Bereich für die Systemarchitektur (*Architektur*) und für deren Teilsysteme (*Subsysteme*). In dem Bereich der Architektur werden die Beziehungen der Teilsysteme definiert, sowie das globale Verhalten das Systems, wie beispielsweise das Startverhalten. In dem Unterbereich der *Subsysteme* sind exemplarisch die Teilsysteme *Alarmanlage* und Betriebssystem (*OS*) dargestellt. In diesen Paketen werden die Teilsysteme und ihre Schnittstellen beschrieben. Dabei werden die Schnittstellen bei den Teilsystemen definiert, die sie anbieten. Ferner ist das Verhalten der Schnittstellen Teil dieser Pakete.

Zusätzlich gibt es in der Projektstruktur noch Pakete, die das Architekturmodell unterstützen bzw. eine Verbindung zu den vorherigen und nachfolgenden Entwicklungsphasen erlauben. Hierdurch wird die Nachverfolgbarkeit sichergestellt. Zur Unterstützung des Architekturmodells dient der Bereich für die Reviews, so dass Reviewkommentare direkt im Architekturmodell abgelegt werden. Auf diese Art und Weise kann ein Review für die Architektur effizient durchgeführt werden. Die beiden letzten Pakete innerhalb der Projektstruktur sind die Pakete *Anforderungen* und *Test*. Diese dienen der Nachverfolgbarkeit (Traceability) der textuellen Anforderungen bzw. der Testfälle, die aus dem Anforderungsmanagementsystem automatisiert importiert werden. Dies Mit-

tel erlaubt die Anbindung an die Entwicklungsphasen der Anforderungsanalyse und des Tests.

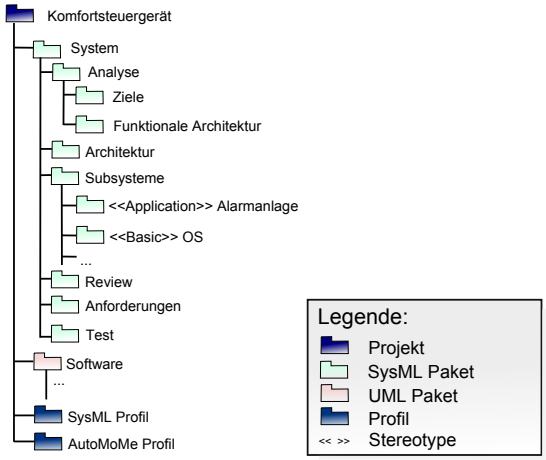


Abb. 4.6. Paketstruktur eines AutoMoMe Projektes

Das Paket (*Subsysteme*) für die Teilsysteme bzw. Komponentenentwicklung wird durch die Verwendung der Stereotypen «Application», «Intermediate» (bei nicht AUTOSAR Systeme) bzw. «Basic» noch einmal unterteilt und entspricht damit der in einem automobilen Steuergerät befindlichen Schichten (siehe Abbildung 2.19). Die Stereotypen sind dabei vom UML Element *Package* abgeleitet (vgl. Abbildung 4.7). Die Verwendung des Stereotyps ermöglicht die eindeutige Zuordnung zu einer der drei Schichten für die im Paket modellierten Komponenten. Dies ist von Belang, da ein Steuergerät aus einer Vielzahl von Komponenten (> 50 bei einem Komfortsteuergerät) besteht. Für eine bessere Übersicht und Navigation innerhalb des Modells ist eine Einteilung in die drei Bereiche vorteilhaft und zahlt sich bei der täglichen Arbeit aus. In diesem Beispiel sind exemplarisch die Komponenten *OS* der Basis-Software und die *Alarmanlagensteuerung* der Applikationsentwicklung zugeordnet.

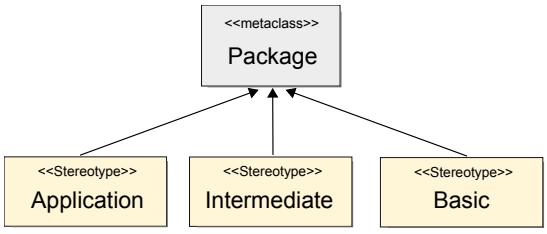


Abb. 4.7. Ableitung der Stereotypen für die Paketstruktur

4.3.1 Die funktionale Sichtweise

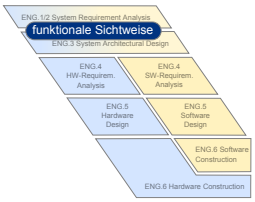


Abb. 4.8. Einordnung der funktionalen Sichtweise in das V-Modell

Die Abbildung 4.8 zeigt die Einordnung der funktionalen Sichtweise in das V-Modell. Innerhalb der funktionalen Sichtweise werden die verschiedenen Funktionalitäten beschrieben, die von einem Steuergerät umzusetzen sind (vgl. Abbildung 4.9). Bislang wurde der funktionalen Sichtweise wenig Beachtung geschenkt, da die Anzahl der Funktionen überschaubar war und in einem Fahrzeug von jeweils einem Steuergerät realisiert wurde. Mittlerweile ist aber ein Paradigmenwechsel eingetreten [DBHW13, HFP+06, MGRK07]. Es ist ein Trend zu erkennen, dass Funktionen nicht mehr nur von einem einzelnen Steuergerät ausgeführt werden, sondern die Steuergeräte interagieren miteinander, um eine Funktion zu realisieren. Dieser Trend ist bei der automobilen Steuergeräteentwicklung in vielen Bereichen festzustellen, da mittlerweile große Steuergeräte aus mehreren Rechenkernen bestehen und die Funktionen verschiedenen Rechenkernen zugeordnet werden. Die funktionsorientierte Sicht ist somit Voraussetzung, um neue Innovationen anstoßen [BS05] und um eine erste Aufwandsschätzung vornehmen zu können [KSSS04].

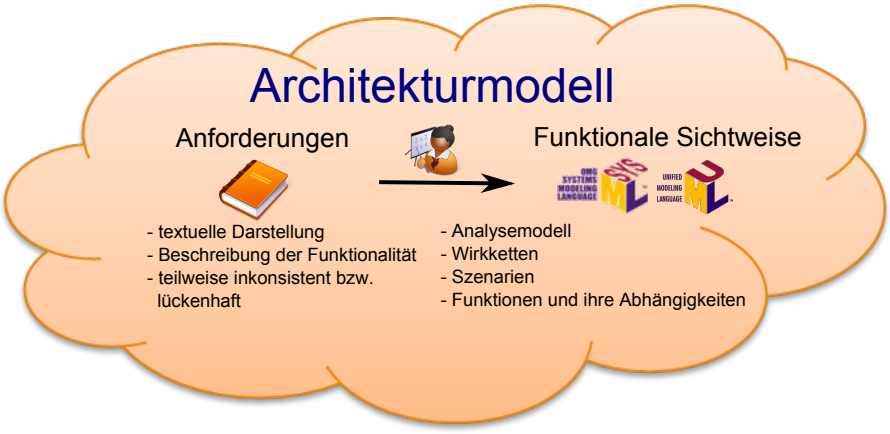


Abb. 4.9. Übergang der Anforderungen zur funktionalen Architektur

Zu Beginn einer Entwicklung geht es darum, die Anforderungen an das System zu analysieren und daraus die notwendigen Funktionalitäten zu definieren. Durch die steigende Komplexität bei den Systemen nimmt ebenso die Größe der Anforderungsbeschreibungen zu [HH04]. Dabei dient die Analyse der

textuellen Anforderungen zur Identifizierung von Inkonsistenzen und Lücken [SFGP05]. Diese sind häufig in den Lastenheften zu finden, da unterschiedliche Abteilungen und damit unabhängig voneinander arbeitende Entwickler auf Seiten der OEMs die Anforderungen erstellen [WW03]. Nur wenn eine frühzeitige Identifizierung gegeben ist, besteht die Chance, in Abstimmung mit dem Automobilhersteller diese zu schließen bzw. auszuräumen [GP04].

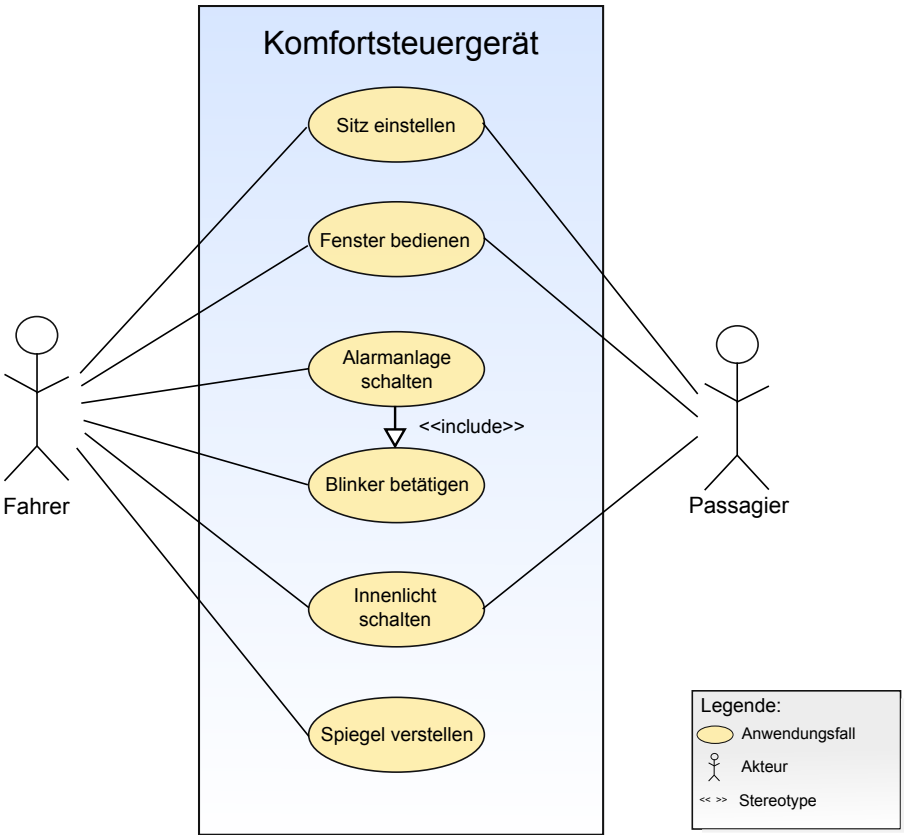


Abb. 4.10. Anwendungsfälle des Komfortsteuergerätes

Die Analyse beginnt zunächst mit der Identifizierung der Ziele des Systems [PK04]. Unter einem Ziel wird die Top-Funktionalität verstanden, die das System zu realisieren hat [PHAB12, PS09, SP10, BSP08]. Sie ist vergleichbar mit einer Packungsbeschreibung eines Produktes [Zör12]. Innerhalb der AutoMoMe werden die Ziele (Goals) des Steuergerätes in Form von Anwendungsfällen (Use-Cases) definiert (vgl. Abbildung 4.10). Jedoch ist die Standardbeschreibungsmöglichkeit der Anwendungsfälle in der SysML für ein automobiles Steuergerät nicht ausreichend [Coc03], da für jeden Anwendungsfall noch

weitere Vor- und Nachbedingungen zu beachten sind, die einzuhalten sind. So ist es nicht hinnehmbar, wenn die Funktionalität *Alarmanlage schalten* bei voller Fahrt aktiviert wird. Von daher darf der Anwendungsfall *Alarmanlage schalten* nur bei einem in Parkposition befindlichen Fahrzeug zur Anwendung kommen (vgl. Abbildung 4.12).

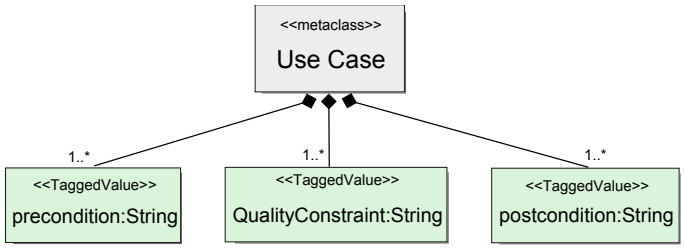


Abb. 4.11. Erweiterungen für ein Anwendungsfall

Ebenso ist es im Bereich der eingebetteten Systeme selbstverständlich, dass Qualitätsanforderungen an die Funktionalitäten gestellt werden. Hier kann exemplarisch auf die zeitlichen Restriktionen verwiesen werden. Bei dem Komfortsteuergerät muss zum Beispiel die Alarmanlage innerhalb von 60 ms aktiviert sein (vgl. Abbildung 4.12). Diese zusätzliche Information muss auf formale Weise im Modell gespeichert werden, damit sie eindeutig ist und bei der späteren Entwicklung umgesetzt wird. Um solche Anforderungen an die Modellierung zu erfüllen, wurde auf die SysML- bzw. UML-Erweiterungsmöglichkeit der Eigenschaftswerte zurückgegriffen (vgl. Abbildung 4.11). Die Eigenschaftswerte erweitern in AutoMoMe die Anwendungsfälle mit den benötigten Informationen. Beispielhaft ist für das Ziel *Alarmanlage schalten* die Analyse erfolgt und folgende Bedingungen konnten identifiziert werden (vgl. Abbildung 4.12).

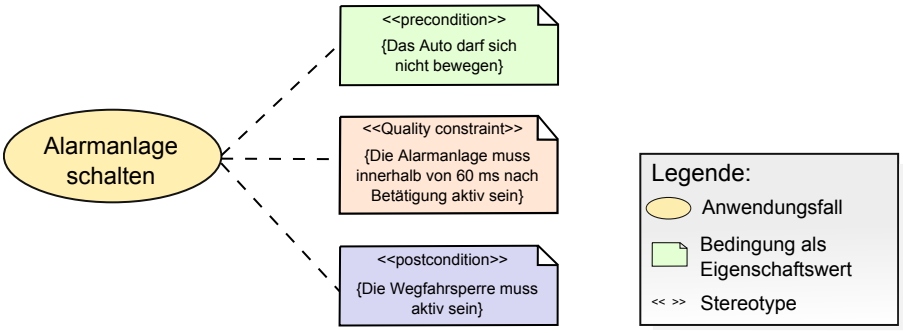


Abb. 4.12. Vor- und Nachbedingungen des Anwendungsfalls *Alarmanlage schalten*

Die Anwendungsfälle beschreiben die wichtigsten Funktionalitäten des Systems. Sie stellen die Managementsicht auf das System dar, da hier die zentralen Funktionalitäten beschrieben und dokumentiert werden. Von daher sind die zuvor erwähnten Anwendungsfälle noch sehr abstrakt und müssen im Nachfolgenden weiter detailliert werden. Hierzu werden in AutoMoMe Aktivitätsdiagramme eingesetzt. Die Aktionen innerhalb der Aktivitätsdiagramme stellen die einzelnen Funktionen bzw. Subfunktionen dar, die für die Erreichung des Ziels benötigt werden. Für die Modellierung wird ein Aktivitätsdiagramm verwendet, da in diesem sowohl Kontroll- als auch Datenflüsse modellierbar sind [FMS12]. Dies entspricht in etwa der Verfahrensweise der Arbeitsgruppe *Funktionale Architektur für Systeme (FAS)* [LW10, KLW11]. Jedoch wurde das Vorgehen an die Gegebenheiten der Steuergeräteentwicklung – vor allem der Echtzeit (siehe Kapitel 4.3.3 und der Wirkketten) – angepasst.

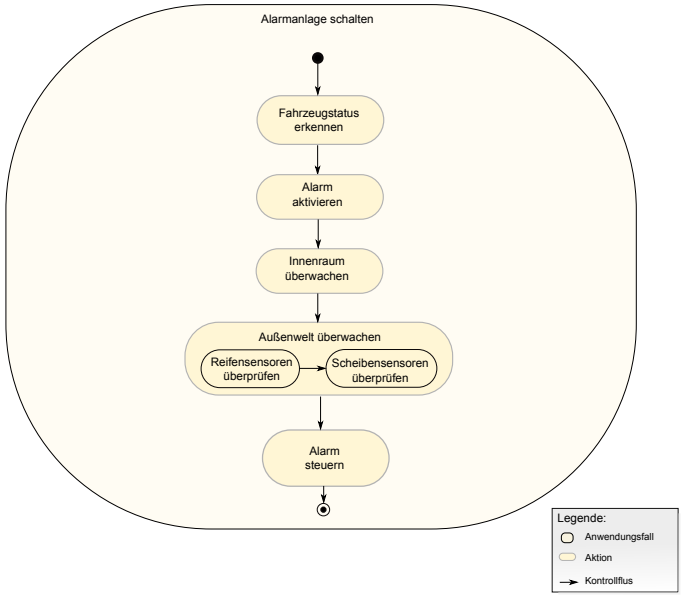


Abb. 4.13. Dekomposition des Ziels *Alarmanlage schalten* in einzelne Funktionen

Das Ziel *Alarmanlage schalten* ist zunächst noch sehr abstrakt. Die genauen Funktionen, die dieses Ziel beinhalten, sind nicht sofort erkennbar. Von daher ist in der Abbildung 4.13 die Dekomposition des Ziels in einzelne Funktionen bzw. Unterfunktionen dargestellt. Hierdurch wird ersichtlich, dass zuerst die Funktion *Fahrzeugstatus erkennen* benötigt wird, um festzustellen, ob die Alarmanlage aktiviert werden kann oder nicht, da sie nur bei einem parkenden Auto aktiviert werden darf. Ferner ist zu erkennen, dass zwei Funktionen existieren, die die notwendigen Eingabedaten für die Steuerung der Alarmanlage bereitstellen (*Innenraum* und *Außenwelt überwachen*).

Bei der Dekomposition werden für jedes Ziel Funktionen und Unterfunktionen identifiziert. Es ist aber häufig der Fall, dass gleiche Funktionen bei unterschiedlichen Zielen benötigt werden. Ebenso muss die Möglichkeit gegeben sein, für einzelne Funktionen das Verhalten, z. B. in Form von Zustandsdiagrammen, zu beschreiben. Dazu werden funktionale Blöcke – sogenannte «FunctionalBlock» – genutzt. Ihnen werden einzelne Funktionen zielübergreifend zugeordnet. Dies ist in Abbildung 4.14 dargestellt, wo der funktionale Block *Fahrzeugstatus erkennen* sowohl von der Alarmanlagen- als auch von der Lichtsteuerung genutzt wird. Der Austausch von Daten zwischen den funktionalen Blöcken erfolgt durch Objektflussports (FlowPorts) und Flüsse (Flows).

Die funktionalen Blöcke sind weiterhin Ausgangspunkt für die Suche nach geeigneten bereits realisierten Teilsystemen. So existieren bereits in den jetzigen Steuergeräten oftmals Teilsysteme, die sich bewährt haben und folglich wiederverwendet werden sollten. Der Vorteil bei der Wiederverwendung ist, dass dadurch Arbeitszeit und -kosten eingespart werden [RR09]. Deshalb ist anzustreben, möglichst viele Teilsysteme durch bereits existierende zu realisieren [LSL+09, FF95]. Die Schwierigkeit bei der Wiederverwendung ist die Identifikation von geeigneten Teilsystemen. Sie beginnt in der AutoMoMe mit den funktionalen Blöcken. Anhand der Anforderungen kann auf das Verhalten geschlossen werden. Dies geschieht derzeit noch manuell. Ebenso werden die Ein- und Ausgabedaten herangezogen. Anhand dieser kann festgestellt werden, ob die bestehenden Teilsysteme funktional sind und mit den erforderlichen Daten übereinstimmen oder nicht. Falls letzteres nicht zutrifft, wird noch überprüft, ob eine Transformation den Einsatz des Teilsystems ermöglicht. Somit wird durch die funktionale Architektur die Wiederverwendung gefördert.

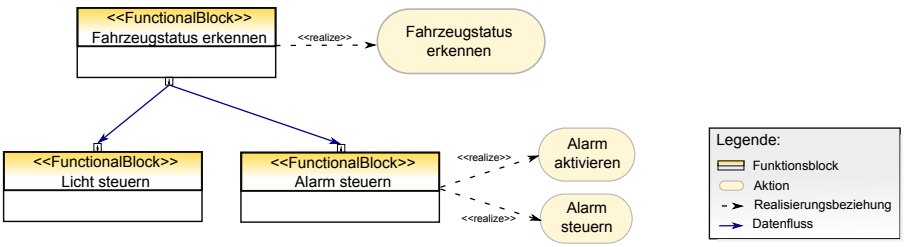


Abb. 4.14. Zuordnung zu funktionalen Blöcken

Falls bereits ein Verhalten für einen funktionalen Block vorliegt, so besteht dies häufig aus Zuständen und Ereignissen. Das zugehörige Zustandsdiagramm wird dem funktionalen Block zugeordnet. Die standardmäßigen Zustandsdiagramme der SysML bzw. der UML sind für die Entwicklung von automobilen Steuergeräten aber nicht ausreichend. Bei eingebetteten Systemen müssen zusätzlich die zeitlichen Informationen bei den Zuständen und den Zustandsübergängen beachtet werden. Diese können bei den bisherigen Zus-

tandsdiagrammen nur informal in Form von Kommentaren angegeben werden (vgl. Abbildung 4.15). So ist es mit den derzeitigen Zustandsdiagrammen kaum möglich, das Verlassen eines Zustandes nach einer gewissen Zeit (wie hier mit 500 ms beschrieben) zu modellieren. Für den Ansatz AutoMoMe werden daher die Zustandsdiagramme erweitert, so dass sie den Real-Time Statecharts (RTSCs) des Sonderforschungsbereiches 614 entsprechen (siehe Kapitel 2.4).

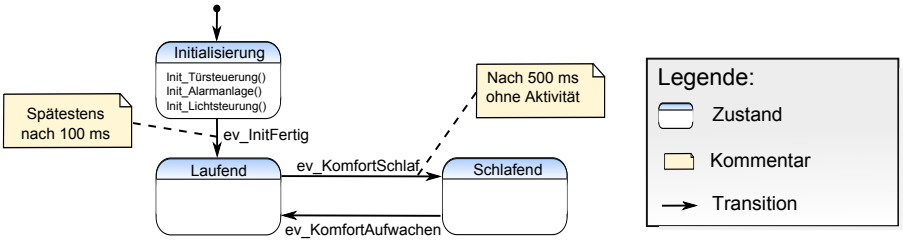


Abb. 4.15. Zustandsdiagramm zur Spezifikation des Aufwachverhaltens des Komfortsteuergerätes

Durch die funktionalen Blöcke wird die funktionale Architektur abgebildet. Sie beschreibt, welche Funktionen untereinander Daten austauschen und somit abhängig voneinander sind. Sie dient als Basis für die weitere Systementwicklung. Die funktionale Architektur ist dabei in unterschiedlichen Detaillierungsstufen (Funktionen und funktionalen Blöcken) aufgeteilt und zwar ähnlich den Detaillierungen im SPES 2020 Metamodell (vgl. Kapitel 3.11) [SFB09, PHAB12].

4.3.2 Erweiterung des Architekturmodells um Echtzeit

Um die Funktionalität der Real-Time Statecharts insbesondere die zeitliche Modellierung in das Architekturmodell zu integrieren, ist zunächst zu analysieren, welche Funktionalitäten bereits zur Verfügung stehen und welche zusätzlichen Funktionen von den RTSCs zu integrieren sind, um die zeitlichen Abläufe bereits innerhalb der funktionalen Architektur auf formale Weise zu modellieren. Dazu sind in der Tabelle 4.1 die Funktionalitäten der Zustandsdiagramme und der Real-Time Statecharts gegenübergestellt.

Eigenschaft	SysML/UML Statechart	Real-Time Statechart
Zustände mit Entry, Do und Exit Methoden	X	X
Hierarchische Schachtelung	X	X
Parallele Zustände	X	X
History Zustände	X	X
Uhren		X
Invarianten	X	X
Zeitinvarianten		X
Uhren-Resets		X
Synchrone Kommunikation		X
Asynchrone Kommunikation	X	X
Guards	X	X
Ereignisse	X	X
Prioritäten		X
Deadlines		X
Worst Case Execution Times (WCET)		X

Tabelle 4.1. Vergleich SysML Statecharts mit Real-Time Statecharts [SFB09]

Der Vergleich lässt erkennen, dass die Zustandsdiagramme der SysML bzw. der UML Defizite im Bereich der Modellierung von Uhren, Deadlines und Ausführungszeiten aufweisen. Gerade aber für die Modellierung von komplexen und vor allem von vernetzten eingebetteten Systemen werden diese Modellierungsmöglichkeiten jedoch benötigt. So ist beispielsweise zu modellieren, dass nach einer vorgegebenen Zeit der Aktivität sich die Alarmanlage wieder auszuschalten hat. Ein solcher Modus ist wichtig, da die Alarmanlage nur im abgeschalteten Zustand des Fahrzeuges betrieben werden darf. In bestimmten Zeitabständen überprüfen die Sensoren, ob ein Einbruchsversuch stattgefunden hat oder nicht. Der Übergang in den Ruhemodus ist auch deshalb von Wichtigkeit, um die Batterie des Fahrzeuges zu schonen. Sollte nach einer Überprüfung das Steuergerät nicht in den Ruhemodus wechseln, wird Energie benötigt und verbraucht, was letztendlich dazu führen kann, dass unter Umständen die Batterie des Fahrzeuges bei der nächsten Startphase nicht über ausreichend Strom verfügt. Die Effektivität dieses oder eines ähnlichen Szenar-

ios nimmt stetig zu, da immer mehr Steuergeräte in einem Automobil verbaut werden.

Die RTSCs besitzen die benötigten Funktionalitäten für dieses Szenario, um die Anforderungen entsprechend zu spezifizieren. Nachfolgend wird das Konzept beschrieben, durch das die Statecharts der SysML erweitert werden, um die zeitlichen Konzepte der Real-Time Statecharts zu realisieren. Die Erweiterungen der Real-Time Statecharts, die sich mit der Modellierung von hybriden Systemen und deren Rekonfiguration beschäftigen [GBS04], werden in dieser Arbeit außer Acht gelassen, da die Modellierung des kontinuierlichen Verhaltens nicht im Fokus der Arbeit steht.

Zustandsdiagramme sind sowohl Bestandteil der SysML als auch der UML. In beiden Modellierungssprachen wird dasselbe Metamodell für die Zustandsdiagramme genutzt [Gro10]. Dies vereinfacht das weitere Vorgehen, da nur ein Metamodell analysiert werden muss, um die Funktionalität der RTSCs sowohl auf System- als auch auf Softwareebene zur Verfügung zu stellen. Bei der Analyse des Metamodells wurden Erweiterungspunkte identifiziert, an denen die Konzepte der Real-Time Statecharts integriert werden konnten. Die beiden Elemente *State* und *Transition* sind die zentralen Bestandteile der Zustandsdiagramme und somit die Startpunkte für die Erweiterung. Die beiden Elemente bieten aber keine Modellierungsmöglichkeiten für Zeitvorgaben wie sie z. B. mittels Uhren, Deadlines etc. modelliert werden. Von daher besteht nur die Chance, über UML Constraints die zeitlichen Informationen in Form von Bedingungen an eine Transition oder eines Zustandes zu spezifizieren. Diese Vorgehensweise ist aber keine formale Spezifikation von Zeiten und kann daher von den Entwicklern unterschiedlich beurteilt oder bewertet werden. Diese Unzulänglichkeit wird noch dadurch verstärkt, dass die Zeiten auf Grund der fehlenden formalen Notation nicht automatisiert weiterverwendet werden können, beispielsweise als Grundlage einer formalen Verifikation. Von daher sind die derzeitigen Modellierungsmöglichkeiten von Zeiten für eine zukünftige Modellierung nicht ausreichend.

Zur Vermeidung von Irritationen und zur automatisierten Weiterverwendung der Daten muss die formale Notation der RTSCs auf die Zustandsdiagramme übertragen werden. Hierzu wurden zunächst die beiden Metamodelle hinsichtlich ihrer Funktionalität analysiert, die die Elemente der Zustandsdiagramme und der Real-Time Statecharts definieren. Es bleibt festzuhalten, dass die Real-Time Statcharts alle - mit Ausnahme des After- und des When-Elementes - Eigenschaften der Zustandsdiagramme aufweisen. Somit ist kein Grund gegeben, die Real-Time Statecharts nicht einzusetzen. Die Ausdrucksmöglichkeiten dieser beiden fehlenden Elemente sind implizit in der Semantik der Real-Time Statecharts vorhanden. Jedoch sind sie in Elementen verborgen, die eine darüber hinausgehende Spezifikation von Zeiten erlauben. Von daher lassen sie sich ohne weiteres nach modellieren.

Bei der Analyse der Metamodelle wurde des Weiteren festgestellt, dass durch die Erweiterung des SysML/UML Metamodells mit den nachfolgenden Elementen die Funktionalität der Real-Time Statecharts übernommen werden kann. Ein SysML bzw. UML Zustand muss um die nachfolgenden Attribute ergänzt werden, damit eine Spezifikation von Zeiten, wie in den Real-Time Statecharts vorhanden, möglich ist [Bur02]:

- Zeitinvarianten
- Uhren-Resets, die mit den entry()- und exit()- Methoden assoziiert werden
- Worst Case Execution Times (WCET) zu den entry()-, do()- und exit()-Methoden
- Periodenintervall für die do()-Methode

Neben der Erweiterung des Elementes *Zustand* muss auch das Element *Transition* erweitert werden. Hier sind Anpassungen um folgende Konstrukte erforderlich:

- Zeitguards
- Uhren-Resets
- Prioritäten
- Deadlines
- Worst Case Execution Times (WCET)
- Synchronisationskanal und Synchronisationsart

Auf der Grundlage dieser Anpassungen bzw. Ergänzungen entsteht ein Metamodell eines erweiterten Zustandsdiagramms, in dem die Möglichkeit zur Modellierung von Zeiten und verteilter Kommunikation machbar ist (vgl. Abbildung A.1 im Anhang). Die Anwendung der im Metamodell spezifizierten Elemente ist der Abbildung 4.16 zu entnehmen.

In der Darstellung 4.16 ist das Verhalten der Alarmanlage als Teil des Komfortsteuergeräts in Form eines Real-Time Statecharts dargestellt, wie es bereits in der Abbildung 4.15 in Form eines SysML Zustandsdiagramms abgebildet wurde. Die Vorteile des RTSCs sind sofort erkennbar. Die Zeiten werden dabei auf formale Art und Weise in Form von Uhren (t_1,t_2 und t_3) spezifiziert. So ist festgelegt, dass der Initialisierungszustand nach 100 ms verlassen sein muss. Des Weiteren ist definiert, dass der Zustand *Laufend* nach 500 ms der Inaktivität verlassen wird. Hierdurch wird erreicht, dass, falls kein Alarm innerhalb dieser Zeit auftritt, das Steuergerät wieder in den Ruhemodus wechselt. So ist sichergestellt, dass wirklich nur die notwendigste Energie zur Überwachung des Fahrzeugs verbraucht wird. Ferner ist spezifiziert, dass der Zustand *Schlafend* alle 5 Sekunden unterbrochen wird, um die Alarmanlage wieder zu aktivieren, um eine Überprüfung durchzuführen.

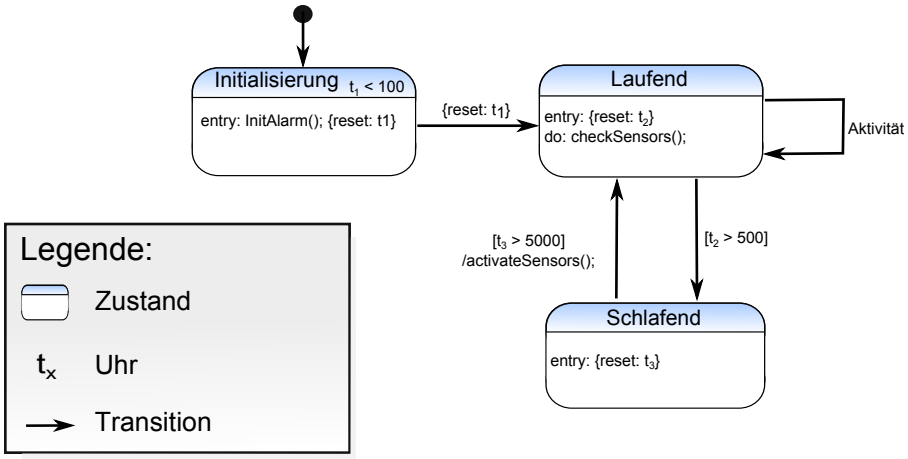


Abb. 4.16. Verhalten der Alarmanlage (Teilkomponente des Komfortsteuergerätes) spezifiziert als RTSC

Modellierung des Kommunikationsverhaltens durch RTSCs

Eine weitere Anforderung an eine zukünftige Entwicklung im Bereich der automobilen Steuergeräte betrifft die Modellierung der Zusammenarbeit von verteilten Systemen. Dies kann sowohl zwischen verschiedenen Rechenkernen innerhalb eines Steuergerätes als auch zwischen verschiedenen Steuergeräten erfolgen. Die Modellierung des Kommunikationsverhaltens ist von Bedeutung, da sie einen Teil des Gesamtverhaltens – insbesondere des zeitlichen Verhaltens – ausmacht.

Eine oftmals verwendete Möglichkeit, das Kommunikationsverhalten zu spezifizieren, ist der Einsatz von Protokollstatecharts basierend auf den UML Zustandsdiagrammen [RQZ07]. In diesen Diagrammen werden die jeweiligen Zustände bei der Kommunikation mit anderen Komponenten modelliert. Jedoch ist, wie bereits bei den Zustandsdiagrammen zuvor beschrieben, noch nicht die Möglichkeit gegeben, die zeitlichen Informationen adäquat zu spezifizieren. Um dies zu erreichen, sind zusätzliche zeitliche Informationen hinzuzufügen, die den Kommunikationsaustausch exakt definieren. Von daher werden in AutoMoMe zur Beschreibung des Kommunikationsverhaltens RTSCs verwendet.

Bei der Modellierung des Kommunikationsverhaltens spielen die verschiedenen Kommunikationsbusse natürlich eine besondere Rolle, da – ausgehend von ihren Eigenschaften – unterschiedliche Zeiten für die Übertragung der Daten möglich sind. So ist es bei einem CAN Bus, der nur über Ereignisse gesteuert wird, schwierig, die Antwortzeit zu bestimmen, da sie immer abhängig von der gerade auftretenden Buslast, d. h. den zu verschickenden Nachrichten ist.

Einfacher ist dies beim LIN Bus zu spezifizieren, da dieser feste Zeitvorgaben hat, wann Informationen ausgetauscht werden. Dies kann mit den erweiterten Zustandsautomaten modelliert werden. In Abbildung 4.17 ist ein Protokollstatechart für den LIN-Bus, beispielsweise bei der Kommunikation zur Türsteuerung, dargestellt.

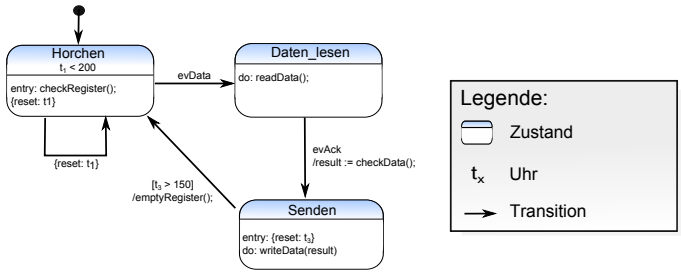


Abb. 4.17. Protokollzustandsautomat mithilfe von Real-Time Statecharts

In dem Protokollstatechart ist das Verhalten bei der Kommunikation wie folgt spezifiziert: alle 200 ms wird überprüft, ob das entsprechende Empfangsregister gefüllt ist oder nicht. Sofern es gefüllt ist, wird signalisiert, dass Daten für das Steuergerät übertragen werden. Gleichzeitig wird das Ereignis *evData* ausgelöst und in den Zustand *Daten lesend* übergegangen, um die Nachricht zu lesen und zu verarbeiten. Nach Abschluss des Vorgangs wird das Ereignis *evAck* ausgelöst und so dann in den Zustand *senden* übergegangen. Gleichfalls wird beim Übergang überprüft, ob die Nachricht korrekt empfangen wurde. Das Ergebnis wird nun an den Sender übermittelt. Der Sendevorgang darf dabei höchstens 150 ms beanspruchen.

Durch die Verwendung von Protokollzustandsautomaten ist es machbar, das Kommunikationsverhalten für ein automobiles Steuergerät zu spezifizieren. Sofern für den Sender als auch für den Empfänger und den Bus entsprechende Automaten eingesetzt werden, ist die Kommunikation vollständig beschrieben. In der Mechatronic UML wird dies auch als Echtzeitkoordinationsprotokoll bezeichnet. In dem Protokoll wird für einzelne Kommunikationsbeziehungen durch die Real-Time Statecharts das Verhalten und die zeitlichen Abfolgen bei der Kommunikation von verschiedenen Komponenten abgebildet [SFB09].

Ein Echtzeitkoordinationsprotokoll abstrahiert von den Real-Time Statecharts, die für die Kommunikationsspezifikation verwendet werden [GHH08]. Die Abbildung 4.18 veranschaulicht, welche verschiedenen Real-Time Statecharts bei einem Koordinationsprotokoll genutzt werden. Zunächst wird jedem Port ein Statechart zugewiesen. Hierdurch wird das Verhalten der Rolle des jeweiligen Ports bei der Kommunikation definiert. Zugleich erhält jede Verbindung auf logischer Ebene ein RTSC, das den Kommunikationsbus symbolisiert. Da es verschiedene Kommunikationsbusse gibt, werden an dieser

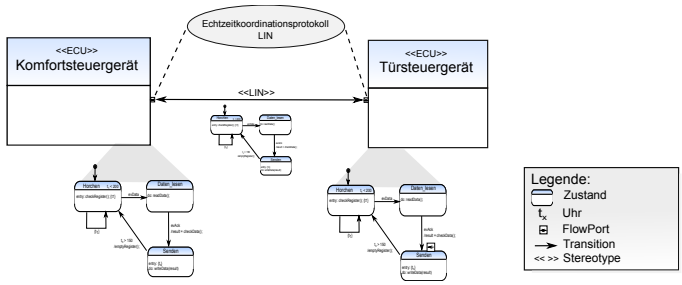


Abb. 4.18. Nutzung der Real-Time Statecharts bei Echtzeitkoordinationsprotokolle

Stelle auch unterschiedliche Real-Time Statecharts verwendet. Zukünftig ist es durchaus überlegenswert, verschiedene Echtzeitkoordinationsprotokolle mit den entsprechenden Real-Time Statecharts vorzuhalten (Bibliothek-Konzept), um für eine zügige Entwicklung zu sorgen. Insgesamt erfüllen die Echtzeitkoordinationsprotokolle die Vorgaben zur Spezifikation der Schnittstellen (Anforderung drei für die zukünftige Steuergeräteentwicklung 4.1) und der Modellierung von verteilten Systemen (Anforderung vier für die zukünftige Steuergeräteentwicklung 4.1) an eine zukünftige Entwicklung.

4.3.3 Erweiterung der Architektur um Wirkketten

Die automobilen Steuergeräte realisieren immer neuere und komplexere Funktionen. Hierfür sind immer mehr Komponenten bei der Verarbeitung erforderlich. Bei der Betrachtung des Systems aus der funktionalen Sichtweise sind noch keine Komponenten vorhanden, sondern nur die einzelnen Funktionen und die Anforderungen an diese. Somit sind zunächst die jeweiligen Anforderungen an die Funktionen zu analysieren und ggf. eine weitere Verfeinerung der Funktionen vorzunehmen. Die Qualitätsanforderungen geben dabei das Funktionsverhalten aus einer Art Black-box Sicht wieder. Es existieren Auslöser beispielhaft von einem Sensor und darauf aufbauende Handlungen z. B. bei einem Aktuator. Die dazwischen befindlichen Schritte sind aus den Anforderungen nicht ersichtlich und müssen durch das vorhandene Erfahrungswissen des Entwicklers ergänzt werden. Man spricht hierbei auch von einer Wirkkette (siehe Definition Wirkkette). Bei dem Komfortsteuergerät kann dies beispielsweise das Verschließen des Autos mittels einer Fernbedienung (Eingangsgröße) mit gleichzeitiger Aktivierung der Alarmanlage (Ausgangsgröße) sein.

Definition (Wirkkette). Eine Wirkkette (engl. event chain) ist eine Abfolge von Ereignissen, die nacheinander ausgeführt werden, um eine Funktion zu realisieren. Dabei können die atomaren Ereignisse durchaus unterschiedlichen Systembestandteilen (also auch Hardwareknoten) zugeordnet sein. Normalerweise spezifiziert im Automobilbereich eine Wirkkette die Ereignisse von einem Sensor bis hin zu einem Aktor. Dazwischen können beliebig viele Ereignisse liegen [RHER07].

Die Wirkketten werden für systemrelevante oder aber auch für kritische Funktionalitäten genutzt. Aus der Modellierung der einzelnen Wirkketten wird ersichtlich, welche verschiedene Funktionen bei der Abarbeitung beteiligt sind. Ebenso ist der Datenfluss zu erkennen und kann so später analysiert werden. Die Modellierung der Wirkketten ändert sich im Laufe der Entwicklung. Zunächst werden in der funktionalen Sichtweise die einzelnen Funktionen betrachtet. Bei der späteren Entwicklung werden die Funktionen durch Komponenten ersetzt, die die Funktionen später realisieren.

Bei den Wirkketten ist es nicht unüblich, dass die Abarbeitung zwischen Eingangs- und Ausgangssignalen innerhalb einer festgesetzten Zeit zu erfolgen hat. Dies muss bei der Modellierung der Wirkketten berücksichtigt werden, da die zeitlichen Vorgaben sich aus den Anforderungen des Kunden ergeben und von daher ein wichtiges Abnahmekriterium für das spätere Gesamtsystem ist. Um die Einhaltung der vorgegebenen Zeiten in den frühen Entwicklungsphasen zu verifizieren, werden die relevanten Wirkketten durch eine Zeitanalyse über-

prüft (vgl. Kapitel 5). Besonders für kritische Pfade ist dieses Vorgehen opportun, da durch die Simulation bereits in den frühen Phasen (zeitliche) Fehler identifiziert und behoben werden können. Dies spart Zeit und dementsprechend Entwicklungskosten [Bür12].

Neben der Zeitmodellierung für eine Wirkkette muss bei der funktionalen Sichtweise ebenso berücksichtigt werden, dass die Zuordnung der einzelnen Funktionen zu Komponenten noch nicht zwingend gegeben ist. Dies bedeutet, dass zunächst nur einzelne Funktionen und die zu übertragenden Daten nur sehr abstrakt bekannt sind, z. B. Schlüsseldaten, die zwischen Benutzer und Fahrzeug ausgetauscht werden. Wie dies im Detail aussieht, wird in der logischen Sichtweise des Systems definiert. Die Modellierung der Wirkketten muss demzufolge so flexibel sein, dass sie in der frühen Phase (funktionale Sichtweise) die Modellierung mit Funktionen und abstrakten Daten erlaubt. In den späteren Entwicklungsphasen (logische Sichtweise) muss aber die Modellierung mittels Komponenten und Schnittstellen erlaubt sein, die genau definieren, welche Informationen zwischen den Komponenten ausgetauscht werden [Wei08].

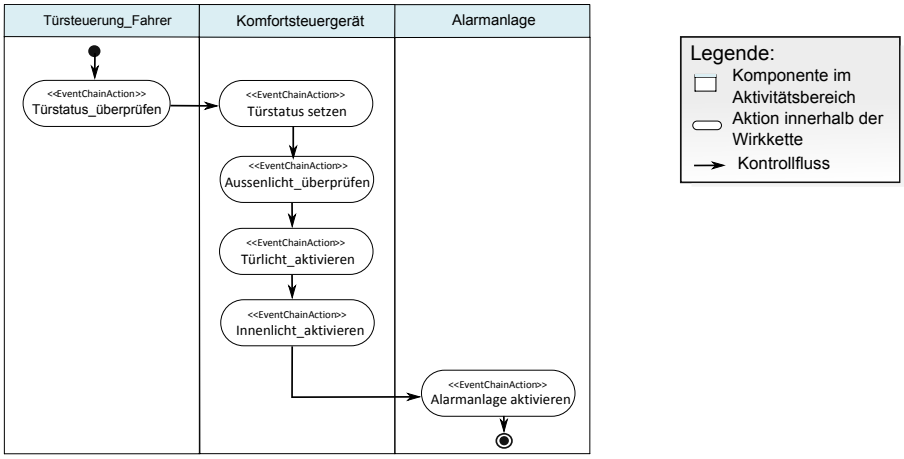


Abb. 4.19. Wirkkette, die die Aktivitäten beim Öffnen einer Tür beschreibt

Weder die SysML noch die UML weisen – wie zuvor beschrieben –eine explizite Modellierung von Wirkketten auf. Von daher wurde im Rahmen der AutoMoMe eine Erweiterung zur Modellierung von Wirkketten erarbeitet. Durch die gestellten Anforderungen für die Modellierung der Wirkketten ist eine Erweiterung der Aktivitätsdiagramme vorteilhaft, wie sie ebenfalls in [HG10] benutzt wird. Hierbei kommt eine leichtgewichtige Ergänzung durch Stereotypen und Eigenschaftswerten zum Einsatz [BH08, Sel07]. Die erweiterten Aktivitätsdiagramme haben den Vorteil, dass sie sowohl Kontroll- als auch Datenflüsse abbilden. Des Weiteren ist die Modellierung von sowohl abstrak-

ten Elementen der Architektur als auch die Referenzierung von konkreten Operationen möglich. Sie setzen somit alle gestellten Anforderungen mittels der Erweiterungen um.

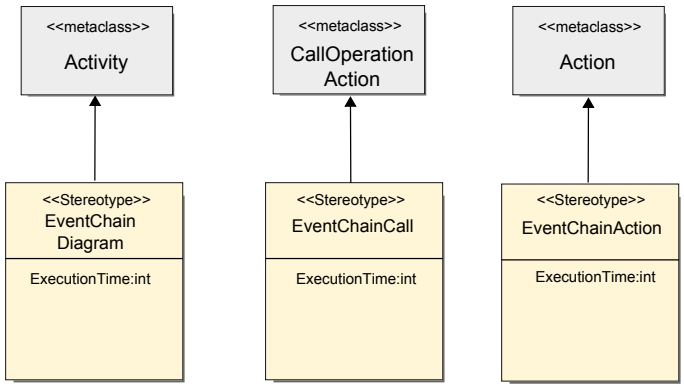


Abb. 4.20. Erweiterungen für die Wirkkettenmodellierung

Als Erweiterungen stehen Stereotypen für die Aktionen im Aktivitätsdiagramm zur Verfügung («EventChainAction» bzw. «EventChainCall») (vgl. Abbildung 4.20). Der Stereotyp «EventChainAction» wird dabei für Funktionen (Aktionen) eingesetzt, die noch nicht innerhalb der logischen Sichtweise in Form von Operationen definiert sind. Im Beispielfall sind es die Funktionen, die bei der Verfeinerung des Anwendungsfalls *Alarmanlage schalten* analysiert wurden. Wenn für die Funktion bereits eine Realisierung existiert, dann wird der Stereotyp «EventChainCall» verwendet, der einen Link auf die entsprechende Operation realisiert. An beiden Stereotypen hängen Eigenschaftswerte, die die Dauer der einzelnen Funktionen vorgeben.

Der Einsatz eines erweiterten Aktivitätsdiagramms bietet darüber hinaus den Vorteil, dass der Entwickler beim Übergang von der funktionalen zur logischen Sichtweise unterstützt wird. So kann im Aktivitätsbereich (Swimlanes) festgelegt werden, welche Funktionen zusammen in einer Komponente zu realisieren sind. Ein Aktivitätsbereich ermöglicht die Zusammenfassung von Aktivitäten mit gemeinsamen Eigenschaften wie beispielsweise Rollen, Verantwortlichkeiten oder Zugehörigkeiten zu einer Hardware [RQZ07]. In diesem Beispiel wird die Zugehörigkeit zu einer Komponente modelliert.

So ist festgelegt, welche Komponenten in der logischen Architektur entstehen. Des Weiteren wird für die Wirkkette schon ein Datenfluss modelliert, der die Datenübergabe zwischen den zu realisierenden Komponenten beschreibt. Aus diesen Datenübergaben werden sodann in der logischen Architektur die Schnittstellen der Komponenten festgelegt. Die (semi-) formale Notation erlaubt es außerdem zu überprüfen, ob die Spezifikation der funktionalen mit

der logischen Sichtweise übereinstimmt. Dies kann z. B. in Form von automatisierten Überprüfungen (Checks) erfolgen [MH13] (vgl. Kapitel 7).

In der Abbildung 4.19 wird eine Wirkkette nach der eben entwickelten Erweiterung präsentiert. Sie spezifiziert die Funktionen, die beim Öffnen bzw. Schließen einer Tür erforderlich sind. So wird zunächst der Status der Tür zum Komfortsteuergerät gesendet, welches wiederum intern den Status setzt und einen Timer aktiviert, um nach einer festgelegten Zeit ggf. die Außenlichtfunktionen wieder zu deaktivieren. Danach wird das Licht in den Türen eingeschaltet, bevor das Außenlicht überprüft wird. Anschließend wird das Innenlicht in Betrieb genommen und erst dann wird der Status der Alarmanlage geändert. Diese Funktionen werden immer dann aktiviert, wenn sich der Status einer Tür ändert. Die zeitliche Vorgabe für die Wirkkette wird durch Eigenschaftswerte spezifiziert (vgl. Abbildung 5.16). In diesem Beispiel müssen alle Funktionen innerhalb von 250 ms ausgeführt sein. Die Wirkkette inklusive der zeitlichen Anforderung kann innerhalb der Systemsimulation (vgl. Abschnitt 5) frühzeitig überprüft werden.

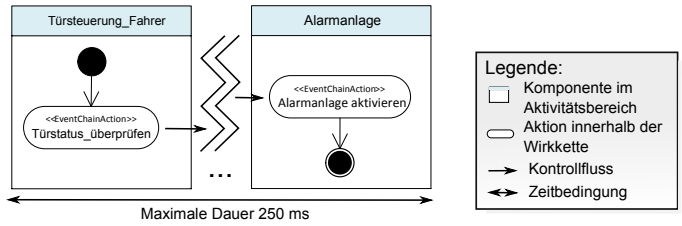


Abb. 4.21. Zeitliche Anforderung an die Wirkkette

Das Ergebnis der funktionalen Sichtweise im Architekturmodell gibt somit einen Überblick über die wesentlichen Funktionalitäten des Systems, die bereits in einzelne Funktionen unterteilt sind. Für die systemrelevanten und kritischen Funktionen sind entsprechende Wirkketten definiert, die bereits frühzeitig auf zeitliche Probleme und später ebenso auf Datenkonsistenz überprüft werden können. Des Weiteren werden die Artefakte der funktionalen Sichtweise zur Erstellung der logischen Sichtweise herangezogen.

4.3.4 Die logische Sichtweise

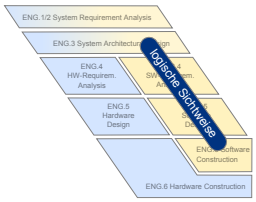


Abb. 4.22. Einordnung der logischen Sichtweise in das V-Modell

Die Abbildung 4.22 zeigt die Einordnung der logischen Sichtweise in das V-Modell. Die logische Sichtweise baut auf der funktionalen Sichtweise auf und detailliert diese, so dass eine logische Architektur entsteht. Die logische Architektur beschreibt dabei die Struktur und das globale Verhalten des Systems bzw. der Software auf einer hardwareunabhängigen Art und Weise (siehe Definition Logische Sichtweise). Die enthaltenen Elemente variieren dabei sehr in ihrem Detaillierungsgrad. Von daher bestehen unterschiedliche Abstraktionsstufen innerhalb der logischen Sichtweise. Ebenso sind aus diesem Grund viele Verfeinerungen innerhalb der logischen Architektur zu finden.

> **Definition (Logische Sichtweise).** Die logische Sichtweise detailliert die funktionale Sichtweise der Architektur und beschreibt die Struktur und das Verhalten des Systems und seiner (Teil-)Systeme. Die logische Sichtweise der Architektur erfolgt dabei hardwareunabhängig [PHAB12], so dass sie bei unterschiedlichen Steuergeräten wiederverwendet werden kann.

Das Ergebnis der Entwicklung eines automobilen Steuergerätes muss ein AUTOSAR konformes System sein. Der Einsatz des AUTOSAR Standards bereits zu Beginn der Entwicklung scheidet aus, da es in diesem nicht möglich ist, bestimmte Aspekte der logischen Architektur abzubilden. So ist es in AUTOSAR nicht möglich, eine vollständige Systemarchitektur abzubilden. Exemplarisch kann hier auf mechanische Bestandteile verwiesen werden, die durch Hydraulik betrieben werden. So kann der AUTOSAR Standard nur die Teile des Systems beschreiben, die mit der Software zusammenarbeiten. Ferner ist kein Beschreibungsmittel für das globale Verhalten vorhanden. Dies ist aber ein wesentlicher Bestandteil der System- als auch der Softwarearchitektur. Von daher kann erst zu einem späteren Entwicklungsschritt innerhalb der logischen Sichtweise der Übergang nach AUTOSAR vollzogen werden (vgl. Kapitel 6). Somit ist ein anderes modellbasiertes Vorgehen für den Ansatz AutoMoMe notwendig, welches im Folgenden näher beschrieben wird.

Systemarchitektur im Architekturmodell

Zunächst geht es bei der logischen Sicht darum, einzelne Funktionen zu Teilsystemen zusammenzufassen und daraus eine erste Struktur des zu entwickelnden

Systems zu erstellen. Die Definition eines Systems wird in der nachfolgenden System Definition gegeben. Bei der Modellierung des Systems ist es unerheblich, in welcher Disziplin die einzelnen Teilsysteme später realisiert werden. Es geht vielmehr darum, dass eine interdisziplinäre (semi-) formale Spezifikation des Systems erfolgt. Hieraus leitet sich einerseits die Struktur des Systems ab. Andererseits sind die Schnittstellen – vor allem zwischen den einzelnen Disziplinen – zu erkennen. Diese Festlegung vermeidet später Irrtümer und Fehler, die auf Grund einer nicht vorgenommenen Festlegung ggf. entstehen können [WFO09, KDHM13].

Definition (System). Ein System kann nach [IEC99] definiert werden als „Menge von Elementen, die nach einem Entwurf in gegenseitiger Beziehung [zueinander] stehen. Ein Element eines Systems kann zugleich ein anderes System sein, genannt Teilsystem, welches ein steuerndes oder ein gesteuertes System sein und Hardware, Software und menschliche Eingriffe beinhalten kann.“ Systeme in der Automobilindustrie bestehen dabei aus verschiedenen Mechanik-, Hardware- und Softwarebestandteilen [MHDZ07].

Die Modellierung einer interdisziplinen Systembeschreibung wird auch im Systemmodell des Sonderforschungsbereiches 614 vorgenommen (vgl. Kapitel 2.4). Hierfür werden die Partialmodelle *Anwendungsszenarien*, *Funktionen* und *Umfeld* sowie der *Wirkstruktur* eingesetzt. Jedoch können diese Modelle in dieser Form nicht für internationale industrielle Serienprojekte genutzt werden, da die Notationsformen nicht weltweit bekannt sind und daher beim OEM viel Erklärungsbedarf notwendig ist. Ferner sind bei den Partialmodellen Randbedingungen zu beachten. So ist die Wirkstruktur noch lösungsfrei und es muss nach Lösungen gesucht werden. Die Lösungen sind aber oftmals in der Steuergeräteentwicklung aus vorherigen Projekten oder der Vorentwicklung bekannt und können daher sofort explizit modelliert werden.

Ebenso ist die Modellierung einer Wirkstruktur im Automobilbereich nicht etabliert, so dass der Austausch mit dem OEM oder einem Tier-2 sich dann schwierig gestalten kann. Darüber hinaus ist die Werkzeugunterstützung auch noch nicht vollständig gewährleistet. Im Sonderforschungsbereich 614 wurden zwar erste Prototypen für die Erstellung der jeweiligen Partialmodelle und deren Übergänge entwickelt. Die Prototypen bieten aber noch keine ausreichende Unterstützung für ein industrielles Serienprojekt. Vor allem im Hinblick auf Wartung und Fehlerbeseitigung. Demzufolge ist zu überlegen, wie die erarbeiteten Techniken und Methoden auf industrielle Werkzeuge übertragbar sind, die eine entsprechende Unterstützung anbieten. Folglich wird eine international standardisierte Modellierungssprache genutzt, die auf (semi-)formale

Weise eine disziplinunabhängige Modellierung des Systems ermöglicht. Dies ist die SysML [Gro10] (vgl. Kapitel 2.3.1).

Die SysML wurde speziell für die Verwendung im System Engineering entwickelt. Sie wird bereits zum Teil in automobilen Projekten eingesetzt (vgl. [Far07, RBvdBS02, GK01]) und von daher sind erste Erkenntnisse bei ihrer Anwendung bekannt. Ferner sind Werkzeuge mit ausreichender Unterstützung vorhanden. In der SysML können die Bestandteile zunächst (disziplinen-) neutral spezifiziert werden und zwar bis zum Zeitpunkt der Entscheidung, ob das Subsystem durch Mechanik, Hard- oder Software entwickelt wird. Zur Spezifikation der Entscheidung wurden entsprechende Stereotypen in AutoMoMe definiert (vgl. Abbildung 4.23). Das Systemmodell und die darin beschriebene Systemarchitektur ist vergleichbar mit dem Partialmodell Wirkstruktur aus dem Sonderforschungsbereich 614 (siehe Kapitel 2.4).

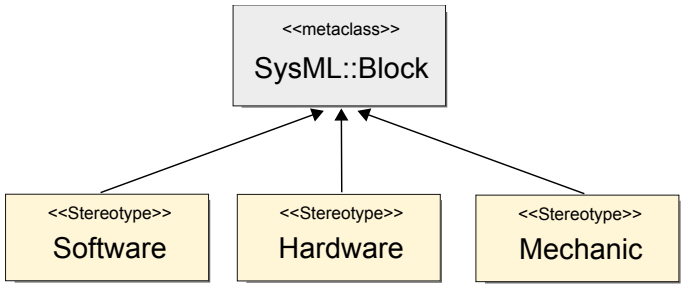


Abb. 4.23. Zuordnung zu Disziplinen durch neue Stereotypen

Bei der Systemmodellierung wird das System aus zwei verschiedenen Sichtweisen spezifiziert. Dies ist zum einen die Blackbox Sicht, die den Kontext des Systems, d. h. die Verbindungen zu anderen Steuergeräten und der entsprechenden Kommunikation spezifiziert. Zum anderen ist dies die Whitebox Sicht, die die Aufteilung des Systems in Teilsysteme betrachtet. Durch beide Sichtweisen wird das komplette System und vor allem die dazu notwendigen Schnittstellen zu anderen Systemen aber auch zu internen Systemen formal beschrieben, so dass keine Inkonsistenten und Fehler auftreten.

Definition (Kontextdiagramm). In [Wei08] wird das Kontextdiagramm wie folgt definiert: „Das (System-)Kontextdiagramm zeigt das System als Blackbox und seine Umgebung sowie die Informationen, die mit der Umgebung ausgetauscht werden.“.

Die Black-Box Sicht des Systems startet mit der Modellierung des Systemkontextes (vgl. Definition Systemkontext). Vergleichbar mit dem Umge-

bungsmodell im Systemmodell des Sonderforschungsbereiches 614. Es werden zunächst die Kommunikationsbeziehungen zu anderen Teilnehmern der Fahrzeugarchitektur modelliert. Hiermit sind vornehmlich die Steuergeräte gemeint, die mit dem Komfortsteuergerät Daten austauschen. Die Modellierung erfolgt in sogenannten Kontextdiagrammen. Das Kontextdiagramm ist kein standardisiertes SysML Diagramm. Vielmehr wird ein angepasstes internes Blockdiagramm (IBD) zur Darstellung eingesetzt [Wei08]. Zur Anpassung des Diagramms wurden innerhalb dieser Arbeit Stereotypen erarbeitet, die die Darstellung einer automobilen Steuergeräteumgebung gestatten (vgl. Abbildung 4.24). Ferner ist für das Kontextdiagramm eine (semi-) formale Spezifikation erarbeitet worden (vgl. Abbildung 4.25), die für ein genaues Verständnis der Inhalte innerhalb des Diagramms sorgt und somit missverständliche Interpretationen ausschließt [KDHM13].

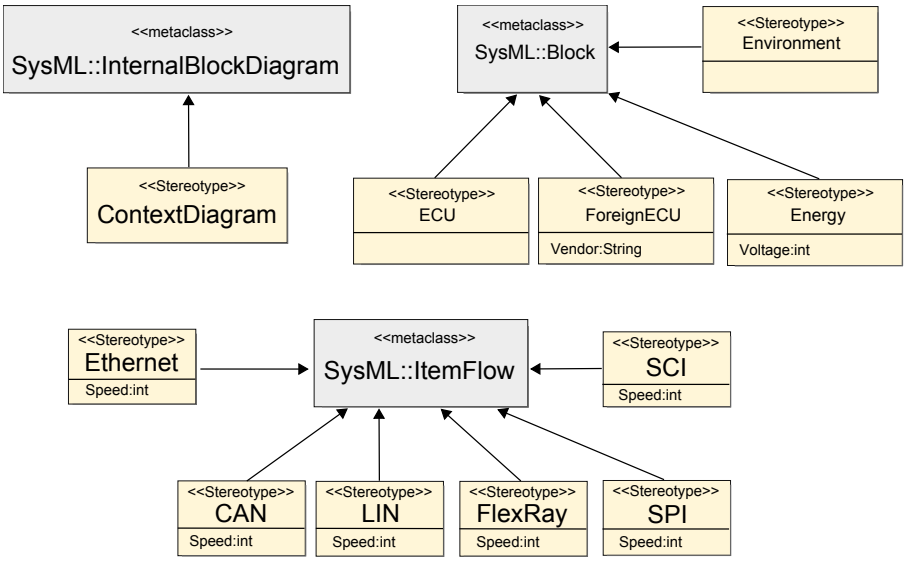


Abb. 4.24. Entwickelte Stereotypen zur Modellierung des Kontextes eines Steuergerätes

Zu den notwendigen Erweiterungen gehört die Einführung von Stereotypen für die vorhandenen Kommunikationsbusse. In einem Fahrzeug gibt es nur eine begrenzte Anzahl von Kommunikationsbussen (CAN, LIN, FlexRay, SPI, SCI und demnächst wohl auch Ethernet (vgl. [ZS08]). Durch die überschaubare Anzahl der Kommunikationsbusse – aber mit unterschiedlichen Eigenschaften – wurde für jeden Bus ein eigener Stereotyp entwickelt, um die entsprechende Kommunikation zu anderen Steuergeräten zu spezifizieren. Dabei werden als Eigenschaften die Geschwindigkeit und die Konfiguration des Busses festgehalten. Ferner wurden die beiden Stereotypen «ECU» und «ForeignECU»

zusätzlich eingeführt, um die Kommunikation zwischen den Steuergeräten darzustellen (vgl. Abbildung 4.24). So kann erkannt werden, ob ein Steuergerät von dergleichen Firma entwickelt wird und daher auf Entwicklungsinformationen zurückgegriffen werden kann oder ob ein anderer Hersteller das Steuergerät bereitstellt. Mit diesen Anpassungen an die Automobilindustrie ist die Erstellung eines Kontextes für das zu entwickelnde Steuergerät verständlich und nachvollziehbar durchzuführen.

In der Abbildung 4.25 ist der Systemkontext des Komfortsteuergerätes wiedergegeben. In dieser Abbildung werden die Schnittstellen zu den anderen Steuergeräten beschrieben. Das Komfortsteuergerät interagiert mit der Lichtsteuerung. Die Zusammenarbeit wird für die Funktionen *Verlassen des Fahrzeugs (Coming Home)* bzw. *Betreten des Fahrzeugs(Leaving Home)* genutzt. Bei diesen Funktionen wird das Licht (sowohl das Außen- als auch das Innenlicht) beim Öffnen/Schließen der Türen automatisch ein- bzw. ausgeschaltet, um den Komfort für die Fahrzeuginsassen zu erhöhen. Diese Kommunikation ist eine interne Kommunikation über den CAN-Bus und wird mit den eben beschriebenen Stereotypen dargestellt (vgl. Abbildung 4.25).

Über die Blinkersteuerung bzw. die Steuerung des Armaturenbretts wird der Zustand des Komfortsteuergerätes nach außen veranschaulicht. Des Weiteren kontrolliert das Komfortsteuergerät den Fahrzeuginnenraum und stellt fest, welche Plätze im Fahrzeug besetzt sind. Diese Information wird der Airbagsteuerung bereitgestellt, so dass bei einem Unfall dann die erforderlichen Airbags ausgelöst werden. Die hier beschriebenen Kommunikationen erfolgen mit fremden Steuergeräten und sind dementsprechend gekennzeichnet.

Neben den Informationsverbindungen werden im Kontextdiagramm ebenso Stoff- oder Energieflüsse abgebildet. Im Fall des Komfortsteuergerätes ist dies der Anschluss an die Stromversorgung des Bordnetzes. Hierzu kommt ein SysML Flow und FlowPorts zum Einsatz. Ebenso wird angezeigt, welche Umwelteinflüsse auf das Steuergerät einwirken. Dies wird durch den Umweltblock verdeutlicht. Er symbolisiert die Umwelteinflüsse, denen das Steuergerät ausgesetzt ist, z. B. Temperaturschwankungen oder Spritzwasserdichtigkeit (vgl. Kapitel 2.6.1).

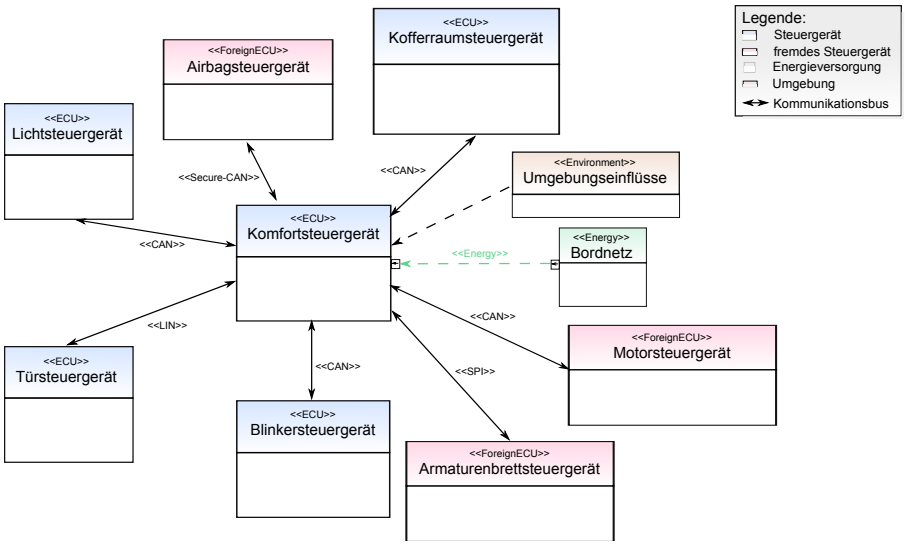


Abb. 4.25. Systemkontextdiagramm des Komfortsteuergeraetes

4.3.5 Kommunikation als Erweiterung des Architekturmodells

Als Anforderung an eine neue Methode für die Steuergeräteentwicklung wurde eine weitestgehende Automatisierung genannt (vgl. Kapitel 1). Eine solche kann bereits bei der Erstellung des Kontextdiagrammes beginnen. Der Zulieferer erhält vom OEM vielfach eine Kommunikationsmatrix, in der die Kommunikationspakete in Form von Botschaften inklusive der zu übertragenden Daten benannt sind. Zusätzlich sind alle Sender und Empfänger der jeweiligen Botschaften beschrieben. Anhand der Kommunikationsmatrix lassen sich die Steuergeräte identifizieren, die Botschaften an das Komfortsteuergerät senden oder von diesem empfangen. Hieraus lässt sich die Netzwerktopologie und somit große Teile des Kontextdiagramms erstellen. Von daher wird dieser Teil der Blackbox Sicht auf das System innerhalb des Ansatzes AutoMoMe automatisiert erstellt. Hierzu wird die Kommunikationsmatrix ausgewertet und entsprechende Modellelemente erstellt. Im Folgenden wird beschrieben, welche zusätzlichen Erweiterungen innerhalb des Ansatzes AutoMoMe entwickelt wurden.

Zu Beginn der Steuergeräteentwicklung waren Sensoren, Aktoren und Steuergeräte über eigene Kabelstränge miteinander verbunden. So bestand zwischen dem Lichtschalter im Armaturenbrett und der Front- und Heckbeleuchtung eine direkte Verbindung [Win08]. Die Automobilhersteller haben aber schon sehr bald erkannt, dass es ökonomisch nicht sinnvoll ist, für jede Kommunikation zwischen den Steuergeräten eine eigene Vernetzung in Form einer Punkt-zu-Punkt-Verbindung zu erstellen. Eine solche Vorgehensweise hat zur Folge, dass zwischen jedem Steuergerät, jedem Sensor und jedem Aktor ein eigener Kabelstrang zu verlegen ist. Diese Verfahrensweise vergrößert nicht nur den Kabelbaum, sondern steigert letztendlich das Gewicht und die Kosten eines Fahrzeuges [SZ13].

Folglich sind die Automobilhersteller dazu übergegangen, eine Verfahrensweise zur Vernetzung der Sensoren, Aktoren und Steuergeräte untereinander zu finden, die das Gewicht des Fahrzeuges möglichst gering hält und ökonomisch rentabler ist. Dies wird durch den Einsatz von Kommunikationsbussen gewährleistet (siehe hierzu auch Kapitel 2.6.2). Ein Kommunikationsbus kann dabei mehrere Nachrichten von verschiedenen Sensoren, Aktoren und Steuergeräten transportieren. Damit nun die jeweiligen Sender und Empfänger die Daten senden bzw. empfangen können, sind spezielle Übertragungsformate einzuhalten. Diese werden meist in Form von sogenannten Kommunikationsmatrizen definiert. Eine Kommunikationsmatrix definiert dabei für einen Bus die möglichen zu übertragenden Daten inklusive Sender und Empfänger.

Die Kommunikationsmatrizen für die im Gesamtfahrzeug verwendeten Busse werden vom Automobilhersteller entwickelt und zusammen mit den Anforderungen dem Zulieferer übergeben. Die Übergabe kann dabei auf unterschiedlichste Art und Weise erfolgen. In AUTOSAR wird die Kommunika-

tionsmatrix in Form des ECU Extrakts [KF09] ausgetauscht. Wird keine AUTOSAR Architektur eingesetzt, erfolgt der Austausch in busspezifischen Formaten. Für den CAN Bus ist dies beispielsweise eine dbc-Datei, die auch für die Spezifikation der Kommunikationsdaten im Komfortsteuergerät Verwendung findet (siehe hierzu Kapitel 4.2).

Die Daten aus den übergebenden Kommunikationsdateien werden in den unterschiedlichsten Artefakten im gesamten Entwicklungsprozess genutzt. Damit die Daten jederzeit in den Entwicklungsphasen verwendet werden können, werden sie in zentralen Dokumenten gespeichert. Die Speicherung erfolgte dabei bisher in Excel-Tabellen oder in DOORS-Modulen in informaler Weise. Eine automatische Verarbeitung der Signale ist hierdurch nur schwerlich zu realisieren, da die Daten immer manuell ins jeweilige Entwicklungsartefakt übertragen werden müssen. Da die Kommunikationsmatrix bei der Entwicklung häufig angepasst wird, minimiert der Ansatz AutoMoMe durch die automatische Verarbeitung den sich daraus ergebenden Aufwand.

Im Systemmodell sind die Daten erforderlich, um dort die Systemschnittstellen zu spezifizieren. Neben den Namen der Signale werden hierfür auch deren Skalierung benötigt, um ggf. Umrechnungen vornehmen zu können. So ist es durchaus realistisch, dass die Eingangswerte eines Subsystems nicht in der Skalierung vorliegen, wie sie von diesem Algorithmus verlangt werden. Zur Korrektur muss eine Umwandlung in Form einer Transformation durchgeführt werden. Diese kann die gespeicherten Skalierungen zur automatischen Umwandlung verwenden. Exemplarisch kann eine Umrechnung von km/h zu m/s automatisiert erfolgen.

Der Nachteil der manuellen Vorgehensweise ist offensichtlich. Zum einen ist eine solche Verfahrensweise recht fehleranfällig, da bei der manuellen Übertragung in das Modell Datenübertragungsfehler etc. sich ergeben können und somit Inkonsistenzen im Systemmodell hervorrufen. Zum anderen erfordert die manuelle Übertragung einen enormen Zeitaufwand. Dieser ergibt sich aus der initialen Übertragung der Daten ins Modell, da eine große Anzahl von verschiedenen Kommunikationssignale pro System einzupflegen sind. Zudem muss immer mit Änderungen seitens des OEMs bei den Kommunikationsmatrizen ausgegangen werden, so dass die Datenbasis häufig aktualisiert werden muss. Daher sind automatisierte Vergleichsmöglichkeiten notwendig, um durch eine Impact Analyse die vorzunehmenden Änderungen herauszufiltern, um somit das Modell an die geänderte Kommunikationsstruktur anzupassen.

Um diese Vorgabe umzusetzen, werden die Kommunikationssignale so formal abgespeichert, dass sie automatisiert weiterverarbeitet werden können. Die formale Speicherung muss dabei zu einem frühen Entwicklungszeitpunkt erfolgen, um auf die Daten in den Entwicklungsphasen jederzeit zurückgreifen zu können. Als optimaler Speicherort bietet sich das Architekturmodell an, da in diesem alle sonstigen Informationen gespeichert sind. Mittels der formalisierten

Speicherung ist ein automatisierter Import der Kommunikationssignale und die Weiterverarbeitung der darin enthaltenen Daten möglich.

Es ergeben sich somit die folgenden Anforderungen an eine formale Speicherung der Daten bzw. deren automatisierten Import:

- In einem Automobil werden unterschiedliche Bustypen eingesetzt. Damit einhergehend sind unterschiedliche Dateiformate für die Kommunikationsmatrix zu berücksichtigen. Die Speicherung der Signale aus den unterschiedlichen Formaten ist eine Zielvorgabe an das zu erarbeitende Konzept.
- Innerhalb des Architekurmodells müssen geeignete SysML Modellelemente bereitgestellt werden. Dazu bedarf es der Modellierung nicht nur der einzelnen Signale, die mit einer Nachricht versandt werden. Vielmehr sind in der Kommunikationsmatrix detaillierte Informationen über die Eigenschaften der Signale hinterlegt. Für jedes Signal ist die Auflösung, das physikalische Minimum und Maximum als auch die Werteeinheit zu definieren. Diese zusätzlichen Werte werden in den späteren Entwicklungsphasen benötigt, um ggf. Umrechnungen vornehmen zu können.
- Für den Import ins Architekturmodell sind entsprechende Transformationsregeln zu erstellen, um einen automatisierten Import der Daten zu gewährleisten.
- Es muss ein Vergleichsmechanismus verfügbar sein, der es ermöglicht, bei vorzunehmenden Anpassungen der Kommunikationsmatrix die Unterschiede zu identifizieren.
- Der Systemkontext kann mithilfe der Kommunikationsmatrix erstellt werden, da die Verbindungen zu anderen Systemen bereits dort spezifiziert sind. Für das zu erstellende System sind die ein- und ausgehenden Daten und somit die Vernetzung zu anderen Steuergeräten bekannt. Dies muss in einem Kontextdiagramm einbezogen werden.
- In der Kommunikationsmatrix werden alle Signale des jeweiligen Kommunikationsbusses definiert. Dies sind die sogenannten technischen Signale. Aus den Anforderungen an das Steuergerät sind die Funktionen und deren Verbindungen untereinander bekannt. Dies sind die logischen Signale. Für die weitere Dekomposition des Systems ist eine Abbildung der logischen auf die technischen Signale unabdingbar. Dies muss bei der Speicherung der Signale im Architekturmodell berücksichtigt werden.

Im Folgenden wird das Lösungskonzept vorgestellt, das die zuvor herausgearbeiteten Anforderungen aufgreift und umsetzt (vgl. Abbildung 4.26). Im Mittelpunkt der Realisierung steht ein Metamodell, das alle Informationen aus der Kommunikationsmatrix aufnehmen kann (siehe Abbildung A.2 im Anhang). Gleichzeitig weist es aber auch die Flexibilität auf, die unterschiedlichen Formate der Kommunikationsmatrix einzulesen. Das Modell wird als *SignalManager* bezeichnet. Das Metamodell ist in Signalgruppen und Signale

aufgeteilt (vgl. Abbildung A.2 im Anhang). Diese beiden Elemente weisen zusätzliche Eigenschaftswerte auf, die die Informationen für die Kommunikation aufnehmen. Sie reichen vom Sender/Empfänger bis hin zu den eigentlichen physikalischen Werten.

Im Zentrum des Modells stehen die Signalgruppen. Diese entsprechen den Busnachrichten, die wiederum mehrere Signale bündeln. Für sie existiert im Metamodell das Element *Signal*. Neben der Speicherung des Signalnamens werden außerdem die Eigenschaften der Signale gespeichert. Da bereits mehrere Projekte und Standards, die sich mit der Speicherung von Kommunikationssignalen beschäftigt haben, verfügbar sind, kann auf etablierte Standards aufgesetzt werden. Hier sind zu nennen der Manufacturer Supplier Relationship Standard (MSR) [fSoAA11], das Field Bus Exchange Format (Fibex) [fSoAA13] und der AUTOSAR Standard [AUT10b]. Diese Standards finden bei der Spezifikation des Metamodells Anwendung.

Das Einlesen der unterschiedlichen Busformate in das Modell des Signal-Managers erfolgt durch entsprechende Adapter (vgl. Abbildung 4.26). In den Adaptern ist das Wissen über die Spezifikation des jeweiligen Busses hinterlegt. Hierdurch ist sichergestellt, dass Daten aus den unterschiedlichen Bussen eingelesen werden können. Der Einsatz von Adaptern bietet überdies den Vorteil, dass die Modellierung der Signalgruppen und ihrer Signale sowie die Transformationsregeln zum Systemmodell gleichbleibend sind. Somit muss bei Änderungen an einem Bus nur der jeweilige Adapter angepasst werden. Eine Anpassung der Transformationsregeln kann entfallen.

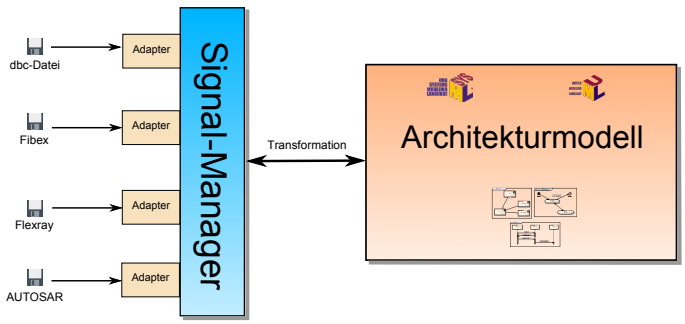


Abb. 4.26. Arbeitsweise des Signal-Managers

Es stellt sich die Frage, warum zunächst ein eigenständiges Modell für die Signalgruppen und deren Signale erstellt werden muss und weshalb die Daten nicht sofort ins Modell importiert werden. Der Vorteil der zuvor skizzierten Vorgehensweise besteht darin, dass die Adapter möglichst klein und handhabbar gehalten werden. Änderungen bei der Modellierung wirken sich nur auf die Transformationsregeln aus. Ferner ist das Metamodell des Signal-Managers

mittels des Eclipse Modeling Framework (EMF) [Fou14] realisiert, so dass auf bestehende Vergleichsmöglichkeiten aus dem Framework zurückgegriffen werden kann [MSW09]. Somit ist ein Abgleich nach einer Aktualisierung durch den Automobilhersteller ohne größeren Aufwand durchführbar. Die sich aus dem Vergleich ergebenden Werte können sodann für eine Impact Analyse herangezogen werden.

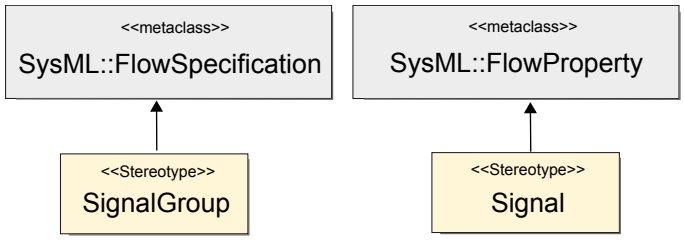


Abb. 4.27. Stereotypen für die eingelesenen Kommunikationssignale

Letztendlich bleibt zu behandeln wie in diesem Konzept die Repräsentation bzw. Darstellung der Daten im Architekturmodell zu erfolgen hat. Hierzu wird für die Signalgruppen ein Stereotyp, abgeleitet von Objektflussspezifikationen (flowSpecification), eingesetzt («SignalGroup»). Die Kommunikationssignale werden innerhalb der Objektflussspezifikationen in Form eines Stereotyps, abgeleitet von Eigenschaften (flowProperties), abgebildet («Signal») (siehe Abbildung 4.27). Die zusätzlichen Informationen wie Wertebereiche, Skalierung und Einheit werden für jede Eigenschaft durch entsprechende Eigenschaftswerte modelliert. Die Objektflussports (FlowPorts) werden innerhalb des Kontextdiagramms dazu genutzt, um die Kommunikation mit weiteren Steuergeräten darzustellen.

```
NS_ :
        NS_DESC_
        SIGTYPE_VALTYPE_

BO_ 259 Lichtwerte: 8 Lichtsteuerung
        SG_ Lichtsensorwert: 0|16@1+ (1.0,0.)[0|+65536] "-" Komfortsteuergerät
        SG_ Klemme58: 16|16@1- (0.003051,0.)[-100|+100] "%" Komfortsteuergerät

BO_ 261 Fahrdaten: 8 Motorsteuerung
        SG_ Geschwindigkeit: 0|16@1+ (1.0,0.)[0|+65536] "-" Komfortsteuergerät, Bremse, ACC
```

Abb. 4.28. Auszug aus der Kommunikationsmatrix des Komfortsteuergerätes

Zum besseren Verständnis des Konzeptes wird diese Vorgehensweise am Beispiel des Komfortsteuergerätes veranschaulicht. Für das Komfortsteuergerät existiert eine Kommunikationsmatrix in Form einer dbc-Datei, da das Komfortsteuergerät an einen CAN-Bus angeschlossen ist (vgl. Abbildung 2.12). Ein Auszug der Kommunikationsmatrix ist der Abbildung 4.28 zu entnehmen. In dem Auszug sind zwei Busnachrichten spezifiziert, die aus drei Signalen bestehen. Die Kommunikationsmatrix wird in den Signal-Manager eingelesen. Die Informationen werden dort in Form von Signalgruppen und Signalen gespeichert. Ein Vergleich mit

bisherigen Matrizen ist nun durchführbar, um vorhandene Unterschiede zu identifizieren.

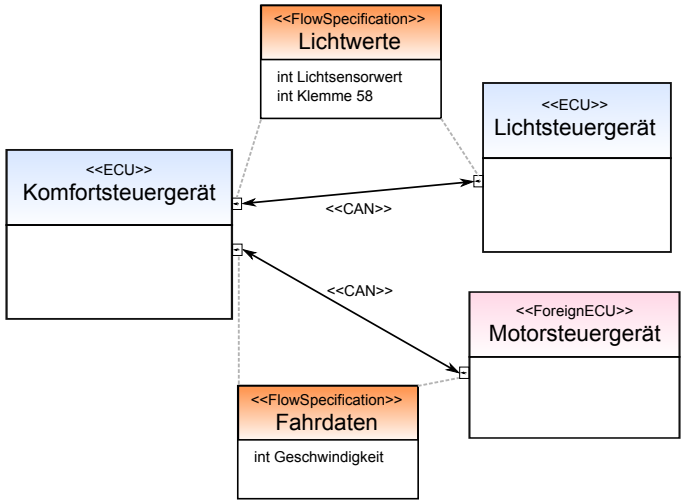


Abb. 4.29. Auszug aus dem Kontextdiagramm mit den importierten Kommunikationsdaten

Die in den Signal-Manager importierten Daten werden dann entsprechend den eben zuvor definierten Transformationsregeln in das Architekturmodell überführt (vgl. Abbildung 4.29). Zudem werden aus Sender- und Empfängerdaten die Verbindungen zu den anderen Systemen identifiziert und dargestellt. Hierzu wird das bereits definierte Kontextdiagramm eingesetzt. Die Informationen aus der Kommunikationsmatrix unterstützen dabei die Erstellung des Kontextdiagramms indem die dazugehörigen Verbindungen automatisiert gezeichnet werden. Hierdurch ist eine Unterstützung bei den ersten Schritten zur Entwicklung des Systems gegeben (vgl. Abbildung 4.25).

Die Whiteboxsicht der Systemarchitektur

Neben der eben beschriebenen Blackbox Sicht ist innerhalb der logischen Sichtweise noch die Whitebox Sicht auf das zu entwickelnde System zu beachten. Bei der Whitebox Sicht (vgl. Abbildung 4.30) geht es vornehmlich darum, die Funktionen aus der funktionalen Sichtweise so zu gliedern und zusammenzufassen, dass Teilsysteme identifiziert werden. Zwischen den Teilsystemen werden dann die Datenflüsse [KKH+08] mithilfe von Objektflussports (FlowPorts) und Objektflussspezifikationen (FlowSpecifications) beschrieben. Hierdurch werden die einzelnen Teilsysteme und ihre benötigten Daten gekapselt. Anhand der Schnittstellen ist zu erkennen, ob das Teilsystem ggf. in anderen Projekten weiterverwendet werden kann.

Jedoch sind nur die hardwareunabhängigen Teile der Systemarchitektur der logischen Sichtweise zuzurechnen. Die hardwareabhängigen Bestandteile zählen zur technischen Sichtweise. Der Vorteil der logischen Architektur liegt darin begründet, dass sie über einen langen Zeitraum stabil bleibt, da sie hardwareunabhängig ist [Teu09]. Das bedeutet, dass es innerhalb der Entwicklung nur selten Änderungen in den Funktionen und der Logik des Systems gibt. Ebenso sind die Abhängigkeiten zwischen den Funktionen aus der funktionalen Architektur bekannt. So ist es möglich, aus den logischen Funktionen geeignete Realisierungen in Form von bereits bestehenden Komponenten zu suchen. Hierdurch kann die Quote der Wiederverwendung innerhalb der Architektur deutlich verbessert und somit die Entwicklungszeiten und -kosten gesenkt werden [HKK04]. Falls keine entsprechenden Komponenten gefunden werden, müssen sie anhand der funktionalen Architektur und den darin analysierten Anforderungen entwickelt werden.

Die entstehende logische Systemarchitektur muss den Vorgaben an das zu entwickelnde System gerecht werden. Besonders geht es darum, die qualitativen Anforderungen (Laufzeit, Speicherbelegung, Wartbarkeit etc.) umzusetzen. Es gilt aber auch, das Verhalten des Gesamtsystems zu beachten. So wird innerhalb der logischen Sichtweise nicht nur die Struktur, sondern ebenso das Verhalten spezifiziert. Dabei spielt es zunächst keine Rolle, ob das Teilsystem später durch Software, Hardware oder Mechanik realisiert wird. In der Systemarchitektur werden alle Disziplinen modelliert. Es werden die Schnittstellen zwischen den verschiedenen Disziplinen auf formale Weise festgelegt. Sie sind daher eindeutig und konsistent.

Es geht es darum, die Funktionen aus der funktionalen Sichtweise zu einer sinnvollen logischen Systemarchitektur zusammenzustellen. Hierzu müssen die Funktionen auf Teilsysteme aufgeteilt und ggf. weiter verfeinert werden. Die Teilsysteme werden dabei mittels der SysML als Blöcke bzw. Parts modelliert. Es werden hierfür die SysML Strukturdiagramme (Block Definitions Diagramm (BDD) und das Interne Block Diagramm (IBD)) verwendet. So wird aus dem Funktionsmodell eine logische Sichtweise erarbeitet, die die einzelnen

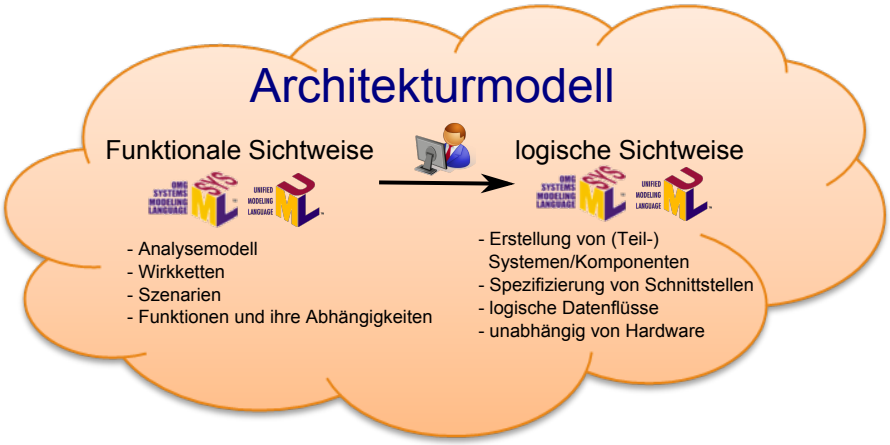


Abb. 4.30. Übergang von der funktionalen zur logischen Sichtweise

Funktionen den Blöcken zuordnet. Dabei kann es durchaus vorkommen, dass eine Funktion von mehreren Blöcken bzw., dass mehrere Funktionen von einem Block erfüllt werden. Es ist somit von einer *m:n* Verteilung auszugehen. Dies ist abhängig von dem Funktionsumfang.

Die Abhängigkeiten der Funktionen untereinander werden durch Kommunikationsverbindungen in der logischen Architektur dargestellt. In der SysML bzw. UML werden sie durch Objektflussports und entsprechende Spezifikationen beschrieben. Die Kommunikationsverbindungen können zum Teil aus den Datenflüssen der funktionalen Sichtweise abgeleitet werden. Aus jedem Datenfluss, der über eine Teilsystemgrenze hinausgeht, wird in der logischen Architektur ein Port. Welche Daten über diesen Port ausgetauscht werden, wird mittels der Schnittstelle spezifiziert.

So ist aus der funktionalen Sichtweise bekannt, dass der Türstatus zum Komfortsteuergerät übertragen wird (vgl. Abbildung 4.19). Bei der logischen Systemarchitektur kann es durchaus vorkommen, dass zwar eine ungefähre Vorstellung der Daten vorhanden ist, die ausgetauscht werden, aber es noch nicht genügend Informationen gibt, um eine Schnittstelle zu definieren. So wird in der Systemarchitektur zunächst ein Datenfluss zwischen den Teilsystemen modelliert. Dieser wird in einer späteren Phase der Softwarearchitektur dann durch eine entsprechende Schnittstelle realisiert. Diese Vorgehensweise ist in der Abbildung 4.31 für den Austausch des Türstatus abgebildet. In der Systemarchitektur sind die Daten zunächst noch nicht bekannt, so dass ein Datenfluss dargestellt wird. Dieser wird in der Softwarearchitektur durch eine Schnittstelle und einen entsprechenden Rückgabewert (in diesem Fall ein uint8 Datentyp) spezifiziert.

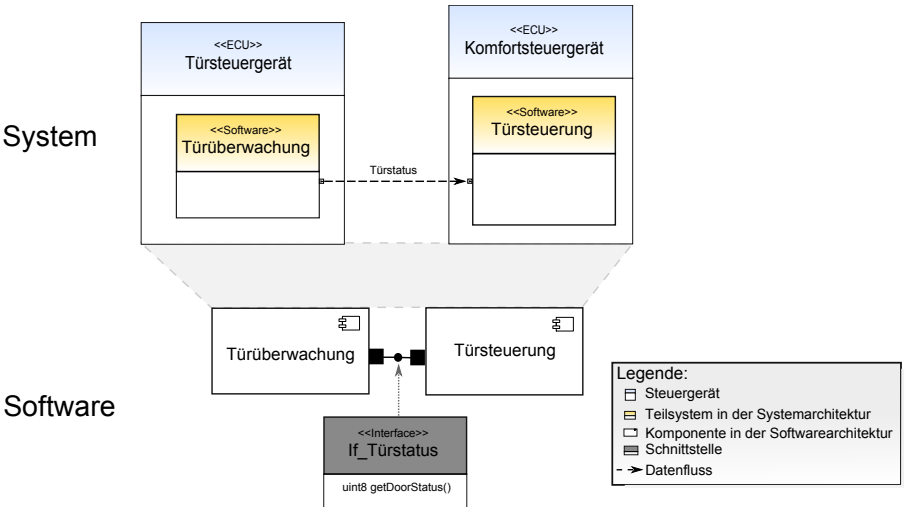


Abb. 4.31. Spätere Verfeinerung der Systemschnittstelle innerhalb der Softwarearchitektur

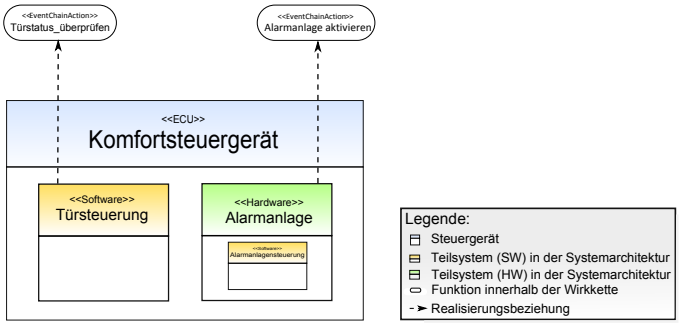


Abb. 4.32. Realisierung von Funktionen in der logischen Sichtweise

Die Abbildung 4.32 vermittelt einen Ausschnitt der logischen Architektur für das Komfortsteuergerät. Aus der funktionalen Sichtweise des Systems ist bekannt (siehe hierzu Abbildung 4.19), dass das Komfortsteuergerät zumindest die Teilsysteme Türsteuergerät und Alarmanlage beinhaltet. Da diese erste Einteilung in Teilsystemen noch sehr abstrakt ist, werden die einzelnen Teilsysteme weiter verfeinert. Damit die Nachvollziehbarkeit zu der funktionalen Sichtweise gewährleistet bleibt, wird eine Abhängigkeit zwischen der Funktion auf der funktionalen Seite und dem Teilsystem auf der logischen Seite gezogen. Die Abhängigkeit ist mit dem Stereotyp «realizeLogicalPart» verbunden (vgl. Abbildung 4.33).

In der Abbildung 4.32 erfolgt eine Verfeinerung der Teilsysteme. So ist das Teilsystem *Alarmanlage* noch in das zusätzliche Teilsystem *Alarmanlagens-*

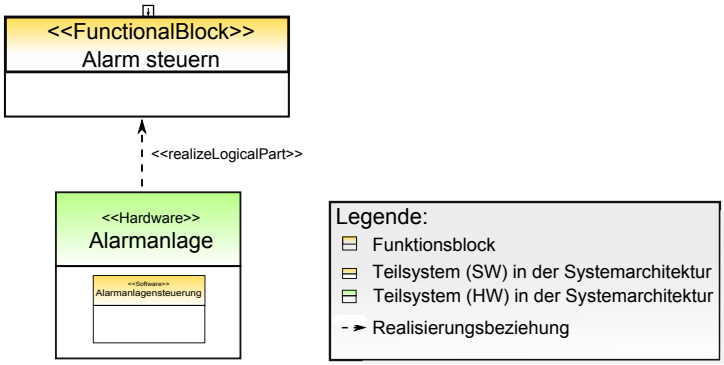


Abb. 4.33. Nachvollziehbarkeit zwischen der funktionalen und logischen Sichtweise

teuerung unterteilt. Diese Dekomposition wird solange fortgesetzt, bis die Teilsysteme einer Disziplin zugeordnet werden können. Hierdurch entsteht ein eindeutiges Kriterium, wann die Struktur der Systemarchitektur abgeschlossen ist und wann in die disziplinenspezifische Entwicklung übergegangen werden kann. Eine Systemarchitektur gilt dann als abgeschlossen, wenn festgelegt werden kann, welche Teilsysteme in Form von Hardware, Mechanik oder welche mittels Software realisiert werden [HHP03]. An dieser Stelle kann die Dekomposition beendet werden. Die Unterscheidung erfolgt durch die Stereotypen «Hardware», «Mechanik» und «Software» (vgl. Abbildung 4.32). Alle Softwarekomponenten müssen dann in der Softwarearchitektur vom Softwarearchitekten weiter verfeinert werden. Ebenso werden die bestehenden Wirkketten auf die entsprechenden Teilsysteme angepasst (vgl. Abbildung 4.32).

Zur logischen Sichtweise gehören neben der Modellierung der Systemstruktur auch das Kommunikations- und Gesamtsystemverhalten. Für das Kommunikationsverhalten können Zustandsdiagramme in Form von Protokollstatecharts genutzt werden. Bei der Modellierung des Kommunikationsverhaltens hat der Faktor *Zeit* eine wesentliche Bedeutung. Von daher muss die Zeit bei der Modellierung berücksichtigt werden. Hierzu reichen die standardmäßigen Mittel der SysML nicht aus. Es wird in AutoMoMe aber auf die Erweiterungen im Architekturmodell bezüglich der Echtzeit zurückgegriffen (vgl. Kapitel 4.3.2). Das Einschaltverhalten des Komfortsteuergerätes veranschaulicht die Abbildung 4.15. Es sind die drei Zustände *Initialisierung*, *Laufend* und *Schlafend* gegeben, die das Steuergerät einnehmen kann. Beim Zustand *Laufend* sind die Funktionen spezifiziert, die während des Zustandes ausgeführt werden. Dies überwiegend ereignisgesteuerte Verhalten wird mittels der Real-Time Statecharts im Architekturmodell modelliert.

Zusätzlich zu dem RTSC Diagramm werden Aktivitäts- und Sequenzdiagramme eingesetzt, um das Verhalten innerhalb der einzelnen Zustände zu spezifizieren. Oftmals werden Sequenzdiagramme verwendet, um mit ihrer

Hilfe positive als auch negative Szenarien für einzelne sicherheitskritische Zustände zu definieren. Diese werden beim Systemtest dann besonders getestet und überprüft. Dazu zählt beispielsweise das Verhalten beim Ein- und Ausschalten (Power-Up bzw. Power-Down) des Systems. In der Abbildung 4.34 ist das Verhalten des Zustandes *Initialisierung* beschrieben. Es werden hier die einzelnen Teilsysteme initialisiert. Aus dem Sequenzdiagramm ist die genaue Reihenfolge der Operationsaufrufe für die Initialisierung ersichtlich. So beginnt der Vorgang bei der *Türsteuerung* und endet bei der *Alarmanlage*. Der gesamte Vorgang darf dabei höchstens 100 ms betragen, wie es bereits im RTSC spezifiziert ist.

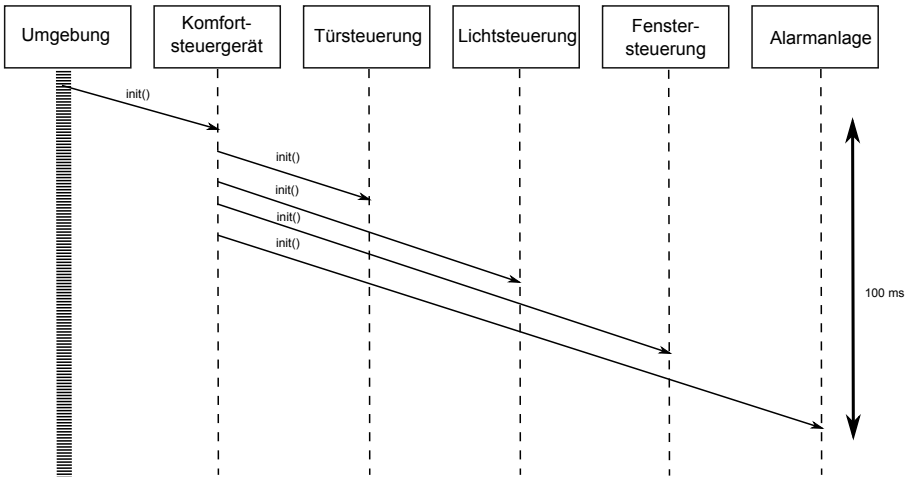


Abb. 4.34. Modellierung der Aufrufreihenfolge der unterschiedlichen init-Methoden

Mithilfe der Systemarchitektur wird innerhalb der logischen Sichtweise zunächst eine disziplinübergreifende Spezifikation geschaffen, die sowohl die Struktur aber auch das Verhalten definiert. In der weiteren Entwicklung werden die einzelnen Teilsysteme nun in den jeweiligen Disziplinen weiterentwickelt. Im Falle des Ansatzes AutoMoMe ist dies für die Softwareanteile die Softwarearchitektur mittels der UML. Der Softwarebereich ist der Bereich, dem ein großes Wachstumspotential vorhergesagt wird, da immer mehr Funktionen durch Software realisiert werden [WFO09]. Daher stehen die anderen Disziplinen nicht im Fokus dieser Arbeit und werden dementsprechend nicht weiter betrachtet.

Softwarearchitektur im Architekturmodell

Zur Modellierung der Softwarearchitektur wird die UML genutzt. Die Wahl ist auf die UML gefallen, da sie sich zum einen auf dem Gebiet der Soft-

wareentwicklung bereits durchgesetzt hat und zum anderen die Modellierung der Struktur und des Verhaltens ermöglicht. Dies ist bei anderen Alternativen nicht der Fall. Ab und an wird die Modellierung von Matlab/Simulink [Mat14] genutzt. Aber hiermit ist nur bedingt eine große Architektur umsetzbar, da das Konzept von Schnittstellen nicht bekannt ist. Somit ist keine Abstraktion der auszutauschenden Daten vornehmbar. Ein zusätzlicher Vorteil der UML ist die Anpassungsfähigkeit. Sie wird speziell auf die Bedürfnisse der Domäne Steuergeräteentwicklung abgestimmt, so dass die Entwickler direkt ihre Lösung modellieren können, ohne sich groß in die Modellierung einzuarbeiten [Tho04]. Die Nachvollziehbarkeit zwischen der System- und der Softwarearchitektur, somit zwischen der SysML und der UML, wird mittels Abhängigkeiten hergestellt (siehe Abbildung 4.35).

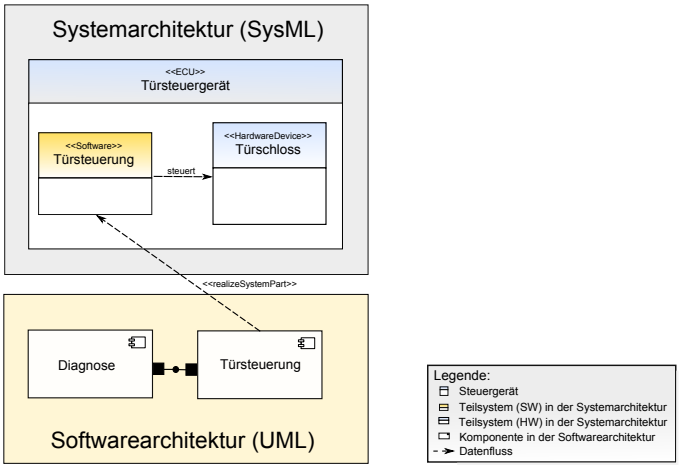


Abb. 4.35. Übergang von der Systemarchitektur zur Softwarearchitektur

Der Softwarearchitekt ist für die Realisierung der Softwarekomponenten und deren Zusammenspiel verantwortlich. Hierzu sind die vorgegebenen Systembestandteile ggf. weiter zu dekomponieren, d. h. es werden kleinere Softwarekomponenten mit einem überschaubaren Verhalten definiert. Die Komponenten werden über Schnittstellen miteinander verbunden. Hierzu werden UML Komponentendiagramme mit Schnittstellen eingesetzt (vgl. Abbildung 4.35). Dabei ist essenziell, dass die Schnittstellen aus der Systemarchitektur mit den Schnittstellen der Softwarearchitektur in Verbindung gebracht werden, um die Nachverfolgbarkeit zwischen den Modellen sicherzustellen [BLY09]. Die Nachverfolgbarkeit wird durch Abhängigkeitsbeziehungen (Dependencies) gewährleistet. Für die Qualitätskontrolle und die Prozesskonformität wird eine Metrik erstellt (vgl. Kaptitel 7.1.1). So wird garantiert, dass alle Systemschnittstellen der Software abgeholt und in der Softwarearchitektur umgesetzt werden.

Abb. 4.36. Ressourceninformationen für das Komfortsteuergerät über den Ressourcenverbrauch

Eine weitere Aufgabe, die der Softwarearchitekt zu erfüllen hat, ist die Festlegung von Zeit- und Ressourceninformationen für die jeweiligen Softwarekomponenten [FSA05]. Mit den Informationen erhält der Softwareentwickler eine Vorgabe, die die Softwarelösung bei der späteren Implementierung erfüllen muss. Die frühzeitige Festlegung auf Softwarearchitekturebene führt dazu, dass bei der Integration keine Überraschungen auf Grund von Fehlinterpretationen auftreten. Ferner können die Informationen für die Zeit und Ressourcen bei der Simulation bereits in einer frühen Phase überprüft und ggf. angepasst werden (vgl. Kapitel 5).

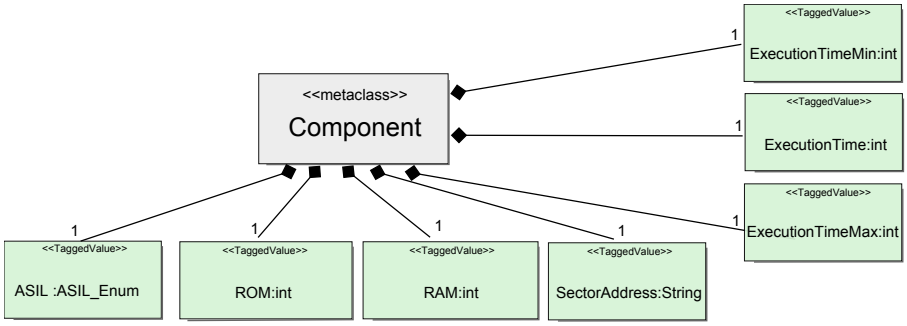


Abb. 4.37. Eigenschaftswerte für eine Komponente

Die Modellierung der Werte erfolgt mittels Eigenschaftswerte (vgl. Abbildung 4.37). Ein Vorteil besteht darin, dass über die Eigenschaftswerte eine tabellarische Darstellung erzeugt werden kann. Hierdurch ist die Arbeit vergleichbar mit der bisherigen dokumentenbasierten, obwohl die Daten (semi-) formal gespeichert werden. Dieser Vorteil ist durch die Verwendung des MARTE Profils nicht gegeben, da hier die Zeiten als zusätzliche Referenz dem Element hinzugefügt werden [GOO02]. In den Eigenschaftswerten werden die Laufzeit mit der Best Case Execution Time (BCET) und der Worst Case Execution Time (WCET) sowie der Speicherverbrauch, die Einstufung der Komponen-

ten hinsichtlich ihrer Sicherheitsstufe (engl. Automotive Safety Integrity Level (ASIL)) und die Testtiefe bestimmt. Für die Laufzeit und den Speicherverbrauch kann nur ein geschätzter Wert angegeben werden. Die Schätzungen beruhen dabei auf Prototypen oder Produkten, die bereits entwickelt wurden. In der Abbildung 4.36 ist ein Beispiel für die Festlegung der Werte dargestellt.

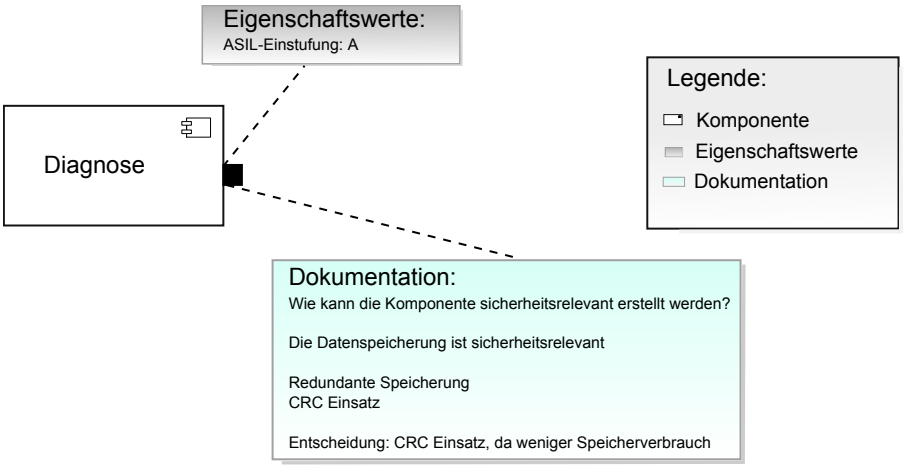


Abb. 4.38. Dokumentation der Entscheidung wie die Sicherheitsrelevanz umgesetzt wird

Es gehört des Weiteren zu den Aufgaben des Architekten nicht nur eine Architektur zu erstellen, sondern diese auch zu dokumentieren [HS09, DGH03, HS06]. Die Dokumentation sorgt dafür, dass nicht nur das Ergebnis – nämlich die Softwarearchitektur – dieser Entwicklungsphase bekannt ist, sondern dass ebenfalls die Entscheidungen, warum gerade diese Lösung gewählt wurde, nachgehalten und dokumentiert sind [Dei13]. Erst wenn die genauen Umstände bei der Architekturerstellung bekannt sind, kann nachvollzogen werden, ob bei einer späteren Generation die Entscheidung revidiert wird oder beibehalten werden kann. Durch die Dokumentation der Entscheidungen kann verhindert werden, dass einzelne Spezialisten zu einem Kapazitätsengpaß werden [Amb11]. Darum beinhaltet AutoMoMe eine Methode zur Dokumentation von Entscheidungen, die angelehnt ist an die Vorlage aus [Zör12]. Die Ableitung des entsprechenden Stereotyps ist in Abbildung 4.39 zu sehen. Die Anwendung ist in Abbildung 4.38 dargestellt, wo begründet wird, wie die Automotive Safety Integrity Level (ASIL) Einstufung in die Komponente umgesetzt wird. Aufbauend auf der formalen Dokumentation sind die Entscheidungen innerhalb der Architektur durch Analyse (wie beispielsweise durch Architecture Tradeoff Analysis Method (ATAM)) zu überprüfen [WFA07].

Hinsichtlich der Sicherheitsstufe ASIL weist der Ansatz AutoMoMe eine weitere Erleichterung für den Architekten auf. Die textuellen Anforderungen und

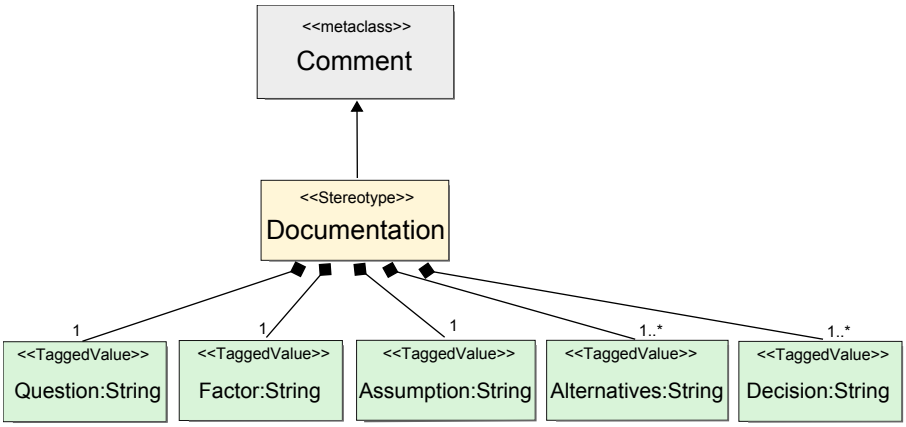


Abb. 4.39. Stereotyp für die Dokumentation

Funktionen werden im funktionalen Sicherheitskonzept (FSC) mit entsprechenden Sicherheitsstufen versehen. So übernimmt die Implementierung der AutoMoMe bei der Zuweisung von Anforderungen auf ein Architekturelement automatisch die höchste Einstufung und setzt diese beim Löschen der Anforderung auch automatisch zurück. Hierdurch ist auf der einen Seite eine konsistente Sicherheitseinstufung gegeben und auf der anderen Seite wird der Architekt von manueller Arbeit entlastet [HMM12]. So kann er nach der Zuweisung der Anforderungen bereits durch eine farbliche Kennzeichnung die abzusichernden Kommunikationsverbindungen innerhalb der Architektur erkennen. In der Abbildung 4.36 sind durch die Zuweisung der Sicherheitseinstufung die Komponenten und der Kommunikationskanal rot eingefärbt.

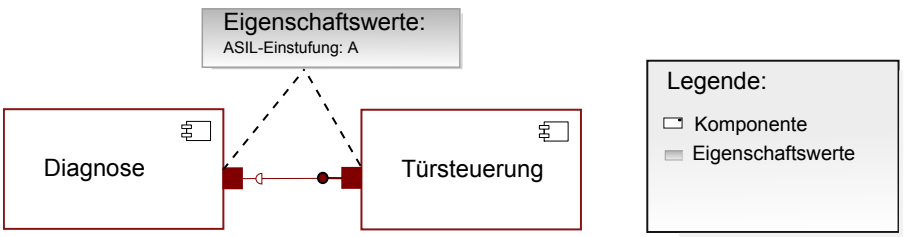


Abb. 4.40. Farbliche Einfärbung von sicherheitskritischen Komponenten und Kommunikationskanälen

Ein weiterer Aspekt bei der Architekturmodellierung ist die Spezifikation des Übergangs zum Feindesign. Es muss jederzeit sichergestellt sein, dass die Architektur- und die Feindesignmodellierung konsistent sind. In AutoMoMe wird für das Feindesign größtenteils Klassen- oder Kompositionsstrukturdiagramme der UML für die weitere Strukturmodellierung genutzt. Lediglich

wenn reglerbasiertes Verhalten modelliert werden muss, wird auf die Modellierung mit Matlab/Simulink [Mat14] zurückgegriffen. Das Verhalten wird in UML durch Aktivitäts- oder Zustandsdiagramme gewährleistet. Hierzu werden die Schnittstellen und Ports aus der Architekturmodellierung weiterverwendet. Falls Änderungen in der Architekturmodellierung auftreten, werden sie automatisch in das Feindesign übernommen, so dass der Entwickler seine Funktionen anpassen kann. Dies ist von Vorteil, wenn die Modellierung innerhalb eines Modells erfolgt.

Feindesign im Architekturmodell

Nach der Modellierung der Softwarearchitektur schließt sich das Feindesign an. In diesem werden (falls notwendig) die innere Struktur, die notwendigen Datenstrukturen und -typen sowie das Verhalten der jeweiligen Komponenten beschrieben. Das Feindesign an sich ist genau genommen nicht mehr Bestandteil der Architektur. Jedoch wird sie in der Architektur referenziert und hat Auswirkungen auf die Architektur beispielsweise in der AUTOSAR Transformation. Deswegen wird sie ebenfalls an dieser Stelle von AutoMoMe dargestellt, um eine umfassende Darstellung zu haben.

Die innere Struktur wird mittels Klassen- oder Kompositionsstrukturdiagramme spezifiziert. Hierdurch ist es möglich, die einzelnen Komponenten in kleinere Bestandteile zu unterteilen. Ebenfalls können Ports und deren Schnittstellen an die verfeinerten Elemente delegiert werden. So ist eine weitere Aufteilung der zu realisierenden Operationen gegeben. In der Abbildung 4.41 ist ein Beispiel für die Softwarekomponente *Alarmanlagensteuerung* beschrieben. Diese ist in die Klassen *Timer*, *Diagnose Alarm* und *Überwachung* aufgeteilt. Die Schnittstelle zur Diagnosekomponente wird dabei an die Klasse *Diagnose_Alarm* delegiert, so dass diese die Schnittstelle mit ihren Operationen zu realisieren hat.

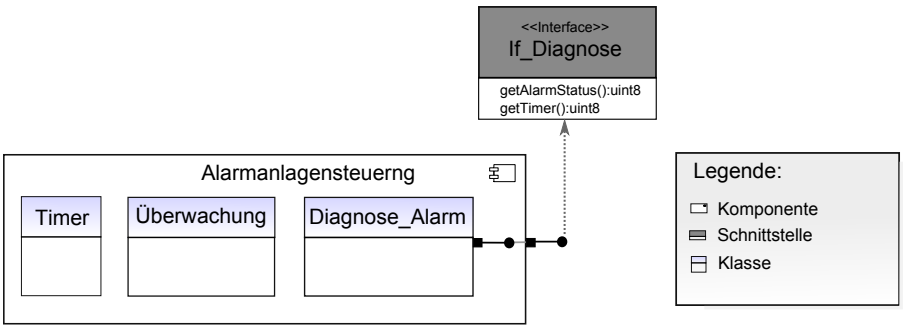


Abb. 4.41. Aufteilung einer Softwarekomponente inklusive Delegation einer Schnittstelle

Zusätzlich zur Verfeinerung der Struktur muss innerhalb des Feindesigns die zu verwendende Datenstruktur mit ihren Datentypen festgelegt werden. Hierfür kommt der UML Datentyp und entsprechende Ableitungen zur Anwendung. So wird beispielsweise der Datentyp *uint16* festgelegt, der einen Zahlenwert mit 16 Bit darstellt. Die im Modell erstellten Datentypen werden dann in den einzelnen Operationen und Attributen verwendet. Innerhalb der AutoMoMe wurden die Datentypen so erweitert, dass sie zusätzlich auch die Skalierung, die verwendete Einheit und eine Beschreibung beinhalten (vgl. Abbildung 4.42). Hierdurch werden notwendige Umrechnungen erkannt. Darüber hinaus werden Datenfehler wie z. B. beim Runden von Zahlen [Huc99] vermieden.

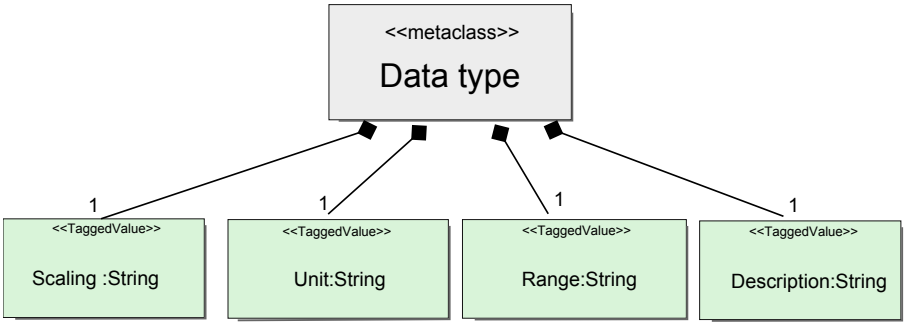


Abb. 4.42. Die Eigenschaftswerte für einen Datentyp

Eine Erweiterung des Ansatzes AutoMoMe betrifft die Operationen, die im Feindesign festgelegt werden. Durch zusätzliche Eigenschaftswerte (Tagged Values) werden die Laufzeiten der Operationen bzw. die Aktivierungsform (benötigt für die AUTOSAR Transformation) spezifiziert. Durch die Spezifikation der Laufzeiten werden bei der Ausführung der Echtzeitsimulation bessere Ergebnisse erzielt, da die Simulation nun nicht mehr auf die Abschätzungen der Tasks angewiesen ist, sondern die genaueren Daten der Operation verwenden kann. Die Laufzeiten werden zunächst geschätzt und dann durch entsprechende Messungen an einem Prototyp genauer vorgegeben. Die Ermittlung der Ausführungszeiten wird ebenfalls gespeichert, um zu dokumentieren wie genau die Ausführungszeiten zu bewerten sind. Die Art der Aktivierung der Operation wird bei der späteren Transformation nach AUTOSAR benötigt und daher bereits hier vorgenommen (siehe Kapitel 6.2.2). In der Abbildung 4.43 ist dies für die Operation *aktiviereAlarmanlage* dargestellt.

Das Verhalten der Operationen wird in AutoMoMe mittels Aktivitätsdiagramme definiert. Für besonders sicherheitskritische Operationen sind zusätzliche Sequenzdiagramme vorgesehen. Bei diesen Sequenzdiagrammen ist jedoch zu beachten, dass sie nur ein einzelnes Szenario beschreiben. Für eine allgemeine Verhaltensbeschreibung sind daher vielfach mehrere Szenarien erforderlich [Gre11]. Für das Beispiel des Komfortsteuergerätes ist das Verhalten

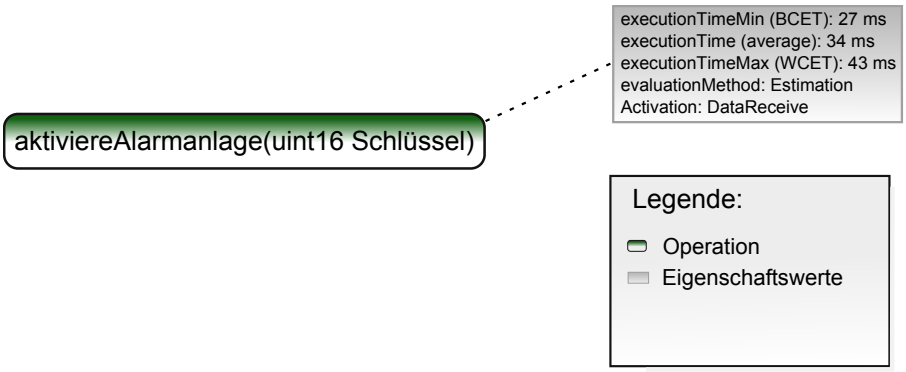


Abb. 4.43. Erweiterungen bei einer Operation

der Operation mittels eines Aktivitätsdiagramms *aktiviereAlarmanlage(uint 16 Schlüssel)* dargestellt (vgl. Abbildung 4.44). Es ist in dem Diagramm zu erkennen, dass die Operation als Argument Schlüsseldaten erhält, die dann gleich validiert werden. Anschließend wird der Innenraum überprüft, bevor die Alarmanlage eingeschaltet wird. Aus diesem oder ähnlichen Diagrammen wird dann später durch Codegenerierung die Struktur im Quellcode automatisch erzeugt, der dann durch den Entwickler vervollständigt wird.

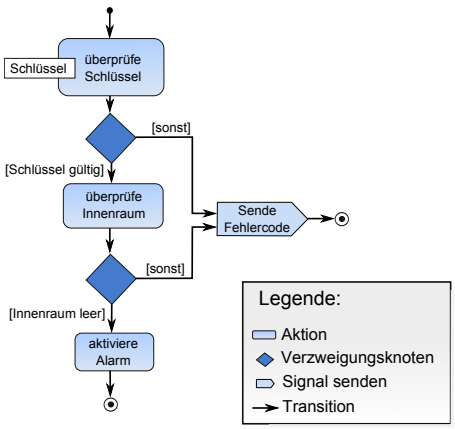


Abb. 4.44. Spezifikation der Operation *aktivereAlarmanlage* mittels eines Aktivitätsdiagramms

Feindesign mit Matlab/Simulink

Es kann aber nicht immer davon ausgegangen werden, dass jede Modellierung mithilfe der UML vorgenommen werden kann. So wird für das Feindesign schon seit geraumer Zeit Matlab/Simulink [Mat14] und darauf aufbauende Werkzeuge, wie Targetlink [dSP14], erfolgreich eingesetzt. Dies geschieht vor allem im Bereich der Reglerentwicklung, in der vielfach kontinuierliche Daten benötigt werden. Aus diesem Grunde sind bereits viele Modelle in Matlab/Simulink vorhanden, die wiederverwendet werden. Ferner wurden Techniken für die Prototypenentwicklung (Rapid Prototyping) entwickelt, die weiterhin eingesetzt werden. Von daher ist eine Synchronisation mit der Architekturmodellierung erforderlich, so dass es zu keinen Inkonsistenzen zwischen den jeweiligen Modellen kommen kann. Eine Arbeit in diesem Bereich ist bereits vorhanden [SPC+13]. Jedoch sorgt dieser Ansatz nur dafür, dass eine durchgängige Simulation zwischen beiden Modellen erfolgen kann. Die Verwendung von inkonsistenten Schnittstellen ist hierbei nicht ausgeschlossen. Ein anderer Ansatz [SPC+13] sorgt für die Generierung der Ports und Schnittstellen. Jedoch unterteilt er nicht das Simulink Modell in mehrere Subsysteme, so dass dort keine Wiederverwendung in Form von Bibliotheken erfolgen kann. Vielmehr wird ein Konzept benötigt, dass die Schnittstellen automatisch erstellt bzw. abgleicht und gleichzeitig die Erzeugung von Subsystemen ermöglicht. Dabei müssen beide Richtungen (Matlab/Simuling –> UML aber auch UML –> Matlab/Simulink) unterstützt werden, um ein Top-Down bzw. Bottom-Up Verfahren zu ermöglichen. Ein solches Konzept wird in AutoMoMe umgesetzt. Die notwendigen Daten werden durch eine Textdatei ausgetauscht und durch Automatismen in die jeweiligen Modelle importiert bzw. exportiert (vgl. Abbildung 4.45).

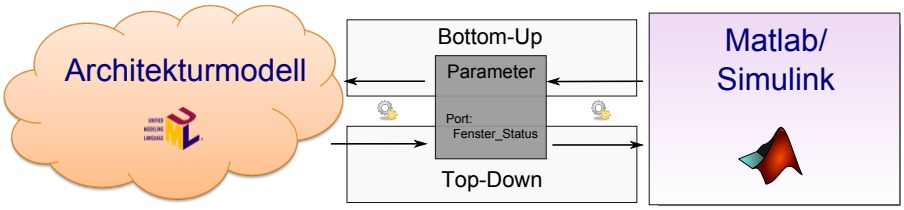


Abb. 4.45. Austausch zwischem dem Architekturmodell und Matlab/Simulink

In der Architekturmodellierung mit UML wird eine Komponente, die in Matlab/Simulink realisiert wird, durch den Stereotyp «Matlab» eindeutig gekennzeichnet. Die Kommunikation zu den anderen Komponenten wird in Form von Datenfluss-Schnittstellen (FlowPorts) beschrieben. Diese Schnittstellen werden unter zur Hilfenahme der Textdatei in das entsprechende Matlab/Simulink Format transformiert. Hierzu werden die

Ports und Schnittstelleneigenschaften in die Blockschaltbilder von Matlab/Simulink überführt. Beim Komfortsteuergerät wird beispielsweise der Klemmschutz der Fenster in Matlab/Simulink realisiert. Dies wird in Abbildung 4.46 verdeutlicht. Aus den Ports und den Schnittstellen wird nun ein Blockschaltbild generiert, in dem die ein- und ausgehenden Signale entsprechend den Architekturschnittstellen definiert sind (vgl. Abbildung 4.46). Dabei wird der Block als Untersystem angelegt, um ihn innerhalb von Matlab/Simulink weiter zu nutzen.

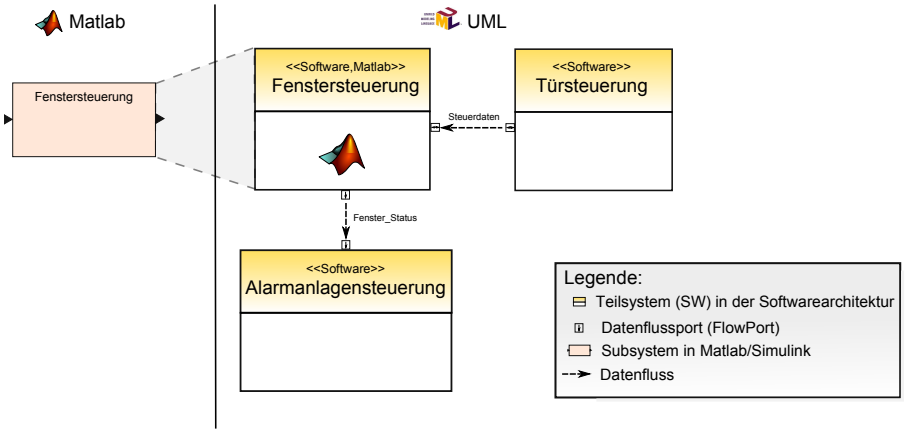


Abb. 4.46. Integration von Matlab/Simulink in das Architekturmodell

Somit ist es möglich, aus der Architektur eine Anbindung an Matlab/Simulink herzustellen. Dieser Top-Down Ansatz garantiert die Konsistenz trotz Nutzung zweier verschiedener Modelle. Jedoch kann nicht immer von einem Top-Down Ansatz ausgegangen werden, da bereits schon Modelle in Matlab/Simulink vorliegen bzw. solche parallel zur Architekturmodellierung erstellt werden. Deswegen kann diese Technik gleichermaßen als Bottom-Up Ansatz genutzt werden. Aus dem Matlab/Simulink Modell wird das äußere Subsystem mit seinen Ports exportiert und in die Architekturmodellierung importiert. Hier wird daraus eine Komponente mit den entsprechenden Ports angelegt.

Modellierung der Basis-Software

Bei der Entwicklung ohne AUTOSAR Standard erfolgt die Modellierung und Implementierung entsprechend der eben beschriebenen Methodik für das Feindesign (vgl. Kapitel 4.3.5). Dabei werden die Klassen mit ihren Schnittstellen in Operationen und Attribute weiter verfeinert und das Verhalten wird durch Aktivitäts- und Zustandsdiagramme spezifiziert. Bei der Modellierung einer

AUTOSAR Basis-Software Komponente ist dieses Vorgehen auf die Erstellung der Implementierung übertragbar.

Jedoch ist noch ein zusätzlicher Aspekt zu berücksichtigen. Für jede AUTOSAR Basis-Software Komponente muss eine Konfigurationsdatei auf Basis des AUTSOAR XML Formats (Arxml) erstellt werden. Dies ist die sogenannte *Basis Software Module Description (BSWMD)* Datei. In dieser sind die möglichen Konfigurationsparameter aufgelistet, aus denen im Projekt die spezifischen Konfigurationsparameter ausgewählt werden. Von daher ist die BSWMD Datei für jede eigene erstellte Basis-Software Komponente zu erzeugen. Da ein Teil der Basis-Software vom automobilen Zulieferer selbst entwickelt wird, muss er für die Erstellung der BSWMD Datei für diese Komponenten sorgen.

Bislang ist die Erstellung der BSWMD Datei eine manuelle Aufgabe, die zum Großteil in entsprechenden XML-Editoren durchgeführt wird. Nachteilig hieran ist, dass auf Grund der Größe der Datei bzw. der Anzahl der Konfigurationsparameter und der häufigen Referenzen innerhalb der Konfigurationsdatei keine gute Übersicht vorhanden ist. Ebenso bieten viele AUTOSAR Konfigurationswerkzeuge nur eine tabellarische Darstellung der vorhandenen Parameter an, so dass es für den Entwickler bzw. dem Integrator mühevoll ist, die geeignete Konfiguration vorzunehmen. Aus diesem Grunde wurde in AutoMoMe eine grafische auf UML Diagrammen basierende Methode zur Darstellung der Konfigurationsparameter für eine BSWMD Datei erarbeitet.

Zum besseren Verständnis für die erstellte Lösung wird zunächst der Aufbau der Konfigurationsdatei vorgestellt. Die einzelnen Konfigurationsparameter werden in Form von AUTOSAR Elementen (beispielsweise *Integer-Param-DEF* für ganzzahlige Werte oder aber *Float-Param-DEF* für Gleitkommazahlen) spezifiziert. Diese Elemente werden anschließend in Gruppen (sogenannte *Module-Param-DEFs*) zusammengefasst. Die einzelnen Gruppen sind ineinander verschachtelt, um dadurch einen Baum aus Konfigurationsparametern entstehen zu lassen. Vorbild für diese Modellierungsform ist die Spezifikation der AUTOSAR XML Datei mittels UML [AUT10j], nur dass in diesem Fall eine BSWMD Arxml-Datei erstellt wird. Für die grafische Notation werden UML Objekte eingesetzt, um die AUTOSAR Gruppen und Elemente zu repräsentieren. Die Schachtelung der einzelnen Elemente ineinander wird mittels von UML Kompositionen vorgenommen. Die Referenzen innerhalb der Konfigurationsparameter werden durch UML Abhängigkeiten (Dependencies) dargestellt. Vorteilhaft bei der Verwendung der UML ist, dass die Konfiguration auf zahlreiche Diagramme verteilt werden kann, so dass die Übersichtlichkeit auch bei größeren Konfigurationsdateien gegeben ist. Dies ist ein Nachteil bei der Erstellung innerhalb von XML-Editoren.

In der Abbildung 4.47 ist ein Ausschnitt aus der Konfigurationsdatei für die Basis-Software Komponente des ADC Treibers von dem Komfortsteuergerät dargestellt. Ein ADC Treiber ist für die Initialisierung und Steuerung der

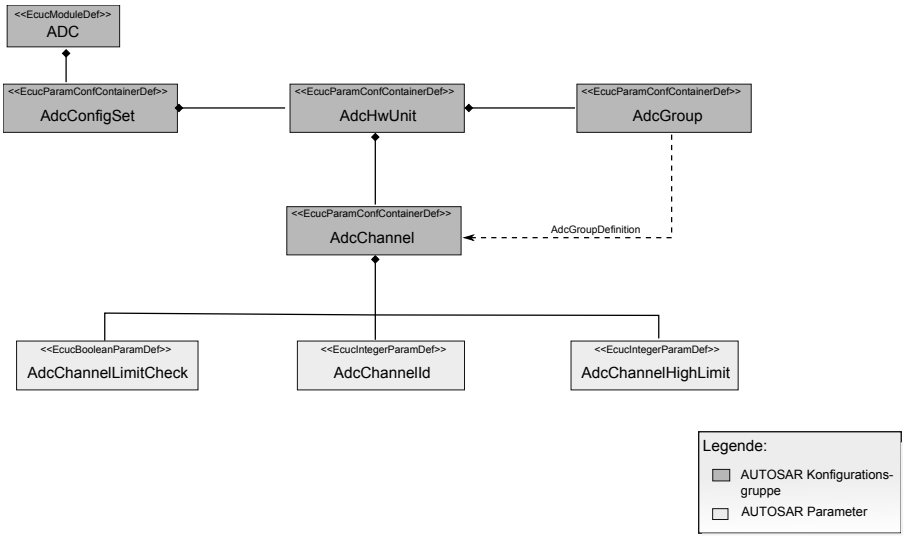


Abb. 4.47. Ausschnitt aus der Konfiguration des ADC Treibers

internen Analog/Digital Konvertierung eines MikroControllers zuständig. In dem hier vorgestellten Ausschnitt wird die Konfiguration eines Kanals für die Konvertierung beschrieben. Es ist ersichtlich, dass bereits für diese Konfiguration eine tiefe Verschachtelung der Elemente vorliegt. Erwähnenswert ist, dass besonders die Referenzen innerhalb der Konfigurationsdatei zu erkennen sind. Hierdurch wird die Übersichtlichkeit und die Navigierbarkeit deutlich verbessert.

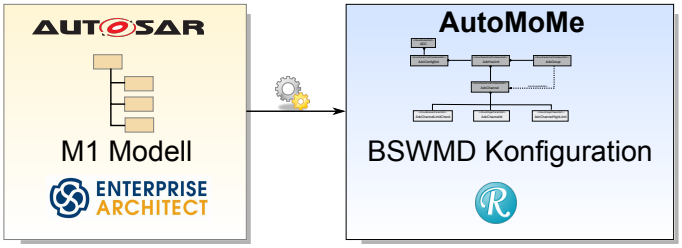


Abb. 4.48. Import von BSWMD Konfigurationsdaten aus dem AUTOSAR M1 Modell

Neben der Erstellung der grafischen Darstellung der Konfigurationsparameter wird auch eine schnelle Erstellung der Diagramme berücksichtigt. Hierzu kann auf das AUTOSAR Metamodell [AUT10b] zurückgegriffen werden. In dem Abschnitt der M1 Modellierung des Metamodells wird für die meisten Basis-Softwaremodule eine Standard Konfiguration bereitgestellt. Diese wird

nun mittels einer Modelltransformation in die für AutoMoMe entwickelten Diagramme übertragen (vgl. Abbildung 4.48). Dabei kann auf eine 1:1 Abbildung zurückgegriffen werden, da auf beiden Seiten die gleichen Modellelemente zu finden sind. Mittels der Transformation ist eine Anfangskonfiguration gegeben, die anschließend an das Design und die Implementierung des spezifischen Moduls angepasst werden kann. Auf Grund der Anfangskonfiguration sind bereits weite Teile modelliert, die nun nicht mehr von Grund auf erstellt werden müssen. Insofern wird durch den automatisierten Import der Konfigurationsdaten eine Zeitersparnis bei der Modellierung erreicht werden.

Integration von bestehender Software

Der Ansatz AutoMoMe unterstellt, dass die Software neu entwickelt wird bzw. dass entsprechende UML Modelle vorhanden sind, um daraus die Gesamtarchitektur für das Steuergerät zu entwickeln. Es muss jedoch davon ausgegangen werden, dass einige Komponenten bereits seit einiger Zeit verwendet werden und daher die Entwicklungszeit einige Jahre zurückliegt. Damals wurden andere Modellierungsformen verwendet bzw. es wurde direkt implementiert, so dass kein oder nur rudimentäre Modelle zur Verfügung stehen. Um diese Komponenten trotzdem in das Architekturmodell zu integrieren, gibt es zwei wesentliche Ansätze. Falls ältere Modelle existieren, so können Transformationen entwickelt werden, um die Modelle nach UML zu übertragen. Exemplarisch kann auf [Fri06] verwiesen werden, in dem eine Transformation von der strukturierten Analyse [DeM79], die häufig bei der Modellierung von eingebetteten Systemen verwendet wurde, nach UML definiert wird.

Falls keine Modelle existieren, kann ein Reverse Engineering vom Code durchgeführt werden. Für die strukturellen Elemente bieten dies bereits viele UML Werkzeuge an [IBM14b]. Jedoch kommt es nicht nur auf die strukturellen Elemente an, sondern ebenso auf das Verhalten. Für ein ereignisbasiertes Verhalten wurde in [HMSN10c, HMSN10b, HMSN10a] mittels von Lern-Algorithmen ein Ansatz entwickelt, um aus bestehenden Code entsprechende UML Zustandsdiagramme zu generieren. Die durch Reverse Engineering erzeugten Modelle können dann anschließend im Architekturmodell genutzt und wiederverwendet werden.

Somit existieren bereits Ansätze und Techniken wie bereits vorhandene Komponenten im Architekturmodell weiter genutzt werden können. Von daher ist neben dem Top-Down Ansatz ebenso ein Bottom-Up Ansatz bei der Erstellung des Architekturmodells möglich.

4.3.6 Die technische Sichtweise

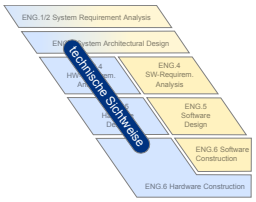


Abb. 4.49. Einordnung der technischen Sichtweise in das V-Modell

Neben der funktionalen und der logischen Sichtweise existiert noch eine dritte und zwar die technische Sichtweise. In der Abbildung 4.49 ist die Einordnung der technischen Sichtweise in das V-Modell zu sehen. Beginnend bei der Systemarchitektur betrifft sie hauptsächlich die hardwareabhängigen Bereiche des Systems. Sie ist bei den automobilen Steuergeräten, wie auch bei anderen eingebetteten Systemen, eine besondere Sichtweise. Bei der technischen Sichtweise werden die physikalischen Bestandteile wie beispielsweise die Rechenkerne oder der Speicher spezifiziert (vgl. Abbildung 4.50). Des Weiteren wird die Verteilung der logischen Elemente auf die physikalischen Bestandteile definiert. Diese Verteilung kann jedoch enorme Auswirkungen auf die Leistung eines Systems haben. Von daher hat die technische Sichtweise eine besondere Bedeutung. In AutoMoMe wird sie durch die SysML und entsprechenden Anpassungen an den automobilen Steuergerätesektor realisiert und ist Bestandteil der Systemarchitektur. Somit vervollständigt die technische Sichtweise die Systemarchitektur.

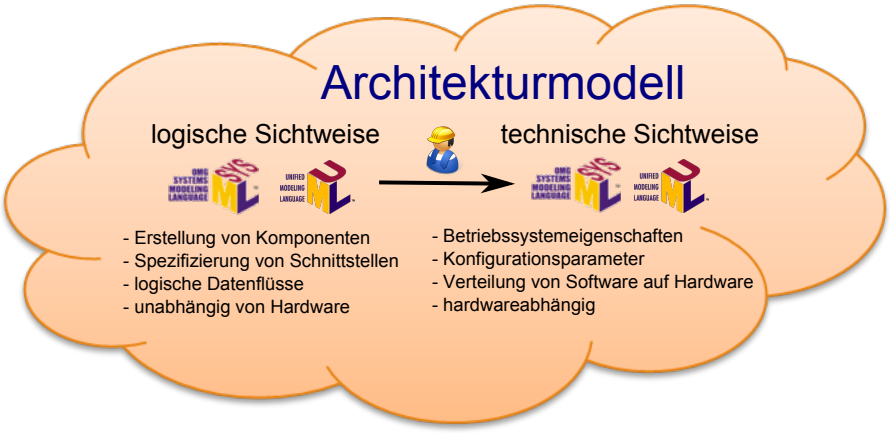


Abb. 4.50. Unterschiede der logischen und technischen Architektur im Architekturmodell

Durch die zunehmende Nutzung von mehreren Prozessoren innerhalb eines Steuergerätes (sogenannte Multiprozessorsysteme [Tan02]) bzw. die Zusammenarbeit von Steuergeräten untereinander, wird bei einem automobilen Steuergerät oftmals von einem verteilten System gesprochen. Hierdurch erhöht sich

die Bedeutung der technischen Sichtweise der Systemarchitektur. Es wird auf der einen Seite sichergestellt, dass die qualitativen Anforderungen erfüllt werden. Die technische Sichtweise sorgt somit dafür, dass die Skalierbarkeit, die Performance und die Verfügbarkeit als exemplarische Bestandteile der qualitativen Anforderungen eingehalten werden [Sta09b]. Hierzu werden bereits in der Analysephase Zeitabhängigkeiten und Ressourcenprobleme spezifiziert [BS05]. Diese müssen sich aber in der Architektur ebenfalls wiederfinden, damit sie entsprechend berücksichtigt werden. Auf der anderen Seite sorgt sie für eine Beherrschung der Komplexität, da die Architektur das Zusammenspiel der einzelnen Komponenten und daher die Struktur der Software aufzeigt. Dies muss bereits frühzeitig in der Entwicklung erfolgen, da ansonsten viele Informationen und Entscheidungen mehrdeutig sind und dann jeweils zu einem erhöhten Kommunikations- bzw. Mehraufwand führen [Zör12].

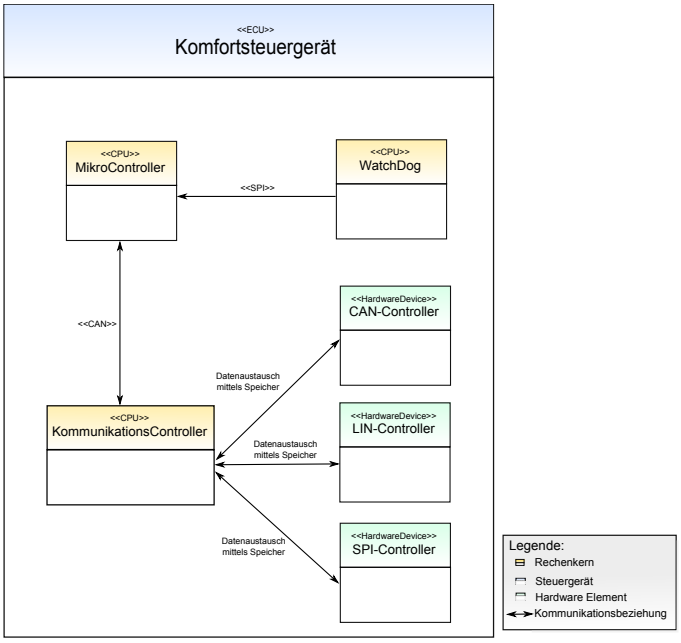


Abb. 4.51. Oberste Hardwaretopologie des Komfortsteuergerätes

Zunächst müssen die physikalischen Bestandteile modelliert werden. Der Ausgangspunkt ist das Kontextdiagramm aus der logischen Sichtweise (vgl. Abbildung 4.25). In diesem Diagramm wird bereits die Netzwerktopologie zu den anderen Steuergeräten beschrieben. Es wird festgelegt, an welchen Kommunikationsbussen das Komfortsteuergerät angeschlossen ist. Hieraus ergeben sich die notwendigen Netzwerkadapter, die das Steuergerät aufweisen muss. Zusätzlich müssen dann noch die Rechenkerne und der Speicher modelliert werden, um

die physikalischen Bestandteile zu vervollständigen. In Abbildung 4.51 ist ein Ausschnitt der technischen Sichtweise des Komfortsteuergerätes dargestellt. Es werden die drei CPUs (MikroController, WatchDog und KommunikationsController) und die drei Netzwerkadapter (CAN-, LIN- und SPI-Controller) abgebildet. Diese sind durch die neu erstellten Stereotypen «CPU» und «HardwareDevice» gekennzeichnet. Hierdurch werden sie formal beschrieben und es werden ihnen zusätzliche Eigenschaftswerte zugeordnet. So wird bei einer CPU beispielsweise die Geschwindigkeit des Prozessors oder bei einem Kommunikationsbus die Übertragungsgeschwindigkeit hinterlegt. Dies sind für die Laufzeit entscheidende Informationen. Sie werden in der Entwicklung an vielen Stellen weiter benötigt.

Die Ressourcennutzung und die entstehenden Laufzeiten müssen explizit modelliert werden, damit eine spätere Analyse bzw. Simulation zur Verifikation des Systems erfolgen kann. Denn nur wenn die Daten formal spezifiziert sind, sind sie automatisiert zu verarbeiten. Ebenso ist es möglich, bei der Verteilung der Software auf die Hardware Unterstützung zu geben. So werden Planungsalgorithmen eingesetzt, um eine optimale Verteilung von Software auf Hardware – auch unter sicherheitskritischen Punkten – vorzunehmen (siehe hierzu [KMTH10][NSH+10]).

Als weiteren Vorteil ist herauszustellen, dass bei der späteren Integration keine Komplikationen auftreten, die nur mit enormen Ressourcenaufwand behoben werden können, da eine eindeutige Beschreibung vorliegt. Von daher muss es Ziel sein, dass bereits in frühen Entwicklungsphasen ein virtuelles System aus der technischen und der logischen Sichtweise entsteht, in dem die grundlegenden Fragestellungen analysiert und verifiziert werden [BS05]. Hierfür muss die SysML jedoch noch weiter an den automobilen Sektor angepasst werden. So müssen zusätzlich zu den physikalischen Bestandteilen in der technischen Sichtweise auch noch die Betriebssystemeigenschaften und auch die Speicherplatzbelegung modelliert werden. Diese im Ansatz AutoMoMe entwickelten Erweiterungen werden in den nächsten Abschnitten (siehe Kapitel 4.3.7 und 4.3.8) detailliert beschrieben.

Nachdem die technische Sichtweise modelliert ist, werden anschließend die Komponenten der logischen Softwarearchitektur auf die technische Architekturen abgebildet. Es besteht die Möglichkeit, eine logische Architektur auf verschiedene technische Architekturen abzubilden. So werden unterschiedliche Generationen von Steuergeräten entwickelt, die auf unterschiedlichen Hardwareelementen basieren. Hierdurch ist es möglich, dass ein System an spezielle Kriterien angepasst werden kann wie z. B. die Performance [Bec06]. Das Verteilen der logischen Komponenten auf Elemente der technischen Architektur ist ein manueller Arbeitsschritt (vgl. Abbildung 4.50), d. h. es muss eine Abbildung zwischen den logischen Komponenten und der technischen Architektur erstellt werden. Hierzu wird die standardmäßige «allocate» Beziehung der SysML eingesetzt.

4.3.7 Erweiterung des Architekturmodells um Betriebssystemeigenschaften

Die automobilen Steuergeräte benötigen wie jeder Computer ein Betriebssystem, das die Verarbeitung regelt. Wie bei allen eingebetteten Systemen hat das Betriebssystem einen großen Einfluss auf die Performance des Systems, da nur beschränkte Ressourcen zur Verfügung stehen und trotzdem eine Echtzeitverarbeitung erfolgen muss [Kop97]. Von daher wird bei einem automobilen Steuergerät ein Echtzeitbetriebssystem eingesetzt. In den überwiegenden Fällen ist dies ein OSEK [OSE05] oder ein AUTOSAR Betriebssystem (vgl. Kapitel 2.6.5). Das AUTOSAR Betriebssystem ist eine Erweiterung des OSEK Betriebssystem. Nachfolgend wird dargestellt, wie die Spezifikation eines Betriebssystems auf formale Art und Weise erfolgt.

Das OSEK Betriebssystem ist durch eine hohe Konfiguration und statische Allokation gekennzeichnet [OSE05]. Eine statische Allokation bedeutet, alle Einstellungen, die Ressourcenanforderungen und -belegungen sowie die benutzten Dienste sind bereits bei der Entwicklung bekannt. Daher unterstützt ein OSEK Betriebssystem keine dynamischen Belegungen und kann aus diesem Grunde deutlich schlanker und effizienter entwickelt werden [OSE05]. So ist es möglich, die kompletten Betriebssystemeigenschaften bereits zur Entwicklungszeit festzulegen und zu verifizieren.

Die bisherige informelle Speicherung in externen Dateien (Excel Dateien oder DOORS Modulen) ist keinesfalls ausreichend, wie in der Motivation (vgl. Kapitel 1.1) bereits ausgeführt. Es bietet sich an, die Informationen im Architekturmodell abzulegen. Hierdurch werden sie eindeutig definiert und stehen zusätzlich für eine weitergehende Analyse oder Weiterverwendung automatisch zur Verfügung (wie beispielsweise in [LMD+09, LSG+09]). In dem hier beschriebenen Ansatz AutoMoMe werden die Betriebssystemeigenschaften sowohl automatisiert in die Echtzeitsimulation übernommen als auch zur Konfiguration des Betriebssystems genutzt. Zum letzteren Einsatzzweck wird für nicht AUTOSAR Systeme die OSEK Implementation Language (OIL) eingesetzt (vgl. Abbildung 4.52), um die Daten in einem Konfigurator-Werkzeug weiterzuverwenden. So wurde in AutoMoMe sowohl ein Import- als auch ein Exportmechanismus entwickelt, um auch bestehende Daten (in Form von OIL-Dateien) wieder zu verwenden. Für AUTOSAR Systeme wird die Beschreibung mittels der ECUC Datei importiert bzw. exportiert.

Die Modellierung von Betriebssystemeigenschaften

Eine Modellierung von Betriebssystemeigenschaften ist in der UML und ebenso in der SysML standardmäßig nicht vorgesehen [TGDT08]. Eine Option zur Modellierung der Betriebssystemeigenschaften ist die Nutzung des SPT

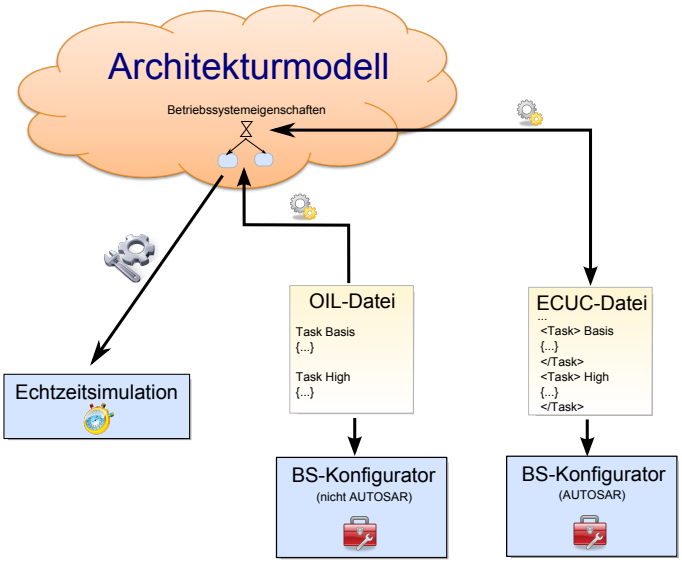


Abb. 4.52. Import/Export von Betriebssystemeigenschaften

[GWS03, Moo02] oder des MARTE Profils [TGDT08, TRGD07] (vgl. Kapitel 2.3.2). In diesem Profil werden in den Paketen *Hardware Resource Modeling - HRM bzw. Software Resource Modeling SRM* Stereotypen definiert, die eine Beschreibung von Betriebssystemen gestatten. Jedoch sind mit dem MARTE Profil noch nicht alle benötigten Eigenschaften für die Modellierung von automobilen Steuergeräten gegeben. So ist beispielsweise nicht die Reihenfolge von Operationsaufrufen modellierbar. Als einen weiteren Nachteil ist anzusehen, dass im MARTE Profil sehr allgemeine Formulierungen verwendet werden, die für den Automobilbereich weiter detailliert werden müssen, damit die Entwickler eine wirkliche domänenspezifische Sprache verwenden können, die ihrem Sprachgebrauch angepasst ist. Nur so sind alle Vorteile von domänenspezifischen Sprachen bei der Modellierung von Betriebssystemeigenschaften auszuschöpfen [Tho04].

Im Folgenden wird anhand des Beispiels des Komfortsteuergerätes die Modellierung der Betriebssystemeigenschaften dargestellt. Dabei wird zunächst auf die Modellierung durch das MARTE Profil eingegangen. Anschließend wird die darauf aufbauende Erweiterung im Rahmen des Ansatzes AutoMoMe vorgestellt, die die vom MARTE Profil zur Verfügung gestellten Stereotypen an die automobile Sprache und an die entsprechenden Bedingungen anpasst. Das daraus resultierende Metamodell ist in Abbildung A.3 im Anhang abgebildet.

Zunächst gilt es, die entsprechenden Rechenkerne und ihre Kommunikationsbusse zu spezifizieren, auf denen die einzelnen Softwarekomponenten rechnen und somit ihre Arbeit verrichten. Im MARTE Profil werden Rechenkerne

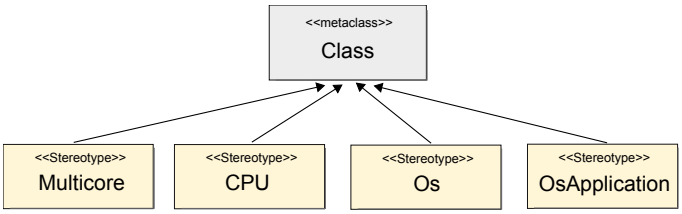


Abb. 4.53. Stereotypen für die Betriebssystemkonfiguration

durch den Stereotyp «HW_Processor» definiert. Dieser wird in der Automobilbranche oft als CPU bezeichnet. Somit wird ein entsprechender Stereotyp vom MARTE Element abgeleitet. Ferner wird der Stereotyp «Multicore» eingefügt, um den zunehmenden Einsatz von Mehrprozessorsystemen zu unterstützen (vgl. Abbildung 4.53). Dies wird ebenso im AUTOSAR 4.0 Standard so gesehen und eine entsprechende Unterstützung eines Mehrprozessorsystems ist dort vorgesehen [AUT10h, SKS10]. Jedoch bedarf es hierbei einer frühzeitigen Betrachtung innerhalb der Entwicklung, um die Abhängigkeiten aufzuzeigen und zu berücksichtigen [BSER11].

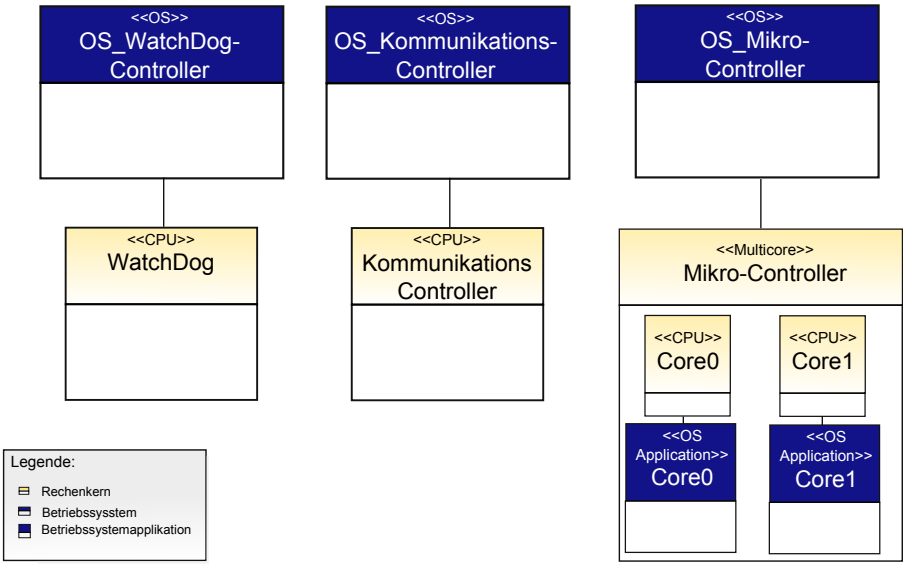


Abb. 4.54. Spezifikation der Rechenkerne für das Komfortsteuergerät

Kommunikationsbusse werden im MARTE Profil durch den Stereotyp «CommunicationMedia» beschrieben. Dieser Begriff ist im Automobilbereich nicht bekannt, da nur eine begrenzte Anzahl von Kommunikationsbussen eingesetzt werden. Deswegen werden die Stereotypen «LIN», «CAN», «FlexRay» und

«SPI» definiert, die die Kommunikation eindeutig beschreiben (vgl. Abbildung 4.24). Im Komfortsteuergerät gibt es die drei Rechenkerne *MikroController*, *KommunikationsController* und *WatchDog* (vgl. Abbildung 4.51), die durch den Stereotype «CPU» beschrieben sind und mittels eines SPI- und eines CAN-Busses miteinander kommunizieren.

Auf jedem dieser drei Rechenkerne läuft ein Betriebssystem, welches die Reihenfolge der zu berechnenden Softwareanteile festlegt. Dies wird in der Abbildung 4.55 gezeigt, in der die Rechenkerne und die Betriebssysteme für das Komfortsteuergerät modelliert sind. Dabei wird zur Vervollständigung der *MikroController* ein Multiprozessorsystem modelliert. In einem Steuergerät kann eine Software auf verschiedene Arten Zugriff auf den Rechenkern erhalten. Dies ist zum einen die zyklische Aktivierung in Form von Tasks und zum anderen die ereignisgesteuerte Aktivierung mittels Ereignisse (Interrupts). Mit der zyklischen Aktivierung werden die wiederkehrenden Funktionalitäten des Steuergerätes in Form von Tasks oder erweiterten Tasks (extendedTasks) modelliert [LLP+09]. Im Komfortsteuergerät sind dies beispielsweise die Überprüfung des MikroControllers oder die Abfrage von Sensoren in einem vordefinierten Zeitabstand. Die Ereignisse, die abhängig von dem Benutzer sind, werden durch Interrupts modelliert. Im Komfortsteuergerät ist dies z. B. die Betätigung des Fensterhebers zum Öffnen/Schließen eines Fensters.

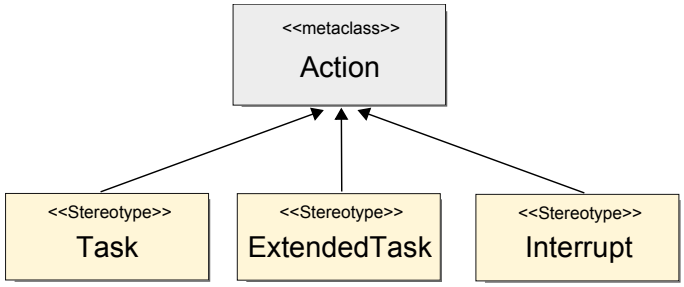


Abb. 4.55. Stereotypen zur Modellierung von Tasks und Interrupts

Betrachtet man zunächst die zyklischen Aktivierungen, so muss eine Task und die Funktionen innerhalb der Task spezifiziert werden. In [Dou11] wird für eine Task eine aktive Klasse verwendet. Hierdurch ist aber keine zyklische Aktivierung modellierbar. Im MARTE Profil werden Tasks mittels des Stereotyps «SchedulableRessource» modelliert. Jedoch besitzt dieser Stereotyp Eigenschaften, die beim OSEK bzw. AUTOSAR Betriebssystem nicht benötigt werden wie z. B. Services. Für die Echtzeitsimulation werden dagegen teilweise noch zusätzliche Informationen wie eine Ausführungszeit bzw. der Offset benötigt. Deswegen wird der Stereotyp «Task» von diesem Stereotyp abgeleitet (vgl. Abbildung 4.55). Die Zuordnung der Task zu einer CPU erfolgt mittels des Stereotyps «allocate». In der Darstellung 4.56 ist abgebildet,

dass die Tasks *Licht* und *Basis* dem MikroController während die Task *Kommunikation* dem *KommunikationsController* zugeordnet sind. Ferner kann der Abbildung entnommen werden, dass die erweiterte (extended Task) *Datenverarbeitung* dem *KommunikationsController* zugewiesen ist. Im Gegensatz zu einer Task kann die erweiterte Task wieder in den wartend Zustand wechseln. Dies geschieht, wenn sie auf ein Ereignis warten muss.

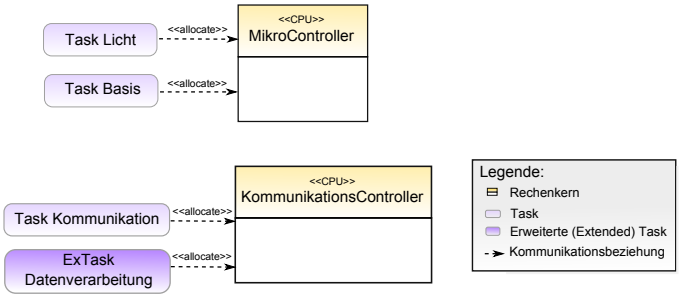


Abb. 4.56. Zuordnung von Tasks zu einer CPU

In Abbildung 4.57 werden die Tasks und ihr zyklischer Aufruf für den MikroController und dem KommunikationsController gezeigt. Durch das Sanduhrsymbol wird die zyklische Aktivierung dargestellt. In diesem Fall wird die Task *Basis* alle 25 ms, die Task *Licht* alle 10 ms und die Task *Kommunikation* alle 5 ms aktiviert. Die zusätzlich benötigten Informationen werden in Form von Eigenschaftswerten spezifiziert. In diesen werden beispielsweise die Prioritäten definiert und ob die Tasks unterbrechbar sind oder nicht (reentrant). So besitzt die Task *Kommunikation* mit der Priorität 25 die höchste Einstufung, damit diese Task ggf. eine andere Task verdrängen kann, um die Kommunikation zu jeder Zeit sicherzustellen. Ferner ist in den Eigenschaftswerten der Offset spezifiziert, dass zu Beginn die Tasks nicht alle gleichzeitig aktiviert werden, da ihre zyklische Aktivierung ein Vielfaches von 5 ist. Für die Task *Basis* sind die einzelnen Softwarebestandteile noch nicht spezifiziert. So kann für die gesamte Task eine Laufzeitabschätzung erfolgen. In diesem Beispiel wurde vom Vorgängerprojekt eine Laufzeit von 12 ms für die Task *Basis* festgesetzt. Dies wird entsprechend im Modell festgehalten. Da die Softwarekomponenten für die anderen Tasks bereits im Modell zur Verfügung stehen, weisen die Tasks keine Laufzeiten auf. Vielmehr finden sich die Laufzeiten bei den entsprechenden Softwarefunktionen wieder.

Für die Modellierung der Betriebssystemeigenschaften ist nicht nur die Modellierung der Tasks von Bedeutung, sondern ebenso die Zuordnung von Operationen auf die einzelnen Tasks. Im MARTE Profil geschieht dies durch den Stereotyp «entryPoint». Jedoch ist es anhand dieser Zuordnung nicht möglich festzustellen, in welcher Reihenfolge die Operationen innerhalb der Tasks aufgerufen werden. Ebenso ist es nicht realisierbar, die weitverbreitete Be-

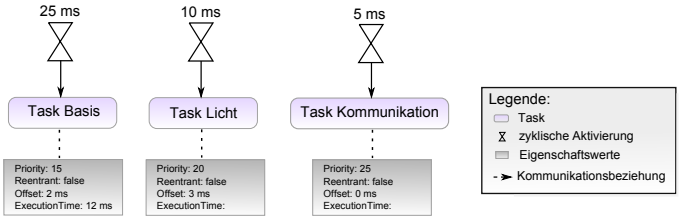


Abb. 4.57. Zyklische Aktivierung von Tasks

nutzung von Operationen nur bei einem bestimmten Taskaufruf auszuführen. Hierdurch wird ein zusätzlicher Taskwechsel im Betriebssystem erspart. Dies ist für die Performance vorteilhaft.

Die Abbildung 4.58 zeigt die Zuordnung der Operationen zur Task *Kommunikation*. Durch die Zahlenfolge an den Pfeilen ist die Reihenfolge der Operationen festgelegt bzw. es ist definiert, ob Operationen nur innerhalb eines bestimmten Taskaufrufes ausgeführt werden. Innerhalb des Taskablaufs wird zunächst die Kommunikation über den CAN Bus bearbeitet. So werden zunächst die Daten vom Bus eingelesen (*(readCANData()*) bevor anschließend die Daten auf den CAN Bus geschrieben werden (*sendCANData()*). In jedem zweiten Aufruf der Task werden die Operationen aufgerufen, die für die Datenverarbeitung auf dem LIN Bus vorgesehen sind. Daran anschließend wird an dritter Stelle die Operation zum Lesen der LIN Daten (*readLINData()*) bzw. an vierter Stelle die Operation zum Schreiben der LIN Daten (*sendLINData()*) aufgerufen.

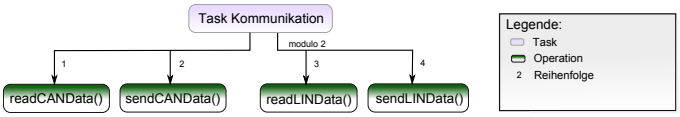


Abb. 4.58. Aufrufreihenfolge von Funktionen

Für die zyklische Aktivierung von Tasks werden im OSEK und somit auch im AUTOSAR Betriebssystem Alarme genutzt. Alarme sind dabei einem Rechenkern zuzuordnen und werden daher als Teil von diesem modelliert. Sie besitzen einen Zähler, der nach einer bestimmten Anzahl von Ticks eine Task aktiviert. Da die Dauer eines Ticks für einen Rechenkern bekannt ist, kann eine exakte Zeit eingestellt werden. Deswegen werden sie in AutoMoMe dazu verwendet die ScheduleTables aus dem AUTOSAR Betriebssystem zu realisieren. In Abbildung 4.59 wird die Modellierung eines Alarms für die zyklische Aktivierung der Task *Basis* gezeigt. Sie wird dabei alle 25 ms aktiviert, wie es bereits innerhalb der zyklischen Aktivierung in Abbildung 4.57 spezifiziert wurde.

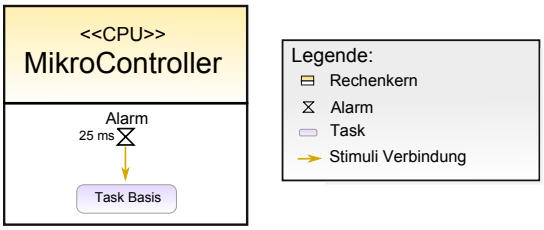


Abb. 4.59. Modellierung von Alarmen

Neben der zyklischen Aktivierung gibt es noch die ereignisgesteuerte Aktivierung. Diese Aktivierungsform ist zur Umsetzung von Ereignissen gedacht, die abhängig vom Benutzer (beispielsweise dem Fahrer) sind. Ein Merkmal dieser Ereignisse ist, dass nicht bekannt ist, wann sie auftreten. Sie können schnell nacheinander auftreten oder es kann eine längere Zeit vergehen, bis sie eintreten. Diese Ereignisse werden mithilfe von Sensoren erfasst. Die Sensoren melden das jeweilige Ereignis an das Steuergerät. Dies kann in Form von analogen oder digitalen Signalen erfolgen. Beim Steuergerät werden die Signale eingelesen und anschließend ausgewertet. Dies erfolgt über die Pins des Steuergerätes. Wird ein Ereignis erkannt, wird ein entsprechender Interrupt im Betriebssystem ausgelöst. Der ausgelöste Interrupt besitzt idealerweise eine höhere Priorität als die zyklischen Tasks. So wird der Interrupt durch den Scheduler bevorzugt verarbeitet und verdrängt ggf. bereits laufende Tasks. Dadurch ist sichergestellt, dass die mit dem Interrupt verbundenen Operationen vorrangig ausgeführt werden.

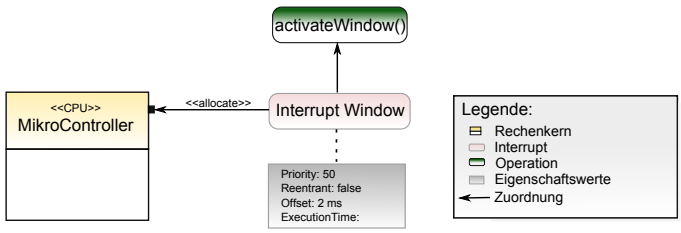


Abb. 4.60. Modellierung von Interrupts

Die Modellierung eines Interrupts wird durch das Ereignis *Fenster öffnen/schließen* beschrieben. Im MARTE Profil wird ein Interrupt durch den Stereotyp «NotificationResource» abgebildet. Dieser wird in dieser Arbeit durch den Stereotyp «Interrupt» – der Sprache im Automobilbereich – angepasst. Die Abbildung 4.60 beschreibt wie ein Interrupt mittels einer «allocate» Beziehung zu einem Port, der für die Modellierung von Interrupts definiert wurde, der CPU zugewiesen wird. Des Weiteren werden die Eigenschaften des Interrupts in Form von Eigenschaftswerten definiert. Die Eigen-

schaftswerte sind dabei ähnlich zu denen einer Task. So besitzt der Interrupt die Priorität 50 und ist nicht unterbrechbar. Bei Aktivierung des Interrupts wird die Operation *activateWindow* ausgeführt, die für das Öffnen/Schließen eines Fensters im Fahrzeug zuständig ist.

Zur Spezifikation der Betriebssystemeigenschaften gehört neben der Modellierung der Tasks und der Zuordnung von Funktionen auf Tasks ebenso der Zugriff auf Ressourcen [Dou00]. Die Zugriffe haben für ein eingebettetes System einen besonderen Stellenwert. Auf der einen Seite wird dadurch festgelegt, ob zusätzliche Sicherungsmaßnahmen bei der Implementierung notwendig sind. Beispielsweise ist bei einem exklusiven Zugriff unterschiedlicher Funktionen auf eine Ressource sicherzustellen, dass der Ressourcenzugriff auf eine Funktion beschränkt ist. Dies kann z. B. mithilfe eines Mutexes oder eines Semaphors erfolgen. Jedoch dürfen diese Absicherungsmaßnahmen sich nicht gegenseitig behindern [Möl09]. Auf der anderen Seite sind die Ressourcenzugriffe von Belang, wenn es um die Performance des Systems geht. Sie können zeitliche Abweichungen hervorrufen, wenn eine Ressource bereits von einer anderen Funktion genutzt wird und somit blockiert ist. Dies ist bei der Entwicklung zu beachten. Die Abbildung 4.61 veranschaulicht den Zugriff der Funktionen *readCANData* und *sendCANData* auf die Ressource *CAN Register*. Die Ressource wird dabei mit dem Stereotyp «Ressource» spezifiziert. In diesem Fall ist ein exklusiver Zugriff bei der Operation *sendCANData* gegeben. Wenn ein gemeinsamer Zugriff auf die Ressource erfolgen kann, wird dies durch den Stereotyp «shared» am Kontrollfluss angegeben, wie in der Abbildung bei *readCANData*. Der Entwickler kann nun geeignete Maßnahmen ergreifen, um das Register entsprechend abzusichern.

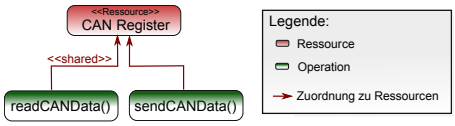


Abb. 4.61. Zuordnung von Funktionen zu Ressourcen

4.3.8 Erweiterung des Architekturmodells um das Speichermanagement

Neben der Rechenzeit gibt es noch eine weitere knappe Ressource. Dies ist der Speicher des Systems. Von daher muss der Speicher sorgfältig verwaltet werden [Tan02]. Die Verwaltung des Speichers erfolgt in einem automobilen Steuergerät im Speichermanagement, das sogenannte Non-Volatile Memory (NVM). Im Speichermanagement wird zum einen festgelegt, welche Daten im Speicher abgelegt werden. So werden sowohl die schreibbaren als auch die nicht beschreibbaren Speicher verwaltet. Zum anderen wird festgelegt, an welcher Stelle die Daten abgelegt werden. Dies ist für die Implementierung unerlässlich, um Speicherzugriffsfehler zu vermeiden.

Die Festlegung der Daten im Speichermanagement ist insbesondere bei der Entwicklung von sicherheitskritischen Systemen ein wesentlicher Aspekt. Denn sicherheitskritische Daten müssen besonders vor Manipulationen, d. h. Veränderung geschützt werden. Dies wird auch von den Standards ISO 26262 [fS10b] und IEC 61508 [IEC99] gefordert. Zur Absicherung können unterschiedliche Verfahren wie das Prüfsummenverfahren, die zyklische Redundanzprüfung (engl. Cyclic Redundancy Check (CRC)) oder aber zusätzliche Absicherungs- bzw. redundant ausgelegte Komponenten eingesetzt werden.

Bei der derzeitigen Entwicklung wird die Aufteilung und die Belegung des Speichers oftmals in informaler Weise dokumentiert, z. B. in Form von Excel-Tabellen. Dies hat jedoch den Nachteil, dass die Absicherung der Daten nicht durch eine automatische Überprüfung erfolgen kann, sondern dass immer ein aufwendiges Review vonnöten ist. Damit eine durchgängige modellbasierte Entwicklung erfolgen kann, sind die Informationen entsprechend zu formalisieren.

Einige Ansätze zur Modellierung des Speichers sind bereits vorhanden, beispielsweise im MARTE Profil (vgl. Kapitel 2.3). Dort befinden sich im Paket *Hardware Resource Modeling (HRM)* Erweiterungen in Form von Stereotypen, die für die Modellierung des Speichers eingesetzt werden können. Die dort vorhandenen Stereotypen «HW_Memory», «HW_RAM» und «HW_ROM» besitzen zahlreiche Eigenschaftswerte, die die Art des Speichers und dessen Größe spezifizieren.

Durch das MARTE Profil sind jedoch nur die physikalischen Eigenschaften des Speichers spezifizierbar. Sie umfassen die Speichergröße, die Speicherart und die Zugriffsgeschwindigkeit. Welche Daten abgelegt und in welcher Form die Ablage der Daten im Speicher realisiert wird, kann nicht spezifiziert werden. Dies ist aber für ein eingebettetes System unerlässlich. Entsprechende Modellierungsmöglichkeiten gerade für sicherheitskritische Bereiche müssen daher geschaffen werden.

Somit beinhaltet AutoMoMe ein Konzept zur Modellierung der Speicherbelegung. Es werden zunächst die physikalischen Eigenschaften des Speichers

beschrieben. Hierzu wird das Paket *Hardware Resource Modeling (HRM)* aus dem MARTE Profil eingesetzt [TRGD07]. Die in dem Paket befindlichen Stereotypen beschreiben die Größe der Hardware, das Gewicht und die Zugriffsgeschwindigkeit.

Diese Beschreibungsform wird erweitert und an die Bedürfnisse der Automobildomäne angepasst. So werden für die einzelnen Steuergeräte immer die gleichen Speicherbereiche wie beispielsweise Konfigurations- oder Statusdaten verwendet. Von daher wird das Speichermanagement in unterschiedliche Bereiche aufgeteilt. Dies ist in der Abbildung 4.62 exemplarisch für das Komfortsteuergerät dargestellt. Das Steuergerät ist in die Speicherbereiche *ROM, Konfigurations-, Status-, Kommunikations- und Sicherheitsdaten* aufgeteilt. Die Zuordnung der Werte auf die einzelnen Speicherbereiche erfolgt mittels Verbindungen (Dependencies). Diese Verbindungen haben zusätzliche Eigenschaftswerte, die festlegen, ob und in welcher Form die Daten gesichert werden müssen. So ist für redundant abgelegte Daten der doppelte Speicherplatz vorgesehen.

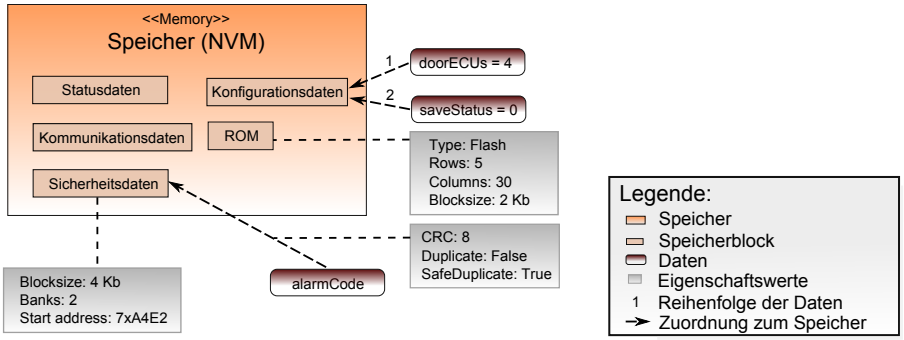


Abb. 4.62. Speicheraufbau des Komfortsteuergerätes

Diese Vorgehensweise wird am Beispiel des *AlarmCodes* in der Abbildung 4.62 beschrieben. Durch die Eigenschaftswerte ist definiert, dass der Alarmcode mit einer Prüfsumme und außerdem redundant auf unterschiedliche Speicher abgelegt werden muss. Durch diese Verbindung wird entsprechender Speicherplatz – inklusive der Prüfsumme – auf unterschiedlichen Speicherbänken bereitgestellt. Ebenso wird festgelegt, an welcher Stelle im Speicher die Daten abgelegt werden. So werden im Konfigurationsdatenspeicherbereich zuerst die Anzahl der Türen (*doorStatus*) und anschließend der Speicherstatus (*saveStatus*) gespeichert. Dies wird durch die Nummerierung an den Verbindungen festgehalten.

Natürlich ist für die Speicherbelegung eine tabellarische Darstellung von Vorteil, da eine Menge von Daten abgebildet werden. Von daher wird auf die tabellarische Notation der UML zurückgegriffen [RQZ07]. Hierdurch kann

eine formale Beschreibung durch die UML erfolgen und gleichzeitig wird eine vorteilhafte Darstellung ermöglicht. Durch die formale Spezifikation sind automatisierte Checks möglich, die überprüfen, ob u. a. Speicherplatz für die Prüfsummen oder für redundante Daten zur Verfügung steht. Auch wird überprüft, ob für sichere Daten auch eine Belegung auf unterschiedlichen Speicherbänken vorgenommen wird. Somit wird sichergestellt, dass bei der Speicherbelegung keine offensichtlichen Fehler bei der Spezifikation vorgenommen werden.

Durch die zuvor vorgestellten Erweiterungen erfolgt eine Verbesserung und Optimierung der Speicherbelegung eines eingebetteten Systems. Eine bestehende Lücke im Entwicklungsprozess wird hiermit geschlossen.

4.4 Zusammenfassung des Architekturmodells

Zu Beginn der Arbeit werden die Anforderungen aus dem Automotive SPICE Referenzprozess herausgearbeitet, die von jeder Entwicklungsmethodik im Steuergeräteentwicklungsprozess erfüllt werden müssen (vgl. Anforderungen in Abbildung 4.63). Ein Teil der Anforderungen wie die Nachverfolgbarkeit, Konsistenz und die Beschreibung des dynamischen Verhaltens werden durch den Einsatz der SysML/UML bereits abgedeckt. Diese sind daher in der Abbildung 4.63 nicht besonders gekennzeichnet. Vielmehr sind nur die Herausforderungen besonders dargestellt, für die AutoMoMe neue Methoden bereitstellt. In der Grafik ist die Zuordnung der Anforderungen zu den Modellerweiterungen durch grüne Pfeile gekennzeichnet. Es fehlt lediglich die Einbettung der Echtzeitsimulation, die im nächsten Kapitel beschrieben wird. Durch die vorgenommenen Erweiterungen ist sichergestellt, dass alle gestellten Anforderungen (vgl. Kapitel 1.1) durch die entwickelte Methodik erfüllt werden.

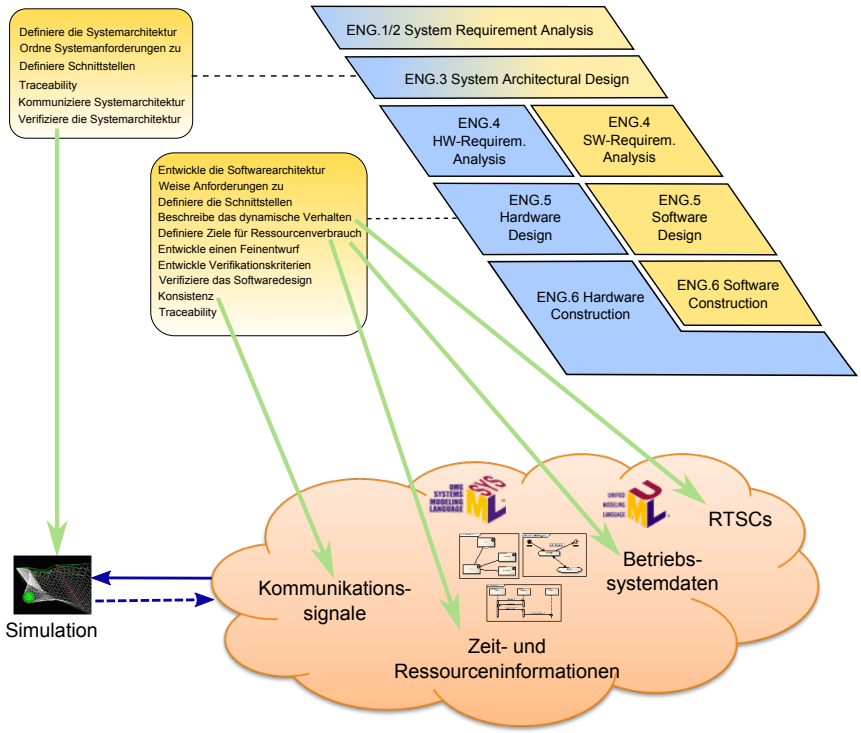


Abb. 4.63. Zuordnung von Automotive SPICE Vorgaben zu den Modellerweiterungen

Zeitverschwendung ist die leichteste aller Verschwendungen.

Henry Ford (1863 - 1947)

5 Echtzeitsimulation

Die neuartigen kooperierenden Funktionen setzen ein korrektes Zeitverhalten voraus, damit das Zusammenspiel der einzelnen Funktionen erfolgen kann. Die Basis für das Zusammenspiel ist die Architektur, die bereits bei der Entwicklung auf diese Qualitätsanforderung hin überprüft werden muss. Von daher sind Tests, beispielsweise durch HIL Simulationen, im Entwicklungsprozess zu spät angesiedelt, um die Architekturerstellung zu unterstützen. Aus diesem Grunde wird die Echtzeitanalyse/-simulation eingesetzt, um bereits in den frühen Entwicklungsphasen einen virtuellen Prototyp [KM10a] für Ausführungszeiten und Aufrufreihenfolge zu erhalten. Die Echtzeitanalyse/-simulation überprüft nicht die funktionale oder physikalische Richtigkeit. Diese Überprüfung wird durch die Simulation der einzelnen Funktionen und Algorithmen im MIL-Verfahren beispielsweise durch Matlab/Simulink [Mat14] durchgeführt. Somit ergänzt die Echtzeitanalyse/-simulation die bereits existierenden Simulationen auf Architekturebene.

Die Echtzeitanalyse/-simulation eröffnet viele Vorteile. Daher ist sie ein Bestandteil der Entwicklungsmethode AutoMoMe (siehe Abbildung 5.1). Die Unterscheidung zwischen einer Echtzeitanalyse und einer Echtzeitsimulation ist bereits im Kapitel 2.5 beschrieben. Da grundsätzlich beide Ansätze in AutoMoMe zur Verfügung stehen, wird auf Grund der häufigeren Verwendung und des sonst zu langen Begriffs nur noch von der Echtzeitsimulation gesprochen. Im Folgenden werden die Vorteile der Echtzeitsimulation und die Integration in die Methodik genauer beschrieben. Mit der Echtzeitsimulation werden folgende Ziele verfolgt:

- zeitliche Fehler in frühen Entwicklungsphasen zu identifizieren und zu beheben
- Hilfestellung bei der Analyse von zeitlichen Fehlern
- Unterstützung bei der Integration von Systemen

- Hilfestellung bei der Ermittlung von Zeitbudgets für die einzelnen Systemkomponenten
- frühzeitige Ermittlung von Lasten auf Rechenkernen und Kommunikationsbussen

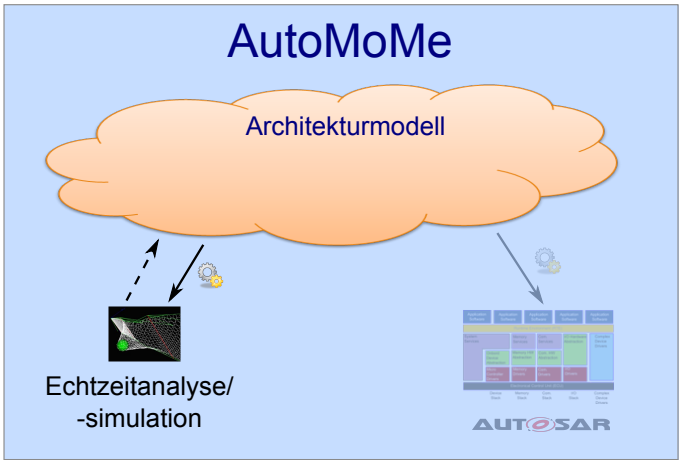


Abb. 5.1. Integration der Echtzeitanalyse/-simulation in AutoMoMe

Die Echtzeitsimulation senkt das Risiko bei der Systementwicklung [AELG11]. Am Anfang der Entwicklung gibt es zugegebenermaßen eine Vorstellung wie die Architektur aussehen kann. Jedoch sind noch viele Einzelheiten unbekannt und es sind noch viele Entscheidungen zu treffen. Demzufolge ist das Risiko, dass die Architektur den Anforderungen und insbesondere den Qualitätsanforderungen nicht genügt, noch sehr groß bzw. wird das Risiko durch die fehlende Unwissenheit ignoriert (Status *Meta-Unwissenheit*) (vgl. Abbildung 5.2). Um das Risiko zu minimieren, muss mehr Wissen über die Gesamtarchitektur gewonnen werden, damit alle Facetten des Systems bekannt sind (Status *Fehlendes Unwissen*). Die Echtzeitsimulation liefert dabei Fakten über das dynamische und insbesondere das zeitliche Verhalten.

Auf Grund der Informationen über das zeitliche und dynamische Verhalten ist die Echtzeitsimulation eine Unterstützung bei wichtigen Architekturentscheidungen, beispielsweise bei der Verteilung von Software auf Rechenkerne [SR08a] oder aber die Integration von zugelieferter Software. Ohne eine Simulation kann der Architekt diese Entscheidungen nur nach Gefühl treffen. Er muss hoffen, dass er die richtige Entscheidung trifft. Das Ergebnis kann aber erst nach der Integration bzw. nach der Erstellung von Prototypen ausgewertet werden (vgl. Abbildung 5.3). Dies erfolgt bei der bisherigen Entwicklung aber erst in einer sehr späten Phase [BRS11]. Daher muss zur Korrektur einer fehlerhaften Entscheidung ein Großteil der Entwicklung nochmals durch-

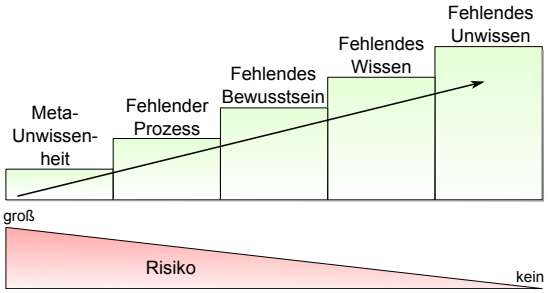


Abb. 5.2. Fünf Stufen des Risikomanagements nach [AM08]

laufen werden. In der Abbildung 5.3 ist dies durch die Vielzahl von V-Modellen dargestellt.

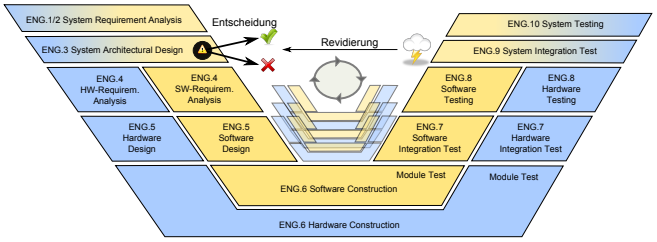


Abb. 5.3. Iterationen durch das V-Modell durch Fehlerfindung im Integrationstest

Die Echtzeitsimulation ermöglicht dagegen eine frühzeitige virtuelle Integration bereits während der Architekturerstellung [KM10a, HG10, PSW10]. Dank dieses Umstands werden die zeitlichen und performancetechnischen Qualitätsanforderungen frühzeitig überprüft. Es wird sichergestellt, dass die Anforderungen von dem zu entwickelnden System mit der Architektur eingehalten werden [KM09, RHER07]. Hierdurch wird die Anzahl der notwendigen Durchläufe durch das V-Modell deutlich verringert. Auf Grund dieser Tatsache werden entsprechende Kosten gespart, da die Entwickler keine Zeit für die Nacharbeit und Fehlerkorrektur aufwenden müssen. Ebenso kann die Anzahl der zu erstellenden Prototypen verringert werden. Wegen der frühzeitigen virtuellen Simulation kommt es bei der anschließenden realen Integration zu deutlich weniger Fehlern, und es werden weniger Prototypen benötigt [SR08b]. Dies spart Ressourcen.

Neben der Qualitätsverbesserung und der Kostenersparnis weist die Echtzeitsimulation darüber hinaus einen weiteren Vorteil auf. Dies ist die Optimierung des Systems [Kra09]. Auf Grund der grafischen Darstellung des dynamischen Verhaltens erhält der Architekt einen sehr genauen Überblick in das System. Dieser ermöglicht kritische Schwachstellen und Engpässe (Bottlenecks) zu iden-

tifizieren. Durch geeignete Veränderungen, beispielsweise durch die Anpassung von Konfigurationsparametern, erfolgt eine Optimierung des Gesamtsystems, so dass die Auslastung bzw. die Ausführungszeiten minimiert werden. Hierdurch sind Wettbewerbsvorteile gegenüber den Mitbewerbern möglich [Har09].

Aus diesen Ausführungen ist zu erkennen, dass viel dafür spricht, eine Echtzeitsimulation einzusetzen. Gleichwohl ist der damit verbundene Aufwand nicht zu unterschätzen. Es muss immer abgewogen werden, ob die Ergebnisse diesen Aufwand rechtfertigen. So ist bei einem größtenteils durch Hardware dominierten System die Abwägung eher negativ zu beurteilen. Damit nicht für jedes Projekt eine aufwendige Abschätzung erfolgen muss, wurden innerhalb des Ansatzes AutoMoMe in Zusammenarbeit mit der Firma Inchron [Inc14] die nachfolgenden Projektmerkmale erarbeitet, für die sich der Einsatz der Echtzeitsimulation auf jeden Fall rechtfertigt.

Die Echtzeitsimulation ist insbesondere dann einzusetzen, wenn in dem zu entwickelnden Steuergerät mehrere Rechenkerne bzw. Steuergeräte enthalten sind [Kra12b]. Hierdurch wird eine interne Kommunikation vorgegeben, die einen wesentlichen Einfluss auf das zeitliche Gesamtverhalten hat. Der Einsatz der Simulation ist auch für Projekte ratsam, die komplexere funktionale Wirkketten abarbeiten. Innerhalb von Wirkketten kann es zu Dateninkonsistenzen kommen, wenn Daten nicht rechtzeitig bereitgestellt werden [KM10b]. Dies kann durch eine rechtzeitige Simulation überprüft werden. Darüber hinaus kann die Simuation genutzt werden, um eine Hilfestellung bei der Auswahl der Hardwarekomponenten zu geben. So können durch die Simulation die erwarteten Lasten auf den Rechenkernen bzw. den Bussen abgeschätzt werden. Dies kann bei der Ermittlung und Festlegung der benötigten Hardware hilfreich sein. Anhand der Ergebnisse können die effektivsten und gleichzeitig kostengünstigsten Hardwarekomponenten ausgewählt werden, die den Anforderungen der Software aber immer noch entsprechen.

Zusammenfassend ergeben sich folgende Merkmale und Faktoren, die für eine Echtzeitsimulation sprechen:

- **Hoch riskantes Zeitverhalten:** Ein hochriskantes Zeitverhalten kann durch mehrere Faktoren in einem Projekt gegeben sein. So kann eine Integration von zugelieferten Komponenten (Zulieferer oder vom OEM) bereits ein zeitliches Risiko für das Projekt verursachen [Kra12a]. Ebenso sind sehr enge zeitliche Anforderungen in diesem Zusammenhang zu nennen, wenn es um den Eingriff in sicherheitsrelevante Funktionen geht.
- **Mehrere CPUs oder ECUs (mit unterschiedlichen Uhren):** Wenn unterschiedliche Rechenkerne in einem System zum Einsatz kommen, spielt die Verteilung der Softwarekomponenten auf die Rechenkerne und die Kommunikation zwischen ihnen für das zeitliche Verhalten eine entscheidende Rolle.

- **Lange Wirkketten:** Sofern viele einzelne Komponenten benötigt werden, um ein Verhalten zu realisieren, kann es selbst bei kleinsten Abweichungen zu großen Auswirkungen im Gesamtverhalten kommen. Von daher muss sichergestellt werden, dass die komplette Wirkkette aufeinander abgestimmt ist und optimal funktioniert.
- **Komplexes Scheduling:** Falls mehrere Interrupts und Tasks in einem Projekt zu finden sind, muss überprüft werden, ob sie sich nicht gegenseitig stören bzw. verdrängen. Daher ist bei einem hohen Auftreten besonders von Interrupts das Scheduling zu überprüfen.
- **Entscheidung für neue Hardware (Re-Design):** Für eine neue Generation eines Steuergerätes wird vielfach eine neue Systemarchitektur entwickelt. Oftmals wird hierfür eine verbesserte Hardwarearchitektur ausgewählt, da für gewöhnlich die Größe der Software zugenommen hat. Ob die ausgewählte Hardware für die Software ausreichend bemessen ist, muss bereits bei der Angebotserstellung bedacht werden, da ansonsten die Preiskalkulation nicht auskömmlich ist. Daher ist eine Kontrolle durch eine Echtzeitsimulation obligatorisch.

Das Komfortsteuergerät besteht aus mehreren Steuergeräten und somit unterschiedlichen Rechenkernen. Ebenfalls sind Wirkketten, die über mehrere Komponenten verteilt sind, vorhanden, so dass bereits zwei Gründe für die Durchführung einer Echtzeitsimulation erfüllt sind. Daher wird für das Komfortsteuergerät eine Echtzeitsimulation durchgeführt, in dem automatisiert aus dem Architekturmodell ein Simulationsmodell erstellt wird.

5.1 Stand der Technik Echtzeitsimulation in der Steuergeräteentwicklung

Das Thema Echtzeitanalyse/-simulation hat in den letzten Jahren deutlich an Bedeutung gewonnen. Dies ist hauptsächlich auf die wachsende Komplexität der Funktionalität und der Verwendung von Mehrprozessorsystemen zurückzuführen. Die Vielzahl der Arbeiten in diesem Bereich bezieht sich aber auf die Durchführung und die Ergebnisse der Echtzeitsimulation [Kra12b, KM10b, KM10a, PPE+08, RER06, RHER07]. Weiterhin existieren Arbeiten in diesem Bereich, die eine automatische Optimierung der Systeme zum Ziel haben [KHTW08, MBAG11, SR08a, KA98]. Ein weiterer Bereich von Arbeiten befasst sich mit der Transformation von Daten aus SysML Modellen in Simulationsmodelle [HRM07]. Hierbei geht es hauptsächlich darum die physikalischen Werte in Simulationen wie z. B. Dymola zu bringen [JPB08, BPL01]

All diese Ansätze beschäftigen sich aber weniger mit der Integration der Echtzeitsimulation in den Entwicklungsprozess und wie die Daten in die

Echtzeitsimulation gelangen. In der Arbeit [HH07] wird dieses Thema aufgegriffen. Jedoch wird in diesem Ansatz der Schluss gezogen, dass die Architekturmodelle mit den Informationen für die Echtzeitsimulation überfrachtet werden [HH07]. Aus den bisherigen Erfahrungen innerhalb von Pilotprojekten kann ich nur den gegensätzlichen Schluss ziehen, da die Entwickler möglichst viele Informationen an einer Stelle sehen und editieren möchten. Hierdurch wird auch die Konsistenz zwischen den einzelnen Artefakten sichergestellt.

Diese Schlussfolgerung wird ebenfalls in [HG10] gesehen. In diesem Ansatz wird ein UML Modell mithilfe des Marte Profils um Zeitinformationen ergänzt und die Informationen werden anschließend an ein Echtzeitsimulationswerkzeug (in diesem Fall SymTa/S [Sym14]) weitergeleitet. Dieser Ansatz hat ein vergleichbares Vorgehen, wie die in AutoMoMe entwickelte Anbindung der Echtzeitsimulation. Jedoch werden in der AutoMoMe Integration weitergehende Informationen übertragen, die für ein automobiles Steuergerät besonders von Bedeutung sind, wie Interrupts, Operationsreihenfolgen und Datenkonsistenz.

Im Folgenden wird der AutoMoMe Ansatz dargestellt, wie die Informationen aus dem Architekturmodell für die Echtzeitsimulation des automobilen Steuergerätes genutzt werden.

5.2 Eingabedaten der Simulation

Die Eingabedaten zur Echtzeitsimulation bzw. -analyse werden von der Architektur bereitgestellt. Dabei gilt es jedoch zu berücksichtigen, dass in den frühen Entwicklungsphasen die exakten Zeiten der Systemfunktionen noch nicht bekannt sind. Es muss daher versucht werden, möglichst genaue Abschätzungen zu erhalten, damit die Simulation auch aussagekräftige Daten liefern kann. Bei Neuentwicklungen ist grundsätzlich anzumerken, dass kaum eine Entwicklung auf grüner Wiese startet [HH04], sondern es fast immer Vorgängerprojekte oder aber Prototypen gibt. Die hierbei gewonnenen Messergebnisse können als Vorgaben für die jeweiligen Komponenten genutzt werden. Ebenso kann bestehender Quellcode analysiert und die Ausführungszeit bestimmt werden [Inc14, Abs14]. Somit steht ein Großteil der notwendigen zeitlichen Informationen zur Verfügung. Für die restlichen neu zu entwickelnden Komponenten werden zunächst Schätzungen vorgenommen. Ausgehend von diesen Schätzungen und den Ergebnissen der Simulation werden zeitliche Eckdaten als Anforderungen für die spätere Entwicklung genutzt. Sofern diese Zeiten für die Funktionen eingehalten werden, ist sichergestellt, dass das System zeitlich korrekt arbeitet. So kann ausgehend von der Echtzeitsimulation eine Budgetierung der einzelnen Komponenten innerhalb der Architektur vorgenommen werden.

Sowohl die System- als auch die Softwarearchitektur stellt Eingabedaten zur Verfügung (vgl. Abbildung 5.4). Diese Daten stehen in unterschiedlicher detail-

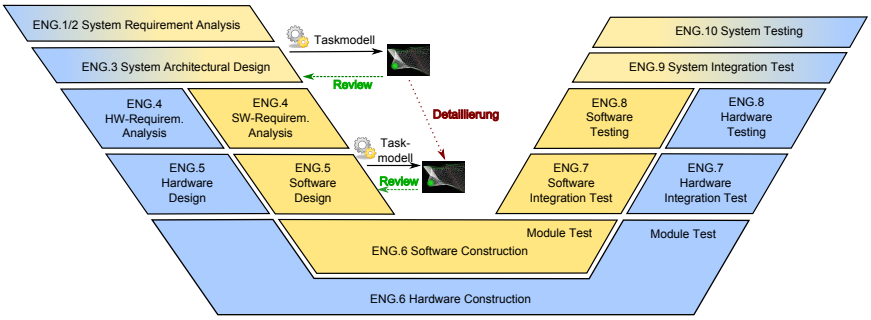


Abb. 5.4. Einbettung der Echtzeitsimulation im Entwicklungsprozess

lierter Form zur Verfügung. Zu Beginn einer Neuentwicklung liegen teilweise keine oder nur ungenaue Informationen vor. Falls keine Informationen vorliegen, ist es ratsam, Schätzungen vorzugeben. Diese Annahmen bilden die Grundlage für die Durchführung einer Simulation. Aus der Echtzeitsimulation werden Anforderungen in Form von Vorgaben für Entwicklungszeiten für spätere Entwicklungsphasen gewonnen, so dass ein Gefühl für die notwendige Leistung der zu realisierenden Komponenten entsteht. Wenn die realisierte Komponente den Vorgaben genügt, so ist das gesamte Systemverhalten korrekt [BRS11]. Die geschätzten Werte werden in der Simulation als Grundlage verwendet. Es wird überprüft, ob mit den vorhandenen oder geschätzten Werten die Architektur die Anforderungen erfüllt. Ist dies der Fall, so können Zeitbudgets in Form von Zeitintervallen dem Softwareentwickler vorgegeben werden. Die Simulation ist dann jedoch sukzessive mit gewonnenen Echtzeitdaten zu wiederholen, um eine möglichst genaue Simulation durchzuführen und die Ergebnisse zu verifizieren.

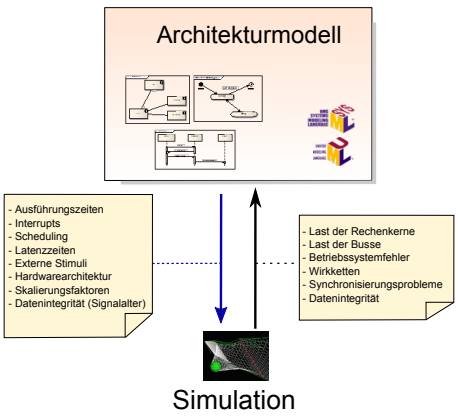


Abb. 5.5. Ein- und Ausgaben der Echtzeitsimulation

In der Abbildung 5.5 sind die Ein- und Ausgabedaten für die Echtzeitsimulation dargestellt. Dabei ist zu beachten, dass die Eingabedaten durch eine Modelltransformation automatisiert in das Echtzeitsimulationsmodell übertragen werden (siehe Kapitel 5.3). Hierdurch wird der iterativen Erstellung des Architekturmodells und dem mehrmaligen Ausführen der Echtzeitsimulation Rechnung getragen. Falls eine Änderung am Architekturmodell erfolgt, kann diese durch Ausführen der Transformation in die Echtzeitsimulation übernommen werden. Falls während der Simulation Änderungen am Simulationsmodell vorgenommen werden, um auf Simulationsergebnisse zu reagieren, werden diese Daten derzeit nicht automatisiert ins Architekturmodell übertragen. Sie müssen manuell in das Architekturmodell eingepflegt werden.

Nachfolgend werden die Informationen beschrieben, die für die Echtzeitsimulation zwingend erforderlich sind (vgl. Abbildung 5.5). Dies sind zum einen die einzelnen Softwarekomponenten und die Interrupts einschließlich der Antwortzeiten, die in dem System vorhanden sind. Zum anderen müssen dem Scheduling die Latenzzeiten und die externen Stimulis zur Verfügung gestellt werden. Diese ergeben sich zum Teil aus der Modellierung der Betriebssystemeigenschaften (vgl. Kapitel 4.3.7). Darüber hinaus werden für die Simulation noch Informationen von der Hardwarearchitektur benötigt. Diese werden innerhalb der Systemarchitektur spezifiziert. Die Informationen umfassen den generellen Aufbau des Steuergerätes einschließlich der Prozessoren, die Anbindung an die Busse, sowie die Allokation der Softwarekomponenten auf die einzelnen Rechenkerne (vgl. Kapitel 4.3.7). Zur Überprüfung der Zeiten der Wirkketten sind diese ebenfalls zu spezifizieren. Die Modellierung inklusive der Spezifikation der Zeiten erfolgt dabei innerhalb der Architekturmodellierung (vgl. Kapitel 4.19).

Aber nicht alle Eingabedaten sind immer zwingend für eine Echtzeitsimulation erforderlich. So werden für die Berechnung der Kommunikationslast auf den Bussen oder der Rechenlast auf den Kernen nicht die Verbindungen von einzelnen Funktionen untereinander benötigt. Von daher ist eine Abhängigkeitsmatrix erstellt worden, aus der abzulesen ist, für welche Ziele, welche Eingaben zwingend erforderlich sind. Das Ergebnis kann in der Abbildung 5.6 nachgelesen werden. Es ist ersichtlich, dass manche Daten, wie die Liste aller Komponenten bzw. deren Ausführungszeiten, immer benötigt werden. Wohingegen die Datenintegrität nur bei wenigen Zielen Voraussetzung ist.

5.3 Vom Architekturmodell zur Echtzeitsimulation

Auf der Grundlage dieser Informationen kann eine Echtzeitsimulation durchgeführt werden. Jedoch müssen die Informationen zunächst in einem Simulationsmodell bereitgestellt werden. In der Arbeit [HH07] wird ein eigenes Simulationsmodell erstellt, in dem die Informationen spezifiziert werden. Nachteilig

Notwendige Informationen	CPU Last	Bus Last	Datenverlust	RTOS Fehler	Wirkketten	WCRT	Synchronisierung	Datenintegrität
Ausführungszeiten der Komponenten	x	x nur für Senderverhalten	x	x	x	x	x	x
Liste aller Interrupts	x	x nur für Senderverhalten	x	x	x	x	x	x
Liste aller Module	x	x nur für Senderverhalten	x	x	x	x	x	x
Scheduling (Priorität, Verdrängungsstrategie,...)	x	x für die Kommunikationsframes	x	x	x	x	x	x
Kontextwechsel	(x)	(x) für den Sender	(x)	x	(x)	(x)	(x)	(x)
Eingangs- und Ausgangslatenzen	(x)	(x) für den Sender	(x)	x	(x)	(x)	(x)	(x)
Basis-Software	(x)	(x) für den Sender	(x)	(x)	(x)	(x)	(x)	(x)
Kommunikationslatenz	-	x	-	-	-	-	(x)	x
Externe Stimuli	(x)	(x) für den Sender	(x)	(x)	(x)	(x)	(x)	(x)
Skalierungsfaktoren (Safety, AUTOSAR, ...)	(x)	(x) für den Sender	(x)	(x)	(x)	(x)	(x)	(x)
Aktivierungskonditionen	x	x für den Sender	x	x	x	x	x	x
Hardwarearchitektur	x	x	x	x	x	x	x	x
Alter von Signalen	-	-	-	-	x		-	x
Funktionale Abhängigkeiten	-	-	-	-	x	-	x	-
Ereignisverarbeitung (API calls: setEvent/waitEvent/...)	(x)	(x)	(x)	(x)	(x)	(x)	x	(x)

Legende:
x zwingend erforderlich
(x) optional
- nicht erforderlich

Abb. 5.6. Verbindung zwischen Ergebnissen und benötigten Eingabedaten

bei diesem Vorgehen ist jedoch, dass möglicherweise Inkonsistenzen zwischen dem Architektur- und dem Simulationsmodell entstehen. Dies kann bei häufigen Änderungen bzw. bei schnellen Entwicklungszyklen der Fall sein wie sie standardmäßig in der Steuergeräteentwicklung gegeben sind [HH04]. Außerdem entsteht ein größerer Entwicklungsaufwand, da das Simulationsmodell zusätzlich zu erstellen ist.

Um dem entgegenzuwirken, wird in AutoMoMe ein anderer Weg verfolgt. Da ein Großteil der Informationen bereits im Architekturmodell auf eine (semi-)formale Art vorliegt, ist es nur folgerichtig, diese Informationen konsequent weiter zu verwenden und so zu transformieren, dass daraus ein Simulationsmodell entsteht. Änderungen in der Architektur sind dann schnell und ohne großen Aufwand zu überprüfen. Ein vergleichbares Vorgehen ist bereits in anderen Arbeiten zu sehen [BPL01, HRM07, JPB08, HG10]. Diese Vorgehensweise ermöglicht, Auswirkungen von Architekturentscheidungen zu analysieren und mit Werten zu versehen. Diese werden anschließend genutzt, um die am Besten geeignete Alternative auszuwählen.

Es sind bereits mehrere kommerzielle Simulationswerkzeuge am Markt verfügbar [Sym14, Inc14]. Infolgedessen muss kein eigenes Werkzeug entworfen und implementiert werden, sondern es kann auf vorhandene und etablierte Werkzeuge zurückgegriffen werden. Für die Integration der Echtzeitsimulation in die Entwicklungsmethodik AutoMoMe ist die Wahl auf die Toolsuite der Firma Inchron gefallen [Inc14]. Ausschlaggebend war, dass neben einer Echtzeitsimulation mit demselben Simulationsmodell ebenfalls eine Echtzeitanalyse vorgenommen werden kann (siehe Abbildung 5.7) [KM10b]. Hierdurch kann die Aussagekraft des Endproduktes noch einmal gesteigert werden. Das Werkzeug für die Echtzeitsimulation lautet *chronSIM* während die formale Analyse mit dem Werkzeug *chronVAL* erfolgt. Zusätzlich zu den bereits vorhandenen Importen (Anforderungen und AUTOSAR) wurde in AutoMoMe noch eine Anbindung an die SysML/UML mit dem Werkzeug IBM Rational Rhapsody [IBM14b] erstellt [NMK10].

Beide Werkzeuge der Toolsuite verwenden als Simulationsmodell ein sogenanntes Taskmodell, das auf textueller Basis in Form einer C-ähnlichen Sprache definiert ist. Der Vorteil hierbei ist, dass bereits bestehender Code mit in das Simulationsmodell integriert werden kann, um so eine möglichst exakte Simulation zu ermöglichen. Dies ist ein Baustein der in Abbildung 5.4 dargestellten Detaillierung zwischen der System- und der Softwarearchitektur. Für die Erstellung des Simulationsmodells ist eine dementsprechende Modell-zu-Text Transformation (M2T) erforderlich (siehe Transformationsschritte 1-6 in Abbildung 5.8). Hierfür bietet der Ansatz AutoMoMe entsprechende Vorlagen (Templates) (siehe Kapitel A.1.3).

Für den Übergang zwischen dem Architektur- und dem Simulationsmodell sind im Rahmen von AutoMoMe Modelltransformationen erstellt worden (siehe Kapitel A.1.3). Dafür ist es zunächst notwendig, die Zusammenhänge zwis-

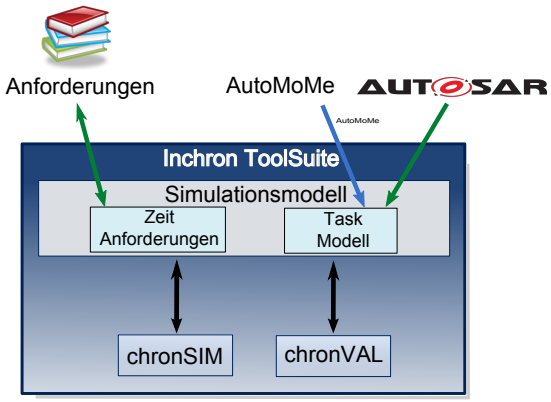


Abb. 5.7. Inchron Toolsuite mit den in AutoMoMe erweiterten Eingabemöglichkeiten

chen dem Architekturmodell und dem Simulationsmodell herzustellen. In der Abbildung 5.8 ist veranschaulicht, welche Informationen aus dem Architekturmodell (Beispiele aus dem Kapitel 4.3) in welche Bereiche des Simulationsmodells fließen. Das Simulationsmodell ist in drei verschiedene Bereiche aufgeteilt. Dies sind ganz links die Zeittabelle, in der Mitte der aus der Struktur und dem Verhalten generierte Quellcode und im rechten Bereich die Simulationsprojektdatei (siehe Abbildung 5.8).

In diese drei Bereiche werden die Daten aus dem Architekturmodell transformiert. Ein erster Transformationsschritt transformiert die Ausführungszeiten der Komponenten und Operationen in einer separaten Datei, der sogenannten *Zeittabelle* (siehe Transformationsschritt eins in Abbildung 5.8). Hierbei werden die drei Eigenschaftswerte (siehe 4.37) für die minimale, durchschnittliche und maximale Ausführungszeit ausgelesen und deren Wert in die Tabelle eingetragen. Der Zweck dieser Datei ist, dass innerhalb der Simulation dynamisch zwischen den Zeiten gewechselt werden kann. Dies geschieht durch eine einfache Variable, die während der Echtzeitsimulation gesetzt wird. So ist ein Wechsel zwischen der minimalen und der maximalen Ausführungszeit ohne Änderungen am generierten Simulations-Code möglich.

Ein zweiter Transformationsschritt generiert aus den Komponenten und ihren Operationen einen entsprechenden Simulations-Code (siehe Transformationsschritt zwei in Abbildung 5.8). Dieser beinhaltet die Abhängigkeiten der Komponenten in Form von includes und ebenso die Stellen an denen die Ausführungszeiten benötigt werden. Dies ist in der Abbildung durch das C-Makro DELAY zu erkennen. Dieses greift wie bereits erwähnt per Variable auf die Ausführungszeiten aus der *Zeittabelle* zu. Ergänzt wird der Simulations-Code durch die Definition der Wirkketten. Ausgangspunkt sind die im Architekturmodell spezifizierten Wirkketten (siehe Kapitel 4.3.3). Aus ihnen werden

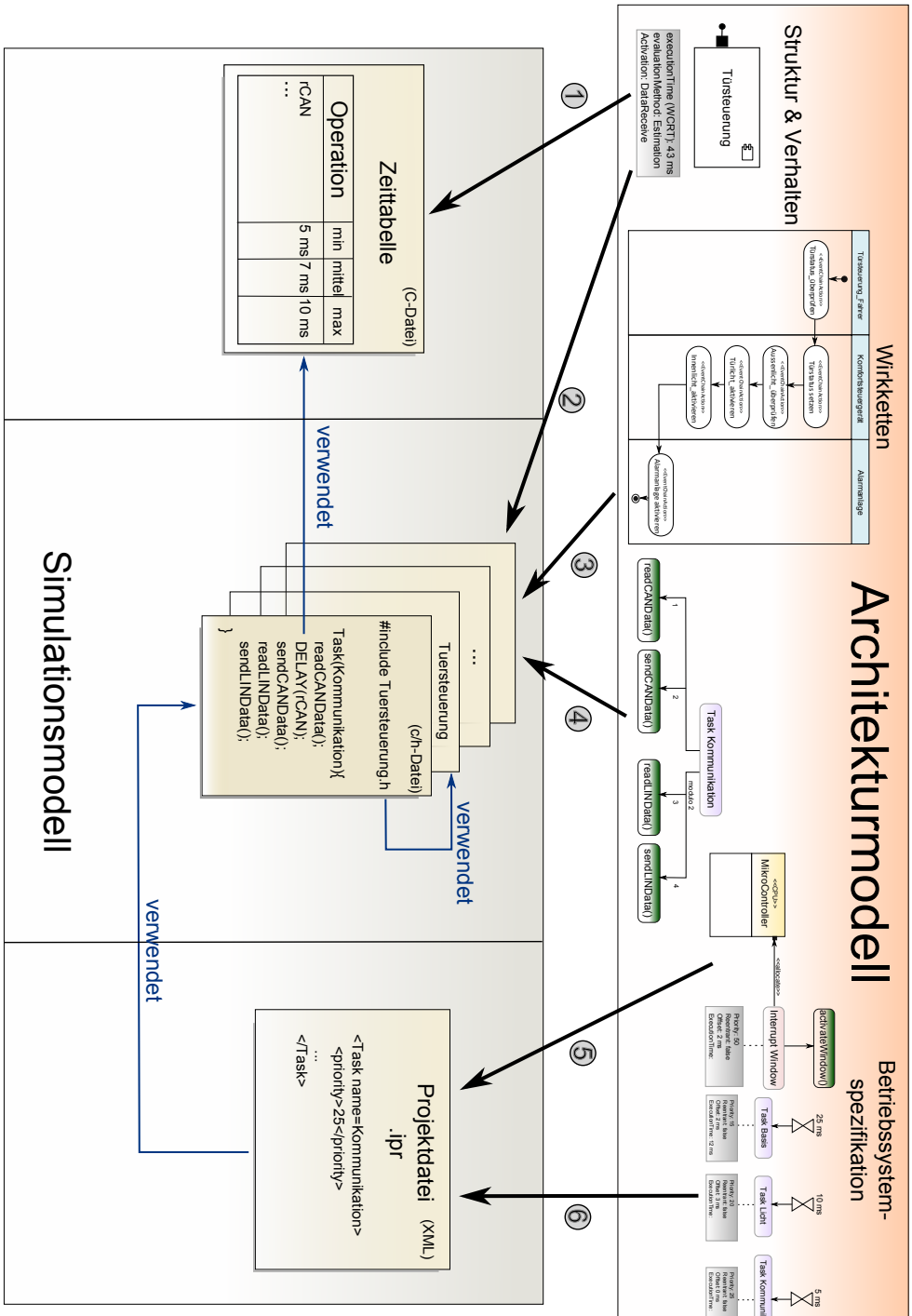


Abb. 5.8. Verteilung der Informationen aus dem Architektur- in das Simulationsmodell

entsprechende C-Makros in den Simulationscode generiert (siehe Transformationsschritt drei in Abbildung 5.8). Hierdurch werden sie bei der Echtzeitsimulation erkannt und ausgewertet. Im Transformationsschritt vier werden die Task und die Zuordnung von Operationen zu Tasks ausgewertet. Hierdurch werden die Tasks mit den Operationsaufrufen in den Simulations-Code generiert und dieser somit vervollständigt. Ebenso wird die Reihenfolge der Funktionsaufrufe innerhalb der Task abgebildet (siehe Transformationsschritt vier in Abbildung 5.8).

Der dritte Bereich des generierten Simulationsmodells ist die XML-basierte Projektdatei. In ihr sind die Einstellungen der Betriebssystemspezifikation und die für die Simulation notwendige Architektur, beispielsweise Peripheriegeräte oder Kommunikationsbusse, festgelegt (siehe Transformationsschritt fünf in Abbildung 5.8). Es werden dort die Rechenkerne, Tasks und Interrupts definiert und mit den entsprechenden Quelldateien verbunden. Hierdurch wird die Verknüpfung mit den einzeln generierten C-Dateien und den darin befindlichen Ausführungszeiten hergestellt (erster und zweiter Bereich der Simulation). Ebenso befinden sich in der Projektdatei die Einstellungen für die Aktivierungen der Tasks und Interrupts. Diese werden aus den Spezifikationen der Betriebssystemeigenschaften aus dem Architekturmodell erstellt (siehe Transformationsschritt sechs in Abbildung 5.8). Insbesondere für die Interrupts muss die Stimulation vorgegeben werden, da das Auftreten sehr variabel erfolgen kann. Hierzu werden die Werte der Stimuli Verbindungen aus Kapitel 4.3.7 genutzt, um eine automatisierte Erstellung für das Simulationsmodell zu gewährleisten.

5.4 Ergebnisse der Simulation

Die Ergebnisse der Simulation müssen für die Überprüfung der Architekturentscheidungen analysiert und ausgewertet werden. Dazu werden die Ergebnisse der Echtzeitsimulation in verschiedenen Diagrammen betrachtet, um unterschiedliche Fragestellungen zu überprüfen. Durch die Analyse bekommt der Architekt zu einem bestimmten Zeitpunkt einen genauen Überblick über das Verhalten des Systems. Auf Grund des nun vorhandenen Wissens sind ggf. Alternativen zu überdenken und notfalls erforderliche Anpassungen vorzunehmen. In Abbildung 5.5 sind die verschiedenen Ergebnisse der Simulation als Rückfluss an das Architekturmodell abgebildet. Im Folgenden werden die einzelnen Diagramme der Echtzeitsimulation und die daraus resultierenden Ergebnisse vorgestellt.

In der Abbildung 5.9 ist das Ergebnis der Simulation der Lastverteilung auf den Rechenkernen abgebildet. Dabei sind die Tasks und Interrupts farblich unterschiedlich dargestellt, so dass erkennbar ist, welcher Task oder Interrupt die größte Rechenlast verursacht. Auf dem *MikroController* (MicroCon-

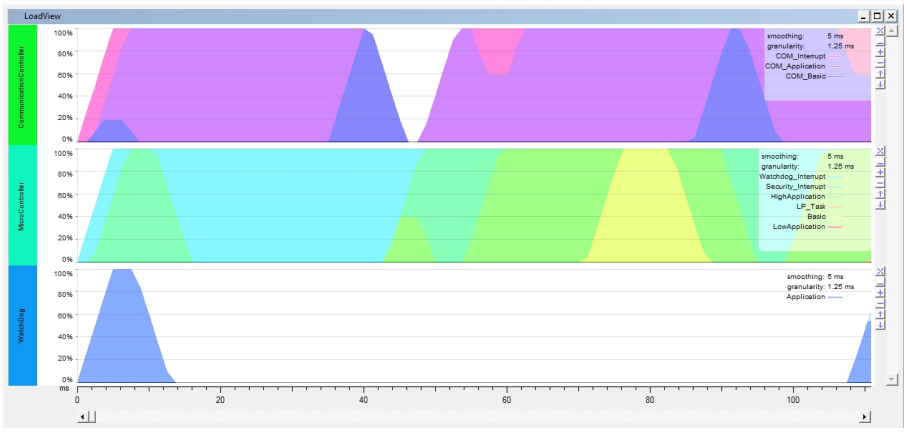


Abb. 5.9. Lastverteilung auf den Rechenkernen

troller) und dem *KommunikationsController* (CommunicationController) ist erkennbar, dass beide Kerne zu über 100 % ausgelastet sind und infolgedessen nicht jede Software ausreichend Rechenzeit zugeteilt erhält. Besonders beim *MikroController* ist eine dauerhafte Auslastung auszumachen. Somit erfüllt die Architektur nicht die Qualitätsanforderungen.

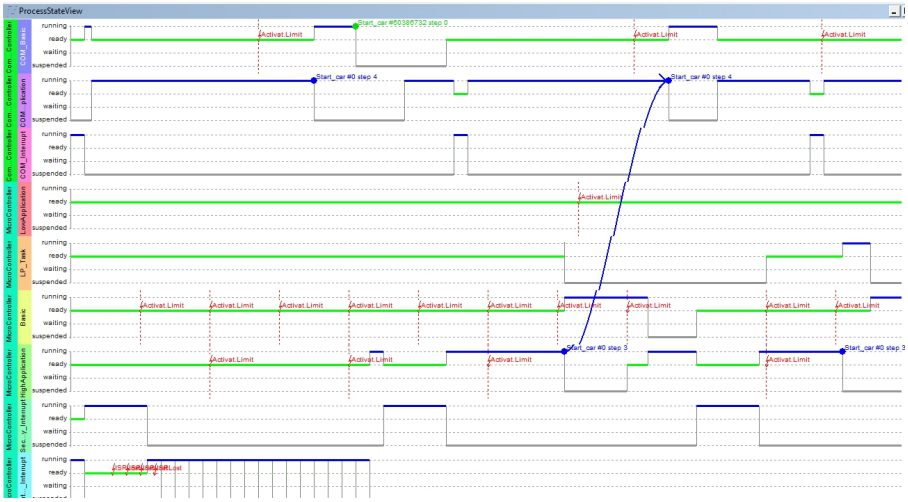


Abb. 5.10. Tasks und ihre Eigenschaften in der Echtzeitsimulation

Wie sich bereits im Lastdiagramm angedeutet hat, ist ebenfalls im Diagramm für die Zustände der Tasks (siehe Abbildung 5.10) ersichtlich, dass die Leistung der Rechenkerne nicht ausreichend dimensioniert ist. Im Simulationsergebnis

sind sehr viele Verletzungen von Task- oder Interruptaktivierungen auszumachen. Diese sind erkennbar an der rot gestrichelten Linie und dem Begriff *Activation Limit*. Besonders bei Tasks, die dem *MikroController* zu geordnet sind, sind solche Verletzungen zu sehen. Exemplarisch ist die *Basis* (Basic) Task hervorzuheben. Sie überschreitet bereits innerhalb des kleinen simulierten Zeitabschnitts die Aktivierung über 10 mal.

Ferner ist aus der Abbildung zu erkennen, dass sich sehr viele Tasks im Zustand wartend (ready) befinden und nicht auf dem Rechenkern ausgeführt werden. Besonders trifft dies bei vielen Tasks beim Start zu. Dies deutet daraufhin, dass der Offset der Tasks nicht optimal gewählt ist. Die Tasks starten alle gleichzeitig und befinden sich sofort im Zustand *wartend*, da immer nur eine Task auf einem Rechenkern laufen kann. Dieser Befund ist zum Anlass zu nehmen, eine Optimierung durch zeitliche Staffelung und Aktivierung der Tasks vorzunehmen.

In Abbildung 5.11 ist die Datenintegrität einer Wirkkette wiedergegeben (definiert in Kapitel 4.3.3). Es ist ersichtlich, dass einzelne Funktionen der Wirkkette aufgerufen werden. Die Daten werden aber nicht anhand der Wirkkettenspezifikation weitergegeben. Nur in der Zeitspanne zwischen 70 und 90 ms ist eine Datenweitergabe festzustellen. Die Wirkkette wird demzufolge an mehreren Stellen unterbrochen und eine Datenintegrität ist so nicht gegeben.

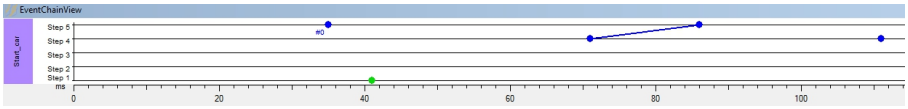


Abb. 5.11. Simulation der Wirkkette im Komfortsteuergerät

Als Resümee der Echtzeitsimulation kann festgehalten werden, dass die Architekturentscheidungen nicht zu dem gewünschten Ergebnis geführt haben und weitere Iterationen für die Architekturerstellung notwendig sind.

Es ist zu konstatieren, dass auf dem Rechenkern *MikroController* eine Last vorzufinden ist, für die der Rechenkern nicht ausgelegt ist. Auch bei einer Lastverschiebung wird keine nachhaltige Verbesserung eintreten, so dass eine Entscheidung für den Einbau eines Multicore Rechenkerns mit zwei Kernen zu treffen ist. Dies ist immer die letzte Lösung, da ein anderer Prozessor meist höhere Kosten verursacht. Eine solche Entscheidung muss deshalb bereits frühzeitig in der Akquisephase getroffen werden, da ansonsten die eingeplanten Kosten für das Komfortsteuergerät nicht einzuhalten sind. Damit einhergehend ist die Lastverteilung auf dem *KommunikationsController* (CommunicationController) zu ändern. Nach Anpassung der Betriebssystemspezifikation im Architekturmodell und anschließender Transformation in das Simulationsmodell ist eine weitere Echtzeitsimulation durchzuführen. Eine solche Iteration

muss so oft wiederholt werden, bis die Architektur den gestellten Anforderungen entspricht.

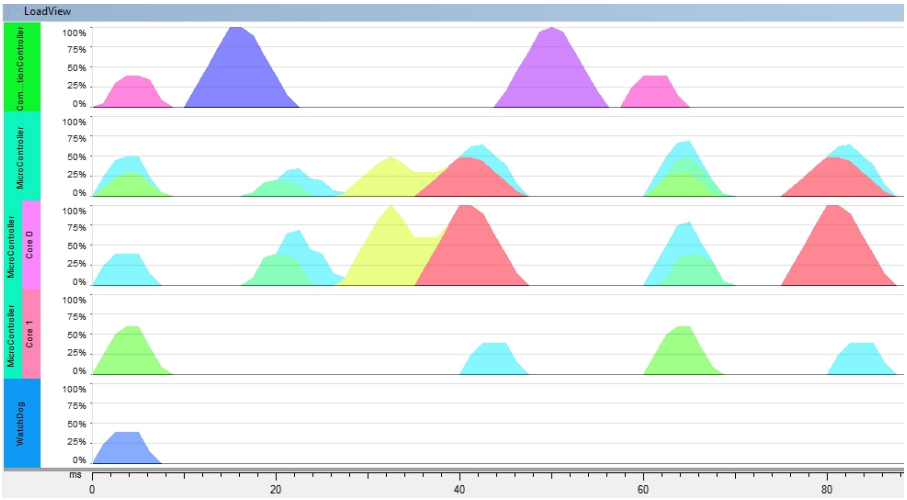


Abb. 5.12. Lastverteilung nach der Einarbeitung der Ergebnisse aus der ersten Simulation

Die Abbildung 5.12 gibt das Ergebnis der Echtzeitsimulation nach der Anpassung der Rechenlast wieder. Es ist zu erkennen, dass auf Grund der Verwendung eines Mehrprozessorsystems zwei weitere Ebenen in dem Lastdiagramm auftauchen. Dies ist zum einen die Wiedergabe des neuen Rechenkerns und zum anderen eine kombinierte Darstellung der beiden Kerne (Cores) des Mehrprozessorsystems. Als Ergebnis ist dabei festzustellen, dass durch die vorgenommenen Maßnahmen die 100 % Auslastung nicht mehr erreicht wird. Insofern war die Entscheidung, einen Multicore Rechenkern mit 2 Kernen einzusetzen, richtig.

Neben der Lastverteilung haben sich verständlicherweise die Taskaufrufe und ihre Zustände geändert. In Abbildung 5.13 ist das Resultat abgebildet. Es sind keine Aktivierungsverluste von Tasks oder Interrupts mehr auszumachen. Dies ist aus den nicht mehr vorhandenen roten Kennzeichnungen im Diagramm ersichtlich. Ferner ist festzustellen, dass die Aktivierungen der Tasks optimal aufeinander abgestimmt sind. So gibt es nur in der Task *LP_Task* eine kurze Situation, in der die Task *wartend* (ready) ist und auf die Zuteilung des Rechenkerns wartet. Ansonsten können die Tasks immer sofort starten. Durch die vorgenommene Echtzeitsimulation und Erarbeitung einer geänderten Betriebssystemspezifikation im Architekturmodell bzw. Veränderung der Rechenkerne wird eine deutlich verbesserte Aufrufreihenfolge und somit auch eine geringere Lastverteilung hergestellt.

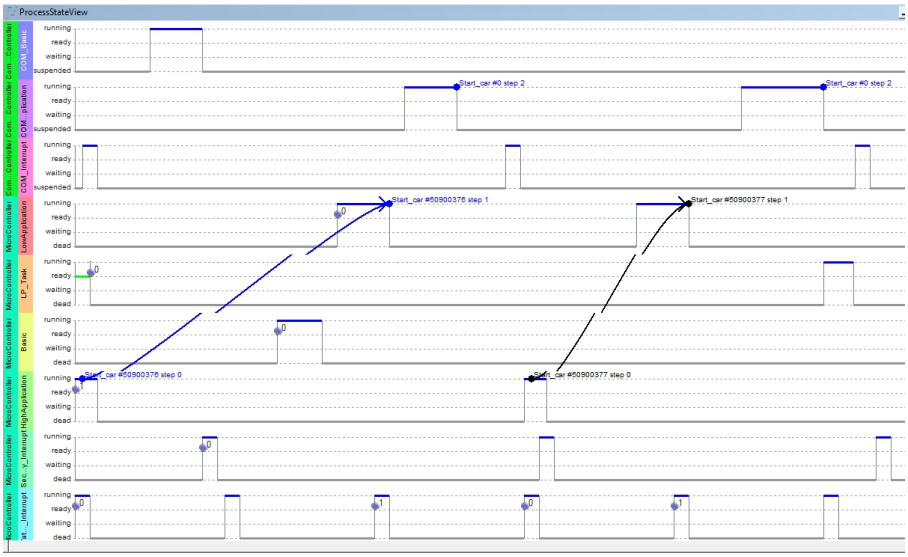


Abb. 5.13. Korrigierte Task Eigenschaften

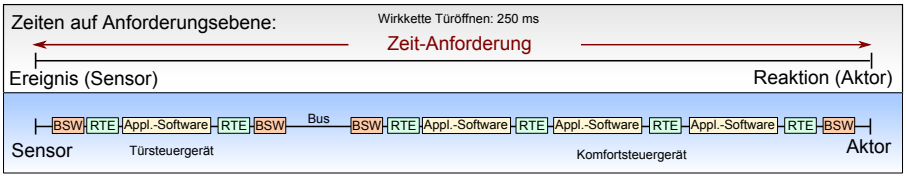


Abb. 5.14. Zeitliche Anforderungen und ihre technische Umsetzung

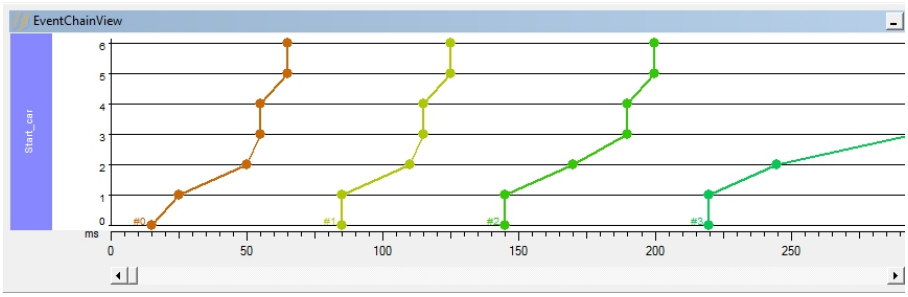


Abb. 5.15. Korrigierte Wirkkette nach den Änderungen am Architekturmodell

Ergänzend zu der Lastreduzierung und der Optimierung der Aktivierung der Tasks ist die Realisierung der Wirkketten zu betrachten [KM09, KM10b]. Bei der ersten Simulation konnte festgestellt werden, dass die Wirkkette abbricht und die Datenintegrität nicht mehr gewährleistet ist. Durch die Behebung der Aktivierungsfehler hat sich die Datenintegrität der Wirkkette ebenfalls verbessert. In der Abbildung 5.15 ist die korrigierte Wirkkette dargestellt. Die Überprüfung von Wirkketten ist bei einer AUTOSAR Architektur von zentraler Bedeutung, da mehrere Komponenten an der Realisierung der Wirkkette beteiligt sind, die zusätzlich noch durch eine Kommunikation mittels Kommunikationsbusse ergänzt wird (siehe Abbildung 5.14) [ROH+07]. Die Spezifikation der Wirkkette ist in Abbildung 4.19 dargestellt. Hierbei ist vorgegeben, dass die Wirkkette höchstens eine Zeitspanne von 250 ms (Ende-zu-Ende Zeit) in Anspruch nehmen darf (vgl. Abbildung 5.16). Diese formal spezifizierte Anforderung wird automatisch als zeitliche Anforderung in das Architekturmodell übernommen und bei der Echtzeitsimulation überprüft. Dies geschieht in Form von zeitlichen Anforderungen. In Abbildung 5.16 ist die Spezifikation für die zeitliche Überprüfung der Wirkkette im Simulationswerkzeug dargestellt. Neben der Ausführungszeit von 250 ms kann ein zusätzliches Intervall angegeben werden, in dem bereits eine Warnung ausgegeben wird, obwohl die zeitliche Anforderung noch eingehalten werden kann. In diesem Fall beträgt der Warnwert 20 % für die spezifizierte Wirkkette des Komfortsteuergerätes.

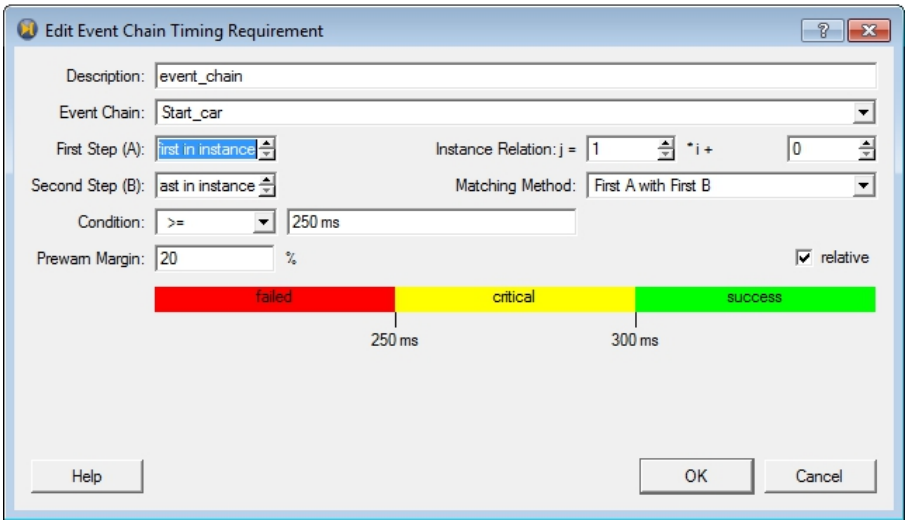


Abb. 5.16. Zeitliche Überprüfung von Wirkketten

Wenn die anderen glauben, man ist am Ende,
dann muß man erst richtig anfangen!
Konrad Adenauer (1876 - 1967)

6 Übergang zur Softwarearchitektur in AUTOSAR

Im Kapitel 4.3 wird die Architekturmodellierung mit SysML bzw. UML erläutert. Für die zukünftige Entwicklung von Steuergeräten ist die Verwendung des AUTOSAR Standards unumgänglich. Das Endergebnis der Steuergeräteentwicklung ist somit in Zukunft ein AUTOSAR Modell bzw. ein zu AUTOSAR kompatibler Code. Der AUTOSAR Standard enthält jedoch keine Methoden zur Analyse bzw. zur Verhaltensmodellierung. Diese sind aber ebenso erforderlich für den vorgestellten Top-Down Ansatz, wie für den Bottom-Up Ansatz. Ein Bottom-Up Ansatz liegt dann vor, wenn der OEM Komponenten hinzu liefert, die der Zulieferer in das Steuergerät zu integrieren hat. Hier sieht der AUTOSAR Standard vor, dass dem Zulieferer ein Extrakt aus der Steuergerätebeschreibung (ECU Extract of System Description (ECU-C)) zur Verfügung gestellt wird. Zusätzlich erhält der Zulieferer noch textuelle Anforderungen. Er hat sicherzustellen, dass diese zu dem zugelieferten Modell kompatibel sind. Dies ist nur durch eine eingehende Analysephase zu gewährleisten, die gleichzeitig eine Nachvollziehbarkeit (Traceability) zwischen den Anforderungen bzw. der Analyse und dem Modell herstellt [Mey14]. Deshalb ist die Kombination von UML/SysML mit AUTOSAR eine Lösungsmöglichkeit, um eine durchgängige Entwicklungsmethodik für den Automobilbereich herzustellen [Bol09] und wird deshalb im Ansatz AutoMoMe angewandt (vgl. Abbildung 6.1).

Die bisher erarbeiteten Entwicklungsartefakte werden aber nicht verworfen, sondern sie werden im AUTOSAR Modell weiterverwendet [KPLJ08]. Da die SysML bzw. UML und AUTOSAR unterschiedliche Metamodelle verwenden, ist eine Überführung nicht ohne weiteres durchführbar. Vielmehr wird eine Modell-zu-Modell Transformation (M2M) benötigt. Es werden Daten innerhalb von einer Abstraktionsebene von einem Modell zu einem anderen übertragen. Deswegen kann von einer horizontalen Transformation gesprochen werden [GPR06]. Bei der Transformation ist zu beachten, dass nicht alle Daten

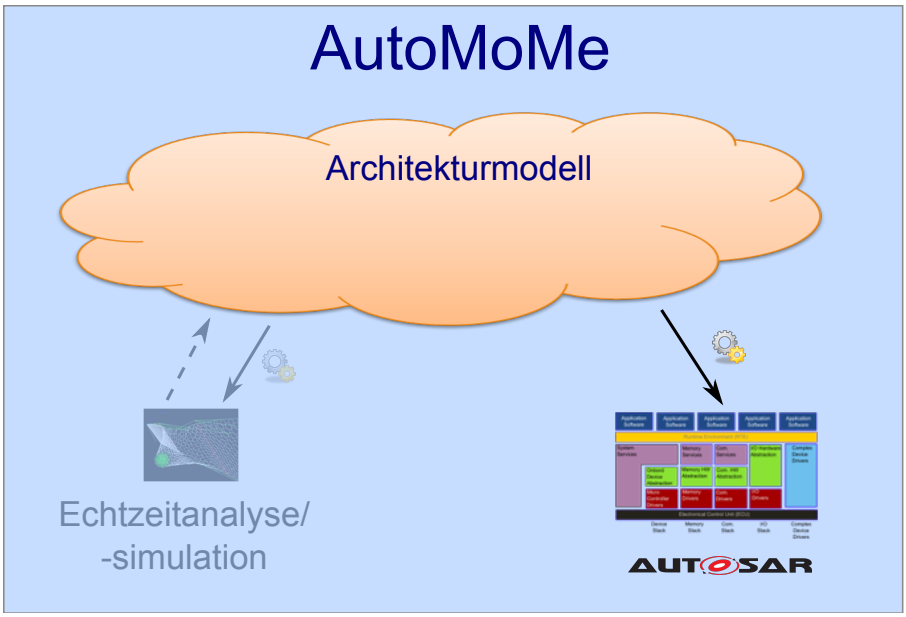


Abb. 6.1. Transformation des Architekturmodells nach AUTOSAR

zwischen dem Architekturmodell und dem AUTOSAR Modell ausgetauscht werden müssen. So werden aus dem Architekturmodell die gesamten Analysedaten nicht im AUTOSAR Modell benötigt. Desgleichen gibt es in AUTOSAR Detaillierungen, die für das Architekturmodell nicht von Belang sind, wie die AUTOSAR Konfigurationsdaten. Diese werden für die Generierung des AUTOSAR Codes benötigt. Sie stellen eine Detaillierung dar und werden auf der Architekturebene nicht benötigt. Dieser Sachverhalt wird in Abbildung 6.2 gezeigt. Der Kreis deutet an, dass nicht alle, sondern nur ausgewählte Daten zwischen den Modellen ausgetauscht werden. Im vorgestellten Ansatz AutoMoMe wird für die Transformation zwischen UML und AUTOSAR somit keine vollständige Transformation beschrieben, aber es werden Regeln für die Elemente aufgestellt, die in beiden Modellen benötigt werden und somit immer konsistent gehalten werden müssen. Es werden daher aufbauend auf den Modellierungsregeln von AutoMoMe entsprechende Transformationsregeln erstellt. Diese werden durch Syntheseschritte ergänzt [Kar04], da im Fall der AutoMoMe mehrere Eingaben aus dem SysML/UML Architekturmodell zu einem Element in der AUTOSAR Architektur kombiniert werden (vgl. Definition Synthese).

Definition (Synthese). Unter einer Synthese versteht man die Kombination von mehreren Eingaben, die dann zu einer Ausgabe führen [ALSU08].

Die Transformation und die damit einhergehende Synthese (siehe Definition Synthese) erfolgt in zwei Schritten. Zunächst werden die Transformationsregeln beschrieben, die zur Übernahme der Informationen der Softwarearchitektur erforderlich sind. Hierbei werden zunächst die Applikations-Software und die Daten zur Konfiguration der RTE transformiert. Anschließend wird in einem weitergehenden Verfahrensschritt dargelegt, wie Informationen außerdem aus dem Architekturmodell dazu genutzt werden können, Teile der Basis-Software zu erstellen bzw. zu konfigurieren. In einem zweiten Schritt wird erläutert, wie Daten aus der Systemarchitektur genutzt werden, um ein AUTOSAR System zu erstellen, in dem die Topologie des Komfortsteuergeräts in Form von Kommunikationsbussen und Rechenkernen beschrieben wird. Eine Übersicht über die Transformation von der Architektur in SysML/UML nach AUTOSAR ist der Abbildung 6.2 zu entnehmen.

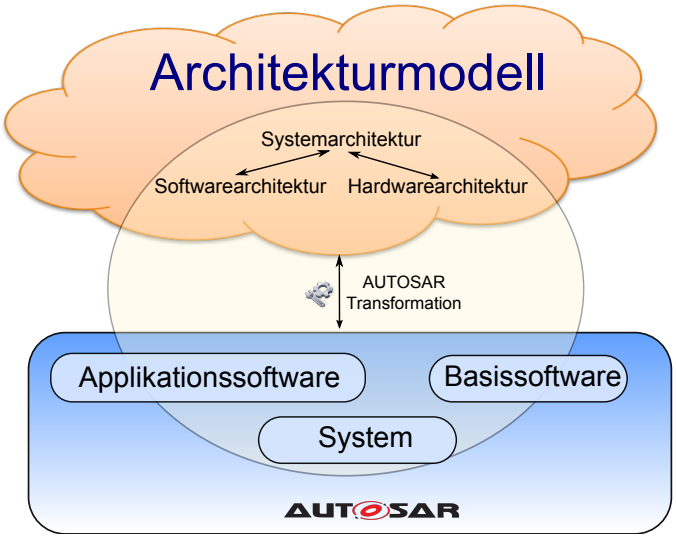


Abb. 6.2. Übersicht über die Transformation nach AUTOSAR

Die Transformation von UML nach AUTOSAR ist dabei für die Entwicklung bedeutsam. Es darf hier zu keinem Verlust von Daten kommen. Daher muss die Transformation entsprechend abgesichert werden [SBC+13], so dass es nicht vorkommen kann, das Daten nicht transformiert werden. Im Fall von

AutoMoMe wird bei der Transformation zusätzlich eine Navigation zwischen dem UML und dem AUTOSAR Modell durch die Speicherung der jeweiligen IDs ermöglicht. Diese werden zum Ende der Transformation verwendet, um sicherzustellen, dass alle Elemente, die nach AUTOSAR übertragen werden mussten, auch wirklich übertragen sind. Hierdurch ist eine erste und doch wertvolle Absicherung für die Transformation gegeben.

6.1 Stand der Technik Modelltransformationen zur Integration des AUTOSAR Standards

Die Idee den AUTOSAR Standard mittels Modelltransformationen in die Entwicklung von Steuergeräten einzubinden, ist bereits in anderen Arbeiten aufgegriffen worden. In diesem Abschnitt werden diese Ansätze vorgestellt und die Unterschiede zum AutoMoMe Ansatz herausgearbeitet.

In [SWCD13] wird eine Studie vorgestellt, wie aus einem UML Modell mittels einer Modelltransformation ein AUTOSAR Modell entsteht. Von daher ist eine ähnliche Problemstellung und ein vergleichbarer Lösungsansatz gegeben. Jedoch beschränkt sich dieser Ansatz auf die Transformation des System Templates vom AUTOSAR Standard [AUT10g], d. h. es werden vornehmlich die Steuergeräte und die AUTOSAR Instanzen übertragen. Die Definition von Datentypen, Runnables und Schnittstellen oder aber die Konfiguration der Basis-Software spielen in dem Ansatz keine Rolle. Dies sind aber die Probleme, mit denen ein automobiler Zulieferer zu kämpfen hat. Von daher ist der Ansatz und die verwendeten Techniken ein guter Startpunkt, aber noch nicht ausreichend.

In [GHN09] sind bereits Transformationsregeln zwischen einem SysML Modell und der AUTOSAR Applikations-Software definiert. Bei der Evaluation dieses Ansatzes im Rahmen des SPES 2020 Forschungsprojektes [SPE09] hat sich gezeigt, dass innerhalb der Systemarchitektur (SysML) noch nicht die notwendigen Daten vorliegen. Beispielsweise sind die Datentypen für die Software Schnittstellen noch nicht definiert. Darüber hinaus fehlen Transformationsregeln für die RTE-Ebene. Ferner hat sich gezeigt, dass die Transformationsregeln hauptsächlich auf die Einführung von Stereotypen in SysML basieren, die denen von AUTOSAR entsprechen. Somit wird die Systemarchitektur nur durch die AUTOSAR Begrifflichkeiten ergänzt, ohne einen Mehrwert an Übersichtlichkeit oder eine zusätzliche Detaillierung zu erreichen. In Anbetracht, dass ein Großteil der in [GHN09] genutzten Stereotypen durch die Einhaltung entsprechender Entwicklungsregeln überflüssig werden, werden in AutoMoMe weitergehende eigene Transformationsregeln und Syntheseschritte erstellt, die im Folgenden vorgestellt werden.

6.2 Transformation der Applikations-Software

In einem ersten Transformationsschritt geht es darum, die Informationen der Applikations-Software von der UML nach AUTOSAR zu transformieren. Zum besseren Verständnis, welche Informationen in AUTOSAR benötigt werden und welche nicht, ist es zunächst sinnvoll, auf den Aufbau von AUTOSAR innerhalb der Applikations-Software näher einzugehen.

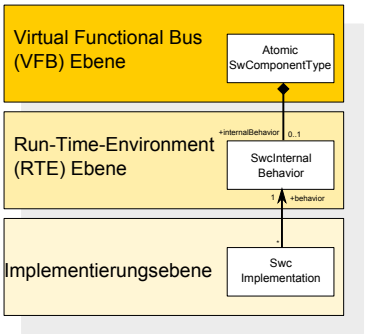


Abb. 6.3. Spezifikation einer Applikationskomponente in AUTOSAR [AUT10b]

Der AUTOSAR Standard beginnt mit der Softwarearchitektur und benötigt somit keine Daten aus der Analysephase. Von daher werden diese Ergebnisse bei der Transformation nicht berücksichtigt. Die Applikations-Software in AUTOSAR spezifiziert die Struktur der Komponenten und ihre Kommunikationsbeziehungen. Jede AUTOSAR Komponente wird dabei in drei Bereiche unterteilt (vgl. Abbildung 6.3). In der VFB Ebene werden die einzelnen Komponenten, ihre Ports und Schnittstellen spezifiziert. In der darunterliegenden Ebene wird das Kommunikationsverhalten also die Operationen – in AUTOSAR Runnables genannt – und ihre Anbindungen zur RTE beschrieben. Diese beiden Ebenen werden für die RTE Generierung benötigt. Es ist erforderlich sowohl für die VFB- als auch für die RTE-Ebene Transformationsregeln zu entwickeln, um alle notwendigen Daten aus dem Architekturmodell in AUTOSAR weiter zu verwenden. Unterhalb der RTE befindet sich in AUTOSAR die Implementierung, die den Code vorhält, der das Verhalten der Runnables wiedergibt. Da AUTOSAR keine Verhaltensmodellierung enthält, wird hier nicht das Design beschrieben, sondern nur die Möglichkeit gegeben, Code in das Modell einzuhängen. Ob der Quellcode manuell implementiert wurde oder modellbasiert entstanden ist, ist dabei nicht entscheidend. AUTOSAR verlässt sich an dieser Stelle auf die sogenannten *Behavior Modeling Tools (BMT)* Werkzeuge [AUT10b]. Dies können beispielsweise Matlab/Simulink [Mat14], TargetLink[dSP14] oder UML Modelle sein [Gro11d].

6.2.1 Die Transformation der VFB-Ebene

In der VFB-Ebene werden die jeweiligen AUTOSAR Komponenten und ihre Ports und Schnittstellen spezifiziert. Dies kann anhand des Beispiels der Komponente *Alarmanlagensteuerung* demonstriert werden. In der Abbildung 4.41 wird im Architekturmodell diese Komponente bereits modelliert. Bei der Transformation in die AUTOSAR VFB-Ebene muss sowohl die Komponente an sich, ihre Ports als auch die Schnittstellen transformiert werden (siehe Transformationsregeln 1 - 7 im Anhang A.9). Bei der Komponente gilt es zu beachten, dass in AUTOSAR unterschiedliche Komponenten vorhanden sind, z. B. *SoftwareComponentType, SensorActuatorComponentType, ServiceType*. Ebenso ist zu berücksichtigen, dass im AUTOSAR Standard eine Instanziierung der Komponenten erfolgt. Dies ist ähnlich zur Klassen-/Objekt-Beziehung in der UML zu sehen.

So ist für die Transformation zunächst zu entscheiden, ob eine Komponente des Architekturmodells der Applikations- oder Basis-Software zuzuordnen ist. Diese Alternative wird dadurch entschieden, in welchem Bereich des Modells die Komponente spezifiziert ist. Die Aufteilung in Applikations- und Basis-Software ist durch Stereotypen für die einzelnen Pakete definiert (siehe Kapitel 4.3). So werden bei der Transformation in die VFB-Ebene von AUTOSAR nur Komponenten berücksichtigt, die aus dem Applikationsbereich stammen. Die Unterscheidung, ob es sich um eine Applikations- oder Servicekomponente handelt, kann anhand der Serviceports bestimmt werden (siehe Transformationsregeln 5 im Anhang A.9). Sollten solche vorhanden sein, handelt es sich um eine Servicekomponente. Die Unterscheidung, ob es sich um eine *SensorActuator* Komponente handelt, kann anhand der Lage der Komponente im Modellbaum entschieden werden. Ist die übergeordnete Komponente ein Sensor oder Aktuator, dann wird die Komponente zu einer *SensorActuator* Komponente transformiert. Somit sind die Entscheidungen, welche Transformationsregel anzuwenden sind, gegeben. Die Transformationsregeln werden nachfolgend anhand des Beispiels des Komfortsteuergerätes noch einmal verdeutlicht.

Im Beispiel der Alarmanlagensteuerung ist die Komponente im Applikationsbereich des Modells definiert (siehe Paketstruktur in Abbildung 6.4). Die Komponente weist weder Service Ports auf noch ist das übergeordnete Element ein Sensor oder Aktuator. Somit wird aus der Alarmanlagensteuerung die *SoftwareComponenteType* Alarmanlagensteuerung (vgl. Abbildung 6.4), welche sich in dem AUTOSAR Paket *ComponentLibrary* befindet. Da die Komponente den Stereotyp «Singleton» aufweist und sie daher nur einmal im Modell instanziiert wird, kann auch gleichzeitig im *Alarmanlagen* Paket eine entsprechende *SoftwareComponentPrototype* Komponente erzeugt werden (siehe unterstes Paket in Abbildung 6.4).

In der Abbildung ist ebenfalls veranschaulicht, dass die Ports der Komponente zu einem RPort bzw. PPort transformiert werden. Dies ist abhängig

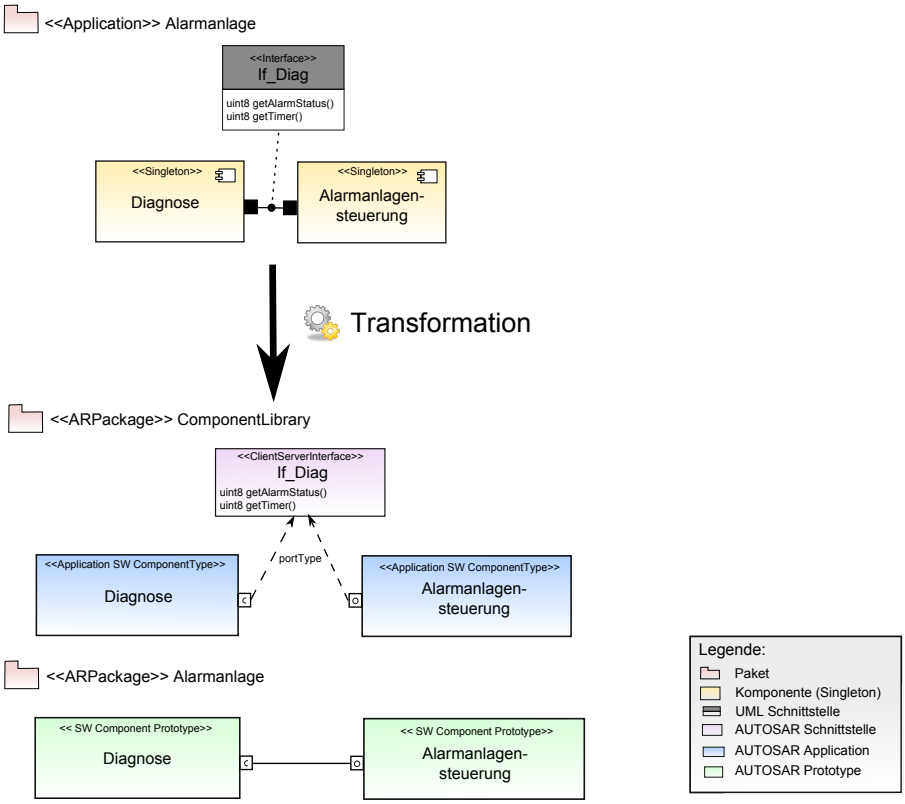


Abb. 6.4. VFB Transformation anhand des Beispiels der Alarmanlagensteuerung

davon, ob der Port eine erfüllende (required) oder anbietende (provided) Schnittstelle aufweist. Im Beispiel der Alarmanlagensteuerung wird daher ein PPort verwendet. Dieser hat eine *portType* Verbindung zur Schnittstelle, so dass auch diese transformiert wird. In AUTOSAR sind zwei verschiedene Arten von Schnittstellen vorhanden: *SenderReceiver* und *ClientServer* Schnittstellen, wobei letztere auch von einem sogenannten Service Port genutzt wird, um die Verbindung zur Basis Software herzustellen.

Die Unterscheidung in UML erfolgt anhand der Inhalte der Schnittstellen. Falls Operationen in der Schnittstelle definiert sind, kommt eine *ClientServer* Schnittstelle zur Anwendung. Sind nur Attribute in der Schnittstelle vorhanden, wird eine *SenderReceiver* Schnittstelle verwendet. Verbindet dagegen eine ClientServer Schnittstelle Komponenten von der Applikation zur Basis Software, wird der entsprechende Port zu einem Service Port. In diesem Beispiel wird eine ClientServer Schnittstelle zur Verbindung der beiden Applikationskomponenten genutzt. Es wird so exemplarisch aufgezeigt wie eine Transformation von UML in die VFB-Ebene von AUTOSAR durch die in AutoMoMe

entwickelten Transformationsregeln erfolgt (siehe Transformationsregeln 6 - 7 im Anhang A.9).

6.2.2 Die Transformation der RTE-Ebene

In der RTE-Ebene geht es um die Instanziierung der AUTOSAR Komponenten in Form von sogenannten Prototypen und dessen Verbindungen untereinander. Ferner zählt zu der RTE-Ebene auch das Kommunikationsverhalten einer AUTOSAR Komponente, die mittels einer Internal Behavior beschrieben wird. Hierunter fallen die Festlegungen der Runnables, der internen Runnable Variablen und der ExclusiveAreas. Diese Daten werden für die RTE-Generierung benötigt (vgl. Abbildung 6.3). Wie bereits eben ausgeführt, werden die Komponenten und ihre zugehörigen Ports nach AUTOSAR übertragen. Auf der RTE-Ebene werden die Komponenten nun instanziiert und miteinander verbunden. Da die Verbindungen der Komponenten ebenfalls in der UML Softwarearchitektur zu finden sind, können sie ebenfalls transformiert werden. So wird aus der *Alarmanlagensteuerung* vom Typ AtomicSoftwareComponentType ein Prototyp, der diesen Typ realisiert (vgl. Abbildung 6.3).

Innerhalb der RTE-Ebene findet aber nicht nur die Instanziierung der Komponenten aus der VFB-Ebene statt. Vielmehr wird innerhalb dieser Ebene auch das Kommunikationsverhalten der Komponenten definiert. Dies geschieht im sogenannten Internal Behavior, welches einer AUTOSAR Komponente zugeordnet ist (siehe Abbildung 6.3 und siehe Transformationsregeln 9 im Anhang A.9). Innerhalb des Internal Behaviors werden die Funktionen und ihre Aufrufe in sogenannten Runnables definiert. Daneben befinden sich in diesem Teil des AUTOSAR Standards aber auch gemeinsam genutzte Variablen (Interrunnables) und exklusive Bereiche (Exclusive Areas) (siehe Transformationsregeln 12 - 14 im Anhang A.9). Im Folgenden werden die einzelnen Bestandteile und ihre Transformationsregeln aus der UML beschrieben.

Runnables

Im Internal Behavior sind Runnables die zentralen Elemente (siehe Transformationsregeln 10 im Anhang A.9). Die Runnables stellen in AUTOSAR Funktionen dar, die durch ein Ereignis (Event) aufgerufen werden. Einer AUTOSAR Komponente können dabei eine oder mehrere Runnables zugeordnet sein. Innerhalb der Runnables ist das eigentliche Verhalten, z. B. ein Algorithmus, beschrieben bzw. implementiert. Hierfür gibt es in dem AUTOSAR Standard jedoch keine Beschreibungsform, sondern es wird der entsprechende Code in das Modell hinzugefügt. Die Runnables sind vergleichbar mit Operationen in der UML. Das Verhalten der Operationen wird im Ansatz AutoMoMe in Form von Aktivitätsdiagrammen beschrieben (siehe Kapitel 4.3.5).

Öffentliche Operationen in UML werden zu Runnables in AUTOSAR transformiert. Für die Transformation wird zusätzlich aber noch die Information benötigt, durch welches Ereignis das Runnable ausgeführt wird. Diese Information ist bereits in der Modellierung des Architekturmodells berücksichtigt (vgl. Abbildung 4.43). Die Ereignisse zum Starten der Runnables werden innerhalb der RTE verarbeitet, und die RTE ruft die betreffenden Runnables auf. Folgerichtig werden die Ereignisse als RTE Events bezeichnet. Die AUTOSAR Version 4.0 beinhaltet zwölf verschiedene Ereignisse, die für die Runnables genutzt werden können (vgl. Tabelle 6.1). Die RTE Events starten dabei entweder das Runnable oder wecken ein Runnable an dem spezifizierten Wartepunkt auf [AUT10e].

RTE Event	**Beschreibung**
TimingEvent	Auslösung des Events durch ein zeitliches Event, d.h. nach dem eine vorgegebene Zeit verstrichen ist. Dies kann z.B. durch ein OSEK Alarm erfolgen.
BackgroundEvent	Wird eingesetzt, um im Hintergrund Aktivitäten zu starten. Es ist einem Timing Event ähnlich, jedoch besitzt dieses Event keine feste Zeitperiode. Typischerweise wird das Runnable, welches durch das Event ausgelöst wird, nur mit der geringsten Priorität gestartet.
SwcModeSwitchEvent	Nach dem Umschalten in einen neuen Betriebsmodus
ModeSwitchedAckEvent	Ein Runnable wird durch das RTE Event über eine ModeSwitch Bestätigung informiert
ExternalTriggeroccurredEvent	Ein Runnable wird auf Grund eines externen Events gestartet.
InternalTriggerOccurredEvent	Ein Runnable wird durch einen internen Auslöser gestartet.

Tabelle 6.1. Allgemeine RTE Events für alle Schnittstellen [AUT10e]

Die Information, welches Ereignis die jeweilige Operation aufruft, ist im UML Modell nicht explizit angegeben. Daher wurde das Architekturmodell an dieser Stelle mittels eines Eigenschaftswertes ergänzt, so dass im Ansatz AutoMoMe diese Informationen vorliegen und für die Transformation nach AUTOSAR verwendet werden (siehe Abbildung 4.43). Im nachfolgenden Beispiel wird die Transformation der Klasse *Innenlicht* von UML nach AUTOSAR dargestellt (siehe Abbildung 6.5). Die Klasse *Innenlicht* besitzt die beiden Operationen *lightOn()* bzw. *lightOff()*, die durch ein Ereignis aktiviert werden, das die Ankunft von Daten repräsentiert (DataReceiveEvent). In der AUTOSAR Notation ergibt sich daraus eine AtomicSoftware Komponente namens *Innenlicht*, die ein Internal Behavior aufweist. In dem Internal Behavior befinden sich die beiden Runnables *lightOn* und *lightOff*, die von entsprechenden Ereignis-

RTE Event	Beschreibung
DataReceivedEvent	Dieser Event wird ausgelöst, wenn eine Datenbotschaft empfangen wird
DataReceiveErrorEvent	Dieser Event wird ausgelöst, wenn ein Fehler beim Übertragen einer Datenbotschaft auftaucht
DataSendCompletedEvent	Nach einer Sendebestätigung für eine Datenbotschaft wird dieser Event ausgelöst. (explizite Kommunikation)
DataWriteCompletedEvent	Das Event wird ausgelöst, wenn eine Bestätigung über eine erfolgreiche Kommunikation vorliegt. (implizite Kommunikation)

Tabelle 6.2. Ereignisse bei einer Sender/Receiver Schnittstelle [AUT10e]

RTE Event	Beschreibung
OperationInvokedEvent	Dieser Event wird nach dem Aufruf einer Server-Funktion gestartet.
AsynchronousServerCallReturnsEvent	Abschluss des asynchronen Aufrufs einer Serverfunktion

Tabelle 6.3. Ereignisse bei einer Client/Server Schnittstelle [AUT10e]

sen (DataReceiveEvent) aus der RTE aktiviert werden (siehe Transformationsregeln 11 im Anhang A.9).

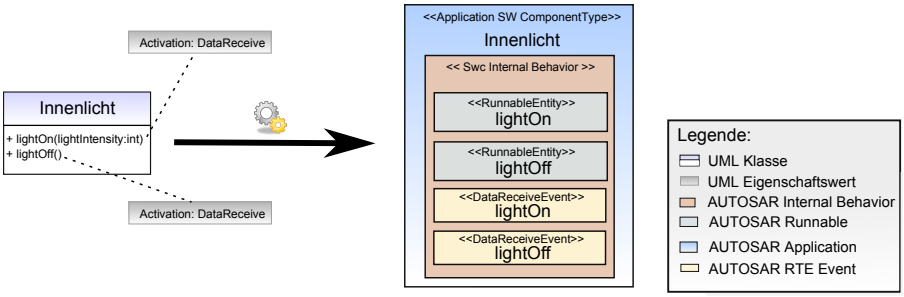


Abb. 6.5. Transformation von Operationen zu AUTOSAR Runnables

Neben der Aktivierung eines Runnables muss zusätzlich noch der Datenzugriff (Data Access) angegeben werden. Bei dem Datenzugriff geht es darum festzulegen, welche Daten aus der Schnittstelle von dem Runnable genutzt werden. Ein Runnable kann sowohl lesend als auch schreibend auf Daten zugreifen. Die notwendigen Informationen zur Spezifikation in AUTOSAR befinden sich in der Modellierung der UML Schnittstelle. Innerhalb der Schnittstelle sind die verschiedenen Argumente aufgeführt, auf die zugegriffen werden kann. Gleichzeitig sind für die jeweiligen Funktionen die Aufrufe mit ihren Argumenten

bekannt. Ausgehend hiervon kann über eine Namensgleichheit abgeprüft werden, ob ein Runnable entsprechende Daten verwendet oder nicht. Aus der UML Schnittstelle kann somit abgeleitet werden, ob die Daten lesend oder schreibend genutzt werden. Es wird für jedes Argument die entsprechende Richtung (in, out bzw. in/out) festgelegt. Bei einem Funktionsargument mit der Richtung *in* wird von der Runnable nur lesend auf das Argument zugegriffen. Die anderen Richtungen können dementsprechend gesetzt werden.

Im nachfolgenden Beispiel wird die Definition des Datenzugriffs aus dem UML Modell wiedergegeben. Es wird dabei wieder auf das Beispiel der Innenlichtsteuerung zurückgegriffen, die bereits für die Transformation der Runnables verwendet wurde. Es existiert in der Schnittstelle ein Datenelement *lightIntensity*. Dieses Datenelement wird als Argument der Operation *lightOn()* verwendet. Dabei wird durch die Richtung (Direction) vorgegeben, dass dies Argument in die Operation eingeht. So ist definiert, dass die Operation nur lesend auf das Argument zugreift. Nach der Transformation zu AUTOSAR wird der Datenzugriff mittels eines «DataReadAccess» modelliert. Hierdurch wird in AUTOSAR ausgedrückt, dass das Runnable lesend auf das Argument zugreift.

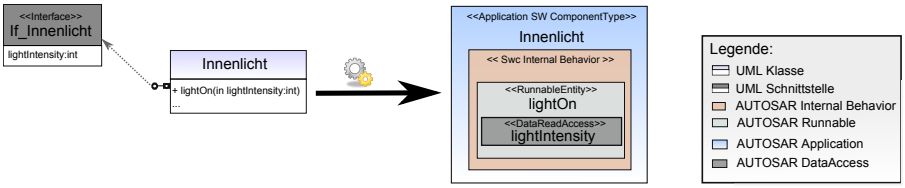


Abb. 6.6. Datenzugriff von Runnables

Interrunnable Variablen

Im AUTOSAR Standard wird durch die RTE eine Middleware zur Verfügung gestellt, die bei nicht AUTOSAR Systemen oft nicht vorhanden ist. Sie bietet somit weitere Funktionalitäten an. Eine dieser zusätzlichen Funktionen ist die Verwendung von Interrunnable Variablen. Mittels der Interrunnable Variablen ist es möglich, dass Runnables eines InternalBehaviors ihre Daten untereinander austauschen. Der Austausch kann dabei sogar zwischen nebenläufigen Runnables durchgeführt werden ([AUT10f] 7.4.2). Die RTE stellt dabei automatisch Speicherplatz zur Verfügung. Die Verwendung von Interrunnable Variablen kann aus dem Architekturmodell abgeleitet werden. Sofern innerhalb einer Operation eine andere Operation aufgerufen wird, die die Eigenschaft einer Runnable erfüllt, wird sie somit zu einer Interrunnable Variable transformiert, wenn sie gleichzeitig in derselben Klasse liegt. Es ist dann noch festzulegen, ob die Runnable lesend oder schreibend auf die Interrunnable Vari-

able zugreift. Dies kann anhand der Argumente bzw. der Rückgabewerte festgelegt werden.

Eine InterRunnable Variable kann dabei zwei verschiedene Verhaltensmuster aufweisen. Dies ist zum einen ein implizites und zum anderen ein explizites Verhalten. Der Unterschied zwischen beiden Verhaltensmustern ist die Art der Übergabe der Daten. Bei einem impliziten Verhalten werden die Daten kopiert, d. h. sie sind über die Laufzeit des Runnables stabil. Dies optimiert einerseits die Laufzeit des Systems, andererseits wird mehr Speicherplatz benötigt ([AUT10e] 4.2.5.6.1). Beim expliziten Verhalten können die zugreifenden Applikationen sofort auf die neuesten Daten (direkter RTE Zugriff) mittels eines Operationsaufrufes zugreifen. Dabei wird ein Sicherungsmechanismus genutzt, so dass von Seiten der RTE die Datenkonsistenz sichergestellt ist. Der Nutzer kann beim Übergang von dem Architekturmodell zur Softwarearchitektur bei der Synthese entscheiden, ob ein speicheroptimierter Übergang zum Einsatz kommen soll. Falls dies gewünscht wird, werden explizite Interrunnable Variablen angelegt. Andernfalls werden implizite erzeugt. Die Eigenschaft, ob implizites oder explizites Verhalten gewünscht ist, wird durch Eigenschaftswerte (TaggedValues) definiert.

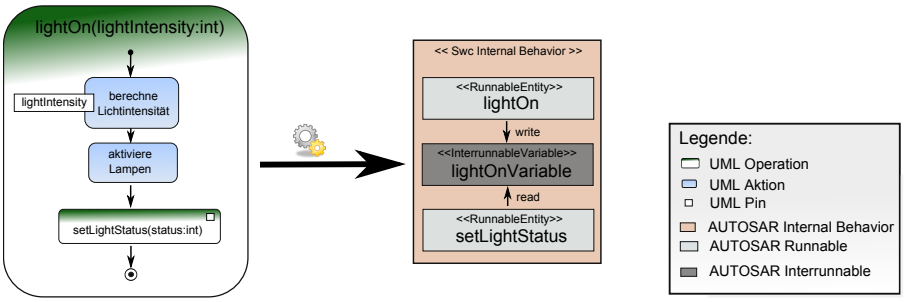


Abb. 6.7. Spezifikation zur Erstellung von Interrunnable Variablen

Im Beispiel (vgl. Abbildung 6.7) ist die Situation zur Erstellung einer Interrunnable Variable dargestellt. In der Operation *lightOn()* wird die Operation *setLightStatus()* aufgerufen. Dies erfolgt mittels eines Verweises (einer sogenannten CallBehavior) auf die Operation. Es wird eine Interrunnable Variable erzeugt, die den Namen der Operation aufweist. Daran anschließend muss noch spezifiziert werden, welche Runnable lesend und welche schreibend auf die Interrunnable Variable zugreift. Dies erfolgt anhand der Argument- bzw. der Rückgabewerte. Da im Beispielfall die aufgerufene Operation ein Argument besitzt, somit der Austausch hierüber erfolgt, greift die Runnable *lightOn* schreibend und das Runnable *setLightStatus* lesend darauf zu.

ExclusiveAreas

Für die Steuergeräteentwicklung im Automobilbereich ist die Datenkonsistenz von grundlegender Bedeutung. Falls Daten verspätet beim Empfänger ankommen oder alte und neue Daten zusammen versandt werden, kann dies zu Fehlern führen, da die Daten nicht stimmig sind. Um eine solche Vorgehensweise auszuschließen, muss eine dauerhafte Datenkonsistenz gewährleistet sein. Die Konsistenz kann in AUTOSAR auf unterschiedlichste Art und Weise hergestellt werden. Auf der einen Seite kann durch die Verwendung von komplexen Datentypen die Datenkonsistenz gewährleistet werden. In diesem Fall ist die RTE dafür verantwortlich, dass die Daten konsistent zueinander sind und zum Empfänger übertragen werden.

Eine andere Möglichkeit, die Datenkonsistenz zu erhalten, ist die Nutzung von Exclusive Areas. Die Exclusive Areas kommen dann zur Anwendung, wenn ausgeschlossen werden soll, dass einzelne Runnables oder Teile davon nebenläufig abgearbeitet werden. Durch Zuordnung der verschiedenen Runnables zu einer Exclusive Area wird festgelegt, dass nur jeweils ein Runnable innerhalb der Exclusive Areas gestartet wird und nicht von den anderen Runnables der Exclusive Area unterbrochen werden kann ([AUT10f] 7.4.1). Die Identifizierung solcher Bereiche erfolgt im Architekturmodell durch die Ressourcenmodellierung (siehe Abbildung 4.61).

Ein Beispiel ist in der Abbildung 6.8 dargestellt. Die beiden Operationen *readCANData()* und *sendCANData()* (vgl. Abbildung 4.61) greifen auf dieselbe Ressource CAN Register zu. Da nicht modelliert ist, dass ein shared Zugriff erfolgt, muss davon ausgegangen werden, dass nur jeweils eine Operation auf die Ressource zugreifen darf. Von daher muss die Resource besonders geschützt sein. Dies erfolgt in AUTOSAR durch die ExclusiveArea, der die beiden transformierten Runnables zugeordnet sind.

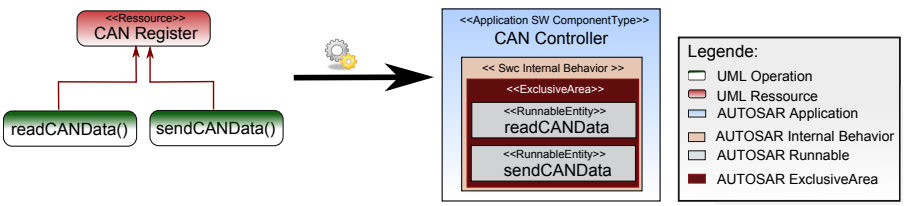


Abb. 6.8. Synthese zu einer ExclusiveArea

Die Realisierung einer Exclusive Area kann dabei unterschiedlich erfolgen. Eine mögliche Implementierung ist die Zuordnung der einzelnen Runnables zu einer gemeinsamen Task des Betriebssystems. Durch das Scheduling wird sichergestellt, dass die verschiedenen Runnables nicht nebenläufig ausgeführt werden. Es ist lediglich darauf zu achten, dass das Betriebssystem nur jeweils

eine Task ausführen kann ([AUT10f] 7.4.1.1). Eine weitere Möglichkeit, eine Exclusive Area zu implementieren, ist die Nutzung von Semaphoren. Beim Start einer Runnable wird ein entsprechender Mutex gesetzt, der erst beim Verlassen des Runnables wieder entfernt wird. Dies kann beispielsweise im Betriebssystem (OSEK) erfolgen [OSE05]. Durch die Verwendung der Semaphore wird die nebenläufige Abarbeitung der jeweiligen Runnables verhindert.

PerInstanceMemory

Falls eine Komponente mehrfach instanziiert werden kann, benötigt sie für ihre Daten zusätzlichen Speicherplatz, um die jeweiligen Instanzinformationen abzulegen. Ein Beispiel für eine mehrfache Instanziierung im Automobilbereich ist der Non-volatile memory (NVM) Ram mirror (siehe Abbildung 6.9). Dies ist eine Spiegelung des NVM Rams und sorgt wegen der redundanten Auslegung für eine zusätzliche Sicherheit. Es ist dann Aufgabe der RTE, dass die jeweilige Instanz auf den ihr zugeordneten Speicher zugreifen kann. Hierzu müssen aber zunächst die notwendigen PerInstanceMemorys angelegt werden. Ferner muss ein initialer Wert definiert sein, der die Größe des Speichers angibt ([AUT10f] 7.7).

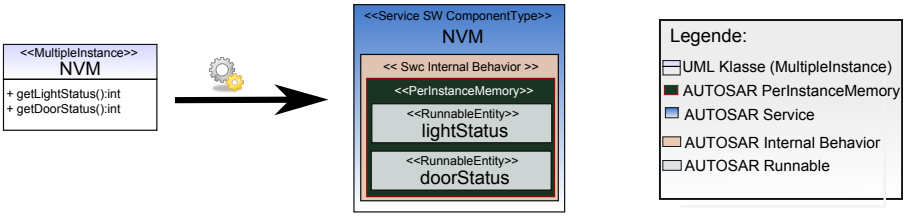


Abb. 6.9. Transformation zu einem PerInstanceMemory

Bei der automobilen Steuergeräteentwicklung ist die Nutzung von mehrfach instanziierbaren Komponenten der Ausnahmefall. Von daher ist jede Komponente nur einmal instanziierbar, es sei denn, sie weist den Stereotyp *«MultipleInstance»* auf (vgl. Abbildung 6.9). In einem solchen Fall ist bei der Transformation nach AUTOSAR bekannt, dass ein PerInstanceMemory angelegt werden muss. Zu diesem werden dann die Runnables und Daten hinzugefügt, die mehrfach instanziiert werden. In diesem Beispiel sind dies die Runnables *lightStatus* and *doorStatus*.

Servicekomponenten

Bei dem Extrakt aus der System Description von AUTOSAR, der für die Definition der ECU Description benötigt wird, sind nur die Applikationskomponenten enthalten. Die Basis-Softwarekomponenten und die RTE sind dort

nicht gespeichert (siehe Kapitel 2.3.3.1 [AUT10d]). Sie sind aber Bestandteil eines Steuergerätes und müssen somit für eine vollständige Spezifikation (ECU Description) vorhanden sein. So fehlt beispielsweise in dem ECU Extrakt die Verbindung zu den Servicekomponenten. Unter einer Servicekomponente werden im AUTOSAR Standard Komponenten verstanden, die dadurch gekennzeichnet sind, dass sie bei der Interaktion mit der Applikations-Software standardisierte AUTOSAR Schnittstellen verwenden (vgl. Abbildung 2.21). Sie können als hybrides Konzept angesehen werden, da sie sowohl eine Kommunikation mit der Basis-Software aber ebenso mit der Applikations-Software aufweisen [AUT10f]. Exemplarisch können folgende Komponenten angeführt werden, die in AUTOSAR als Services spezifiziert werden: NvRam Manager, Watchdog Manager, ECU State Manager etc. ([AUT10d] 2.3.3.1).

Im Architekturmodell sind die Servicekomponenten bereits enthalten, denn hier sind alle Komponenten und ihre Verbindungen modelliert. Die Servicekomponenten werden dabei anhand ihrer Ablage im Projektbaum identifiziert. Sie liegen unter dem Servicepackage (erkennbar durch den Stereotype «Basic» vgl. Abbildung 6.10). Für jede Komponente, die in einem Paket mit dem Stereotyp «Basic» liegt, wird eine *Service Sw ComponentType* in AUTOSAR generiert. Zugleich wird ein Internal Behavior und eine Implementierung für diese Servicekomponente angelegt. Die Implementierung richtet sich dabei an die *BasicSoftwareModuleDescription* (siehe Abbildung 6.10). Die Verbindung zwischen der Servicekomponente und der *BswModuleDescription* erfolgt über das Element *SwcBswMapping*. Ein entsprechendes Element wird automatisch beim Übergang nach AUTOSAR erzeugt. Des Weiteren können die Verbindungen zwischen den Applikations- und Servicekomponenten automatisch generiert werden. Hierzu müssen lediglich die entsprechenden Ports mit Serviceschnittstellen bereitgestellt werden. Wichtig ist, dass das Attribut *isService* auf true gesetzt wird ([AUT10f] constr2019), so dass der Port als *ServicePort* in AUTOSAR erkannt wird.

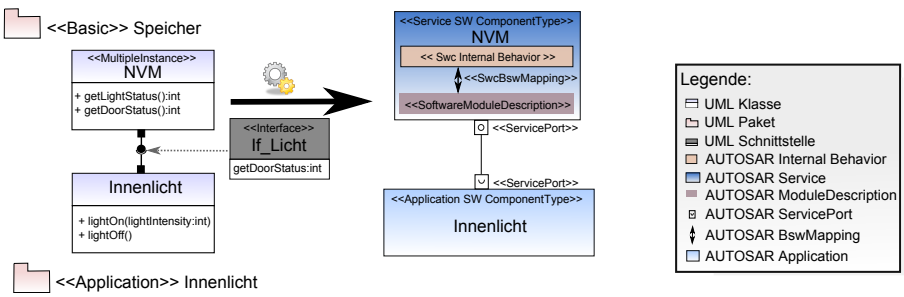


Abb. 6.10. Synthese zu einer Service Komponente in AUTOSAR

Implementierung

Die Abbildung 6.3 zeigt, dass zu einer AUTOSAR Komponente nicht nur das Internal Behavior zählt, sondern ebenso eine Implementierung in Form einer AUTOSAR Implementation. Die Implementierung kann dabei auf unterschiedliche Art und Weise realisiert werden. Es kann ein manuell entwickelter Code, aus einem UML Modell generierter Code oder aber auch ein regelungstechnisches Modell, wie z. B. Matlab/Simulink bzw. TargetLink aus dem der Code für das AUTOSAR Modell entsteht, sein. Der Code muss aber aus dem AUTOSAR Modell referenziert werden, damit er eingebunden werden kann. Wenn der Code aus dem UML Modell generiert wird, kann dies automatisiert erfolgen.

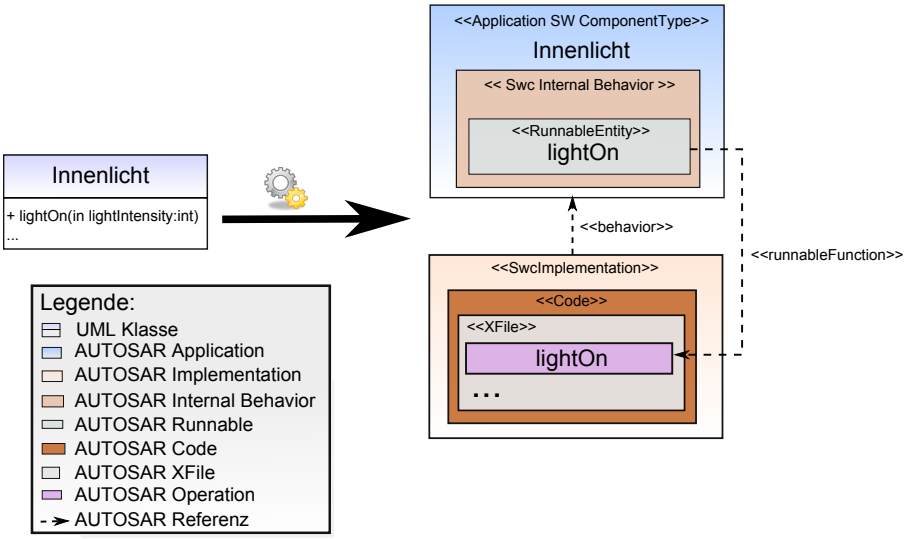


Abb. 6.11. Implementierung einer Runnable aus dem UML Modell

In der Abbildung 6.11 ist beispielhaft die Implementierung des Runnables *lightOn* aus der Komponente Innenlicht dargestellt. Es existiert zunächst ein Implementierungsbereich, in dem mit dem Element *Code* angezeigt wird, dass auf einen Quellcode verwiesen wird. Für jede Komponente gibt es ein sogenanntes *XFile*, in dem die einzelnen Funktionen des Codes noch einmal dargestellt sind. Die Runnables, in diesem Fall die Runnable *lightOn*, verweisen auf die Funktionen, und dass die Implementierung im Sinne von AUTOSAR vorgenommen wird.

Bei einer Implementierung werden ferner Angaben zum Ressourcenverbrauch eingetragen (ResourceConsumption). So werden in diesem Beispiel die Daten der Türsteuerung ins AUTOSAR Modell übernommen (siehe Abbildung 6.12).

Diese Angaben befinden sich gleichfalls im Architekturmodell und werden dort für die Simulation des Systems vorgehalten. Somit werden die Daten aus dem Architekturmodell nach AUTOSAR übertragen und weiterverwendet. Im Architekturmodell werden diese Werte in Eigenschaftswerten abgespeichert. Dies sind beispielsweise die Werte RAM_max bzw. ROM_max. Die Werte werden auf das AUTOSAR Element *MemorySection* abgebildet. So kann beispielsweise ein Speicherverbrauch von 4 Kbyte aus dem Architekturmodell in dem AUTOSAR Element MemorySection spezifiziert werden.

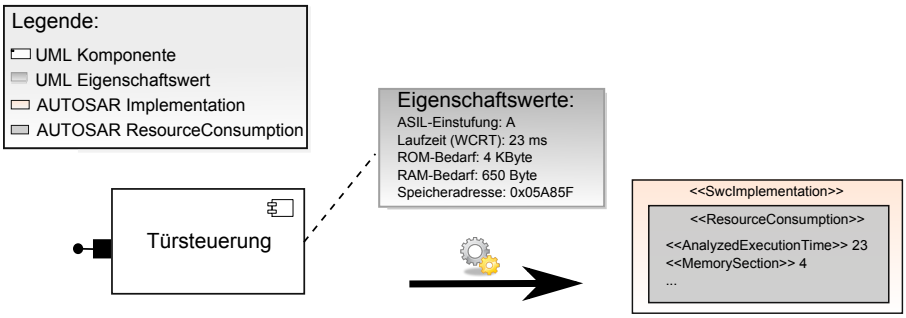


Abb. 6.12. AUTOSAR Resourcen Verbrauch

Neben den Speicherinformationen sind ebenfalls die Zeitinformationen im Architekturmodell hinterlegt. Ab der AUTOSAR Version 4.0 sind zeitliche Informationen spezifizierbar. Der AUTOSAR Standard kennt dabei unterschiedliche zeitliche Informationen (vgl. Abbildung 6.13). Für die Transformation ist besonders das SWC Timing wichtig, da diese Informationen im Architekturmodell abgelegt sind.

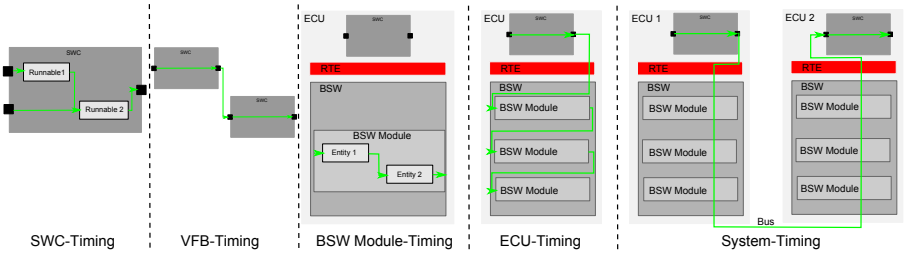


Abb. 6.13. Verschiedene Zeiten innerhalb des AUTOSAR Standards [AUT10b]

Gleichzeitig werden in der *ResourceConsumption* die Ausführungszeiten der Softwarekomponente festgelegt. Die Ausführungszeiten werden in der Architektur durch Eigenschaftswerte spezifiziert. Neben der Festlegung der Zeiten ist ebenso hinterlegt, wie diese Zeiten ermittelt werden. Dies kann durch

Messung, Analyse, Abschätzung oder Simulation erfolgen. Diese Werte werden nach AUTOSAR in die ResourceConsumption transformiert (vgl. Abbildung 6.12).

Das AUTOSAR System

In den bisherigen Abschnitten ist beschrieben, wie die Softwarearchitektur und ihre Elemente nach AUTOSAR transformiert bzw. synthetisiert werden. Ein automobiles Steuergerät besteht aber aus mehr als nur der Software. So spielt auch die Hardware eine gewichtige Rolle. Von daher findet sie auch Eingang in die AUTOSAR Modellierung. Die wichtigsten Bestandteile des Systems, wie die Rechenkerne und die Kommunikationsparameter werden ebenfalls in AUTOSAR definiert. Ferner wird eine Verteilung (Mapping) zwischen der Software und der Hardware vorgenommen. Die Modellierung dieser Aspekte in AUTOSAR ist zwingend erforderlich, um die RTE – also die Middleware – zu generieren.

In Abbildung 4.51 wird die oberste Hardware Topologie mit den Rechenkernen, dem Speicher und den Kommunikationsmodulen dargestellt. Für diese Daten gibt es ebenso Transformationen, die es ermöglichen, die SysML bzw. UML Daten in AUTOSAR weiter zu verwenden. In der Grafik 6.14 ist der Übergang anhand des Beispiels der beiden CPUs *MikroController* und *KommunikationsController* mit der Verbindung eines CAN Busses abgebildet. Es wird zunächst das System Komfortsteuergerät erstellt, in dem die beiden Rechenkerne *MikroController* und *KommunikationsController* zu finden sind. Wie bereits ausgeführt, verwendet der AUTOSAR Standard den Begriff ECU in einer etwas anderen Form. In AUTOSAR bezeichnet eine ECU einen Rechenkern. Von daher wird zum besseren Verständnis die Namensgebung im Architekturmodell anders gewählt.

Die beiden Rechenkerne kommunizieren über einen CAN Bus miteinander. Hieraus ergibt sich, dass neben dem Bus entsprechende Hardwareelemente benötigt werden, um die Daten des Busses zu lesen bzw. zu senden. Daher kann im AUTOSAR Modell durch die Verwendung des CAN Busses im Architekturmodell automatisiert der entsprechende CAN Controller und der CAN Bus erzeugt werden. Durch die Modellierung des CAN Controllers kann das AUTOSAR System die ein- und ausgehenden Daten verarbeiten und an die entsprechenden Komponenten weiterleiten. Dies erfolgt mithilfe des sogenannten COM-Stacks. Da die Daten aber gleichzeitig für die Generierung der RTE benötigt werden, macht es Sinn, sie beim Übergang von dem Architekturmodell hin zum AUTOSAR Modell zu berücksichtigen.

Im AUTOSAR System wird neben der Modellierung von Hardware Bestandteilen gleichermaßen die Verteilung der Software auf die entsprechende

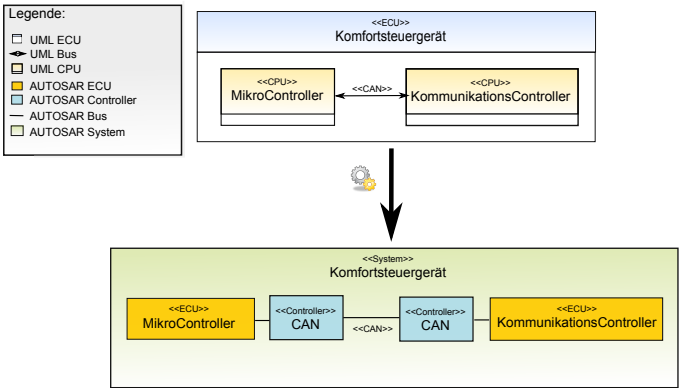


Abb. 6.14. Transformation von Systembestandteilen

Hardware festgelegt. Hierzu werden die AUTOSAR Prototypen einer *ECUInstance* zugewiesen. Diese Zuweisung ist für die RTE Generierung zwingend erforderlich, damit die entsprechenden Informationen vorliegen, ob Softwarekomponenten mittels des Speichers miteinander kommunizieren oder ob eine Kommunikation über einen Bus notwendig ist. Als Entscheidungsgrundlage kann die Spezifizierung des Betriebssystems in dem Architekturmodell dienen, da dort bereits die Verteilung der Software auf Tasks und somit die Zuweisung zu den einzelnen Recheneinheiten erfolgt. In der Abbildung 6.15 ist exemplarisch die Zuordnung der Komponente *Kommunikation* bzw. des Rechenkerns *KommunikationsController* dargestellt. Beide werden über Referenzen auf das Systemmapping abgebildet. Oftmals erfolgt diese Zuordnung nicht in einem Diagramm, sondern in einer Tabelle, da diese Variante bei vielen Einträgen übersichtlicher ist.

6.3 Konfiguration der Basis Software

Beim Übergang von UML zu AUTOSAR in AutoMoMe werden Informationen, die bereits im Architekturmodell vorhanden sind, zur Konfiguration der Basis-Software weiterverwendet. Es wird aber keinesfalls die Basis-Software vollständig generiert. Vielmehr werden nur die vorhandenen Daten übertragen. Die restlichen Konfigurationen werden in AUTOSAR vorgenommen. Sie werden nicht in die UML zurück übertragen. Die Intention der Synthese ist nicht die vollständige Konfiguration von Basis-Softwaremodulen. Die speziellen Konfigurationstools für die einzelnen Basis-Softwaremodule werden durch die Synthese nicht ersetzt. Vielmehr werden sie bei der Erstellung der richtigen Konfiguration unterstützend herangezogen. Bereits vorhandene Informationen werden genutzt und in die entsprechenden AUTOSAR Dateien eingepflegt. Die

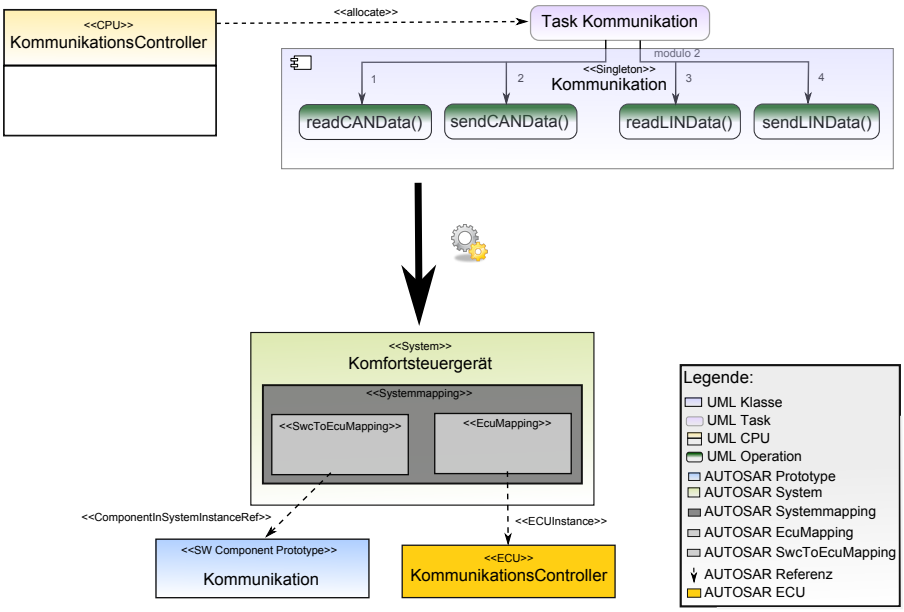


Abb. 6.15. Zuordnung von Komponenten und Rechenkernen im AUTOSAR System

Informationen können von den speziellen Konfigurationstools eingelesen, dort weiter spezifiziert und verfeinert werden.

6.3.1 Konfiguration des AUTOSAR Betriebssystems

Das AUTOSAR Betriebssystem beruht grundlegend auf dem *Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug (OSEK)* Betriebssystem, welches sich bereits derzeit in vielen automobilen Steuergeräten befindet. Die Erweiterungen zum AUTOSAR Betriebssystem ergeben sich aus der Abbildung 6.16. Basis des AUTOSAR Betriebssystem ist weiterhin das OSEK Betriebssystem. Zusätzlich wurden die drei Komponenten Schedule Tabellen, Zeit- und Speicher-Protektion hinzugefügt. Die Schedule Tabellen werden von einer Applikation aus gestartet und werden u. a. für die Synchronisierung mit einer globalen Zeitbasis wie dem FlexRay Bus verwendet. Die Zeit Protektion dient zum Erkennen von zeitlichen Verletzungen. Hierzu müssen aber Deadlines angegeben werden. Sinnvollerweise werden sie bei der Entwicklung durch eine Performanceanalyse (z. B. durch eine Systemsimulation) bestimmt. Die Speicher Protektion bietet Schutz gegen Speicherzugriffsfehler.

Da das AUTOSAR Betriebssystem zum großen Teil auf dem OSEK Standard basiert und dieses bereits für die Spezifikation von Betriebssystemeigenschaften

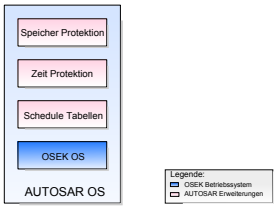


Abb. 6.16. AUTOSAR Betriebssystem [AUT10h]

im Architekturmodell verwendet wird (vgl. Kapitel 4.3.7), erfolgt auch hier eine Transformation vom Architekturmodell nach AUTOSAR, um die Informationen weiter zu verwenden. Folgende Informationen aus dem Architekturmodell werden zur Konfiguration des AUTOSAR Betriebssystems weiter eingesetzt:

- CPUs
- Tasks
- Interrupts
- Alarme
- Funktionsabbildung auf die Tasks
- Taskzuordnung auf die CPUs
- Ressourcenverbrauch von Funktionen

Die notwendigen Transformationsregeln werden auf der Grundlage des Komfortsteuergerätes definiert. Im Kapitel 4.3.7 wird die Erweiterung des Architekturmodells um Betriebssystemeigenschaften beschrieben. So sind die einzelnen Rechenkerne anhand des Stereotyps «CPU» kenntlich gemacht und durch zusätzliche Eigenschaftswerte ergänzt. Für das Komfortsteuergerät existiert beispielsweise der Rechenkern *MicroController* mit den Tasks *Licht* und *Basis* (siehe Abbildung 4.56). Hieraus wird zunächst die AUTOSAR Definition für die entsprechenden Tasks innerhalb des Betriebssystems erzeugt (vgl. linke Seite der Abbildung 6.17), d.h. es werden entsprechende Container innerhalb der Betriebssystemdefinition angelegt. Aus dem Architekturmodell ist ferner bekannt, welche Operationen den Tasks zugeordnet sind. Bei AUTOSAR sind dies die Runnables, deren Transformation bereits beschrieben ist. Für die Task *Licht* sind dies die Runnables *lightOn* bzw. *lightOff*. Diese werden nun durch einen «RteEventToTaskMapping» mittels des RteEvents der Task zugeordnet (vgl. rechte Seite der Abbildung 6.17). Vergleichbar ist die Transformation von Alarmen oder Interrupts zu sehen. Auch sie werden in AUTOSAR Container innerhalb des Betriebssystems definiert.

Neben dem Betriebssystem gibt es eine Komponente, die auf das Scheduling der Basis-Software eingeht. Dies ist die BSW-Scheduling Komponente. Ab der

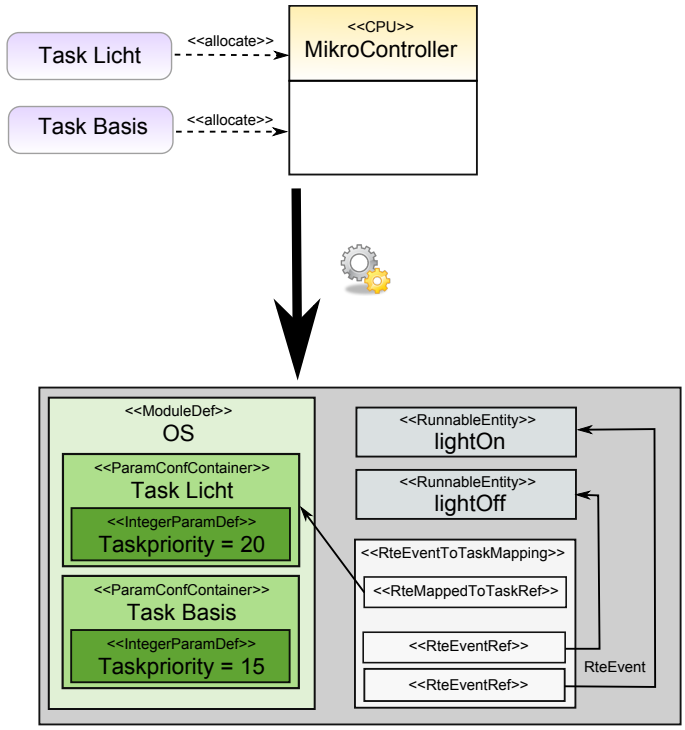


Abb. 6.17. Transformation der Betriebssystemeigenschaften

AUTOSAR Version 4.0 ist es kein eigenständiges AUTOSAR Modul mehr, sondern ist der RTE zugeordnet [AUT10e]. Sie sorgt dafür, dass die Runnables der Basis-Software entsprechend gescheduled werden. Davon ausgenommen ist das Betriebssystem und der *ECU State Manager* (vgl. [AUT10c] 017). Aus der Architekturmodellierung ergeben sich die verschiedenen Komponenten der Basis-Software. Dies sind alle Komponenten, die in einem Paket mit dem Stereotyp «Basic» gespeichert sind. Diese werden durch Events aufgerufen. Die BSW-Events sind dabei vergleichbar denen der RTE Events auf Applikationsebene.

Die Zuordnung der einzelnen Basisfunktionen auf die jeweiligen Tasks erfolgt auf gleiche Art und Weise wie die Zuordnung der Applikations-Software auf Tasks (vgl. Abbildung 6.17). Normalerweise wird für die Basis-Softwaremodule eine eigene Task eingesetzt. Aus den Eigenschaften kann die Zuordnung der Basis Software entnommen und im AUTOSAR Modell entsprechend übertragen werden. Die Werte der Zuordnung können dann in den Basic Scheduler mit den entsprechenden Werten eingetragen werden. Hierzu kommt der Parameter *RteBswEventToTaskMapping* zum Einsatz.

6.3.2 Die Vorkonfiguration des Speichermanagements

Ein weiterer wichtiger Aspekt bei der Entwicklung eines Steuergerätes ist die Spezifizierung des nicht-flüchtigen Speichers. In AUTOSAR wird diese Spezifizierung im NvRam Manager (NvM) durchgeführt. Die Daten werden im NvM in Blöcken gespeichert, den sogenannten NvM BlockDescriptoren. Diese Blöcke beinhalten die Informationen, die in einem Speichersegment abgelegt werden. Der NvM BlockDescriptor kann in AUTOSAR aus Daten von unterschiedlichen nicht flüchtigen Speichern bestehen (vgl. Abbildung 6.18). So ist es durchaus möglich, dass sowohl Daten aus dem Direktspeicherzugriff (RAM) oder aber ebenso aus dem Nur-Lesespeicher (ROM) mittels des NvM verwaltet werden. Im Folgenden wird erläutert, wie der NvM mit Daten aus dem Architekturmodell vorkonfiguriert werden kann.

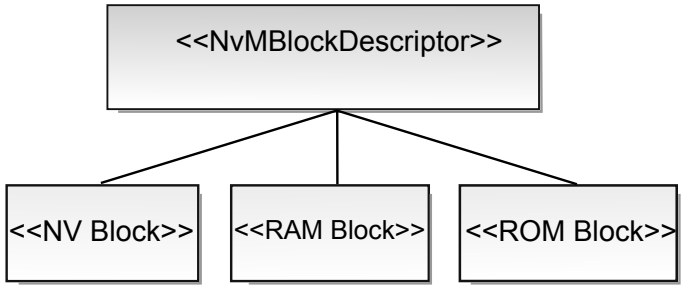


Abb. 6.18. Aufbau eines NvM Blocks

Der NvManager ist ein wesentlicher Bestandteil in einem Steuergerät, da er für die Speicherung der Daten zuständig ist. Dies hat zur Folge, dass alle Applikationskomponenten, die Daten permanent speichern, über die RTE auf den NvManager zugreifen. Durch die RTE werden die Anfragen und die entsprechenden Antworten an die Applikations-Software weitergeleitet. Der NvManager abstrahiert von den darunterliegenden teilweise hardwarespezifischen Eigenschaften. So sind die gesamten Treiber und die Memory Hardware Abstraction nicht für andere Komponenten zugreifbar, sondern der Aufruf und die Steuerung erfolgt über den NvManager (vgl. Abbildung 6.19). Ferner kommuniziert der NvM mit dem *Memory Abstraction Interface* (MemIf). Über diese Kommunikationsverbindung hat das NvM Zugriff auf die verschiedenen Speichertreiber, um so auf die jeweiligen Speichermedien zugreifen zu können. Die Abbildung 6.19 verdeutlicht, wie durch die Kommunikation von NvM und MemIf die verschiedenen Treiber und somit letztendlich der Zugriff auf die unterschiedlichen Speichermedien ermöglicht wird.

Die Konfiguration des NVM kann anhand eines Beispiels des Komfortsteuergerätes verdeutlicht werden. In Abbildung 4.62 wird ein Ausschnitt aus der Speicherbelegung des Komfortsteuergerätes vorgestellt. Dieser wird nun nach

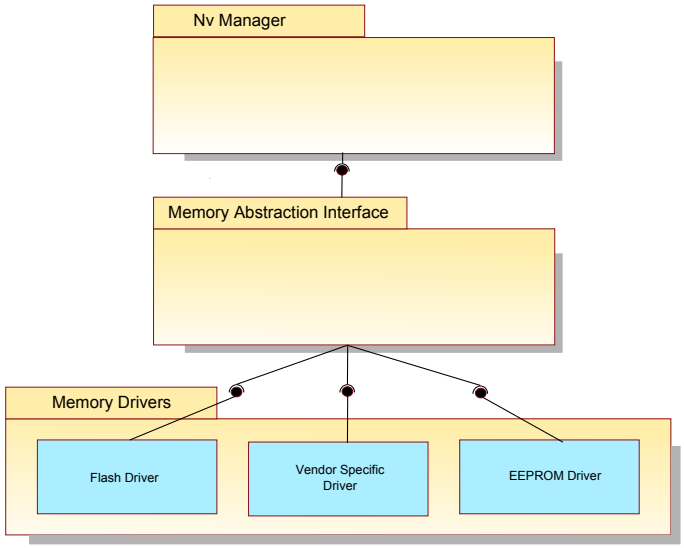


Abb. 6.19. Verbindung zwischen NVRAM Manager und den verschiedenen Speichertreibern

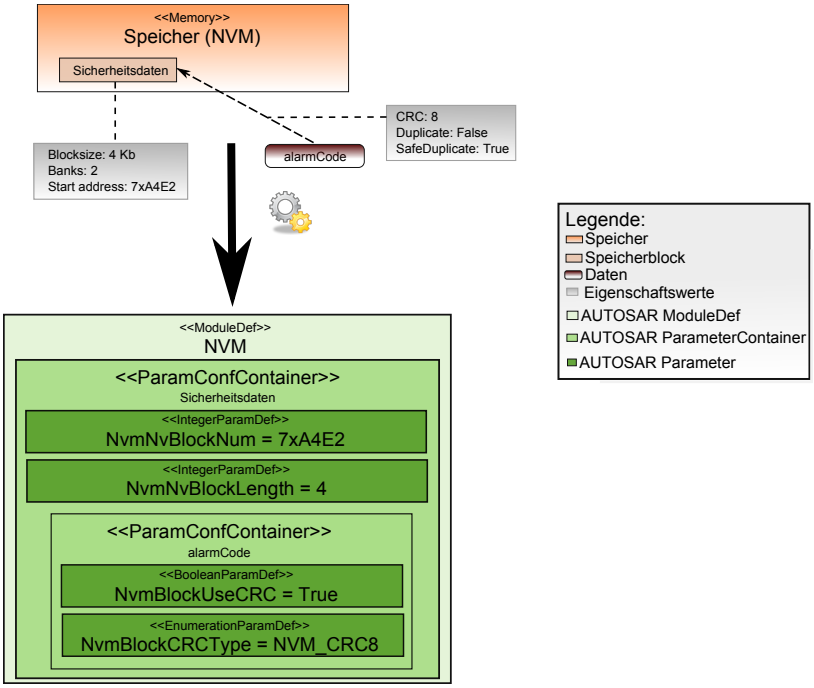


Abb. 6.20. Vorkonfiguration des NVM Managers

AUTOSAR transformiert (siehe Abbildung 6.20). Hierzu wird zunächst der Bereich der Sicherheitsdaten transformiert. Aus dem Speicherblock wird ein AUTOSAR Container, der zwei nummerische Inhalte besitzt. Die Inhalte beschreiben dabei die Startadresse und die Größe des Speichers. Ferner werden auch die Daten innerhalb des Speicherblocks übertragen. Dazu wird ein weiterer AUTOSAR Container namens *alarmDaten* erzeugt, der zwei Inhalte aufweist. Dies ist zum einen die Konfiguration zur Verwendung des CRC Mechanismus und zum anderen die Spezifikation des CRC Typs, in diesem Fall des *CRC8* Typs. Damit werden die Informationen aus dem Architekturmodell in das AUTOSAR Modell übernommen.

Als Fazit kann festgehalten werden, dass der Übergang zwischen dem Architekturmodell (SysML/UML) und dem AUTOSAR Standard realisierbar ist. So werden im Ansatz AutoMoMe Transformations- bzw. Syntheseregeln erstellt, die es erlauben, die wichtigsten Daten auszutauschen und somit beide Modelle konsistent zu halten. Durch die Beschränkung auf bestimmte Informationen, die ausgetauscht werden und unter Berücksichtigung der Designregeln aus der erarbeiteten Entwicklungsmethodik für SysML/UML ist es möglich, mit wenigen zusätzlichen Stereotypen auszukommen und trotzdem eine eindeutige Zuordnung zwischen den Modellelementen herzustellen.

Holzhacken ist deshalb so beliebt, weil man bei dieser Tätigkeit den Erfolg sofort sieht.
Albert Einstein (1879 - 1955)

7 Realisierung und praktische Erprobung des Ansatzes

In den folgenden Abschnitten wird die Implementierung des Ansatzes AutoMoMe und die anschließende praktische Erprobung in industriellen Serienprojekten beschrieben.

7.1 Realisierung in einem Werkzeug

Das in AutoMoMe beschriebene Konzept ist zur Evaluierung in einem Werkzeug zunächst prototypisch umgesetzt, um die Methodik erfolgreich zu unterstützen [WHR+13]. Das Werkzeug ist so gewählt, dass zum einen das Konzept damit realisierbar ist [WHR+13] und zum anderen es in die Werkzeuglandschaft eines automobilen Zulieferers integrierbar ist. Dazu ist ein Werkzeug erforderlich, dass sowohl die SysML/UML beherrscht als auch die Möglichkeit zur Modellierung eines AUTOSAR Modells anbietet. Ferner muss es so anpassungsfähig sein, um das Konzept von AutoMoMe zu implementieren. Eine weitere Voraussetzung ist, dass das Werkzeug zusätzlich eine Anbindung an Anforderungs- bzw. Konfigurationsmanagementwerkzeuge ermöglicht, um eine durchgängige Nachverfolgbarkeit herzustellen. Nach einer systematischen Werkzeugauswahl (vgl. [MDS08b]) ist die Wahl auf das UML Werkzeug IBM Rational Rhapsody [IBM14b] gefallen.

Das Werkzeug IBM Rational Rhapsody erfüllt die aufgestellten Anforderungen. So erlaubt es sowohl die Modellierung von SysML/UML als auch von AUTOSAR Modellen, die mittels eines AUTOSAR Profils erstellt werden [Bol09, SB09]. Des Weiteren besteht eine Anbindung an das Anforderungsmanagementwerkzeug IBM Rational DOORS [IBM14a], das im Automobilbereich sehr verbreitet ist. Hierdurch ist eine vollständige Nachverfolgbarkeit zwischen den Anforderungen und den Modellen gewährleistet. Ebenso ist eine

Anbindung an ein Konfigurationsmanagementsystem über die SourceControl (SCC) Schnittstelle vorhanden [WC01], so dass mehrere Entwickler parallel an einem Modell arbeiten und die Daten gleichzeitig versioniert und gesichert werden. Ferner besitzt das Werkzeug eine Schnittstelle, die durch eigene Funktionalität, den sogenannten Add-ons, erweitert werden kann. Die Schnittstelle kann durch die Programmiersprache Java [Ora14] bedient werden. Die Realisierung der Funktionen und Automatismen aus AutoMoMe wurden überwiegend durch eigene Java Programme durchgeführt, die diese Schnittstelle nutzen.

7.1.1 Die Realisierung des Architekturmodells

Das Konzept AutoMoMe basiert auf einen an den Automobilbereich angepassten Einsatz der SysML bzw. UML im Architekturmodell. Für die Realisierung der Anpassungen wird eine leichtgewichtige Erweiterung des UML Metamodells in Form von Stereotypen und Eigenschaftswerten vorgenommen [BH08, Sel07]. Hierdurch ist zum einen eine einfache und unkomplizierte Erweiterung möglich und zum anderen besteht in ferner Zukunft die Aussicht, ein anderes SysML/UML Werkzeug einsetzen zu können. Um die Erweiterungen in verschiedenen Projekten einsetzen zu können, werden sie in einem Profil zusammengefasst.

Das Profil ist in verschiedene Bereiche aufgeteilt, an denen Erweiterungen und Anpassungen vorgenommen werden (vgl. Abbildung 7.1). So gibt es einen Bereich für die funktionale, logische und technische Architektur. Ebenso sind Bereiche für die Codegenerierung und für die Erweiterungen zur Echtzeitsimulation vorhanden.

Exemplarisch für die Umsetzung der Erweiterungen durch Stereotypen und Eigenschaftswerten wird die Modellierung der Betriebssystemspezifikation erläutert. Hierzu sind zwei vom UML Aktivitätsdiagramm abgeleitete Diagramme (Task- und Allokationsdiagramm) entwickelt worden. In diesen Diagrammen wird zum einen die Taskzuordnung zu den Rechenkernen und zum anderen die Zuordnung der Funktionen auf die Tasks abgebildet. Die im Ansatz verwendeten Beispiele des Komfortsteuergerätes sind im Werkzeug Rhapsody entsprechend umgesetzt.

Die Einbindung der Basis-Software aus dem AUTOSAR Standard

Der AUTOSAR Standard sieht für einen Großteil der Basis-Software eine generelle Definition der Schnittstellen bereits vor [AUT10i]. In dem entsprechenden AUTOSAR Modell sind für die jeweiligen Module der Basis-Software die angebotenen und benötigten Schnittstellen, sowie deren Operationen, beschrieben. Diese Informationen werden als Startpunkt für die Modellierung in das Architekturmodell genutzt. Dazu werden Transformationen

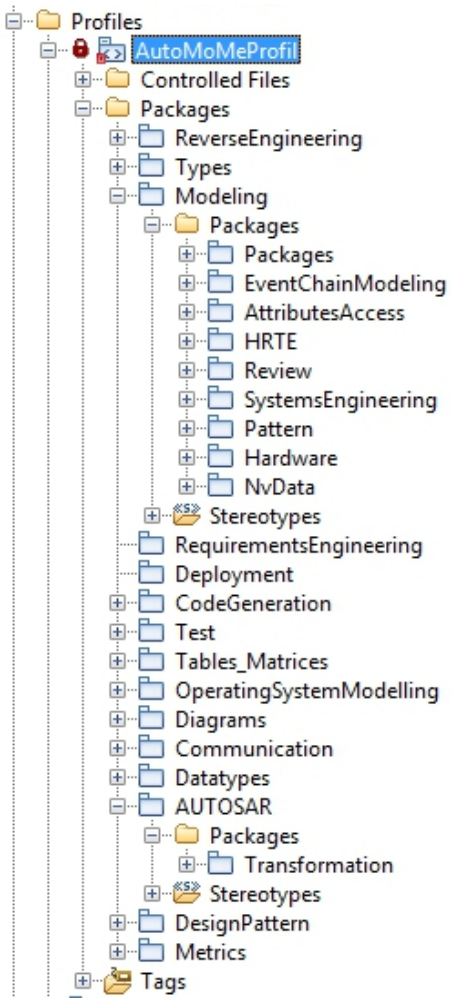


Abb. 7.1. Aufbau des entwickelten Profils

eingesetzt, um sie aus dem Sparx Enterprise Architect [Sys14] Modell in das Architekturmodell in IBM Rational Rhapsody zu überführen. Die überführten Komponenten, Ports und Schnittstellen sind jedoch nur als Startpunkte zu sehen, da viele der Ports noch als optional bzw. konfigurierbar gekennzeichnet sind. Je nach Konfiguration der Basis-Software bleiben die Ports bestehen oder werden überflüssig und können entfernt werden [FKKSP05]. In der Abbildung 7.2 ist der Import der Komponente ComManager für das Kommunikationssteuergerät dargestellt. Besonderes Kennzeichen der Importe ist, dass einige Ports mittels Stereotyp als «optional» bzw. «configurable» gekennzeichnet sind. Diese sind auch in der Abbildung wiederzuerkennen, da sie besonders farblich gekennzeichnet sind.

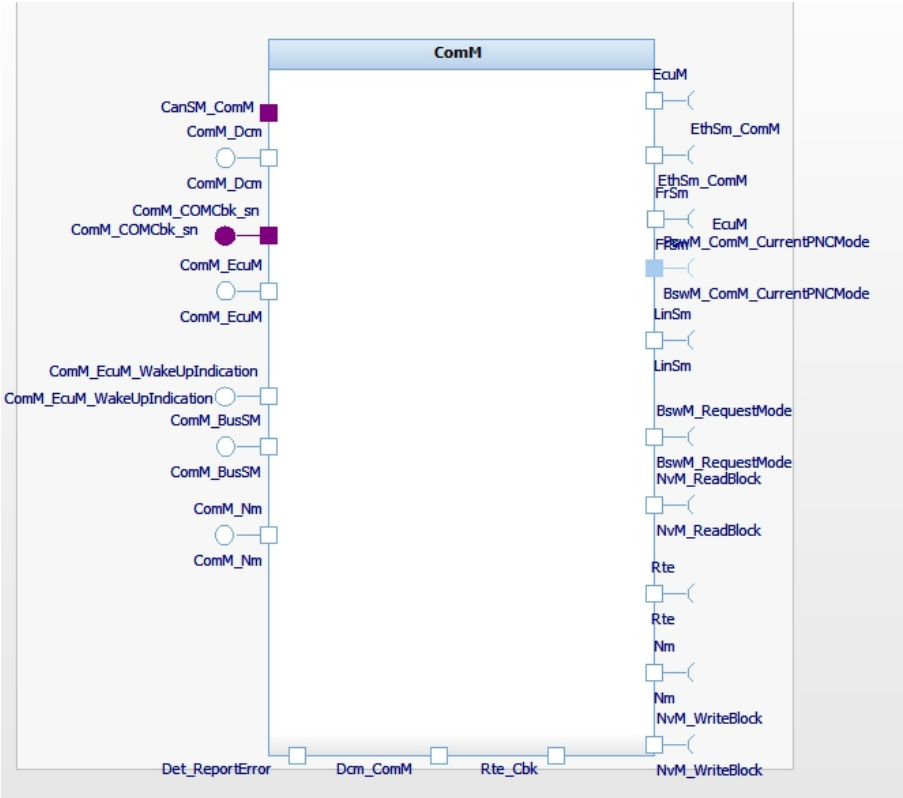


Abb. 7.2. Import der generellen Schnittstellenbeschreibung aus dem UML BSWMD Modell von AUTOSAR

Die Realisierung der AutoMoMe Unterstützung

Die Realisierung der AutoMoMe Unterstützung zeichnet sich nicht nur durch eine Erstellung bzw. Anpassung einer Modellierungssprache an die automobile Steuergeräteentwicklung aus. Vielmehr wird die Modellierung durch Automatismen unterstützt, so dass der Entwickler bei der Nutzung der Methodik entlastet wird. Beispielhaft kann hier auf die automatische Einbettung der ASIL Einstufung verwiesen werden (vgl. Kapitel 4.3.5). Für die Implementierung wird die Erweiterungsmöglichkeit des Werkzeugs IBM Rational Rhapsody mittels der Java API genutzt (siehe Abbildung 7.3). Gleichzeitig werden die Add-ons so in das bestehende Werkzeug integriert, dass für den Nutzer kein offensichtlicher Unterschied zu erkennen ist, ob nun eine bereits vorhandene oder eine AutoMoMe Funktionalität genutzt wird. Hierdurch wird die Nutzung durch den Entwickler vereinfacht, da eine durchgängige Oberfläche (Look & Feel) zur Verfügung steht.

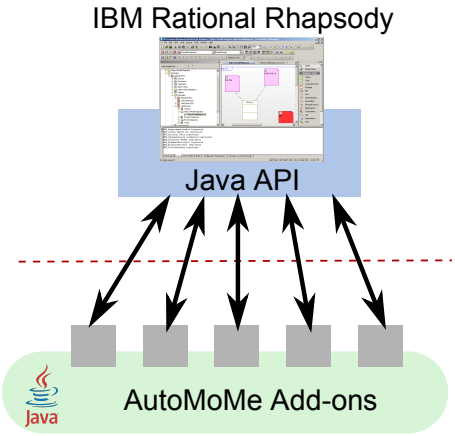


Abb. 7.3. Implementierung der Unterstützungsfunktionen im Ansatz AutoMoMe

Sicherstellung der Korrektheit und Vollständigkeit

Bei jeder Modellierung ist sicherzustellen, dass zum einen die Methodik ordnungsgemäß angewandt wird, und dass zum anderen die Modellierung auch vollständig ist. Man kann auch sagen, dass die Modellqualität hoch ist [FHR08]. Nur so ist sichergestellt, dass ein korrektes Abbild der realen Welt erfolgt. Oftmals wird die Korrektheit und Vollständigkeit eines Modells durch ein manuelles Review gewährleistet. Jedoch ist hiermit ein erhöhter Aufwand verbunden, da mehrere Entwickler verschiedene Diagramme und den Modellbaum überprüfen müssen. Um Ressourcen zu sparen, werden automatische Überprüfungen erstellt [MH13, Mey14]. Als Grundlage für die Erstellung der Überprüfungen wurde die Spezifikation der Modellierungsmethodik AutoMoMe eingesetzt (vgl. Kapitel Anhang A.2). Aus der Spezifikation ergibt sich, welche Elemente in welcher Anzahl im Architekturmodell auftreten dürfen. Hieraus lassen sich Überprüfungen ableiten. So kann beispielsweise eine Funktion nur einem Ziel und in der technischen Architektur eine Task nur einer CPU zugeordnet werden. Die Aufzählung kann noch beliebig fortgeführt werden.

Die Realisierung der Überprüfungen erfolgt durch die Java Programmiersprache und der Integration in das Werkzeug IBM Rational Rhapsody durch die Java API (vgl. Abbildung 7.3). Die Überprüfungen haben eine kurze Beschreibung und sind den entsprechenden Elementen zugeordnet. Darüber hinaus sind sie in drei verschiedene Kritikalitätsstufen (Information, Warnung und Fehler) eingeordnet. Hierdurch wird der Entwickler unterstützt. So ist zu erkennen, ob etwas sofort oder zu einem späteren Zeitpunkt geändert werden muss. In Abbildung 7.4 sind die implementierten Überprüfungen im Werkzeug Rhapsody dargestellt.

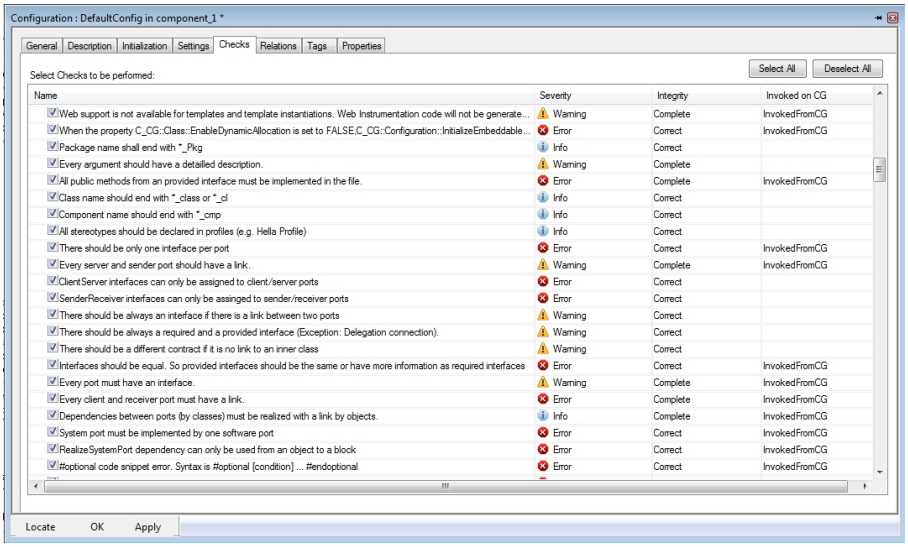


Abb. 7.4. Realisierte Überprüfungen der AutoMoMe im Werkzeug IBM Rational Rhapsody

Die Integration in das Projektmanagement

Neben der Erstellung von automatischen Unterstützungen und Überprüfungen ist die Integration in das Projektmanagement ein weiterer wichtiger Bestandteil für die Implementierung der Methodik. Dies bedeutet, dass es für das Projektmanagement immer von großer Bedeutung ist, zu wissen, wo das Projekt steht und wie viele Aufgaben bereits abgearbeitet sind. Dies betrifft natürlich ebenso die Architekturmodellierung mittels der AutoMoMe. Damit die Methodik dieser Anforderung genügt und eine Integration in das Projektmanagement vorhanden ist, werden innerhalb AutoMoMe einige Metriken entwickelt. Die Metriken geben natürlich nur einen Hinweis und einen Anhaltspunkt, in welchem Stadium die Architekturmodellierung sich befindet und müssen dementsprechend interpretiert werden [Sta09a]. Sie geben dem Projektmanager einen ersten Hinweis und er kann dann die entsprechenden Nachfragen an das Entwicklungsteam richten. So sind im Rahmen der AutoMoMe Metriken für die Fertigstellung von Elementen, die Verlinkung der Anforderungen, das Setzen der ASIL Einstufung, das Verlinken von Ports, die Realisierung von Anforderungen etc. entwickelt worden, um nur einige Beispiele zu nennen. In der Abbildung 7.5 ist exemplarisch die Metrik für die Verlinkung von Anforderungen dargestellt. Es ist zu erkennen, dass für das Komfortsteuergerät bereits 394 Anforderungen mit der Architektur verbunden sind.

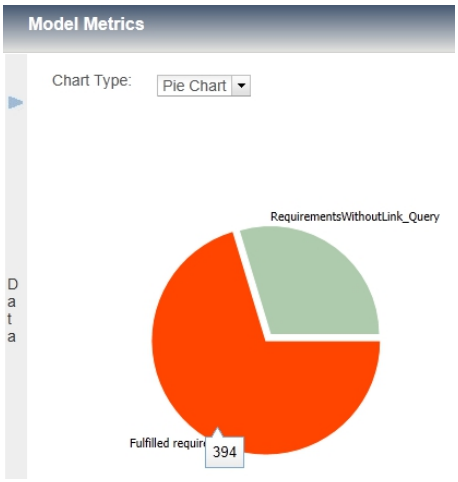


Abb. 7.5. Metrik zum Abgleich von erfüllten und nicht erfüllten Anforderungen

7.1.2 Der SignalManager

Zur Realisierung des SignalManagers, d. h. für den Import der Kommunikationssignale in das Architekturmodell, werden neben der Bereitstellung der Stereotypen und Eigenschaftswerten zur Aufnahme der Kommunikationsdaten noch zusätzliche Einlesemethoden entwickelt. Damit diese möglichst unkompliziert und schlank sind, wird ein eigenes Metamodell konzipiert, so dass der Übergang zu den UML-Elementen einheitlich erfolgen kann, unabhängig davon, welche Kommunikationsdaten vorliegen. Das Metamodell ist im Anhang (vgl. Abbildung Anhang A.2) zu finden und wird mit dem Eclipse Modeling Framework (EMF) [Fou14] realisiert. Der Vorteil des EMF Modells besteht darin, dass viele Funktionen wie das Laden/Speichern von Modellen oder eine Benachrichtigungsfunktion bereits existieren und genutzt werden konnten. Das EMF Modell des SignalManager richtet sich dabei hauptsächlich nach dem Fibex Standard [fSoAA13], der auch beim AUTOSAR Standard zum Einsatz kommt. In der nachfolgenden Abbildung (siehe Abbildung 7.6) ist der Import der Kommunikationsmatrix (CAN-Bus) für das Komfortsteuergerät im Werkzeug Rhapsody dargestellt. Bei dem Import handelt es sich um die verschiedenen Signalgruppen mit deren Signalen und den dazugehörigen Werten. Diese sind dabei im Dialogfenster zu sehen.

7.1.3 Der OIL-Generator

Der Export der Betriebssystemeigenschaften wird durch das openArchitectureWare (OAW) Framework realisiert [Ope09, VG07]. Durch die Nutzung dieses Frameworks ist es möglich, eine Modell-zu-Text (M2T) Transformation

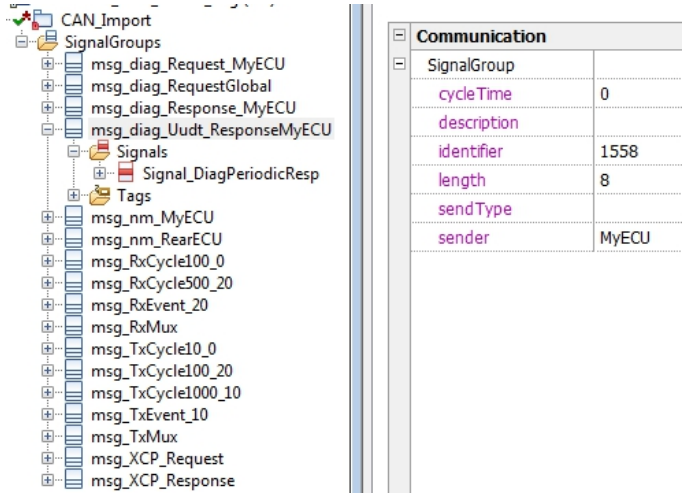


Abb. 7.6. Import der Daten aus der Kommunikationsmatrix des Komfortsteuergerätes

durchzuführen [GPR06]. So werden zunächst aus dem UML Modell die Daten in ein entsprechendes EMF Modell übertragen (das Metamodell befindet sich im Anhang im Kapitel A.3). Die Anbindung des EMF Modells an das Werkzeug Rhapsody erfolgt über die vorgegebene Java Schnittstelle. Für die einzelnen Elemente des EMF Modells bestehen entsprechende Templates aus dem OAW Framework, die die Daten entsprechend generieren. Exemplarisch ist nachfolgend das Template für die Erstellung einer Task abgebildet.

```
«DEFINE Task FOR OSEK_Manager::Task»
        TASK «name» { // *** Defines a task ***
                TYPE = «type»;
                PRIORITY = «priority»;
                SCHEDULE = «schedule»;
                ACTIVATION = «activation»;
                AUTOSTART = «autostart»;
                STACKSIZE = «stackSize»;
                SCHEDULE_CALL = «scheduleCall»;
        };
«ENDDEFINE»
```

7.1.4 Die Anbindung an die Echtzeitsimulation

Die Anbindung an die Echtzeitsimulation wird ähnlich dem OSEK Implementation Language (OIL)-Generator realisiert. Es wird ebenfalls das OAW

Framework eingesetzt, um eine Modell-zu-Text Transformation zu realisieren. Die Daten aus dem Rhapsody Modell werden mittels der Java API und durch die entsprechend entwickelten Funktionen in ein EMF Modell übertragen (das Metamodell ist im Anhang A.1.3 zu finden). Aus diesem werden dann die Dateien für das Inchron Simulationsmodell erzeugt (vgl. Abbildung 7.7). Ein Teil der entsprechenden Textmuster aus dem OAW Framework sind im Anhang zu finden. Bei der Transformation wird eine auf Extensible Markup Language (XML) basierte Projektdatei und entsprechende *.c und *.h Dateien für die einzelnen Komponenten erzeugt. Diese werden dann in das Inchron Werkzeug geladen und für die Simulation vorgehalten.

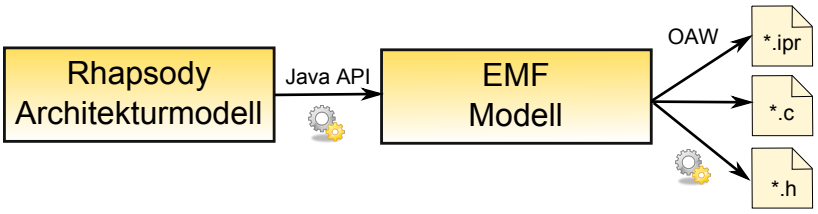


Abb. 7.7. Technische Umsetzung der Anbindung an die Echtzeitsimulation

7.1.5 Der Übergang nach AUTOSAR

Für den Übergang von UML/SysML nach AUTOSAR wird auf die AUTOSAR Profile des Werkzeugs IBM Rational Rhapsody zurückgegriffen [Bol09]. Die AUTOSAR Profile dieses Werkzeugs bilden die verschiedenen Varianten des AUTOSAR Metamodells ab (zum Beispiel V4.0 [AUT10b]). Hierdurch ist es möglich, das AUTOSAR Profil zu nutzen, welches für die Projekte vom Automobilhersteller vorgeschrieben ist. Damit die Aufrufe der Transformationsregeln identisch sind und es für die Transformationen keine Rolle spielt, welche AUTOSAR Version zur Anwendung kommt, wird das Design-Muster *Strategy Pattern* [GHJV04] eingesetzt (vgl. Abbildung 7.8). So werden die Transformationen für die verschiedenen AUTOSAR Versionen unabhängig voneinander erstellt. Lediglich wenn Abweichungen zwischen den einzelnen AUTOSAR Versionen vorhanden sind, werden notwendige Anpassungen bei den Transformationsregeln vorgenommen. Vor allem im Bereich der Datentypen bestehen Unterschiede zwischen den AUTOSAR Versionen 3.x und 4.x, . Hierfür werden eigene Transformationsregeln geschaffen. Erst während des Transformationsablaufs wird anhand der Einstellungen im Architekturmodell entschieden, welche Transformation letztendlich zum Einsatz gelangt.

Die Definitionen der einzelnen Regeln werden formal spezifiziert (vgl. Kapitel 6). Die Regeln sind in der Programmiersprache Java realisiert. Die Wahl fiel auf diese Sprache, da das verwendete Werkzeug IBM Rational Rhapsody eine Java

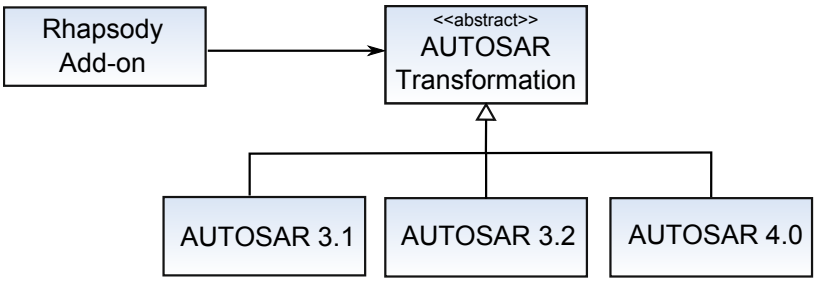


Abb. 7.8. Aufbau der AUTOSAR Transformation

API aufweist, um zusätzliche eigene Programme auszuführen. Das von IBM Rational Rhapsody verwendete SysML bzw. UML Metamodell nur in einem proprietären Format vor, so dass keine Transformationssprachen wie Query View Transformation (QVT) [Gro11a] oder Triple Graph Grammatiken (TGG) [KW07, GPR11] eingesetzt werden kann. Ebenso sind für manche Übergänge von UML nach AUTOSAR – insbesondere im Bereich der Basis-Software – Informationen von verschiedenen Stellen im Architekturmodell zu übernehmen, so dass an dieser Stelle eher von einem Synthese- als von einem Transformationsschritt auszugehen ist.

7.2 Praktischer Einsatz der Arbeit

Der Nachweis der Durchführbarkeit des Ansatzes AutoMoMe wird anhand des durchgängigen Fallbeispiels *Komfortsteuergerät* in der Arbeit erbracht. Es hat sich gezeigt, dass mit dem erarbeiteten Ansatz eine durchgängige Modellierung von der funktionalen Architektur bis hin zur AUTOSAR Modellierung möglich ist. Aber nicht nur die durchgängige Modellierung ist ein Vorteil von AutoMoMe, sondern ebenso die Speicherung der Entwicklungsinformationen in einem zentralen Modell in einer (semi-) formalen Sprache. Hierdurch sind die Informationen in einer Weise verfügbar, dass sie bei Automatisierungen – beispielsweise zur Erstellung eines Echtzeitsimulationsmodells – genutzt werden. Hierdurch ist eine gleichbleibende, da standardisierte und wiederholbare, hohe Qualität der Entwicklungsartefakte gegeben, und dies bei gleichzeitiger Verkürzung der Entwicklungszeit.

Der Ansatz AutoMoMe wurde speziell für den Einsatz in der automobilen Steuergeräteentwicklung entworfen. Insofern bleibt zu klären, inwieweit die Methodik im industriellen Bereich praktikabel und einsatzfähig ist. Vorteilhaft war an dieser Stelle, dass die Arbeit in enger Zusammenarbeit mit einem international agierenden automobilen Zulieferer entwickelt wurde. Von daher konnten die erarbeiteten Lösungen bereits frühzeitig in industriellen Serienprojekten erprobt und evaluiert werden. Es wurden für die einzelnen Teilbereiche der Methodik AutoMoMe Pilotprojekte ausgewählt, in denen die Methodik seriennah eingesetzt werden konnte. Hierbei wurden Projekte aus dem Komfort-, Energie- als auch aus dem Fahrerassistenzbereich herangezogen. Mit dieser Vorgehensweise wurde der Ansatz AutoMoMe Stück für Stück etabliert, bis er vollständig genutzt werden konnte. Erkenntnisse aus dem praktischen Einsatz wurden dabei zügig in die Methodik eingearbeitet, so dass ein hoher praktischer Nutzen erkennbar ist.

Bei der Einführung in die Serienprojekte hat sich gezeigt, dass die Erfolgskurve vergleichbar ist mit einem theoretischen Modell aus der Psychologie/Familien-Therapie namens Satir-Change [SGG91, Smi13]. Dieses Modell gliedert sich in fünf Phasen. Zunächst befindet sich das Projekt im Status Quo. Alle Arbeitsschritte werden mit den vorhandenen und bekannten Methoden und Techniken bearbeitet. Aus Erfahrung ist jedoch allen am Projekt Beteiligten bewusst, dass es noch Verbesserungen und Anpassungsbedarf gibt. Solche Verbesserungsmöglichkeiten bietet der Ansatz AutoMoMe.

Um die Methodik mit den darin befindlichen Techniken einsetzen zu können, ist zunächst eine intensive Einarbeitung inklusive Schulung notwendig. Danach ist die Performance nicht mehr so hoch wie zuvor. Die Einführung von neuen Methoden wird daher skeptisch gesehen. Dies ist in der zweiten Phase durchaus feststellbar und sie trägt daher passenderweise den Namen Widerstandsphase. In der daran anschließenden Phase, die in dem Modell auch als Chaosphase bezeichnet wird, sind vor allem Überzeugungsfähigkeiten gefragt. Die Projek-

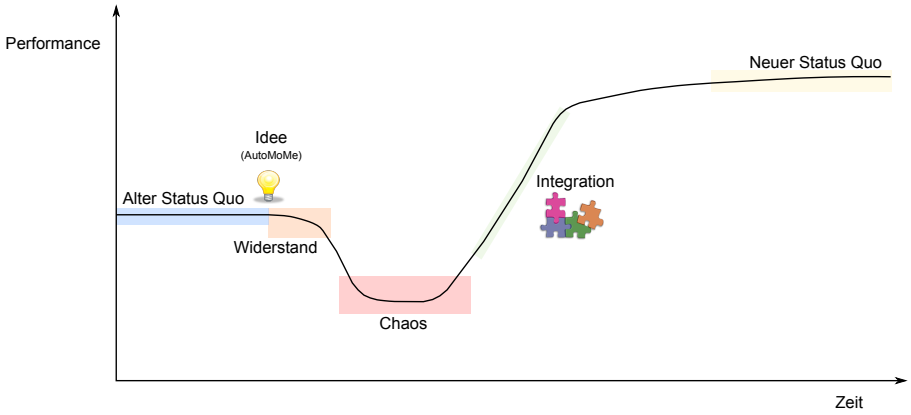


Abb. 7.9. Satir-Change Modell [SGG91] beschreibt die Einführung der AutoMoMe

tmitglieder sind auf Grund der neuen Techniken und Vorgehensweise verunsichert und haben oft Angst diese anzuwenden. Dies war ebenfalls beim Einsatz des Konzeptes AutoMoMe der Fall. So taten die Beteiligten sich schwer, die unterschiedlichen Diagramme und Elemente der AutoMoMe den einzelnen Entwicklungsphasen und den darin befindlichen Aufgaben zuzuordnen. Um diese Phase möglichst kurz zu halten und die Mitglieder auch von den entwickelten neuen Methoden und Werkzeugen zu überzeugen, wurden die jeweiligen Projekte intensiv begleitet und es fand eine dauernde Beratung und Nachschulung statt. So erhielten die Entwickler einen Gesamtüberblick über den Ansatz AutoMoMe und waren nach kurzer Zeit in der Lage, diesen Ansatz problemlos und vollständig umzusetzen. Die in AutoMoMe entwickelten neuen Ansätze im Bereich der Durchgängigkeit, Eindeutigkeit und der automatischen Überprüfungen überzeugten die Entwickler letztendlich, dass die Methodik einen Mehrwert erbringt. Daher war es leicht, in die Integrationsphase überzugehen. In dieser Phase war die Methodik des Ansatzes AutoMoMe verinnerlicht und wurde positiv umgesetzt. Es konnte ein deutlich verbesserter neuer Status Quo erreicht werden. Die Projekte wurden zielorientierter und effizienter bearbeitet. Hierdurch trat der Effekt ein, dass Zeit und Kosten eingespart wurden.

Neben der Einführung in Serienprojekten sind Ergebnisse auch in das nationale Forschungsprojekt Software Plattform Embedded Systems 2020 (SPES2020) [SPE09, PHAB12] für die Entwicklung zukünftiger eingebetteter Systeme eingeflossen und bereits zum Großteil veröffentlicht. Auf Grund dieser Tatsache kann gesagt werden, dass die Ergebnisse nicht nur innerhalb des automobilen Zulieferbetriebes akzeptiert wurden, sondern ebenso in der Forschung angekommen sind.

Was nicht auf einer einzigen Manuskriptseite zusammengefasst werden kann, ist weder durchdacht noch entscheidungsreif.

Dwight D. Eisenhower (1890 - 1969)

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Diese Dissertation beschreibt eine durchgängige und modellbasierte Entwicklungsmethode für die automobile Steuergeräteentwicklung aus der Sicht eines Automobilzulieferers. Kurze Entwicklungszeiten einhergehend mit Kostenreduzierungen inklusive Qualitätsverbesserungen durch eine durchgängige Modellierung stehen dabei im Fokus. Die Verbesserungen werden durch Automatismen erreicht. Hierzu werden durch das Konzept AutoMoMe bisherige manuelle zeit- und ressourcenverbrauchende Entwicklungsschritte automatisiert bzw. teilautomatisiert. Die Arbeit kann in die drei Bereiche unterteilt werden: *Architekturmodellierung inklusive notwendiger Formalisierungen*, *Einbettung der Echtzeitanalyse/-simulation in den Entwicklungsprozess* und (teil-) automatisierter *Übergang nach AUTOSAR*.

In den einleitenden Kapiteln dieser Arbeit werden die bestehenden Schwachstellen der derzeit gängigen Entwicklungsmethodik bei der automobilen Steuergeräteentwicklung herausgearbeitet (vgl. Kapitel 1.1). Dies ist insbesondere die Verwendung von unterschiedlichen und zum Teil nicht formalisierten Notationsformen. Mit der Folge, dass das Datenmaterial nicht automatisierbar gespeichert werden kann und für die nächsten Schritte im Entwicklungszyklus nur bedingt und lediglich mit zusätzlichem Aufwand nutzbar ist.

Durch die Weiterverwendung der Daten in den jeweiligen Entwicklungsphasen und verstärkt durch die Einbindung von dezentralen Entwicklungen ist nicht auszuschließen, dass die Daten redundant und an den unterschiedlichsten Stellen abgelegt werden. Häufige Aktualisierungen sind die Folge und erfordern eine Anpassung der Entwicklungsartefakte. Durch redundante Speicherungen kommt es deshalb häufig zu Inkonsistenzen, die unvermeidlich zu Mehraufwand und sogar zu Fehlern im Entwicklungsprozess führen. Von daher gilt es, Redundanzen weitestgehend zu vermeiden.

Ein weiteres Defizit, das sich aus der Verwendung unterschiedlicher Techniken ergibt, sind im gegenwärtigen Entwicklungsprozess die Brüche zwischen den einzelnen Entwicklungsphasen. Diese entstehen durch unterschiedliche Notationsformen und werden durch die Verwendung von nicht integrierten Werkzeugen noch verstärkt. Hierdurch ist im Entwicklungsprozess eine durchgängige Entwicklungsmethodik nicht gegeben. Die Brüche sind dafür verantwortlich, dass zusätzlicher Arbeitsaufwand in die Erstellung der Nachvollziehbarkeit zwischen den einzelnen Entwicklungsphasen anfällt. Gerade bei der Entwicklung von sicherheitskritischen Systemen ist die Sicherstellung der Nachverfolgbarkeit und Durchgängigkeit ein außerordentlich wichtiger Aspekt.

Die Modellierung von zeitlichen Informationen, vor allem bei komplexen Steuergeräten, ist als Schwachpunkt in der heute gängigen Entwicklung anzusehen. Diese zeitlichen Informationen sind aber notwendig, um das Zusammenspiel der komplexen Funktionen korrekt zu spezifizieren und später dementsprechend zu implementieren. Der Formalisierung ist somit oberste Priorität zuzuordnen. Denn nur die Formalisierung erlaubt eine automatische Weiterverarbeitung der Daten. Gleichzeitig sind diese dann bereits frühzeitig für eine Analyse und Verifikation einsetzbar. Eine Integration in die Entwicklungsmethodik ist daher unerlässlich, so dass jederzeit die Durchgängigkeit gewahrt wird.

Zusätzlich wird die jetzige Entwicklungsmethodik noch erschwert durch den Einsatz des AUTOSAR Standards, da dieser keine Möglichkeit zur Analyse oder zur Spezifikation von dynamischen Verhalten aufweist. Der AUTOSAR Standard ist vorrangig für die Entwicklung und Implementierung von Software ausgelegt. Auf diese Art und Weise ist die Erstellung einer logischen Architektur für das komplette Steuergerät nur zum Teil abgedeckt (vgl. Kapitel 2.7). Eine vollständige, durchgängige und nicht unterbrochene Entwicklung ist aber zwingend notwendig.

Die genannten Problem- und Schwachstellen sowie weitere Defizite der gegenwärtig eingesetzten Entwicklungsmethodik werden aufgegriffen, um eine Lösung in Form des Ansatzes AutoMoMe anzubieten. Die entwickelte Methodik schafft durch konstruktive Art und Weise eine korrekte, transparente, nachvollziehbare und verbesserte automobile Steuergeräteentwicklung.

8.1.1 Formalisierung und Architekturmodellierung

Das Kernstück der erarbeiteten Entwicklungsmethode AutoMoMe bildet das Architekturmodell. Das Architekturmodell wird durch die Sprachen SysML und UML spezifiziert, wobei zusätzliche domänen-spezifische Anpassungen an die automobile Steuergeräteentwicklung vorgenommen sind (vgl. Kapitel 4.3). Die Verwendung dieser Sprachen erlaubt eine eindeutige und konsistente Modellierung. Ferner werden die Sprachen SysML und UML an die Bedürfnisse der

automobilen Zuliefererindustrie angepasst, um einen verbesserten Umgang zu gewährleisten [TK05]. Hierzu kommen Techniken und Notationsformen der Mechatronik UML zur Anwendung (vgl. Kapitel 2.4).

Zunächst wird die Aufteilung in eine funktionale, logische und technische Architektur innerhalb des Ansatzes AutoMoMe herausgearbeitet (vgl. Kapitel 4.3). Zur Modellierung der funktionalen Architektur kommen Aktivitätsdiagramme zum Einsatz, die die Funktionen und ihre Abhängigkeiten untereinander explizit beschreiben. Durch die funktionale Architektur ist es bereits in einer frühen Entwicklungsphase möglich, Wirkketten zu spezifizieren (vgl. Kapitel 4.3.3). Diese sind in der Architekturbewertung von entscheidender Bedeutung.

Darüber hinaus wird eine Technik aufgezeigt, um Kommunikationssignale im Architekturmodell zu modellieren, die als Eingabe von den OEMs zur Verfügung gestellt werden und somit in die Entwicklung zu integrieren sind (vgl. Kapitel 4.3.5). Neben der Entwicklung der notwendigen Spezifikationsformen werden automatische Import- und Exportmechanismen entwickelt, so dass bei Änderung der Kommunikationssignale eine Aktualisierung jederzeit erfolgen kann.

Zusätzlich wird das Architekturmodell um die Spezifikation des Echtzeitverhaltens in Form der Real-Time Statecharts ergänzt. Hierdurch ist die Modellierung von Echtzeitinformationen im Architekturmodell darstellbar und die zeitlichen Informationen und Abhängigkeiten können untereinander abgebildet werden (vgl. Kapitel 4.3.2).

Das Architekturmodell wird außerdem um eine formale Spezifikation von Betriebssystemeigenschaften erweitert. Hierzu werden entsprechende UML-Notationen entwickelt (vgl. Kapitel 4.3.7). Diese Formalisierung ermöglicht die Spezifikation des Schedulings und ist von daher unverzichtbar für eine vollständige Steuergeräteentwicklung.

Die vorgenommenen Formalisierungen bilden die Grundlage, dass das Architekturmodell alle notwendigen Informationen für die Steuergeräteentwicklung enthält und Grundlage für die nachfolgend entwickelten Techniken ist.

8.1.2 Integration der Echtzeitanalyse/-simulation

Ein Vorteil der Modellierung eines Architekturmodells in (semi-)formaler Weise ist die Wiederverwendung der spezifizierten Daten. Im Ansatz AutoMoMe wird eine Technik für die automatisierte Wiederverwendung der Daten in einem Systemsimulationsmodell aufgezeigt. Das Simulationsmodell ermöglicht die Verifikation der Architektur vor allem in zeitlicher und performancetechnischer Sicht. Die erarbeitete Technik wird in den Prozess integriert, so dass eine durchgängige Entwicklung geschaffen wird (vgl. Kapitel 5).

In dem Echtzeitsimulationsmodell werden die getroffenen Architekturdesignentscheidungen bereits frühzeitig gegen die gestellten Anforderungen überprüft. Schwerpunkt ist hierbei die Verifikation der Ausführungszeiten und die Auslastung der Bus- und Rechenkerne. Es wird der Nachweis erbracht, dass die Daten des Architekturmodells für die Erstellung der Echtzeitsimulation genutzt, die Ergebnisse aus der Simulation ausgewertet und in das Modell zurück übertragen werden.

8.1.3 Übergang zum AUTOSAR Standard

Die Wiederverwendung der Daten innerhalb des Ansatzes AutoMoMe ist durch den (teil-) automatisierten Übergang von der SysML/UML Architektur zum AUTOSAR Standard abgedeckt. In der Arbeit wird ein bestehender Ansatz [GHN09] aufgegriffen aber dahingehend erweitert, dass er im industriellen Umfeld eingesetzt werden kann. Hierzu werden zusätzliche Modelltransformationsregeln erstellt, die einen Übergang von UML nach AUTOSAR zulassen (vgl. Kapitel 6). Die bisherige Beschränkung auf die Struktur des Systems wird somit beseitigt, so dass das Kommunikationsverhalten mitberücksichtigt wird. Außerdem werden Teile der Basis-Software bei der Transformation einbezogen. Die Anwendung dieser Regeln sorgt für einen automatisierten und durchgängigen Entwicklungsprozess. Hierdurch wird die bis dahin bestehende Lücke zwischen der Analysephase und dem Design bzw. der Implementierung mit AUTOSAR geschlossen.

Es wird in AutoMoMe eine durchgängige und modellbasierte Entwicklungsmethode für automobile Steuergeräte vorgestellt, die von den Anforderungen bis hin zur Implementierung reicht, so dass die komplette linke Seite des V-Modells abgedeckt wird.

8.2 Praktische Relevanz

Mit den derzeitig bestehenden Schwachstellen und den zusätzlichen Anforderungen an eine zukünftige durchgängige Entwicklungsmethodik haben sich zahlreiche Autoren in wissenschaftlichen Arbeiten auseinandergesetzt [BS05, Bro06]. Diese Arbeit und das Konzept AutoMoMe setzt sich nicht nur mit den derzeitigen Entwicklungsmethoden aus wissenschaftlicher Sicht auseinander, sondern es werden auch weitestgehend industrielle Vorgaben und Erfordernisse einbezogen. Von daher ist es nicht vermessen zu sagen, dass für die aufgezeigten Probleme und Schwachstellen die erarbeiteten Lösungen eine hohe praktische Relevanz aufweisen.

Die praktische Relevanz dieser Arbeit zeigt sich letztendlich auch darin, dass auf Grund der Zusammenarbeit mit einem automobilen Zulieferer ein Großteil

der in dieser Dissertation vorgestellten Methoden und Techniken bereits praxisnah in die Serienentwicklung von Steuergeräten eingeflossen sind und dort mit Erfolg eingesetzt werden. Die hier erarbeiteten Techniken kommen dabei in allen Bereichen der Steuergeräteentwicklung zum Einsatz, angefangen von komplexen Fahrerassistenzsystemen bis hin zu Komfortsteuergeräten. Beim praktischen Einsatz im industriellen Umfeld haben sich die entwickelten Methoden und Techniken bewährt (vgl. Kapitel 7.2). Vor allem durch die Bereitstellung der Automatismen und die damit verbundenen Einsparungen von Ressourcen hat sich die Akzeptanz der neuen Methodik noch erhöht. Besonders die Modellierung der Betriebssystemeigenschaften, die Modellierung und der Import von Bussignalen sowie der Übergang zum Systemmodell werden bereits praktisch genutzt. Ferner ist die Methodik AutoMoMe in das Forschungsprojekt SPES2020 [SPE09, PHAB12] eingeflossen, was ebenfalls die praktische Relevanz verdeutlicht.

8.3 Ausblick

Durch die in AutoMoMe entwickelten Methoden und Techniken ist Vorsorge getroffen, um zukünftigen Entwicklungen mit noch kürzeren Entwicklungszeiten und weiterer Zunahme der Komplexität der Steuergeräte zu begegnen. Hier kann auf die zunehmende Vernetzung der Automobile und ihrer Umgebung hingewiesen werden, die derzeit unter den Begriffen Industrie 4.0 [Bro10, BBB+13], Cyber Physical Systems (CPS) [Lee08] und Internet der Dinge (Internet of Things) (IoT) [Krä14] erforscht werden.

Hierdurch steigt die Vernetzung der Systeme und damit einhergehend die Komplexität eines jeden Steuergerätes. Eine Bewertung, wie erfolgreich diese Arbeit die zukünftigen Herausforderungen besteht, kann nur durch eine Evaluation des Ansatzes AutoMoMe bei der Entwicklung von neuartigen Systemen herausgefunden werden. Eine Evaluation bedingt aber einen großen Aufwand, um verwert- und belastbare Aussagen zu bekommen. So muss zunächst eine geeignete These gefunden werden, die durch passende Datenerhebungen und Auswertungsmethoden analysiert wird [BD06, Yin09, PS02]. Eine Möglichkeit kann beispielsweise ein Experiment mit vergleichbaren Teams sein, wobei ein Team nach den herkömmlichen Methoden und Techniken arbeitet und das andere nach denen von AutoMoMe. Bei einer vergleichbaren Aufgabenstellung ist dann erkennbar, welcher Ansatz effizienter und schneller zum Ziel führt. Eine andere Möglichkeit zur Evaluierung besteht in der Befragung von Experten, die den Ansatz auf Grund ihrer Erfahrung bewerten und beurteilen. Hierzu ist aber eine große Gruppe von Experten notwendig, um valide Aussagen treffen zu können. Auf Grund des damit verbundenen Aufwandes konnte eine Evaluierung in dieser Arbeit nicht mehr durchgeführt werden. Sie kann in einer weitergehenden Arbeit erfolgen.

Die Evaluation des Ansatzes ist nicht der einzige Ansatzpunkt für weitergehende Arbeiten, da in dem AutoMoMe Ansatz nicht alle Problemstellungen für die automobile Steuergeräteentwicklung betrachtet werden konnten. Aufgrund dessen werden im Folgenden Anknüpfungspunkte zur Lösung von weiteren Problemstellungen vorgestellt.

Die erarbeiteten Techniken und Methoden sind auf die Entwicklung von automobilen Steuergeräten abgestimmt. Aber auch andere industrielle Domänen profitieren von der Formalisierung und dem Einsatz von Automatismen. Bei besonders eng verwandten Branchen sind die gleichen Techniken ohne größere Anpassungen einsetzbar [SBG+10, PBK10]. So kann die Echtzeitsimulation und deren Übergang zum Architekturmodell ebenso in der Avionik eingesetzt werden. Darüber hinaus ist der Übergang von dem Analysemodell zu einem in der Domäne verwendeten Standard durch vergleichbare Transformationsregeln, wie die hier eingesetzten, umzusetzen. Bei der Avionik ist dies natürlich nicht der AUTOSAR Standard, sondern die Architecture Analysis & Design Language Architecture Analysis and Design Language (AADL) [FG12].

Für das AUTOSAR Modell ist die Codierung freigestellt, d. h. es sind unterschiedliche Implementierungen möglich. Oftmals werden Implementierungen in Form von Matlab/Simulink bzw. TargetLink Modellen verwendet. Eine andere häufig anzutreffende Variante ist die manuelle Implementierung. Auf Grund der Namensgebung im AUTOSAR Standard ist letztere jedoch fehleranfällig und kompliziert. Von daher kann auf das Feindesign mit UML zurückgegriffen werden, um daraus entsprechenden Code zu generieren, der in einem nächsten Schritt dann über den Code Eintrag in das AUTOSAR Modell zurück gespielt wird. Somit ist in Zukunft mit diesem Ansatz eine weitergehende Möglichkeit zur Integration von UML und AUTOSAR gegeben.

Die in AutoMoMe entwickelten Methoden und Techniken beschränken sich dabei ausschließlich auf den Softwareentwicklungsprozess. In dieser Arbeit wird jedoch nicht der Übergang auf die Hardwarearchitektur oder die Mechanik mit der geometrischen Anordnung der einzelnen Bauteile betrachtet. Dies bleibt weiteren Forschungen vorbehalten. Erste Ansätze sind im Projekt SPES 2020 [SPE09] und dem Sonderforschungsbereich 614 [SFB09]vorhanden.

Letztendlich ist auch eine stärkere Einbeziehung des Anforderungsmanagements anzustreben. Im bisherigen Ansatz wird das Anforderungsmanagement nur in der Form betrachtet, dass die Anforderungen mit den Elementen des Architekturmodells verbunden werden. Hierdurch wird zum einen deutlich, wie viele Anforderungen bereits umgesetzt sind und zum anderen ist die Nachverfolgbarkeit gegeben, d. h. es kann nachvollzogen werden, welches Modellelement welche Anforderungen realisiert. Die Erstellung der Verbindungen zwischen den Anforderungen und den Modellelementen erfolgt jedoch noch manuell, so dass jede Verbindung seperat erstellt werden muss. Dies doch aufwendige Prozedere kann durch geeignete Automatismen verkürzt und verbessert werden. So kann beispielsweise durch den Einsatz von Satzmustern ein er-

stes Systemanalysemodell aus den textuellen Anforderungen, einschließlich der notwendigen Verbindungen, automatisiert erstellt werden [Hol10]. Hierdurch werden die vorgestellten Techniken erweitert und darüber hinaus wird eine Verbindung zu den Anforderungsphasen (ENG.1 und ENG.2 von Automotive SPICE [dAV10]) hergestellt. Ein erster Prototyp wurde bereits mit der Verwendung der AutoMoMe Methodik für die Architekturentwicklung in [SPE09] entwickelt.

Aber nicht nur die Einbeziehung der Anforderungsphasen ist in zukünftigen aufbauenden Arbeiten zu behandeln. Auch gilt es, die Testphasen mit einzubeziehen. So werden bereits in AutoMoMe die Testfälle mit den Modellelementen des Architekturmodells verbunden. Dies ist ähnlich zu sehen, wie die Verbindung der Anforderungen, d. h. die Nachverfolgbarkeit wird auf manuelle Art und Weise hergestellt. Die Prozessanforderungen werden hierdurch zwar erfüllt. Gleichwohl ist eine Verbesserung durchaus denkbar. Als Ausgangspunkt kann hierfür das (semi-) formale Architekturmodell herangezogen werden. Es bestehen bereits Ansätze, um aus einem (semi-) formalen Modell automatisiert Testfälle zu generieren [Sch11, BTC13, SA07]. Diese Ansätze und Verfahren sind dann so anzupassen, dass sie für das Architekturmodell nutzbar sind. Dies bietet den Vorteil, dass der bisherige Aufwand für die Testfallerstellung reduziert werden kann. Die Anpassung der Ansätze ist Aufgabe der weiterführenden Forschungsarbeiten.

Eine Aufwandsreduzierung kann ergänzend noch bei der Architekturmodellierung erfolgen. Teile der Architektur werden so oder in abgewandelter Fassung in vielen automobilen Steuergeräten eingesetzt, so bei der Diagnose oder dem Speichermanagement. Von daher ist die Wiederverwendung ein wichtiges Thema. In AutoMoMe ist bereits ein manueller Ansatz bezüglich der Wiederverwendung von Teilsystemen enthalten (siehe Kapitel4.3.1). In Zukunft ist dieser Ansatz so auszubauen, dass automatisiert entsprechende Teilsysteme identifiziert und notwendige Transformationen erstellt werden. Erforderlich ist hierfür eine Bibliothek mit bereits realisierten Teilsystemen aus denen das zu lösende System gefunden wird [SKFS07, BdALM08]. Hierdurch kann die Wiederverwendung noch besser unterstützt werden.

Um die Wiederverwendung zu steigern und gleichzeitig den Aufwand für die Steuergeräteentwicklung zu reduzieren [PBKS07], ist der Aufbau von Produktlinien mit entsprechenden Varianten eine Möglichkeit [JHCMG10, WFO09]. Durch die Verwendung der AutoMoMe ist bereits die Grundlage für die Erstellung von Produktlinien geschaffen, da eine modellbasierte Modellierung mit der Möglichkeit zur Variantenbildung unter gleichzeitiger virtueller Absicherung vorliegt. Dies sind wesentliche Aspekte für die Erstellung einer Produktlinie im automobilen Bereich [JHCMG10]. Jedoch zählt zur Produktlinienentwicklung nicht nur die Architekturmodellierung, sondern gleichermaßen das Anforderungsmanagement. In diesem muss definiert werden, welche Anforderungen zu einer Variante gehören und welche Anforderungen allgemein gültig sind.

Von daher ist die Einbindung eines Variantenmangementwerkzeuges erforderlich, das sich sowohl mit dem Anforderungsmanagement, als auch mit der Architekturmodellierung in Form der AutoMoMe verbindet. Die Grundlage hierzu ist mit der Nutzung der AutoMoMe gelegt.

Der Schwerpunkt zukünftiger Entwicklungen verlagert sich immer mehr zu sicherheitskritischen Systemen. Bei deren Entwicklung spielt die funktionale Sicherheit eine entscheidende Rolle. Diese ist besonders für die Architekturmodellierung relevant und muss daher bei der Entwicklung mitberücksichtigt werden. Ein Grundstein hierfür ist die Zusammenführung der Daten in einem Architekturmodell [SRM13, MGRKB04], wie es in AutoMoMe vorgesehen ist. Folglich besteht die Möglichkeit, weitergehende Analysen für die funktionale Sicherheit in den hier entwickelten Techniken zu integrieren [MGRKB04]. Beispielsweise kann die Absicherung von Signalen oder Komponenten bei der AUTOSAR Synthese genutzt werden. Diese kann abhängig von der Sicherheitseinstufung automatisiert angewandt werden. Somit ist der Datenpfad ohne zusätzlichen Aufwand abgesichert. Ferner spielt der Einsatz von Redundanzen und damit verbunden die Aufteilung auf unterschiedliche Hardwarekomponenten eine wesentliche Funktion.

Der AUTOSAR Standard mit den standardisierten Schnittstellen ist ein entscheidender Schritt hin zu der beliebigen Verteilung von Software auf die Hardware [Buh07]. Die Verteilung der Komponenten kann manuell oder aber unterstützt durch entsprechende Algorithmen erfolgen, wobei letztere Variante den Vorteil bietet, jeweils sofort die optimale Konfiguration zu erhalten [KMTH10, NSH+10, KHTW08, MBAG11, SR08a, KA98]. Durch die Werkzeugunterstützung kann sichergestellt werden, dass alle Vorgaben und Bedingungen an die Architektur, die z. B. durch das technische Sicherheitskonzept entstanden sind, auch umgesetzt werden. Voraussetzung ist jedoch die (semi-) formale Spezifikation, wie sie in AutoMoMe erfolgt.

In Zukunft kann sogar damit gerechnet werden, dass die Systeme sich automatisch den Gegebenheiten anpassen. In diesem Fall spricht man von selbstadaptiven Systemen [Gei08]. In denen wird die Software durch Konfigurationsdaten den Gegebenheiten angepasst. Derzeit werden diese jedoch aus Gründen der Sicherheit nur selten oder gar nicht eingesetzt. Aber durch neue Techniken ist eine zukünftige Nutzung durchaus vorstellbar [THP+07, ARC+07, ALE+06, GBS04]. Hierdurch sind neue Funktionalitäten insbesondere beim Fahrkomfort zu erwarten.

Bei der Entwicklung von sicherheitskritischen Systemen ist die Verwendung von formalen Verifikationstechniken, beispielsweise Model-Checker, durchaus angebracht [BKKS05]. Durch sie ist es möglich zu beweisen, dass bestimmte Zustände z. B. Verklemmungen (Deadlocks) etc. nicht auftreten [SKB+09]. Durch die Beschreibung der Architektur in (semi-) formaler Weise und der Verwendung von RTSCs ist auch hierfür der Grundstein gelegt, so dass Trans-

formationen in eine formalisierte Sprache und die anschließende Überprüfung durch eine formalisierte Verifikation durchführbar ist [Dru06, DMS09].

Als Resümee gilt es festzuhalten, dass bei der Steuergeräteentwicklung noch Verbesserungspotentiale vorhanden sind, die über die in AutoMoMe dargelegten behobenen Schwachstellen hinausgehen. Mit der Arbeit AutoMoMe ist die Basis und ein Grundstein hierfür gelegt. Die Integration in die im Ausblick beschriebenen Ansätze ist in aufbauenden Arbeiten zu realisieren und zu vertiefen.

A

Anhang zur Arbeit

Im Anhang sind die Diagramme zu finden, die auf Grund von Platzmangel nicht in der Arbeit dargestellt werden konnten. Dies sind hauptsächlich die umgesetzten Metamodelle für die vorgenommenen Modellerweiterungen und die Modelltransformationsregeln, die die Synthese zwischen dem Architekturmodell und dem AUTOSAR Softwarearchitekturmodell definieren.

A.1 Metamodelle der Erweiterungen

A.1.1 Einbindung des Echtzeitverhaltens

Zur Einbindung der Echtzeit wird die SysML um Real-time Statecharts erweitert. In der Abbildung A.1 ist das Metamodell der erweiterten SysML dargestellt.

Die neu eingefügten Elemente sind dabei farblich gekennzeichnet. Die neuen Modellelemente sind gelb und die neuen Assoziationen grün hervorgehoben. Die Erweiterungen beziehen sich dabei zumeist auf die Zeitmodellierung. So können beispielsweise Uhren und zeitliche Bedingungen modelliert werden. Mit dem so erweiterten Metamodell wird die Kombination von SysML (Version 1.2) und Real-Time Statecharts hergestellt.

A.1.2 Metamodell des Signal-Managers

Die einzelnen Bussignale werden in Form des Signal-Managers importiert und anschließend für die Nutzung innerhalb der Systemarchitektur transformiert. Der Signal-Manager ist für verschiedene Kommunikationsbusse anwendbar. Von daher ist es wichtig, ein gemeinsames Metamodell zu erstellen, so dass

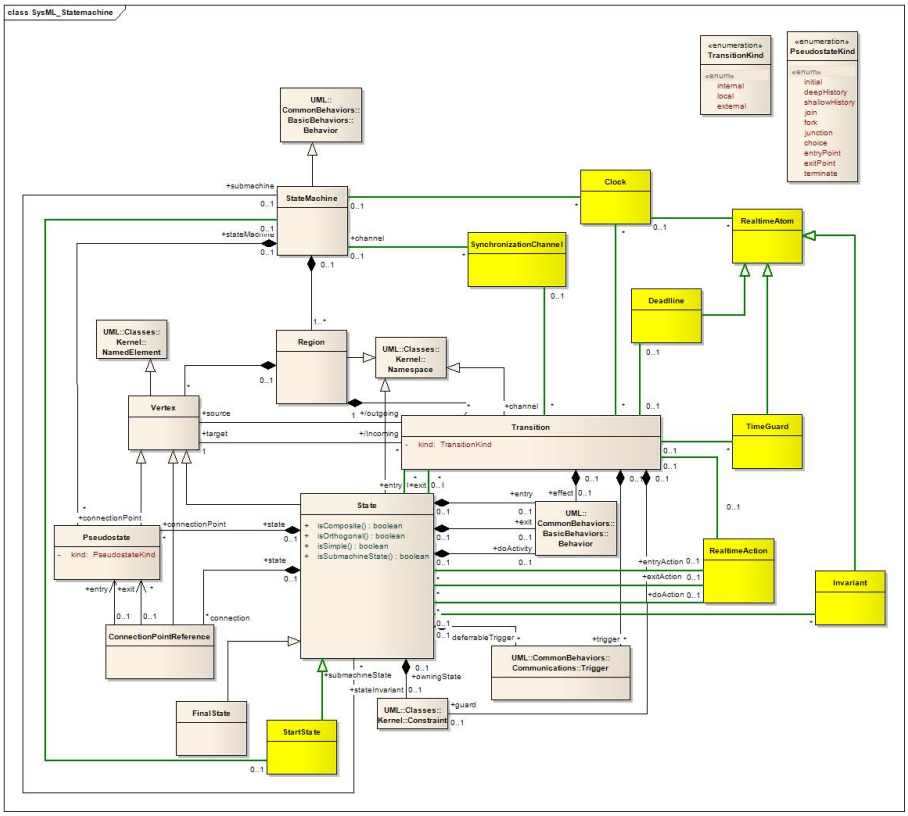


Abb. A.1. Erweitertes Metamodell der SysML/UML Zustandsdiagramme

die entwickelten Transformationen für alle Busse anwendbar sind. In der Abbildung A.2 ist das erarbeitete Metamodell wiedergegeben. Hierbei geht es nicht darum, jede mögliche Information aus den Kommunikationsmatrizen zu importieren. Vorrangig ist vielmehr, die für die Systemarchitektur benötigten Informationen bereitzustellen. Hauptschwerpunkt sind dabei die Signalgruppen mit den jeweiligen Signalen. Sie werden für die Systemarchitektur benötigt und sind daher im Metamodell durch ihre Attribute, wie beispielsweise den *Offset* oder die *Skalierung*, beschrieben. Hierdurch besteht die Möglichkeit, falls Werte nicht mit der gewünschten Güte vorliegen ggf. innerhalb der Systemarchitektur eine notwendige Transformation vorzunehmen, so dass ein bestehendes Teilsystem trotz der zuerst inkompatiblen Signale verwendet werden kann.

A.1.3 Echtzeitsimulationsmodell

In diesem Abschnitt wird das Metamodell beschrieben, dass für die Generierung des Echtzeitsimulationsmodells genutzt wird. Auf Grund der Größe

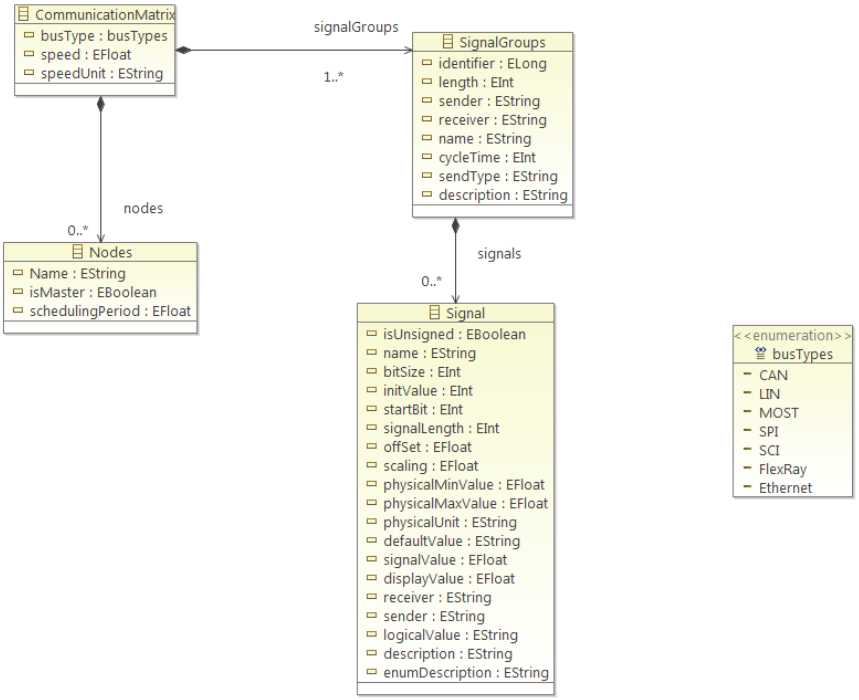


Abb. A.2. Metamodell des Signal-Managers

des Metamodells konnte es in Kapitel 5 nur anhand der gezeigten Beispiele beschrieben werden. In der Abbildung A.3 ist das vollständige Metamodell im EMF Format abgebildet. Das Metamodell besteht dabei aus zahlreichen Klassen, Attributen und Operationen, wobei das Element *SimulationProject* im Mittelpunkt steht und dem alle anderen Elemente untergeordnet sind.

Damit die Daten aus dem Architekturmodell automatisiert in ein von diesem Metamodell abgeleitetes Modell übertragen werden, sind Transformationsregeln notwendig. Dies sind größtenteils 1:1 Abbildungen, da die Architekturmodellierung dem Metamodell entsprechend angepasst ist. In der Abbildung A.4 sind die einzelnen Transformationsregeln dargestellt. Auf der linken Seite befinden sich die Modellelemente aus dem Architekturmodell. Bei abgeleiteten Elementen (eigener Stereotyp) ist dabei stets angegeben, von welchem Element sie abgeleitet sind. Auf der rechten Seite befinden sich die Ziele der Transformation.

Aus einem von diesem Metamodell abgeleiteten Modell wird mittels Templates der Code für die Projektdatei (XML), für die Zeittabelle (C-Source) und für die Tasks und Dateien (C/H-Dateien) generiert. Die Templates sind dabei mittels dem OAW Framework realisiert.

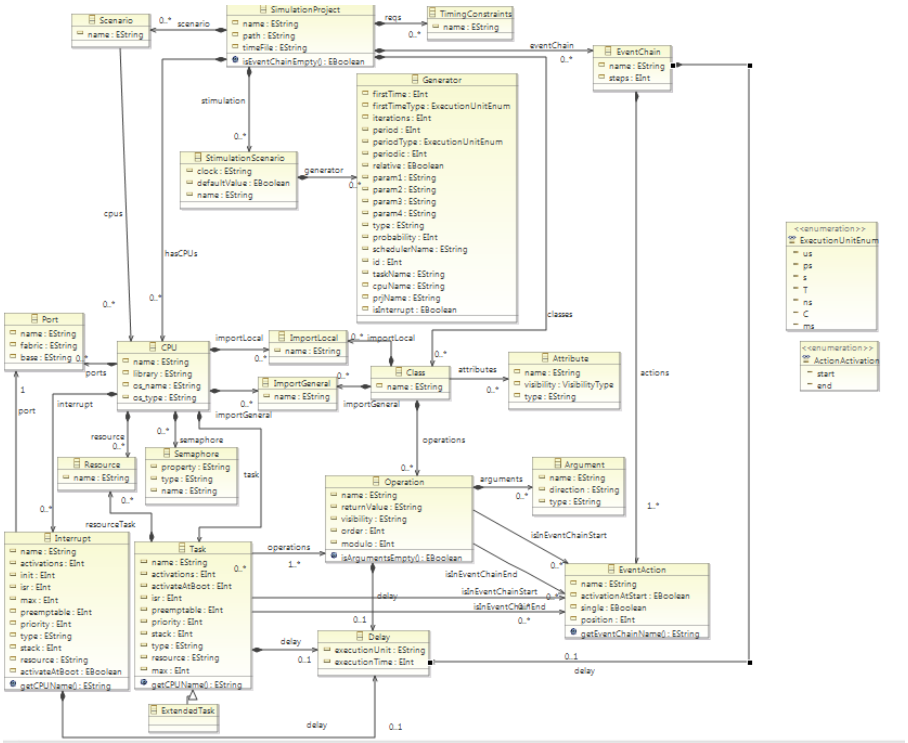
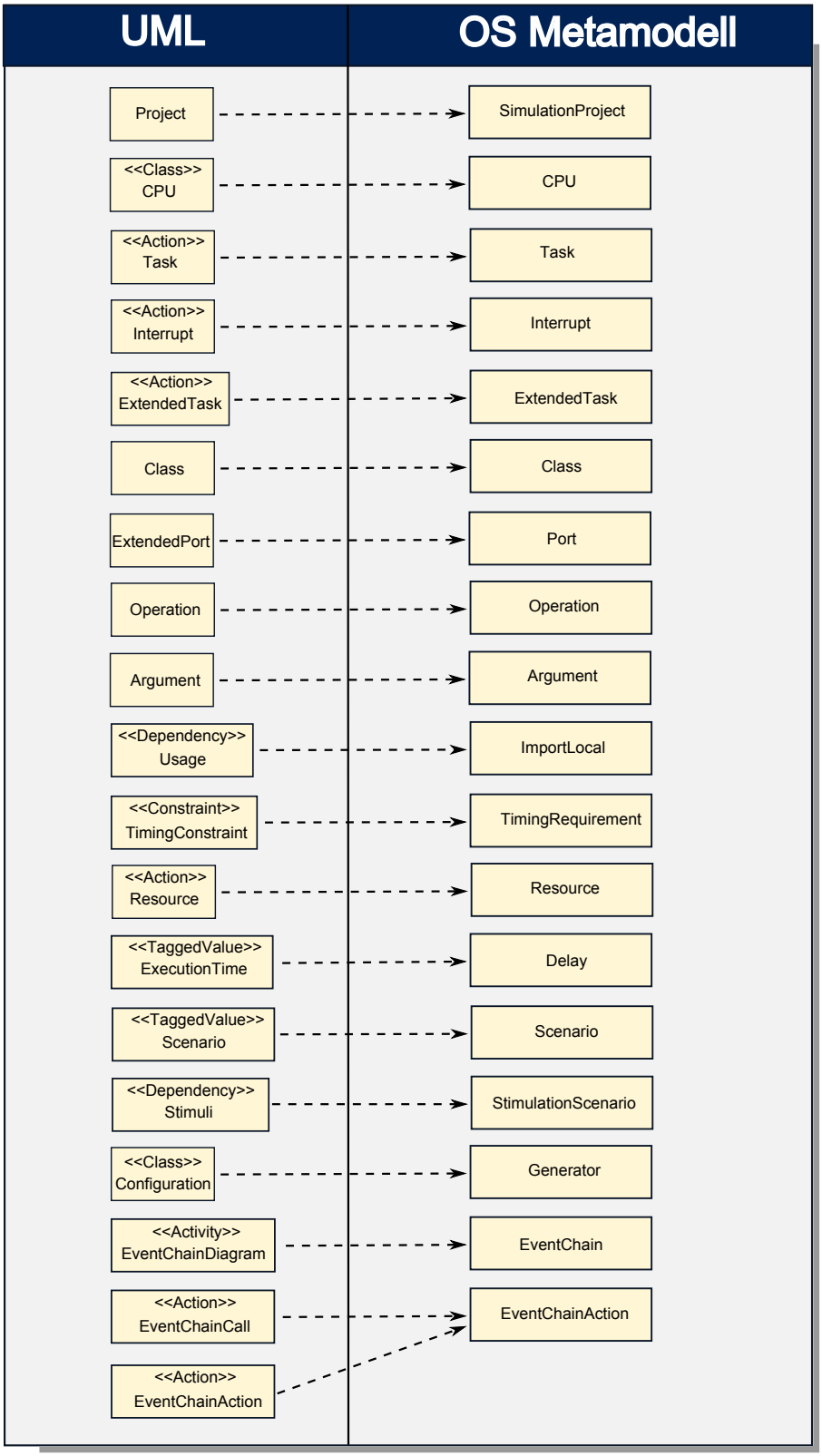


Abb. A.3. Metamodell für die Transformation in die Echtzeitsimulation

A.2 Spezifikation der Modellierungsmethodik

Die neuen Modellelemente und ihre Ableitung von den Meta-Elementen der SysML bzw. UML sind im Kapitel 4 beschrieben. Zusätzlich wird für die entwickelte Modellierungsmethodik AutoMoMe eine (semi-) formalen Spezifikation in Form eines Metamodells definiert. Hierdurch wird sichergestellt, dass alle Entwickler eine Problemstellung auf die gleiche Art und Weise bearbeiten und nicht unterschiedliche Lösungen nutzen [KDHM13]. Des Weiteren kann aus der Spezifikation (automatische) Überprüfungen (Checks) abgeleitet werden und zum anderen dient es zur Schulung der Entwickler in der Methodik. So ist ersichtlich, welche Modellelemente in den einzelnen Bereichen und in welcher Anzahl zugelassen sind. Die Spezifikation erfolgt dabei mittels der UML. Im Folgenden sind die wesentlichen neuen Beschreibungsmittel grafisch abgebildet. So ist in der Abbildung A.5 die Definition des Systemkontextdiagramms dargestellt. Hierdurch wird festgelegt, welche Elemente innerhalb des Systemkontextes beschrieben werden. So werden hauptsächlich die Steuergeräte (*ECU* und *ForeignECU*) sowie deren Zusammenspiel in Form von Bussen


UML
OS Metamodell
Project
SimulationProject
<<Class>> CPU
CPU
<<Action>> Task
Task
<<Action>> Interrupt
Interrupt
<<Action>> ExtendedTask
ExtendedTask
Class
Class
ExtendedPort
Port
Operation
Operation
Argument
Argument
<<Dependency>> Usage
ImportLocal
<<Constraint>> TimingConstraint
TimingRequirement
<<Action>> Resource
Resource
<<TaggedValue>> ExecutionTime
Delay
<<TaggedValue>> Scenario
Scenario
<<Dependency>> Stimuli
StimulationScenario
<<Class>> Configuration
Generator
<<Activity>> EventChainDiagram
EventChain
<<Action>> EventChainCall
EventChainAction
<<Action>> EventChainAction

dargestellt. Die weitere Verfeinerung der Kommunikationsdaten erfolgt mittels Spezifikationen und Eigenschaften (*FlowSpecification* und *FlowProperties*).

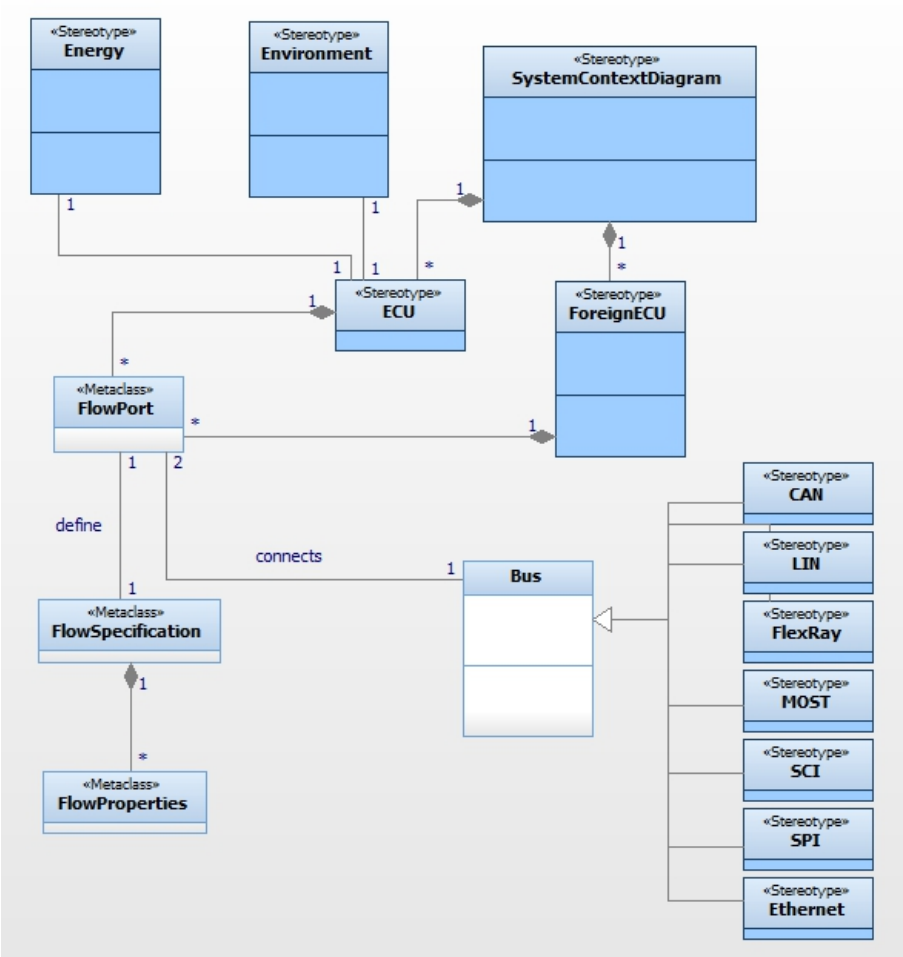


Abb. A.5. Spezifikation für das Systemkontextdiagramm

Innerhalb der Modellierung der AutoMoMe sind ebenfalls Bedingungen notwendig. Für die Definition von Bedingungen (Constraints) kann das Beispiel der Sicherheitseinstufung in Abbildung A.6 genutzt werden. Durch die Bedingung wird innerhalb des Metamodells festgelegt, dass die Sicherheitseinstufung eines Funktionsblocks mindestens so groß wie die Sicherheitseinstufung der darunterliegenden Funktionen sein muss. Eine Bedingung wird dabei im Modell als Constraint dargestellt und die Spezifikation erfolgt mittels der Object Constraint Language OCL [Gro12, WK04].

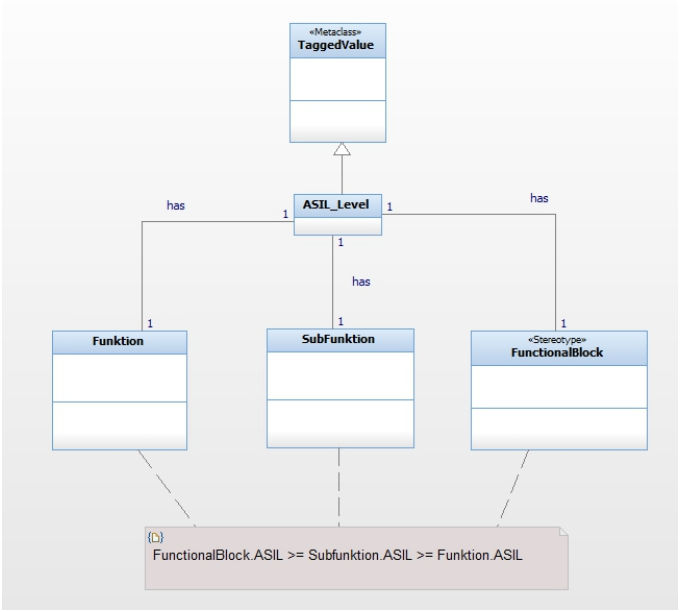


Abb. A.6. Modellierung von Bedingungen (Constraints) innerhalb der AutoMoMe

Neben dem Kontextdiagramm ist eine weitere Ergänzung in AutoMoMe die Modellierung von Wirkketten. Zum besseren Verständnis wird ebenfalls eine Spezifikation erstellt, die in der Abbildung A.7 wiedergegeben wird. Es ist zu sehen, dass mittels der neuen Stereotypen «EventChainAction» und «EventChainCall» die verschiedenen Elemente, die innerhalb einer Wirkkette vorkommen können, abgebildet werden.

Ein weiterer Schwerpunkt des Ansatzes AutoMoMe ist die Spezifikation der Betriebssystemeigenschaften. In der Abbildung A.8 ist die Spezifikation hierfür abgebildet. Neben der Spezifikation der Betriebssystemeigenschaften wie Tasks und Interrupts spielt darüber hinaus die Spezifikation der Wirkketten für die Echtzeitsimulation eine gewichtige Rolle. Daher wird sie ebenfalls bei der Betriebssystemmodellierung mit aufgeführt. Ferner ist neben der Echtzeitsimulation die Konfiguration des AUTOSAR Betriebssystems von Bedeutung. Aus diesem Grund sind die notwendigen Komponenten, z. B. die Betriebssystem Applikation in die Konfiguration mit integriert.

A.3 Modelltransformationen

Im folgenden Abschnitt werden die Transformationsregeln für die UML nach AUTOSAR Transformation vorgestellt. Diese sind zwar im Prototyp mittels

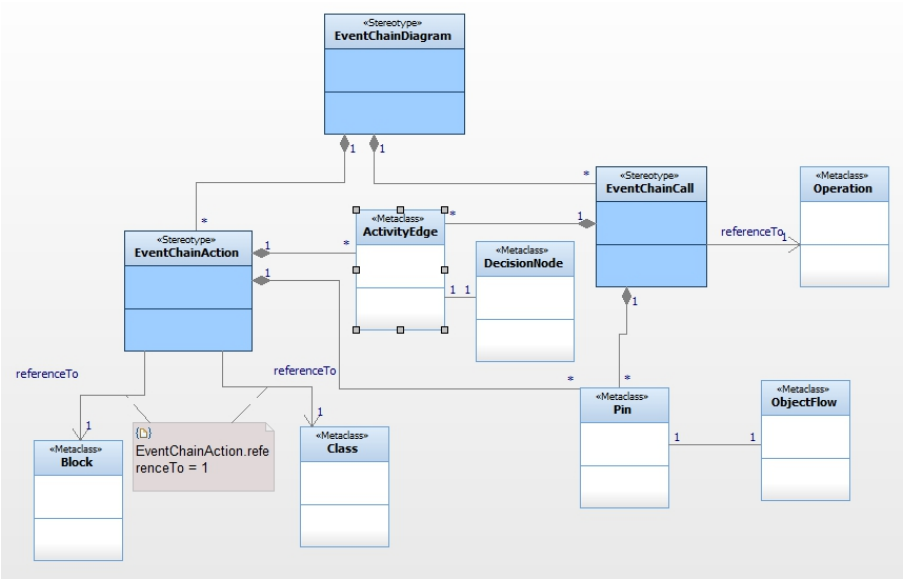


Abb. A.7. Spezifikation einer Wirkkette

der Programmiersprache Java realisiert, werden an dieser Stelle zur besseren Übersicht anlehnend an die QVT abgebildet. In der Abbildung A.9 werden die Transformationsregeln für die Struktur dargestellt, wobei in Abbildung A.10 die Transformationsregeln für die RTE dargestellt sind.

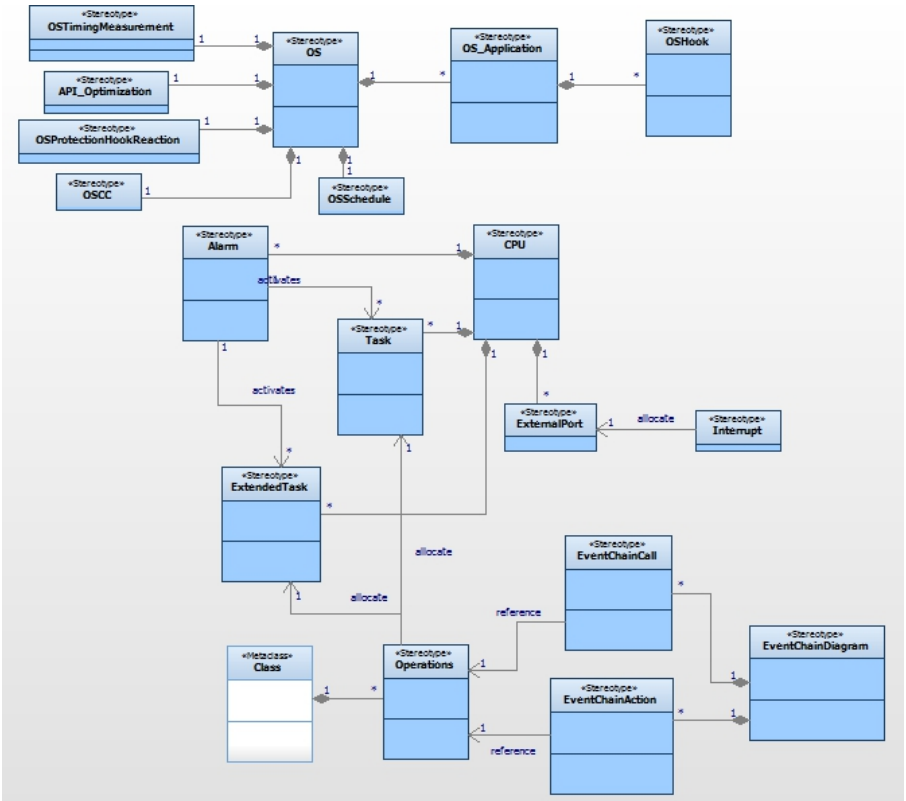


Abb. A.8. Spezifikation der Betriebssystemkonfiguration

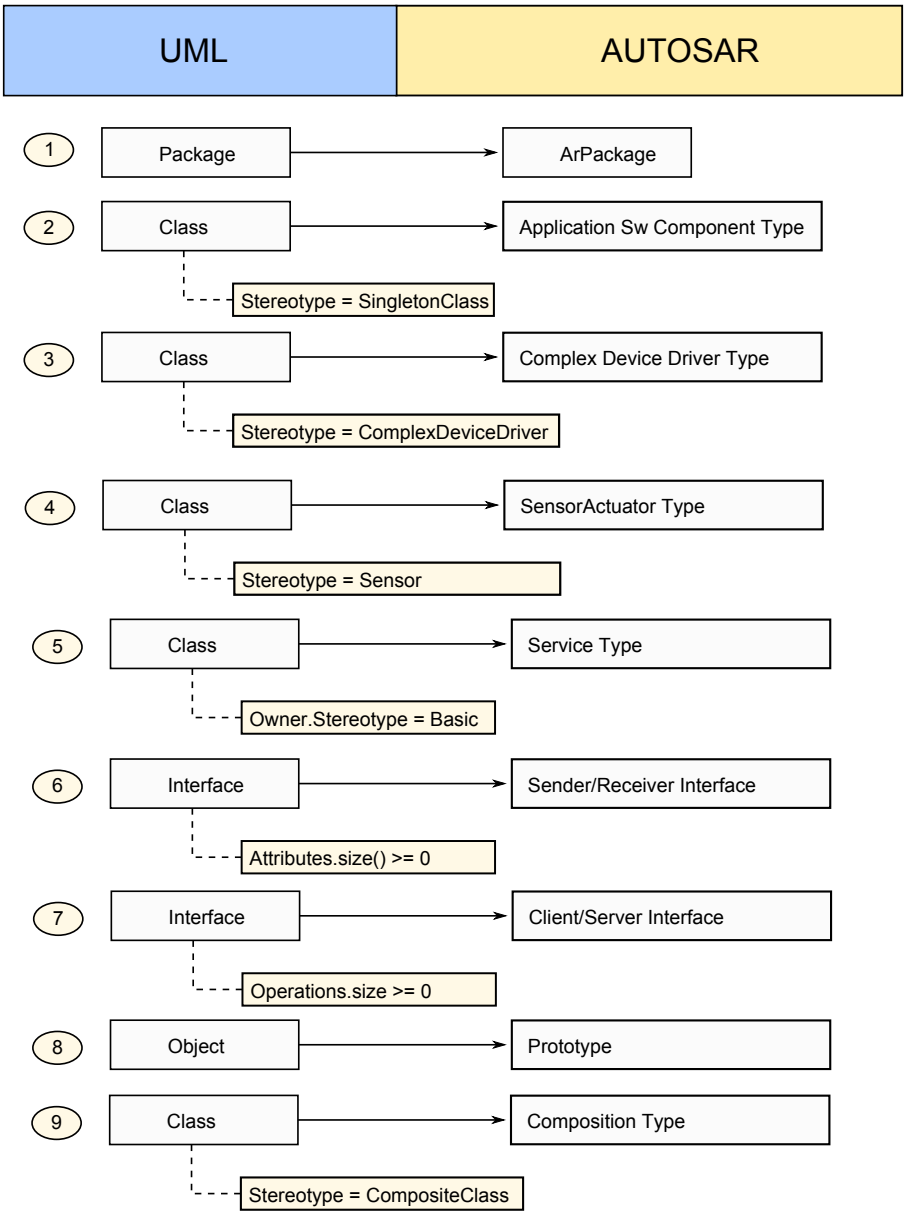


Abb. A.9. Die Transformationsregeln für die Struktur

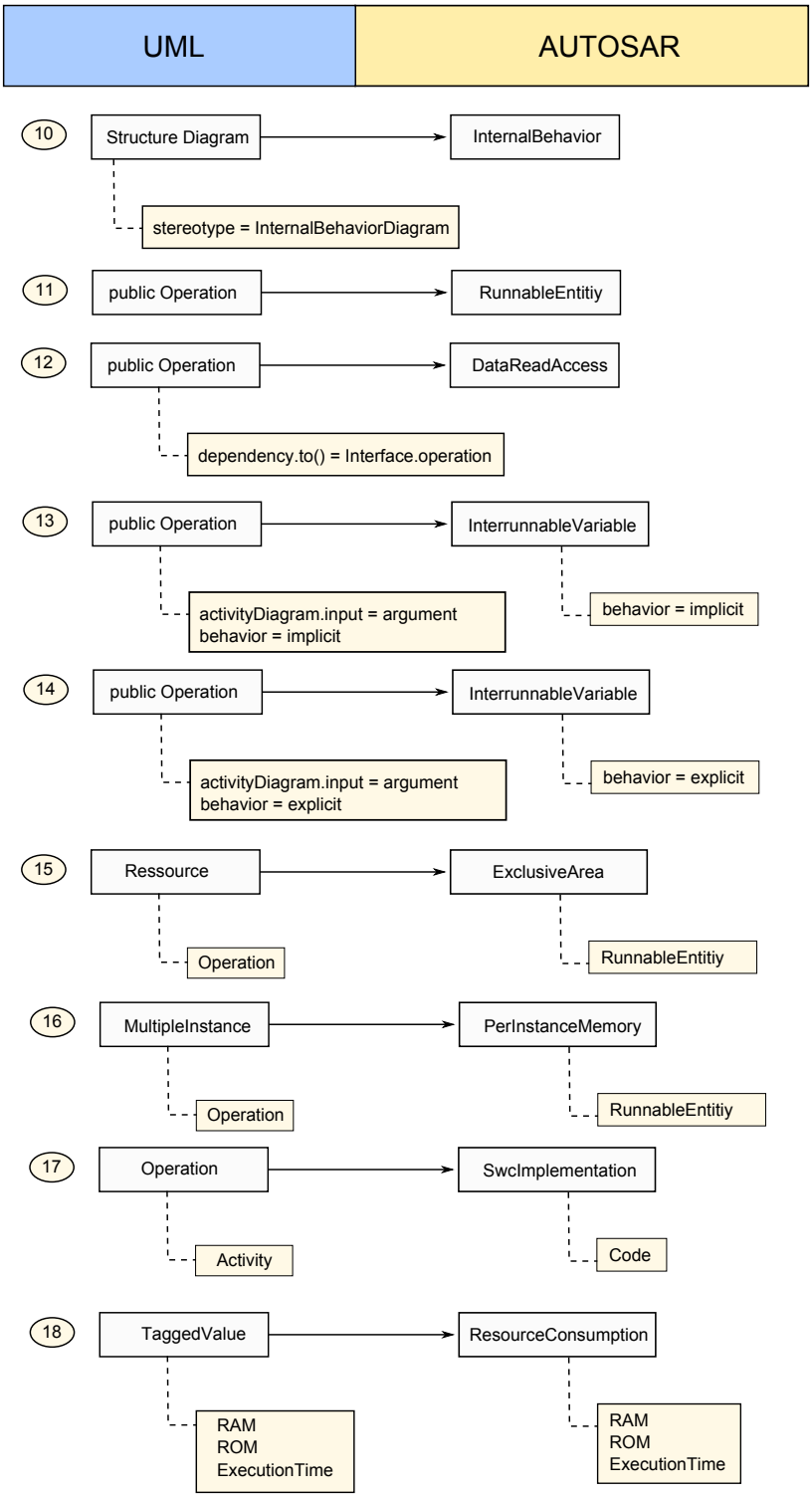


Abb. A.10. Transformationsregeln für die RTE

B

Glossar & Verzeichnisse

Glossar

Frontloading
: Verstärkter Aufwand in den frühen Entwicklungsphasen, um dadurch in den späteren Entwicklungsphasen eine Entlastung zu bekommen. Insgesamt entsteht eine schnellere und günstigere Entwicklung.

Abkürzungsverzeichnis

AADL	Architecture Analysis and Design Language.
ABS	Anti-Blockier System.
ACC	Adaptive Cruise Control.
AML	Automotive Modeling Language.
ASIL	Automotive Safety Integrity Level.
ATAM	Architecture Tradeoff Analysis Method.
AutoMoMe	Automotive Modeling Methodology.
AUTOSAR	Automotive Open System Architecture.
BCRT	Best Case Response Time.
BSWMD	Basis Software Module Description.
CESAR	Cost-efficient methods and processes for safety relevant embedded systems.
CPU	Central Processing Unit.
DOORS	Dynamic Object Oriented Requirements System.
DSL	domänen-spezifische Sprache.

ECU	Steuergeräten (Electronical Control Unit).
ECU-C	ECU Extract of System Description.
EMF	Eclipse Modeling Framework.
ESP	Elektronische Stabilitäts-Programm.
FSC	funktionalen Sicherheitskonzept.
HIL	Hardware-in-the-loop.
IBD	internes Blockdiagramm.
IEC	International Electrotechnical Comission.
IoT	Internet der Dinge (Internet of Things).
ISO	International Organization for Standardization.
KSG	Komfortsteuergerät.
LDW	Fahrstreifenerkennungssystem (Lane departure warning).
M2M	Modell-zu-Modell Transformation.
M2T	Modell-zu-Text Transformation.
MIL	Model-in-the-loop.
NVM	Non-Volatile Memory.
OAW	openArchitectureWare.
OCL	Object Constraint Language.
OEM	Original Equipment Manufacturer.
OIL	OSEK Implementation Language.
OMG	Object Management Group.
OS	Betriebssystem (Operating System).
OSEK	Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug.
Pkw	Personenkraftwagen.
PWM	Pulsweitenmodulation.
QVT	Query View Transformation.
RE	Requirements Engineering.
RTE	Runtime Environment.
RTSC	Real-Time Statecharts.
SA	Strukturierte Analyse.

SADT	Structured Analysis and Design Technique.
SD	Strukturiertes Design.
SOP	Start of production.
SPES	Software Plattform Embedded Systems.
SPES2020	Software Plattform Embedded Systems 2020.
SPICE	Software Process Improvement and Capability Determination.
SWC	Software component.
SysML	Systems Modeling Language.
TADL	Timing Augmented Description Language.
TGG	Triple Graph Grammatiken.
TTA	Time-Triggered Architektur.
UML	Unified Modeling Language.
VFB	Virtual Functional Bus.
WCET	Worst Case Execution Time.
WCRT	Worst Case Response Time.
XML	Extensible Markup Language.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Abs14] AbsInt. aiT - WCET-Analyse, http://www.absint.com/ait/index_de.htm. Internet, 2014.

[AELG11] Peter Altenbernd, Andreas Ermedahl, Björn Lisper, and Jan Gustafsson. Automatic Generation of Timing Models for Timing Analysis of High-Level Code. In S. Faucou, editor, *Proc. 19th International Conference on Real-Time and Network Systems (RTNS)*, 2011.

[AG13] Daimler AG. Die Geburt des Automobils, http://www.daimler.com/dccom/0-5-1322446-49-1323352-1-0-0-1322455-0-0-135-7145-0-0-0-0-0-0-0.html. Internet, 2013.

[ALE+06] Richard Anthony, Alexander Leonhardi, Cecilia Ekelin, Dejiu Chen, Martin Törngren, Gerrit de Boer, Isabell Jahnich, Simon Burton, Ola Redell, Alexander Weber, and Vasco Vollmer. A Future Dynamically Reconfigurable Automotive Software System. In *Proceedings of Elektronik im Kraftfahrzeug, Systeme von Morgen - Technische Innovationen und Entwicklungstrends*, 2006.

[ALSU08] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compiler - Prinzipien, Techniken und Werkzeuge*. Pearson Studium, 2008.

[AM08] Fabian Ahrendts and Anita Marton. *IT-Risikomanagement leben*. Springer Verlag, 2008.

[Amb11] Scott W. Ambler. Generalizing Specialists: Improving Your IT Career Skills. *http://www.agilemodeling.com/essays/generalizingSpecialists.htm*, 2011.

[AMdS07] Charles Andre, Frederic Mallet, and Robert de Simone. Time modeling in MARTE. *Forum on specification and design languages (FDL)*, 2007.

[AQST10] A. Albinet, L. Queran, B. Sanchez, and Y. Tanguy. Requirement management from System modeling to AUTOSAR SW Components. *Conference Embedded Real Time Software and Systems (ERTS)*, 2010.

[ARC+07] Richard Anthony, Achim Rettberg, Dejiu Chen, Isabell Jahnich, Gerrit de Boer, and Cecilia Ekelin. Towards a dynamically reconfigurable automotive control system architecture. *International Federation for Information Processing (IFIP)*, 231:77 – 84, 2007.

[Ass13] EAST ADL Association. EAST-ADL2 Specification V2.1.12, 2013.

[AUT10a] AUTOSAR Gbr. Version 4.0. Application Interfaces (AUTOSAR_MOD_AISpecification), 2010.

[AUT10b] AUTOSAR Gbr. Version 4.0. AUTOSAR Standard, 2010.

[AUT10c] AUTOSAR Gbr. Version 4.0. Specification of BSW Module Description Template (AUTOSAR_TPS_BSWModuleDescriptionTemplate), 2010. Version 4.0.

[AUT10d] AUTOSAR Gbr. Version 4.0. Specification of ECU Configuration (AUTOSAR_TPS_ECUConfiguration), 2010. Version 4.0.

[AUT10e] AUTOSAR Gbr. Version 4.0. Specification of Runtime Environment(AUTOSAR_SWS_RTE), 2010. Version 4.0.

[AUT10f] AUTOSAR Gbr. Version 4.0. Specification of Software Component Template (AUTOSAR_TPS_SoftwareComponentTemplate), 2010.

[AUT10g] AUTOSAR Gbr. Version 4.0. Specification of System Template (AUTOSAR_TPS_SystemTemplate), 2010.

[AUT10h] AUTOSAR Gbr. Version 4.0. Specification of the AUTOSAR Operating System (AUTOSAR_SWS_OS), 2010. Version 4.0.

[AUT10i] AUTOSAR Gbr. Version 4.0. UML BSWMD Model (AUTOSAR_MOD_BSWUMLModel), 2010.

[AUT10j] AUTOSAR Gbr. Version 4.0. UML Meta Model (AUTOSAR_MOD_MetaModel), 2010.

[AxB11] AxBench. AxBench Projektseite. Internet, 2011.

[BBB+13] Gerhard Baum, Holger Borcherding, Manfred Broy, Martin Eigner, Anton S. Huber, Herbert K. Kohler, Siegfried Russwurm, and Matthias Stümpfle. *Industrie 4.0 Beherrschung der industriellen Komplexität mit SysLM.* Springer Verlag, 2013.

[BBRS05] Andreas Bauer, Manfred Broy, Jan Romberg, and Bernhard Schätz. AutoMoDe - Notations, Methods and Tools for Model-Based Development of Automotive Software. *SAE International*, 2005.

[BD06] Jürgen Bortz and Nicola Döring. *Forschungsmethoden und Evaluation (4. Auflage).* Springer Verlag, 2006.

[BdALM08] Vanilson Arruda Buregio, Eduardo Santana de Almeida, Daniel Ludredio, and Silvio Lemos Meira. A Reuse Repository System: From Specification to Deployment. *Proceedings of the International Conference on Software Reuse (ICSR)*, 2008.

[Bec00] Kent Beck. *Extreme Programming Explained.* Addison Wesley, 2000.

[Bec06] Helmut Becker. *High Noon in the Automotive Industry.* Springer Verlag, 2006.

[Bec07] Helmut Becker. *Auf Crashkurs Automobilindustrie im globalen Verdrängungswettbewerb.* Springer Verlag, 2007.

[BFH+10] Manfred Broy, Martin Feilkas, Markus Herrmannsdoerfer, Stefano Merenda, and Daniel Ratiu. Seamless Model-based Development: from Isolated Tools to Integrated Model Engineering Environments. *Proceedings of the IEEE - Special Issue on Aerospace & Automotive*, 2010.

[BGS05] Sven Burmester, Holger Giese, and Wilhelm Schäfer. Model-Driven Architecture for Hard Real-Time Systems: From Platform Independent Models to Code. *Model Driven Architecture - Foundations and Applications ECMDA-FA*, 2005.

[BH08] James Bruck and Kenn Hussey. Customizing UML: Which Technique is Right for you? *eclipse.org*, 2008.

[BKK+12] Manfred Broy, Sascha Kirstan, Helmut Krcmar, Bernhard Schätz, and Jens Zimmermann. What is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry? *Emerging Technologies for the Evolution and Maintenance of Software models*, 2012.

[BKKS05] J. Botaschanjan, L. Kof, C. Kühnel, and M. Spichkova. Towards Verified Automotife Software. *ACM SIGSOFT Software Engineering Notes archive*, 30(4), 2005.

[BKS10] Klaus Bergner, Michael Kempf, and Siegfried Schäfler. Zwischen Totalitarismus und Kleinstaaterei: Grosse Modelle adäquat managen. *Objektspektrum*, 6:14–19, 2010.

[BLY09] Lionel C. Briand, Yvan Labiche, and Tao Yue. Automated traceability analysis for UML model refinements. In *Journal of Information and Software Technology*, volume 51, pages 517–527, 2009.

[BMP08] Simona Bernardi, Jose Merseguer, and Dorina C. Petriu. Adding Dependability Analysis Capabilities to the MARTE Profile. *Springer Verlag*, 2008.

[BMR12] N Md Jubair Basha, Salman Abdul Moiz, and Mohammed Rizwanullah. Model Based Software Development: Issues & Challenges. In *Special Issue of International Journal of Computer Science & Informatics (IJCSI)*, 2012.

[BMT11] Manfred Broy, Klaus Hardy Mühleck, and Dirk Taubner. Informatik in der Automobilindustrie. In *Informatik Spektrum - Informatik in der Automobilindustrie*, volume 34. Gesellschaft für Informatik, Gesellschaft für Informatik, Februar 2011.

[Boe76] Barry W. Boehm. Software Engineering. *IEEE Transactions on Computers*, C- 25:1226 – 1241, 1976.

[Bol09] Richard F. Boldt. Modeling AUTOSAR systems with a UML/SysML profile. Technical report, IBM Software Group, 2009.

[Bor08] Kai Borgeest. *Elektronik in der Fahrzeugtechnik*. Vieweg + Teubner, 2008.

[Bos13] Bosch. 25 Jahre ABS von Bosch. Internet, 2013.

[BPL01] A. Bakshi, V. K. Prasanna, and A. Ledeczi. MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems. In *Workshop on Languages, Compilers and Tools for Embedded Systems*, 2001.

[Bür12] Ralf Bürger. *Frühe Fehler sind teure Fehler*. Ralf Bürger, 2012.

[Bro06] Manfred Broy. Challenges in Automotive Software Engineering. *Proceedings of the International Conference on Software Engineering (ICSE)*, 2006.

[Bro10] Manfred Broy. *Cyber Physical Systems - acatech DISKUTIERT*, volume 0. Springer Verlag, 2010.

[BRR11] Manfred Broy, Günter Reichart, and Lutz Rothhardt. Architekturen softwarebasierter Funktionen im Fahrzeug: von den Anforderungen zur Umsetzung. In *Informatik Spektrum - Informatik in der Automobilindustrie*, volume 34. Gesellschaft für Informatik, Gesellschaft für Informatik, Februar 2011.

[BRS11] Andreas Baudisch, Kai Richter, and Stefan Sollmann. Erweiterte Vorgehensmodelle für die Entwicklung echtzeitfähiger, hochintegrierter, multifumultifunktionaler Steuergeräte-Plattformen. *VDI Baden-Baden*, 2011.

[BS05] Bruno Bouyssounouse and Joseph Sifakis. *Embedded Systems Design*. Springer Verlag, 2005.

[BS13] Hans-Hermann Braess and Ulrich Seiffert. *Vieweg Handbuch Kraftfahrzeugtechnik*. Springer Vieweg, 2013.

[BSER11] Michael Bohn, Jörn Schneider, Christian Eltges, and Robert Rößger. Migration von AUTOSAR-basierten Echtzeitanwendungen auf Multicore-Systeme. *4. Workshop Entwicklung zuverlässiger Software-Systeme der Gesellschaft für Informatik e.V. Fachgruppe Ada*, 2011.

[BSP08] Nadine Bramsiepe, Ernst Sikora, and Klaus Pohl. Ableitung von Systemfunktionen aus Zielen und Szenarien. In *Softwaretechnik-Trends*, volume 28, pages 13 –16. Gesellschaft für Informatik (GI), Gesellschaft für Informatik, 2008.

[BST10] Karsten Berns, Bernd Schürmann, and Mario Trapp. *Eingebettete Systeme Systemgrundlagen und Entwicklung eingebetteter Software*. Vieweg + Teubner, 2010.

[BTC13] BTC. IBM Rational Rhapsody Automatic Test Generation Add On. Internet, 2013.

[Buh07] M. Buhlmann. Vom Design zum Code; ein durchgängiger Ansatz (DECOS). *VDI Berichte Nr. 2000 (Baden-Baden)*, pages 287 – 295, 2007.

[Bur02] Sven Burmester. Generierung von Java Real-Time Code für zeitbehaftete UML Modelle. Master's thesis, Universität Paderborn, 2002.

[CaPL09] Lawrence Chung and and Julio Cesar Prado Leite. Conceptual Modeling: Foundations and Applications. pages 363–379, Berlin, Heidelberg, 2009. Springer-Verlag.

[CCG+07] Philipe Cuenot, DeJiu Chen, Sebastien Gerard, Henrik Lönn, Mark-Oliver Reiser, David Servat, Carl-Johan Sjöstedt, Ramin Tavakoli Kolagari, Martin Törngren, and Matthias Weber. Managing Complexity Automotive Electronics Using the EAST-ADL. *12th IEEE International Conference on Engineering Complex Computer Systems*, 2007.

[CFJ+09] P. Cuenot, P. Frey, R. Johansson, H. Lönn, M.-O. Reiser, D. Servat, R. Tavakoli, and D.J. Chen. Developing Automotive Products Using the EAST-ADL2, an AUTOSAR Compliant Architecture Description Language. Technical report, Mentor Graphics Techpub, 2009.

[Coc03] Alistair Cockburn. *Use Cases effektiv erstellen.* mitp-Verlag, 2003.

[CR11] Oliver Charlet and Sebastian Rothbucher. Flüsterpost ade: Teamintegration durch ein gemeinsames Modell. In *OBJEKTSpektrum*, volume 1, pages 68–72. SIGS Datacom, 2011.

[CS06] Volker Claus and Andreas Schwill. *Duden Informatik A-Z.* Dudenverlag, 2006.

[DAV09] Zamira Daw, Flor Alvarez, and Marcus Vetter. Methode zur Entwicklung sicherheitskritischer eingebetteter Systeme mittels deterministischer UML-Modelle. *Softwaretechnik Trends*, pages 11 – 16, August 2009.

[dAV10] Verband der Automobilindustrie (VDA). Automotive SPICE, 2010.

[DBHW13] Marian Daun, Jennifer Brings, Jens Höfflinger, and Thorsten Weyer. Funktionsgetriebene Entwicklung software-intensiver eingebetteter Systeme in der Automobilindustrie - Stand der Wissenschaft und Forschungsfragestellungen. In *Workshopband Software Engineering 2013, GI-Edition Lecture Notes in Informatics (LNI), Dritter Workshop zur Zukunft der Entwicklung softwareintensiver eingebetteter Systeme (ENVISION2020)*, pages 293 – 302, 2013.

[DEC11] DECOS. DECOS Projektseite. Internet, 2011.

[Dei13] Fabian Deitelhoff. Gegen das Vergessen: Designentscheidungen bewusst wahrnehmen und dokumentieren. *Objektspektrum*, 3:78–85, 2013.

[DeM79] Tom DeMarco. *Structured Analysis and system specification.* Yourdon Press, 1979.

[DGH03] Schahram Dustdar, Harald Gall, and Manfred Hauswirth. *Software Architekturen für verteilte Systeme.* Springer Verlag, 2003.

[dI04] Verein deutscher Ingenieure. Entwicklungsmethodik für mechatronische Systeme, VDI 2206. Internet, 2004.

[dJ07] Gjalt de Jong. A UML-Based Design Methodology for Real-Time and Embedded Systems. Telelogic, 2007.

[DJM+08] H. HeiH. Heinecke. Damm, B. Josko, A. Metzner, H. Kopetz, A. Sangiovanni-Vincentelli, and M. Di Natale. Software Components for Reliable Automotive Systems. In *Proceedings of Design, Automation and Test in Europe (DATE)*, 2008.

[DMS09] Henning Dierks, Alexander Metzner, and Ingo Stierand. Efficient Model-Checking for Real-Time Task Networks. In *Proceedings of International Conference on Embedded Software and Systems*, 2009.

[Dou00] Bruce Powel Douglass. *Real-Time UML Second Edition - Developing Efficient Objects for embedded Systems.* Addison-Wesley, 2000.

[Dou11] Bruce Powel Douglass. *Doing Hard Time.* Addison Wesley, 2011.

[Dru06] Doron Drusinsky. *Modeling and Verification Using UML Statecharts.* Elsevier, 2006.

[dSP14] dSPACE. TargetLink, http://www.dspace.de/de/gmb/home/products/sw/pcgs/targetli.cfm. Internet, 2014.

[DTA+08] Sebastien Demathieu, Frederic Thomas, Charles Andre, Sebastien Gerard, and Francois Terrier. First experiments using the UML profile for MARTE. *IEEE Symposium on Object Oriented Real-Time Distributed Computing*, 2008.

[EAT14] EATOP. EATOP Projektseite http://eclipse.org/proposals/modeling.eatop/. Internet, 2014.

[ECSG09] Huascar Espinoza, Daniela Concila, Brain Selic, and Sebastien Gerard. Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems. *Model Driven Architecture - Foundations and Applications*, 2009.

[EDMG05] Huascar Espinoza, Hubert Dubois, Julio Medina, and Sebastien Gerard. A general structure for the analysis framework of the UML MARTE Profile. *Workshop on Modelling and Analysis of Real Time and Embedded Systems (Models)*, 2005.

[EGG+09] Gregor Engels, Michael Goedicke, Ursula Goltz, Andreas Rausch, and Ralf Reussner. Design for Future Legacy-Probleme von morgen vermeidbar? *Informatik Spektrum*, 32(5):393–397, 2009.

[Ehr06] Klaus Ehrlenspiel. *Integrierte Produktentwicklung Denkabläufe, MethodeMethoden, Zusammenarbeit.* Hanser Verlag, 2006.

[EHV+03] Tiemo Ehlers, Mirko Harms, J.-Uwe Varchmin, Martin Mutz, and Marc Horstmann. STEP-X: Strukturierter Entwicklungsprozess für Automotive Anwendungen. In *Proceedings 23. Tagung Elektronik im Automobil*, 2003.

[ERG08] Huascar Espinoza, Kai Richter, and Sebastien Gerard. Evaluating MARTE in an Industry-Driven Environment: TIMMO's Challenges for AUTOSAR Timing Model, 2008.

[Est08] Jeff A. Estefan. Survey of model-based systems engineering (MBSE) methodologies. *California Institute of Technology, Pasadena, California, USA May*, 25, 2008.

[EtSTA10] Advancing Traffic Efficiency and Safety through Software Technology (ATESST). EAST-ADL tool, http://www.atesst.org/scripts/home/publigen/content/templates/show.asp?P=125&L=EN#2. Internet, 2010.

[eV08] ADAC e. V. Die ADAC Pannenstatistik 2008, 2008.

[Far07] Tibor Farkas. Automotive Software Engineering mit der Unified Modeling Language (UML). *Fraunhofer Institut FOKUS*, 2007.

[FAZ10] FAZ. Gaspedal bremst Toyota aus, 2010.

[FCLT11] Lei Feng, DeJiu Chen, Henrik Lönn, and Martin Törngren. Verifying System BBehavior in EAST-ADL2 with the SPIN Model Checker. In *IEEE International Conference on Mechatronics and Automation*, 2011.

[FF95] William B. Frakes and Christopher J. Fox. Sixteen Questions about Software Reuse. *Communications of the ACM archive Volume 38*, 1995.

[FFKM10] Claudiu Farcas, Emilia Farcas, Ingolf H. Krueger, and Massimiliano Menarini. Adressing the Integration Challenge for Avionics and Automotive Systems - From Components to Rich Services. *Proceedings of the IEEE - Special Issue on Aerospace & Automotive*, 2010.

[FG12] Peter H. Feiler and David P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language.* Addison Wesley, 2012.

[FGDT07] Robert B. France, Sudipto Ghosh, and Trung Dinh-Trong. Model-Driven Development Using UML 2.0: Promises and Pitfalls. *FOSE*, 2007.

[FHP+05] A. Fleischmann, J. Hartmann, C. Pfaller, M. Rappl, S. Rittmann, and D. Wild. Concretization and Formalization of Requirements for Automotive Embedded Software Systems Development. pages 60–65, 2005.

[FHR08] Florian Fieber, Michaela Huhn, and Bernhard Rumpe. Modellqualität als Indikator für Softwarequalität: eine Taxonomie. In *Informatik Spektrum*, volume 31, pages 408–424. GI, 2008.

[FKKSP05] Wolfgang Frieß, Stefan Kubica, Andreas Krüger, and Wolfgang Schröder-Preikschat. Konfigurationsprüfung für Standardsoftware mit Hilfe von Merkmalmodellen. *VDI Baden-Baden*, 2005.

[FKS10] Reiner Friedrich, Jörg Kostelezky, and Ludwig Schwankl. Anwendung von neuen Standards am Beispiel des BMW 5er Gran Turismo. *Automotive - Carl Hanser Verlag*, 3-4:41 – 43, 2010.

[FMS12] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical Guide to SysML The Systems Modeling Language*. Elsevier, 2012.

[Fou14] Eclipse Foundation. Eclipse Modeling Framework (EMF), http://www.eclipse.org/modeling/emf/. Internet, 2014.

[FR07] Robert France and Bernhard Rumpe. Model-driven Development of Complex Software: A Research Roadmap. *FOSE*, 2007.

[Fri06] Terrence P. Fries. A Framework for Transforming Structured Analysis and Design Artifacts to UML. *SIGDOC*, 2006.

[fS10a] International Organization for Standardization. ISO 10681-1: Road vehicles – Communication on FlexRay – Part 1: General information and use case definition, 2010.

[fS10b] International Organization for Standards. ISO/DIS 26262 Road vehicles – Functional safety, 2010.

[fS12] International Organization for Standardization. Software Process Improvement and Capability Determination (SPICE) (ISO/IEC 15504), 2012.

[FSA05] Johan Fredriksson, Kristian Sandström, and Mikael Akerholm. Optimizing Resource Usage in Component-Based Real-Time Sytems. *Component-Based Software Engineering*, 2005.

[fSoAA11] Association for Standardisation of Automation and Measuring Systems (ASAM). ASAM MSRSW V3.0.0,http://www.asam.net/nc/home/standards/standard-detail.html?tx_rbwbmasamstandards_pi1[showUid]=2560&start. Internet, 2011.

[fSoAA13] Association for Standardisation of Automation and Measuring Systems (ASAM). ASAM MCD-2 NET (market name: FIBEX) V4.1.0, http://www.asam.net/nc/home/standards/standard-detail.html?tx_rbwbmasamstandards_pi1[showUid]=2694&start=. Internet, 2013.

[GB03] Holger Giese and Sven Burmester. Real-Time Statechart Semantics. Technical report, Universität Paderborn, 2003.

[GB10] Andreas Graf and Miriam Brückner. AUTOSAR und modellbasierte Softwareentwicklung. *Elektronik automotive*, 2010.

[GBS04] Holger Giese, Sven Burmester, and Wilhelm Schäfer. Modular Design and Verification of Component-Based Mechatronic Systems with Online-Reconfiguration. *Foundations of Software Engineering (FSE)*, 2004.

[Gei08] Kurt Geihs. Selbst-adaptive Software. *Informatik Spektrum*, 31:133–145, 2008.

[GGH+06] Susanne Graf, Sebastien Gerard, Oystein Haugen, Iulian Ober, and Bran Selic. Modelling and Analysis of Real Time and Embedded Systems - using UML. *Workshop MARTES*, 2006.

[GGS12] Marvin Grieger, Baris Güldali, and Stefan Sauer. Sichern der Zukunftsfähigkeit bei der Migration von Legacy-Systemen durch modellgetriebene Softwareentwicklung. In *Softwaretechnik-Trends*, volume 32, pages 37–38. GI, 2012.

[GH06] Holger Giese and Stefan Henkler. A survey of approaches for the visual model-driven development of next generation software-intensive systems. *Journal of Visual Languages & Computing*, 2006.

[GHH08] Holger Giese, Stefan Henkler, and Martin Hirsch. Combining Formal Verification and Testing for Correct Legacy Component Integration in Mechatronic UML. *Architecting Dependable Systems V*, 2008.

[GHJV04] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Entwurfsmuster*. Add, 2004.

[GHN09] Holger Giese, Stephan Hildebrandt, and Stefan Neumann. Towards Integrating SysML and AUTOSAR Modeling via Bidirectional Model Synchronization. In Holger Giese, Michaela Huhn, Ulrich Nickel, and Bernhard Schätz, editors, *Tagungsband Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme*, pages 155 – 164. Informatik-Bericht 2009-01 TU Braunschweig, Institut für Software Systems Engineering Technische Universität Braunschweig, 2009.

[GK01] Marco Götze and Wolfram Kattanek. Erfahrungen mit der UML beim Entwurf von Kfz-Steuerungen. *4. GI/ITG/GMM-Workshop der Fachgruppen 3 und 4 der Kooperationsgemeinschaft Rechnergestützter Schaltungs- und Systementwurf*, 2:87–98, 2001.

[GK02] Abdelouahed Gherbi and Ferhat Khendek. UML Profiles for Real-Time Systems and their Applications. *Journal of Object Technology*, 2002.

[GLT03] Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded Software Engineering: The State of the Practice. *IEEE Software*, pages 61 – 69, 2003.

[Gmb14] Vector Informatik GmbH. PreeVision, http://vector.com/vi_preevision_en.html. Internet, 2014.

[GOO02] Susanne Graf, Ileana Ober, and Iulian Ober. A real-time profile for UML. *Omega-Project*, 2002.

[GP04] Michael Gerdom and Torsten Posch. Pragmatische softwarearchitektur für automative-Systeme. *Objektspektrum*, 2004.

[GPR06] Volker Gruhn, Daniel Pieper, and Carsten Röttgers. *MDA Effektives Software-Engineering mit UML2 und Eclipse*. Springer Verlag, 2006.

[GPR11] Joel Greenyer, Sebastian Pook, and Jan Rieke. Preventing Information Loss in Incremental Model Synchronization by Reusing Elements. In Robert France, Jochen M. Kuester, Behzad Bordbar, and Richard F. Paige, editors, *Proceedings of the Seventh European Conference on Modelling Foundations and Applications (ECMFA)*, volume 6698, pages 144–159. Springer Verlag, 2011.

[GR11] Martin Große-Rhode. aXLang User Guide - Version 0.10.0, 2011.

[Gra11] Andreas Graf. UML und AUTOSAR UML als Ergänzung zu AUTOSAR für die Modellierung von E/E-Systemen. *Elektronik Automotive*, 2011.

[Gre11] Joel Greenyer. *Scenario-based design of mechatronic systems*. PhD thesis, University of Paderborn, 2011.

[Gro05] Object Management Group. UML Profile for Schedulability, Performance and Testing (SPT), V1.1, 2005.

[Gro10] Object Management Group. SysML Specification 1.2, 2010.

[Gro11a] Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT), v1.1. Internet, 2011.

[Gro11b] Object Management Group. MOF 2 XMI Mapping Standard v2.4.1, 2011.

[Gro11c] Object Management Group. UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) v1.1, 2011.

[Gro11d] Object Management Group. UML Specification v2.4.1, 2011.

[Gro12] Object Management Group. OCL Specification (v2.3.1), 2012.

[Gro14] ETAS Group. ETAS ASCET, http://www.etas.com/de /products/ascet_software_products.php. Internet, 2014.

[GRSS11] Joel Greenyer, Jan Rieke, Wilhelm Schäfer, and Oliver Sudmann. The Mechatronic UML Development Process. In Peri L. Tarr and Alexander L. Wolf, editors, *The Continuing Contributions of Leon J. Osterweil*, pages 311–322, 2011.

[Grz07] Andreas Grzemba. *MOST: Das Multimedia-Bussystem für den Einsatz im Automobil*. Franzis Verlag, 2007.

[GSA+11] Jürgen Gausemeier, Wilhelm Schäfer, Harald Anacker, Frank Bauer, and Stefan Dziwok. Einsatz semantischer Technologien im Entwurf mechatronischer Systeme. In Jürgen Gausemeier, Franz Rammig, Wilhelm Schäfer, and Ansgar Trächtler, editors, *8. Paderborner Workshop: Entwurf mechatronischer Systeme (Wissenschaftsforum 2011 Intelligente Technische Systeme)*, number 294, pages 7–35, 2011.

[GTB+03] Holger Giese, Matthias Tichy, Sven Burmester, Wilhelm Schäafer, and Stephan Flake. Towards the Compositional Verification of Real-Time UML Designs. *Proceedings of the 9th European software engineering conference*, 2003.

[GvdW05] Andreas Grzemba and Hans-Christian von der Wense. *LIN-Bus*. Franzis Verlag, 2005.

[GWS03] Zonhua Gu, Shige Wang, and Kang G. Shin. Issues in Mapping from UML Real-Time Profile to OSEK API. *Workshop on Specification and Validation of UML models for Real Time and Embedded Systems (SVERTS)*, 2003.

[Har09] Henning Harbs. Keine Angst vor AUTOSAR... Ein Vergleich Gestern und Heute. *VDI-Beri*, 2009.

[Har11] Markus Hardt. AxBench - Datenblatt, 2011. Fraunhofer ISST.

[HFP+06] J. Hartmann, A. Fleischmann, C. Pfaller, M. Rappl, S. Rittmann, and D. Wild. Feature Net - ein Ansatz zur Modellierung von automobil-spezifischem Domänenwissen und Anforderungen. 2006.

[HG10] Matthias Hagner and Ursula Goltz. Integration of Scheduling Analysis into UML Based Development Process Through Model Transformation. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, pages 797–804, 2010.

[HH04] Nadine Heumesser and Frank Houdek. Experiences in Managing an Automotive Requirements Engineering Process. *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE)*, 2004.

[HH07] Matthias Hagner and Michaela Huhn. Modellierung und Analyse von Zeitanforderungen basierend auf der UML. *GI-Workshop: Automotive Software Engineering*, 2007.

[HHP03] Derek Hatley, Peter Hruschka, and Imtiaz Pirbhai. *Komplexe Software-Systeme beherrschen.* mitp-Verlag, 2003.

[HKK04] Bernd Hardung, Thorsten Kölzow, and Andreas Krüger. Reuse of Software in Distributed Embedded Automotive Systems. 2004.

[HKM+05] Alfred Helmerich, Nora Koch, Luis Mandel, Peter Braun, Peter Dornbusch, Alexander Gruler, Patrick Keil, Roland Leisibach, Jan Romberg, Bernhard Schätz, Thomas Wild, and Guido Wimmel. Study of Worldwide Trends and R&D Programmes in Embedded Systems in View of Maximising the Impact of a Technology Platform in the Area, 2005.

[HLP+10] Florian Hölzl, Christian Leuxner, Birgit Penzenstadler, Martin Haldenmair, Christoph Döbber, and Andreas Wandinger. AutoFOCUS3 - The picture book, 2010.

[HMM12] Wilfried Horn, Jan Meyer, and Thorsten Molt. Systems Engineering im Rahmen der funktionalen Sicherheit. In Maik Maurer and Sven-Olaf Schulze, editors, *Tag des Systems Engineering (TdSE 2012)*, pages 97 –106. Carl Hanser Verlag, 2012.

[HMSN10a] Stefan Henkler, Jan Meyer, Wilhelm Schäfer, and Ulrich Nickel. Legacy Component Integration by the Fujaba Real-Time Tool Suite. 2010.

[HMSN10b] Stefan Henkler, Jan Meyer, Wilhelm Schäfer, and Ulrich Nickel. Reverse Engineering mechatronischer Komponenten. *Proc. 7. Paderborner Workshop Entwurf mechatronischer Systeme*, 2010.

[HMSN10c] Stefan Henkler, Jan Meyer, Wilhelm Schäfer, and Ulrich Nickel. Reverse Engineering vernetzter automotiver Softwaresysteme. *Modellbasierte Entwicklung eingebetteter Systeme (MBEES) Workshop*, 2010.

[HMvD11] Jörg Holtmann, Jan Meyer, and Markus von Detten. Automatic Validation and Correction of Formalized, Textual Requirements. In *Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW) 2011*, 2011.

[Hol10] Jörg Holtmann. Mit Satzmustern von textuellen Anforderungen zu Modellen. *OBJEKTspektrum*, (RE/2010 (Online Themenspecial Requirements Engineering)), June 2010.

[HOT11] Erich Henkel, Martin Ober, and Dirk Taubner. Erfahrungen mit Lean-Konzepten im Management von Softwareprojekten. In *Informatik Spektrum - Informatik in der Automobilindustrie*, volume 34. Gesellschaft für Informatik, Gesellschaft für Informatik, Februar 2011.

[Hoy00] David Hoyle. *Automotive Quality Systems Handbook*. Butterworth-Heinemann, 2000.

[HRM07] Edward Huang, Randeep Ramamurthy, and Leon F. McGinnis. System and simulation modeling using sysml. *Winter Simulation Conference*, 2007.

[HS06] Peter Hruschka and Gernot Starke. Praktische Architekturdokumentation: Wie wenig ist genau richtig? *Objektspektrum*, 1, 2006.

[HS09] Peter Hruschka and Gernot Starke. Softwarearchitekten: Die Zehnkämpfer der IT. In *OBJEKTSpektrum*, volume 4, pages 12–19. SI, 2009.

[HSF+09] Florian Hölzl, Wolfgang Schwitzer, Martin Feilkas, Christoph Döbber, and Andreas Wandinger. AUTOFOCUS 3 Overview, http://autofocus.in.tum.de /index.php/AF3_Overview. Internet, 2009.

[HSS+07] Wolfgang Herzner, Rupert Schlick, Martin Schlager, Bernhard Leiner, Bernhard Huber, András Balogh, György Csertan, Alain LeGuennec, Thierry LeSergent, Neeraj Suri, and Shariful Islam. Model-Based Development of Distributed Embedded Real-Time Systems with the DECOS Tool-Chain. In *Proceedings of 2007 SAE AeroTech Congress & Exhibition*, 2007.

[Huc99] Thomas Huckle. Kleine BUGs, große GAUs, http://www5.in.tum.de/ huckle/bugs.html. Internet, 1999.

[IBM14a] IBM. Rational DOORS ,http://www-03.ibm.com/software/products/de/ratidoor/. Internet, 2014.

[IBM14b] IBM. Rational Rhapsody, http://www-03.ibm.com/software/products/de/ratirhap. Internet, 2014.

[IEC99] Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbar elektronischer Systeme, 1999.

[IEE90] IEEE. IEEE Standard Glossary of Software Engineering Terminology, 1990.

[Inc14] Inchron GmbH. Inchron Toolsuite, www.inchron.de. Internet, 2014.

[Jer07] Jersak. Timing model and methodology for AUTOSAR. *Elektronik automotive Special issue AUTOSAR*, 2007.

[JHCMG10] Martin Jaensch, Bernd Hedenetz, Markus Conrath, and Klaus D. Müller-Glaser. Transfer von Prozessen des Software-Produktlinien Engineering in die Elektrik/Elektronik-Architekturentwicklung von Fahrzeugen. In *Informatik 2010: Service Science - Neue Perspektiven für die Informatik, Beiträge der 40. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Band 2*, 2010.

[Jon97] Capers Jones. *Applied software measurement (2nd ed.): assuring productivity and quality.* McGraw-Hill, Inc., Hightstown, NJ, USA, 1997.

[JPB08] Thomas Johnson, Christiaan J.J. Paredis, and Roger Burkhart. Integrating Models and Simulations of Continuous Dynamics into SysML. *OMG*, 2008.

[KA98] Muhammad Kafil and Ishfaq Ahmad. Optimal Task Assignment in Heterogeneous Distributed Computing Systems. *IEEE Concurrency*, 1998.

[Kap06] Roland Kapeller. Erstellung vollständiger Systemspezifikationen im Embedded Computing. *Softwaretechnik Trends*, 2006.

[Kar04] Gabor Karsai. Automotive Software: A Challenge and Opportunity for Model-Based Software Development. *ASWSD*, LNCS 4147:103 – 115, 2004.

[KC02] Sascha Konrad and Betty H.C. Cheng. Requirements Patterns for Embedded Systems. In *Proceedings of the IEEE International Requirements Engineering Conference (RE02)*, 2002.

[KDHM13] Lydia Kaiser, Roman Dumitrescu, Jörg Holtmann, and Matthias Meyer. Automatic verification of modeling rules in system engineering for mechatronic systems. In *Proceedings of the ASME International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE*, 2013.

[KF09] Olaf Kindel and Mario Friedrich. *Softwareentwicklung mit AUTOSAR.* dpunkt-Verlag, 2009.

[KHTW08] Stefan Kugele, Wolfgang Haberl, Michael Tautschnig, and Martin Wechs. Optimizing Automatic Deployment Using Nonfunctional Requirement Annotations. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation, CCIS*, pages 400–414. Springer Verlag, 2008.

[KK06] Roland Kapeller and Stefan Krause. So natürlich wie Sprechen - Embedded Systeme modellieren -. *Design & Elektronik*, 2006.

[KKH+08] Ji Eun Kim, Rahul Kapoor, Martn Herrmann, Jochen Haerdtlein, Franz Grzeschniok, and Peter Lutz. Software Behavior Description of Real-Time Embedded Systems in Component Based Software Development. *11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, 2008.

[KLW11] Andreas Korff, Jesko G. Lamm, and Tim Weilkiens. Werkzeuge für den Schmied funktionale Architekturen. In Maik Maurer and Sven-Olaf Schulze, editors, *Tag des Systems Engineering (TdSe2011).* Hanser Verlag, 2011.

[KM09] T. Kramer and R. Münzenberger. New Functions, New Sensors, New Architectures - How to Cope with the Real-Time Requirements. *13th Int. Forum on Advanced Microsystems for Automotive Applications (AMAA)*, 2009.

[KM10a] Tapio Kramer and Ralf Münzenberger. Absicherung des Echtzeitverhaltens mittels virtueller Integration. *Vortragsband zur 4. Tagung - Simulation und Test für die Automobilelektronik, IAV*, 2010.

[KM10b] Tapio Kramer and Ralf Münzenberger. Echtzeitverhalten komplexe Systeme Optimal mit beidenMethoden analysieren und beherrschen. In *Embedded Software Engineering Kongress*, 2010.

[KMTH10] Benjamin. Klöpper, Jan Meyer, Matthias Tichy, and Shinichi Honiden. Planning with Utilities and State Trajectories Constraints for Self-Healing in Automotive Systems. In *Proc. of the Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Budapest, Hungary, September 27-October 1, 2010*. IEEE Computer Society Press, 2010.

[Kop97] Hermann Kopetz. *Real-Time Systems Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[KPK+10] Steffen Kollmann, Victor Pollex, Kilian Kempf, Frank Slomka, Matthias Traub, Torsten Bone, and Jürgen Becker. Comparative Application of Real-Time Verification Methods to an Automotive Architecture. In *Proceedings of 18th International Conference on Real-Time and Network Systems*, 2010.

[KPLJ08] Daehyun Kum, Gwang-Min Park, Seohngun Lee, and Wooyoung Jung. AUTOSAR Migration from Existing Automotive Software. *International Conference on Control, Automation and Systems 2008*, 2008.

[KR07] Markus Kühl and Clemens Reichmann. Modellbasierte Architekturentwicklung von E/E-Systemen. *Automobil Elektronik*, 2007.

[Krö13] Michael Kröger. Rückrufaktionen erreichen Rekorde, http://www.spiegel.de/auto/aktuell/autoindustrie-rueckrufaktionen-2012-auf-rekordniveau-a-883625.html. Internet, 2013.

[Krä14] Bernd J. Krämer. Evolution of Cyber-Physical Sytems: A Brief Review. In Sang C. Suh, John N. Carbone, U. John Tanik, and Abdullah Eroglu, editors, *Applied Cyber-Physical Systems*, 2014.

[Kra09] Tapio Kramer. Perfekt synchronisiert. In *Elektronik automotive*, volume 11, pages 42–44. www.elektroniknet.de, 2009.

[Kra12a] Tapio Kramer. Echtzeitsysteme verteilt entwickeln. In *Elektronik Praxis*, pages 24–26. Vogel Business Media, 2012.

[Kra12b] Tapio Kramer. Quo vadis Echtzeitsysteme? In *Elektronik automotive*, volume 5, pages 40–42. www.elektroniknet.de, 2012.

[KRL12] Felix Klanner, Ralph Raßhofer, and Martin Liebner. Eine neue Dimension der Fahrzeugsicherheit. In *Elektronik automotive*, volume 10, pages 20 – 25. www.elektroniknet.de, 2012.

[KSSS04] Christian Kamm, Johannes Siedersleben, Daniel Schick, and Alexandre Saad. Systematische Aufwandsabschätzung für Software im Fahrzeug. *Objektspektrum*, 6, 2004.

[KSW13] Rüdiger Kaffenberger, Sven-Olaf Schulze, and Hanno Weber. *INCOSE Systems Engineering Handbuch*. Hanser Verlag, 2013.

[ku13] ku. Automobilindustrie ist erneut Spitzenreiter. *Elektronik Automotive*, (5):6, 2013.

[KW07] Ekkart Kindler and Robert Wagner. Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios, http://www.cs.uni-paderborn.de/ uploads/tx_sibibtex/tr-ri-07-284.pdf. Internet, 2007.

[LB08] Falk Langer and Susanne Baumer. Das CAMoS-Framework, 2008.

[Lee08] Edward A. Lee. Cyber Physical Systems: Design Challenges. In *Proceedings of International Symposium on Object/Component/Service-Oriented Real-Time Distributing Computing (ISORC)*, 2008.

[Lee09] Edward A. Lee. Computing needs Time. *Communications of the ACM archive Volume 52*, 2009.

[LKM+12] Andrea Leitner, Nermin Kajtazovic, Roland Mader, Christian Kreiner, Christian Steger, and Reinhold Weiß. Lightweight introduction of EAST-ADL2 in an automotive software product line. In *Proceedings 45th International Conference on System Science*, 2012.

[LLP+09] Rongshen Long, Hong Li, Wei Peng, Yi Zhang, and Minde Zhao. An Approach to Optimize Intra-ECU Communication Based on Mapping of AUTOSAR Runnable Entities. In *Proceedings of International Conference on Embedded Software and Systems*, 2009.

[LMD+09] Malte Lochau, Tobias C. Müller, Stefan Detering, Ursula Goltz, and Thomas Form. Architektur-Evaluation von AUTOSAR-Systemen: Adaption und Integration. In *Elektronik automotive congress*, 2009.

[LR05] Peter Liggesmeyer and Dieter Rombach. *Software Engineering eingebetteter Systeme (Grundlagen - Methodik - Anwendungen)*. Elsevier Spektrum Akademischer Verlag, 2005.

[LSG+09] Malte Lochau, Jens Steiner, Ursula Goltz, Tobias C. Müller, and Thomas Form. Optimierung von AUTOSAR-Systemen durch automatisierte Architektur-Evaluation. *VDI-Ber*, 2009.

[LSL+09] Rikard Land, Daniel Sundmark, Frank Lüders, Iva Krasteva, and Adnan Causevic. Reuse with Software Components - A Survey of Industrial State of Practice. *Formal Foundations of Reuse and Domain Engineering - Springer Verlag*, 5791, 2009.

[LT09] Peter Liggesmeyer and Mario Trapp. Trends in Embedded Software Engineering. *IEEE Software*, 2009.

[LW10] Jesko G. Lamm and Tim Weilkiens. Funktionale Architekturen in SysML. In Maik Maurer and Sven-Olaf Schulze, editors, *Tag des Systems Engineering (TdSE 2010)*, pages 109 –118. GfSE, Hanser Verlag, 2010.

[Mal10] Claudia Mallok. Chancen der Autozuliefererindustrie Oliver Wyman-Studie nennt 5 Erfolgsfaktoren für die Zukunft der Autozuliefererindustrie. *Elektronik Praxis*, 2010.

[Mar02] Grant Martin. UML for Embedded Systems Specification and Design: Motivation and Overview. *Design, Automation & Test in Europe (DATE)*, 2002.

[Mar07] Peter Marwedel. *Eingebettete Systeme.* Springer Verlag, 2007.

[Mat14] Mathworks. Matlab/Simulink, http://www.mathworks.com/products/ simulink/. Internet, 2014.

[MBAG11] Indika Meedeniya, Barbora Bunova, Aldeida Aleti, and Lars Grunske. Reliability-driven Deployment optimization for embedded systems. In *Journal of Systems and Software*, pages 835–846. Elsevier, 2011.

[MDD+10] Sergio Montenegro, Frank Dannemann, Lutz Dittrich, Benjamin Vogel, Ulf Noyer, Jan Gacnik, Marco Hannibal, Andreas Richter, and Frank Köster. SpaceSpace BusController+AutomotiveECU = UltimateController. In Gregor Engels, Markus Luckey, Alexander Pretschner, and Ralf Reussner, editors, *Proceedings Software Engineering 2010*, pages 103–114. Gesellschaft für Informatik (GI), 2010.

[MdS08a] Frederic Mallet and Robert de Simone. MARTE: A Profile for RT/E Systems Modeling, Analysis – and SSimulation? In *1st Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools)*, 2008.

[MDS08b] Mathias Maurmaier, Kristian Dencovski, and Engelbert Schmitz. Engineering Challenges - Evaluierungskonzept für Engineering-Werkzeuge. *atp-online*, 1 2008.

[Mey14] Jan Meyer. Die Ergänzung des AUTOSAR Standards für eine durchgängige modellbasierte automobile Steuergeräteentwicklung. In *Modellbasierte Entwicklung eingebetteter Systeme (MBEES) Workshop*, 2014.

[MGRK07] Klaus D. Müller-Glaser, Clemens Reichmann, and Markus Kuehl. Supporting System Level Design of Distributed Real Time Systems for Automotive Applications. *Composition of Embedded Systems. Scientific and Industrial Issues*, 2007.

[MGRKB04] Klaus D. Mueller-Glaser, Clemens Reichmann, Markus Kuehl, and Stefan Benz. Quality Assurance and Certification of Software Modules in Safety Critical Automotive Electronic Control Units Using a CASE Tool Integration Platform. *ASWSD*, pages 15 – 30, 2004.

[MH13] Jan Meyer and Wilfried Horn. Modellbasiertes SystemeSystemeng zur Qualitätsverbesserung bei der Entwicklung eines automobilen Steuergerätes. In Maik Maurer and Sven-Olaf Schulze, editors, *Tag des Systems Engineering (TdSE 2013)*, pages 315 – 324. Gesellschaft für Systems Engineering, Carl Hanser Verlag, 2013.

[MHDZ07] Markus Müller, Klaus Hörmann, Lars Dittmann, and Jörg Zimmer. *AutomotiveSPICE in der Praxis*. dPunkt Verlag, 2007.

[Mic14a] Microsoft. PowerPoint, http://office.microsoft.com/de-de/powerpoint/. Internet, 2014.

[Mic14b] Microsoft. Visio, http://office.microsoft.com/de-de/visio/. Internet, 2014.

[Möl09] Daniel Mölle. Design by Demut. *iX Zeitschrift - Heise Zeitschriftenverlag*, 2009.

[Moo02] Alan Moore. Extending the RT Profile to Support the OSEK Infrastructure. In *Proceedings of the 5th International Symposium on Object-Oriented Real-Time Distributed Computing*, 2002.

[MRZ+05] Joseph F. Maranzano, Sandra A. Rozsypal, Gus H. Zimmerman, Guy W. Warneken, Patricia E. Wirth, and David M. Weiss. Architecture Review: Practice and Experience. In *IEEE Software*, 2005.

[MSW09] Andreas Mülder, Holger Schill, and Lothar Wendehals. Modellvergleich mit EMF Compare. *Eclipse Magazin*, 2009.

[Mut05] Martin Mutz. *Eine Durchgängige modellbasierte Entwurfsmethodik für eingebettete Systeme im Automobilbereich*. Cullivier Verlag Göttingen, 2005.

[Nat07] Marco Di Natale. Virtual Platforms and Timing Analysis: Status, Challenges and Future Directions. *Design Automation Conference*, 2007.

[Nat08] Marco Di Natale. Design and Development of Component-Based Embedded Systems for Automotive Applications. In Fabrice Kordon and Tullio Vardanega, editors, *Reliable Software Technologies - Ada-Europe 2008*, volume 5026 of *Lecture Notes in Computer Science*, pages 15–29. Springer Berlin Heidelberg, 2008.

[NHB05] Thomas Nolte, Hans Hansson, and Lucia Lo Bello. Automotive Communications - Past, Current and Future. In *Proceedings of 10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2005.

[NMK10] Ulrich Nickel, Jan Meyer, and Tapio Kramer. Wie hoch ist die Performance? *Automobil-Elektronik*, 03:36 – 38, Juni 2010.

[Noe05] Tammy Noergaard. *Embedded Systems Architecture*. Newnes, 2005.

[NS08] Oliver Niggemann and Joachim Stroop. Models for Model's Sake. In *Proceedings of the International Conference on Software Engineering (ICSE)*, 2008.

[NSH+10] Florian Nafz, Hella Seebach, Jörg Holtmann, Jan Meyer, Matthias Tichy, Wolfgang Reif, and Wilhelm Schäfer. Designing Self-Healing in Automotive Systems. In *Proc. of the 7th International Conference on Autonomic and Trusted Computing (ATC 2010), Xi'an, China, 26-29 October, 2010*, volume 6407, pages 47–61. Lecture Notes in Computer Science, Springer Verlag, 2010.

[NSL08] Nicolas Navet and Francoise Simonot-Lion. *Automotive Embedded Systems Handbook*. CRC Press, 2008.

[NSV10] Marco Di Natale and Alberto Luigi Sangiovanni-Vincentelli. Moving from Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools. *Proceedings of the IEEE - Special Issue on Aerospace & Automotive*, 2010.

[NT09] Joaquin Nicolas and Ambrosio Toval. On the generation of requirements specifications from software engineering models: A systematic literature review. *Information and Software Technology*, (51):1291–1307, 2009.

[oMVMO06] The International Organization of Motor Vehicle Manufacturers (OICA). The World's Automotive Industry. Internet, 2006.

[Ope09] OpenArchitectureWare. OpenOpenArchitecture 4.3.1, http://www.openarchitectureware.org/index.php. Internet, 2009.

[Ora14] Oracle. JAVA, http://www.java.com/de/. Internet, 2014.

[OSE05] OSEK. OSEK OS Specification Version 2.2.3. Technical report, OSEK-VDX Konsortium, 2005.

[Ost87] Leon Osterweil. Software Processes are Software Too. In *Proceedings of the 9th International Conference on Software Engineering (ICSE)*, 1987.

[OT10] F. Ougier and F. Terrier. Eclipse based architecture of the EDONA platform for automotive system development. *Conference Embedded Real Time Software and Systems (ERTS)*, 2010.

[PA12] J-D. Piques and E. Andrianarison. SysML for embedded automotive Systems: lessons learned. *Embedded Real Time Software and Systems (ERTS)*, 2012.

[PBFG06] Gerhard Pahl, Wolfgang Beitz, Jörg Feldhusen, and Karl-Heinrich Grote. *Pahl/Beitz KonstrKonstruktion: Grundlagen erfolgreicher Produktentwicklkung.Methoden und Anwendung.* Springer Verlag, 2006.

[PBK10] Venkatesh Prasad, Manfred Broy, and Ingolf Krueger. Scanning Advances in Aerospace & Automobile Software Technology. *Proceedings of the IEEE - Special Issue on Aerospace & Automotive*, 2010.

[PBKS07] Alexander Pretschner, Manfred Broy, Ingolf H. Krüger, and Thomas Stauner. Software Engineering for Automotive Systems: A Roadmap. *Future of Software Engineering (FOSE'07)*, 2007.

[PFS08] Marie-Agnes Peraldi-Frati and Yves Sorel. From high-level modelling of time in MARTE to real-time scheduling analysis. *MoDELS 2008*, 2008.

[PHAB12] Klaus Pohl, Harald Hönninger, Reinhold Achatz, and Manfred Broy. *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology.* Springer Verlag, 2012.

[PK04] Alexander Puschnig and Ramin Tavakoli Kolagari. Requirements Engineering in the Development of Innovative Automotive Embedded Software Systems. *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE)*, 2004.

[PMP11] Amalinda Post, Igor Menzel, and Andreas Podelski. Applying Restricted English Grammar on Automotive Requirements - Does it Work? A Case Study. In D. Berry and X. Franch, editors, *Proceedings of Working Conference on Requirements Engineering (REFSQ)*, volume LNCS 6606, pages 166–180. Springer Verlag, 2011.

[POFG05] Georg Pelz, Peter Oehler, Eliane Fourgeau, and Christoph Grimm. Automotive System Design and AUTOSAR. In P. Boulet, editor, *Advances in Design and Specification Languages for SoCs*, pages 293 – 305. Springer Verlag, 2005.

[PPE+08] Traian Pop, Paul Pop, Petru Eles, Zebo Peng, and Alexandru Andreei. Timing analysis of the FlexRay communication protocol. *Real Time Systems*, 2008.

[PR09] Klaus Pohl and Chris Rupp. *Basiswissen Requirements Engineering*. dpunkt.verlag, 2009.

[Pro09] Timmo Projekt. TIMMO Projekt (TIMing MOdel) v2.12, 2009.

[Pro11] Edona Project. The Edona Project. Internet, 2011.

[Pro14] CESAR Project. Cost-efficient methods and processes for safety relevant embedded systems (CESAR), http://www.cesarproject.eu/. Internet, 2014.

[PS02] Peter Prüfer and Angelika Stiegler. Die Durchführung standardisierter Interviews: Ein Leitfaden, http://www.gesis.org/fileadmin/upload/forschung/publikationen/gesis_reihen/howto/How-to11ppas.pdf. Internet, 2002.

[PS09] Klaus Pohl and Ernst Sikora. Cosmod-RE: Verzahnung des Architekturentwurfs mit dem Requirements Engineering. *Objektspektrum*, 2009.

[PS12] Carsten Pflug and Dirk Schüpferling. Von Missverständnissen und guten Kooperationen. In *JavaSpektrum*, volume 6, pages 48 – 51, 2012.

[PSSZ12] Heike Proff, Jörg Schönharting, Dieter Schramm, and Jürgen Ziegler. *Zukünftige Entwicklungen in der Mobilität Betriebswirtschaftliche und technische Aspekte*. Springer Gabler Verlag, 2012.

[PSW10] Sangsoo Park, Kang Geuh Shin, and Shige Wang. Integration of Collorative Analyses for Development of Embedded Control Software. *Proceedings of the IEEE*, 2010.

[QCLT11] Tahir Naseer Qureshi, DeJiu Chen, Henrik Lönn, and Martin Törngren. From EAST-ADL to AUTOSAR Software Architecture: A Mapping Scheme. In Ivica Crnkovic, Volker Gruhn, and Matthias Book, editors, *Proceedings 5th European Conference, ECSA*, pages 328 – 335. Springer Verlag, 2011.

[QTP+11] Tahir Naseer Qureshi, Martin Törngren, Magnus Pessson, De-Jiu Chen, and Carl-Johan Sjöstedt. Towards Harmonizing MultipleArchitecture Description Languages for Real-Time Embedded Systems. In *Real-Time in Sweden (RTiS)*, 2011.

[Rau07] Mathias Rausch. *FlexRay - Grundlagen, Funktionsweise, Anwendung*. Hanser Verlag, 2007.

[RBvdBS02] Martin Rappl, Peter Braun, Dr. Michael von der Beeck, and Dr. Christian Schröder. Automotive Software Development: A Model Based Approach. *Society of Automotive Engineers, Inc.*, 2002.

[Rei10a] Konrad Reif. *Bosch Autoelektrik und Autoelektronik: Bordnetze, Sensoren und elektronische Syteme*. Vieweg + Teubner, 2010.

[Rei10b] Konrad Reif. *Fahrstabilisierungssysteme und Fahrerassistenzsysteme*. Vieweg + Teubner, 2010.

[RER06] Tazvan Racu, Rolf Ernst, and Kai Richter. The Need of a Timing Model for the AUTOSAR software standard. *Workshop on Models and Analysis Methods for Automotive Systems (RTSS Conference), (Rio de Janeiro, Brazil)*, 2006.

[RH94] Jock Rader and Leslie Haggerty. Supporting Systems Engineering with Methods and Tools: A Case Study. Technical report, Hughes Aircraft Company and H&A System Engineering, 1994.

[RHER07] Razvan Racu, Arne Hamann, Rolf Ernst, and Kai Richter. Automotive Software Integration. *DAC*, 2007.

[RLE02] Martin Rohdin, Lars Ljungberg, and Ulrik Eklund. A Method for Model Based Automotive Software Development. *Proc. Work in Progress and Industrial Experience Sessions, 12th Euromicro Conference on Real-Time Systems*, pages 15–18, 2002.

[ROH+07] Michael Rudorfer, Tilman Ochs, Paul Hoser, Martin Thiede, Martin Mössmer, Oliver Scheickl, and Harald Heinecke. Realtime System Design Utilizing AUTOSAR Methodology. *ERTS Embedded Real time software*, 2007.

[Ros77] D. T. Ross. Structured Analysis (SA) A Language for Communicating Ideas. In *IEEE Transactions on Software*, volume SE-3 of *1*, pages 16–34, 1977.

[Roy70] William W. Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the IEEE WESCON*, 1970.

[RP10] Anja Ranft and Andre Pflüger. Strukturierte Modellieren: Ein Modell ist mehr als die Summe seiner Diagramme. *Objektspektrum*, 6:43–49, 2010.

[RQZ07] Chris Rupp, Stefan Queins, and Barbara Zengler. *UML 2 Glasklar*. Hanser Verlag, 2007.

[RR09] Chris Rupp and Anja Ranft. Spezifikation jenseits der grünen Wiese. *Elektronik Praxis - Embedded Software Engineering Report*, pages 10–13, 2009.

[SA07] Sebastian Schlecht and Oliver Alt. Strategien zur Testfallgenerierung aus SysML Modellen. *Software Engineering*, 2007.

[SB09] Renate Stücka and Rick Bolt. Verringerter Aufwand. In *Elektronik*, volume 19, pages 34 – 39. www.elektroniknet.de, 2009.

[SBC+13] Gehan M. K. Selim, Fabian Büttner, James R. Cordy, Juergen Dingel, and Shige Wang. Automated Verification of Model Transformations in the Automotive Industry. In Ana Moreira, Bernhard Schätz, Jeff Gray, Antonio Vallecillo, and Peter Clarke, editors, *Model-Driven Engineering Languages and Systems (MODELS)*, number LNCS 8107, pages 690–706. Springer Verlag, 2013.

[SBG+10] David C. Sharp, Alex E. Bell, Jeffrey J. Gold, Ken W. Gibbar, Dennis W. Gvillo, Vann M. Knight, Kevin P. Murphy, Wendy C. Roll, Radhakrishna G. Sampigethaya, Viswa Santhanam, and Steven P. Weismuller. Challenges and Solutions for Embedded and Networked Aerospace Software Systems. *Proceedings of the IEEE - Special Issue on Aerospace & Automotive*, 2010.

[Sch05a] Wilhelm Schäfer. A Rigorous Software Process for the Development of Embedded Systems. *Unifying the Software Process Spectrum*, 3840/2006, 2005.

[Sch05b] Peter Scholz. *Softwareentwicklung eingebetteter Systeme*. Springer Verlag, 2005.

[Sch11] Matthias Schnelte. Automatische Testfallgenerierung aus kontrolliert natürlichsprachlichen Anforderungsspezifikationen für reaktive Echtzeitsysteme, 2011.

[SDW+10] Alexander Schatten, Markus Demolsky, Dietmar Winkler, Stefan Biffl, Erik Gostischa-Franta, and Thomas Östreicher. *Best Practice Software-Engineering*. Springer Verlag, 2010.

[Sei09a] Michael Seibt. Architekturmodellierung mit East-Adl2 und AUTOSAR. *Carl Hanser Verlag Automotive Journal*, 2009.

[Sei09b] Reinhard Seiffert. *Die Ära Gottlieb Daimlers*. Vieweg + Teubner, 2009.

[SEI10] Capability Maturity Model Integration (CMMI), 2010.

[Sel07] Bran Selic. A Systematic Approach to Domain-Specific Language Design Using UML. In *Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 2007.

[SF11] Markus Schöttle and Elmar Frickenstein. Autosar sorgt für einen Paukenschlag. *ATZ online*, 2011.

[SFB09] Sonderforschungsbereich 614 SSelbstoptimierende Systeme des Maschinenbaus", www.sfb614.de, 2009.

[SFGP05] Bernhard Schätz, Andreas Fleischmann, Eva Geisberger, and Markus Pister. Modellbasierte Anforderungsentwicklung mit AutoRAID. In *Proceeding of the Object-Oriented Software-Engineering (OOSE). Net.ObjectDays-Conference*, 2005.

[SGG91] Virginia Satir, Maria Gomori, and Jane Gerber. *The Satir Model: Family Therapy and Beyond*. Science and Behavior Books, 1991.

[SH09] Gernot Starke and Peter Hruschka. *Software-Architektur kompakt - angemessen und zielorientiert*. Spektrum Akademischer Verlag, 2009.

[SHI+11] R. Stark, H. Hayka, J.H. Israel, M. Kim, P. Müller, and U. Völlinger. Virtuelle Produktentstehung in der Automobilindustrie. In *Informatik Spektrum - Informatik in der Automobilindustrie*, volume 34. Gesellschaft für Informatik, Gesellschaft für Informatik, 2011.

[SI07] Ken Schwaber and Thomas Irlbeck. *Agiles Projektmanagement mit SCRUM*. Microsoft Press, 2007.

[SKB+09] Oliver Sandner, A. Klimm, J. E. Becker, J. Becker, T. Kimmeskamp, J. Formann, K. Echtle, K. Weinberger, and S. Bulach. Sicherung von Zuverlässigkeit und Interoperabilität bei der fahrzeugfahrzeug Kommunikation mittels formaler Verifikation. *VDI-Berichte Nr 2075*, pages 345–356, 2009.

[SKFS07] R. Siwy, N. Kloss, S. Fürst, and H. Schmid. Modellothek: Modellbasierte Entwicklung von wieder verwendbaren AUTOSAR SW-Komponenten für Karosseriefunktionen. *VDI Berichte Nr. 1907(Baden-Baden)*, 2007.

[SKS10] Kathrin D. Scheidemann, Michael Knapp, and Claus Stellwag. Load Balancing in AUTOSAR Multicore-Systemen. *Elektronik Automotive*, 2010.

[Smi13] Steven M. Smith. The Satir Change Model, http://stevensmith.com/ar-satir-change-model. Internet, 2013.

[Som07] Ian Sommerville. *Software Engineering*. Pearson Studium, 2007.

[SP10] Ernst Sikora and Klaus Pohl. Evaluation eines Modellbasierten Requirements-Engineering-Ansatzes für den Einsatz in der Motorsteuerungs-Domäne. In Gregor Engels, Markus Luckey, and Alexander Pretschner, editors, *Proceedings Software Engineering - Workshop ENVISION2020*, pages 127–136, 2010.

[SPC+13] Takashi Sakairi, Eldad Palachi, Chaim Cohen, Yoichi Hatsutori, Junya Shimizu, and Hisashi Miyashita. Model Based Control System Design Using SysML, Simulink, and Computer Algebra System. *Journal of Control Science and Engineering*, Volume 2013, 2013.

[SPE09] SPES. Software Plattform Embedded Systems (SPES2020), www.spes2020.de, 2009.

[SR08a] Oliver Scheickl and Michael Rudorfer. Automotive Real Time Development Using a Timing-augmented AUTOSAR Specification. *ERTS Embedded Real time software*, 2008.

[SR08b] Ulrich Seiffert and Gotthard Rainer. *Virtuelle Produktentstehung für Fahrzeug und Antrieb im Kfz*. Vieweg+Teubner, 2008.

[SRM13] Michael Soden, Andreas Redtenbacher, and Michael Maruhn. Modellgetrieben zum technischen Sicherheitskonzept. *Elektroni-IEEE Software - funktionale Sicherheit*, 2013.

[SRT+05] Chr. Salzmann, M. Rudorfer, M. Thiede, T. Ochs, P. Hoser, M. Mössmer, H. Heinecke, and A. Münnich. Erfahrungen mit der technischen Anwendung einer AUTOSAR Runtime Environment - Migrationsstrategien in die AUTOSAR Architektur -. 2005.

[ST12] T. Streichert and M. Traub. *Elektrik/Elektronik-Architekturen im Kraftfahrzeug: Modellierung und Bewertung von Echtzeitsystemen*. Springer Verlag, 2012.

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer Verlag, 1973.

[Sta09a] Michael Stal. Das beste Rezept. *IX*, 2009.

[Sta09b] Gernot Starke. *Effektive Software-Architekturen.* Carl Hanser Verlag, 2009.

[STBW05] Th. Saul, A. Titze, G. Bikker, and W.Lawrenz. Grafische Formalisierung von Elektronik-Systemen im Kfz zur Bewertung von Architekturen in frühen Entwicklungsphasen. volume 1, pages 535–562, 2005.

[Sto96] Neil Storey. *Safety-Critical Computer Systems.* Pearson Studium, 1996.

[SVN07] Alberto Sangiovanni-Vincentelli and Marco Di Natale. Embedded System Design for Automotive Applications. *IEEE Computer*, 40:42 – 51, Oktober 2007.

[SW00] Connie U. Smith and Lloyd G. Williams. Software Performance AntiPatterns. In *Proceedings 2nd International Workshop on Software and Performance*, 2000.

[SW07] Wilhelm Schäfer and Heike Wehrheim. The Challenges of Building Advanced Mechatronic Systems. *FOSE*, 2007.

[SWCD13] Gehan M. K. Selim, Shige Wang, James R. Cordy, and Jürgen Dingel. Model transformations for migrating legacy deployment models in the automotive Industry. In *Journal of Software and Systems Modeling.* Springer Verlag, 2013.

[Sym14] Symtavision GmbH. SymTA/S, http://www.symtavision.com. Internet, 2014.

[Sys14] Sparx Systems. Enterprise Architect, http://www.sparxsystems.com/. Internet, 2014.

[SZ13] Jörg Schäuffele and Thomas Zurawka. *Automotive Software Engineering (5.Auflage).* Vieweg+Teubner, 2013.

[Tan02] Andrew S. Tanenbaum. *Moderne Betriebssysteme.* Pearson Studium, 2 edition, 2002.

[TB03] Dave Thomas and Brian M. Barry. Model Driven Development - The Case for Domain Oriented Programming. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '03, pages 2–7. ACM, 2003.

[Teu09] Stefan Teuchert. Funktionsorientierter Architektur-Entwurf am Beispiel eines Hybrid-Busses. *VDI Berichte 2075*, 2009.

[TGDT08] Frederic Thomas, Sebastien Gerard, Jerome Delatour, and Francois Terrier. Software Real-Time Resource Modeling. In Eugenio Villar, editor, *Embedded Systems Specification and Design Languages*, pages 169–182. Springer Verlag, 2008.

[Tho04] Dave Thomas. MDA: Revenge of the ModModel or UML Utopia? *IEEE Software*, 2004.

[THP+07] Wolfgang Trumler, Markus Helbig, Andreas Pietzowski, Benjamin Satzger, and Theo Ungerer. Self-configuration and Self-healing in AUTOSAR. In *Proceedings of 14th Asia Pacific Automotive Engineering Conference (APAC)*, 2007.

[TK05] Juha-Pekka Tolvanen and Steven Kelly. Defining Domain specific model Languages to Automate Product Derivation. *Mentor Graphics*, 2005.

[TKL11] Mario Trapp, Thomas Kuhn, and Peter Liggesmeyer. Modellbasierte Entwicklung eingebetteter Softwaresysteme. *Elektronik Praxis - Embedded Software Engineering Report*, pages 14–15, 2011.

[Tol08] Juha-Pekka Tolvanen. Domänenspezifische Modellierung in der Praxis. In *OBJEKTSpektrum*, volume 4, pages 38–40. SIGS Datacom, 2008.

[Tra10] Matthias Traub. *Durchgängige Timing-Bewertung von Vernetzungsarchitekturen und Gateway-Systemen im Kraftfahrzeug.* PhD thesis, Karlsruher Institut für Technologie (KIT), 2010.

[TRGD07] Afouan Taha, Ansgar Rademacher, Sebastien Gerard, and Jean-Luc Dekeyser. An Open Framework for DeDetail Hardware Modeling. In *Proceedings of International Symposium on Industrial Embedded Systems (SIES)*, 2007.

[USJ08] Philipp Ulbrich, Martin Schmuck, and Lutz Jäde. Working Capital Management in der Automobilindustrie - Eine Betrachtung der Schnittstelle zwischen OEM und Zulieferer. *Controlling & Management*, 52(1), 2008.

[vdB06] Michael von der Beeck. Development of logical and technical architectures for automotive systems. *Software System Model*, 2006.

[VG07] Markus Völter and Iris Groher. MoModelltransformation in der Praxis. In *JavaSpektrum*, volume 2, pages 13–17. SI, 2007.

[VHG+04] J.-U. Varchmin, M. Harms, U. Goltz, M. Huhn, M. Mutz, E. Schnieder, M. Horstmann, K. Lange and. C. Krömke, and G. Bikker. Strukturierter Entwicklungsprozess für eingebettete Systeme - vom Lastenheft zur automatischen Codegenerierung. 2004.

[Wat07] Andrew Watson. UML vs. DSLs: A false dichotomy. Technical report, Object Management Group, 2007.

[WC01] Bernhard Westfechtel and Reidar Conradi. Software Architecture and Software Configuration Management. *10th International Workshop on software Configuration Management*, 2001.

[Wei08] Tim Weilkiens. *Systems Engineering mit SysML/UML*. 2. Auflage. dpunkt-Verlag, 2008.

[WFA07] Peter Wallin, Joakim Fröberg, and Jakob Axelsson. Making Decisions in Integration of Automotive Software and Electronics: A Method Based on ATAM and AHP. *ICSE*, 2007.

[WFH+06] D. Wild, A. Fleischmann, J. Hartmann, C. Pfaller, M. Rappl, and S. Rittmann. An Architecture-Centric Approach towards the Construction of Dependable Automotive Software. *SAE 2006*, 2006.

[WFO09] Henning Wallentowitz, Arndt Freialdenhoven, and Ingo Olschewski. *Strategien in der Automobilindustrie*. Vieweg + Teubner, 2009.

[WHR+13] Jon Whittle, John Hutchinson, Mark Rouncefield, Hakan Burden, and Rogardt Heldal. Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem? In Ana Moreira, Bernhard Schätz, Jeff Gray, Antonio Vallecillo, and Peter Clarke, editors, *Model-Driven Engineering Languages and Systems (MODELS)*, number LNCS 8107, pages 1–17. Springer Verlag, 2013.

[WHW10] Hermann Winner, Stephan Hakuli, and Gabriele Wolf. *Handbuch Fahrerassistenzsysteme*. Vieweg + Teubner, 2010.

[Win05] Hans Windpassinger. AUTOSAR Methodology - Erfolgskonzept für Kfz-Software. *Automotive Journal*, 2005.

[Win08] Marco Winzker. *Elektronik für Entscheider*, volume 7. Springer Verlag, 2008.

[WK04] Jos Warmer and Anneke Kleppe. *Object Constraint Language 2.0*. mitp-Verlag, 2004.

[WMK+09] Andreas Wolfram, Mikahail Makarov, Tapio Kramer, Wendel Ramisch, and Ralf Münzenberger. Design of Robust System Architectures for Automotive ECUs. In *Proceedings of Conference on Quality Engineering in Software Technology (CONQUEST)*, 2009.

[WTS08] Wolfram Webers, Christer Thörn, and Kurt Sandkuhl. Connecting Feature Models and AUTOSAR: An Approach Supporting Requirements Engineering in Automotive Industries. *International Working Conference on Requirements Engineering (REFSQ)*, 2008.

[WW03] Matthias Weber and Joachim Weisbrod. Requirements Engineering in Automotive Development: Experiences and Challenges. *IEEE Software*, 20:16 – 24, 2003.

[Yin09] Robert K. Yin. *Case Study Research: Design and Methods (5.Auflage)*. SAGE, 2009.

[ZBF+05] Dirk Ziegenbein, Peter Braun, Ulrich Freund, Andreas Bauer, Jan Romberg, and Bernhard Schätz. AutoMoDe Model-Based Development of Automotive Software. volume 3, pages 171 – 177, 2005.

[ZHLK10] Andreas Zauner, Holger Hoffmann, Jan Marco Leimeister, and Helmut Krcmar. Automotive Software und Service Engineering (ASSE) - Eine Exploration von Herausforderungen und Trends aus Sicht von Branchenexperten. In *Proceedings of Multikonferenz Wirtschaftsinformatik (MKWI), Göttingen*, 2010.

[Zör12] Stefan Zörner. *Softwarearchitekturen dokumentieren und kommunizieren*. Hanser Verlag, 2012.

[ZS08] Werner Zimmermann and Ralf Schmidgall. *Bussysteme in der Fahrzeugtechnik*. Vieweg + Teubner Verlag, 2008.