Entwurfsmethodik für hybride Software- und Systemarchitektur

Zur Erlangung des akademischen Grades eines
DOKTORS DER INGENIEURWISSENSCHAFTEN (Dr.-Ing.)
von der KIT-Fakultät für Elektrotechnik und Informationstechnik des
Karlsruher Instituts für Technologie (KIT)
angenommene

DISSERTATION

von

M.Sc. Philipp Obergfell

Tag der mündlichen Prüfung:	14.06.2021
Referent:	Prof. Dr. Eric Sax
Korreferent:	Prof. Dr. Kristin Paetzold

Kurzfassung

Die Softwaretechnik gewinnt in der Automobilindustrie durch Trends wie das automatisierte Fahren und die Vernetzung von Fahrzeugen mit der Infrastruktur weiter an Bedeutung. Software-basierte Funktionen müssen dabei neben der Implementierung und dem Test in einer frühen Phase der Entwicklung im Rahmen eines gemeinsamen Systems aus Softwarekomponenten und Steuergeräten entworfen werden. Die Spezifikation von Kommunikationsbeziehungen zwischen Softwarekomponenten wird heute über den Ansatz einer Signal-orientierten Architektur erreicht. Die vorliegende Arbeit analysiert das hierzu entsprechende Entwurfsmodell und entwickelt es zu einem hybriden Modell aus Signal- und Service-Orientierung weiter. Die Modellentwicklung wird weitergehend durch eine Entwurfsmethodik gestützt und im Rahmen des Systementwurfs in das V-Modell eingebunden. Hierdurch entsteht eine Alternative zur heutigen Spezifikation von Kommunikationsbeziehungen auf Basis von Signalen durch Services. Speziell bei Änderungen tragen diese den Vorteil, dass Folgeanpassungen an den Kommunikationsschnittstellen eines Steuergeräts reduziert werden. Die Softwareimplementierung, welche heute durch eine strikte Synchronisation mit der Entwicklung und Weiterentwicklung von Steuergeräten beeinflusst ist, wird dadurch vereinfacht.

Abstract

Automotive software increases continuously its importance due to trends such as automated driving and connectivity. Besides implementing and testing software-based functions, architectural blueprints depicting dependencies between software components and ECUs are relevant at an early stage of development. For specifying communication links among software components, signals as information carriers represent the status quo. This thesis analyses the corresponding communication concept and proposes its redesign. The main impact is the inclusion of services as alternative information carriers within one hybrid approach. In order to realise the approach, a corresponding methodology is provided and aligned with the V-model. On this basis, the advantage of services in respect of hardware independent software changes becomes available and omits today's necessity to update software and ECUs simultaneously.

Danksagung

Meinen Dank richte ich zuerst an Herrn Prof. Dr. Sax, der mir die Möglichkeit und das Vertrauen für die Dissertation gegeben hat. Die Betreuung durch kontinuierliches Lesen von Ausarbeitungen und die darauffolgenden Diskussionen waren eine große Unterstützung besonders zum Ende der Arbeit. Ich danke Frau Prof. Dr. Paetzold für die Übernahme des Zweitgutachtens und die Diskussion der Arbeit.

Ich bin im Speziellen Herrn Dr. Matthias Traub und Herrn Dr. Wolfgang Hopfensitz, mit denen ich seit dem Ende meines Masterstudiums zusammenarbeiten durfte und die den Rahmen der Arbeit bei BMW gelegt haben, dankbar. Weitergehend danke ich für eine vielschichtig aufgestellte Arbeitsgruppe am ITIV und für die Zusammenarbeit im Rahmen des Projekts OSBORNE bei BMW. Zuletzt danke ich meinen Eltern und meinen Geschwistern für ihre Unterstützung auf dem Weg bis hierher.

München, im Januar 2021 *Philipp Obergfell*

Inhaltsverzeichnis

E/E Elektrik- und Elektronik

DIN Deutsche Industrienorm

ISO Internationale Organisation für Normung

OMG Object Management Group

MOF Meta Object Facility

ECU Electronic Control Unit

UML Unified Modeling Language

DSL Domain Specific Language

ADL Architecture Description Language

IEEE Institute of Electrical and Electronics Engineers

MARTE Modeling and Analysis of Real-Time and Embedded Systems

VDI Verband Deutscher Ingenieure

XML Extensible Markup Language

EEA-ADL Electric Electronic Architecture - Analysis Design Language

EAST-ADL Electronics Architecture and Software Technology-Architecture Description Language

SymTA/S Symbolische Laufzeitanalyse für Systeme

SiL Software-in-the-Loop

HiL Hardware-in-the-Loop

AUTOSAR AUTomotive Open System ARchitecture

RTC Real-Time Calculus

SysML Systems Modeling Language

CE Consumer Electronics

SOME/IP Scalable service-Oriented MiddlewarE over IP

COM Communication

ID Identifikationsnummer

PDU Protocol Data Unit

IP Internet Protocol

OSI Open Systems Interconnection

CAN Controller Area Network

LIN Local Interconnect Network

MOST Media Oriented Systems Transport

SDN Software-defined Networking

UDP User Datagram Protocol

TCP Transmission Control Protocol

AVB Audio Video Bridging

Kbit Kilobit

Mbit Megabit

RAM Random Access Memory

SRAM Static Random Access Memory

DRAM Dynamic Random Access Memory

ASIL Automotive Safety Integrity Level

KiB Kibibyte

MiB Mebibyte

MHz Megahertz

DDS Data Distribution Service

SEIS Sicherheit in Eingebetteten IP-basierten Systemen

EPI European Processor Initiative

RISC Reduced Instruction Set Computer

SMT Satisfiability Modulo Theories

BCET Best Case Execution Time

AVET Average Execution Time

WCET Worst Case Execution Time

MAC Media Access Control

1 Einleitung

Im Folgenden sind zunächst der Einfluss der *Softwaretechnik* in der Fahrzeugentwicklung und der Beitrag einer *Software- und Systemarchitektur* als Hintergrund dieser Arbeit betrachtet.

1.1 Softwaretechnik im Fahrzeug

Funktionalität in Fahrzeugen erfährt seit Jahrzehnten eine steigende Ausrichtung auf Software als Technologie. Den Beginn dieser Entwicklung bildeten dabei Algorithmen zur Implementierung der Motorsteuerung in den 1970er Jahren. Anschließend in den 1980er und 1990er Jahren wurden erste Eingriffe in die Fahrdynamik eines Fahrzeugs basierend auf Software realisiert. Heute, in den 2000er Jahren, befinden sich nun Software-basierte Implementierungen von Fahrerassistenzsystemen und Funktionen im Zusammenspiel mit der Fahrzeugumwelt im Fokus [Seite 3 f.][104].

Prognose

In Oberklassefahrzeugen finden sich ungefähr 100 Millionen Zeilen Quellcode wieder und tragen somit maßgeblich zu dem Schlagwort eines “Computers auf Rädern” [Seite 128][85] bei. Speziell durch Weiterentwicklungen von Fahrerassistenzsystemen zu automatisierten Fahrfunktionen wird aktuell ein sprunghafter Anstieg im Hinblick auf die Erweiterung dieses Umfangs prognostiziert. Nur für die kommende Generation von Fahrzeugen in den 2020er Jahren ist ein Aufwand von dann mindestens 300 Millionen Zeilen Quellcode geschätzt [Seite 2][7].

Software- und Systemarchitektur

Software-basierte Funktionen in Fahrzeugen stellen im Wesentlichen eine Verteilung auf unterschiedlichen Steuergeräten dar (s. Bild 1.1).

Bild 1.1: Verteilte Software-basierte Funktion

Um entsprechende Zusammenhänge für alle Funktionen eines Fahrzeugs zu verstehen, ist der Entwurf einer Software- und Systemarchitektur ein wesentlicher Schritt. Die Hauptaufgabe der Software- und Systemarchitektur ist dabei, neben einer effizienten Allokation von Rechenressourcen eines Steuergeräts, die Festlegung von Informationsflüssen zwischen Softwarekomponenten in einem Kommunikationsnetzwerk.

Signal-Orientierung

Zur Realisierung dieser Informationsflüsse werden heute Signale als Informationsträger festgelegt und auf Netzwerknachrichten zwischen im Fahrzeug verteilten Steuergeräten abgebildet (s. Bild 1.2).

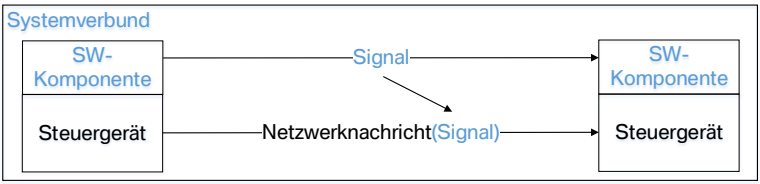

Bild 1.2: Signal-Orientierung: Verbund von Software und physischem Kommunikationsnetzwerk

Hierdurch entsteht die Konfiguration statischer, während der Entwicklungszeit festgelegter Kommunikationsbeziehungen, die einen *Systemverbund* zwischen der *Softwarearchitektur* und dem darunterliegenden *physischen Netzwerk* der Steuergeräte beschreiben.

1.2 Systemgrenze für Software

Aktuell im Fokus steht das Vorhaben ein Kommunikationskonzept für eine Softwarearchitektur zu etablieren, welches den Informationsfluss zwischen Softwarekomponenten vom Kommunikationskonzept des darunterliegenden physischen Netzwerks über Nachrichten entkoppelt (s. Bild 1.3).

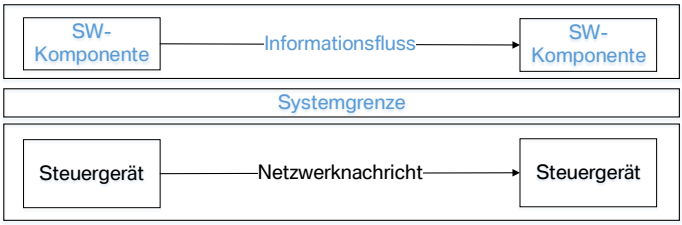

Bild 1.3: Trennung: Software und physisches Kommunikationsnetzwerk

Neben der noch dominanteren Rolle von Software in Bezug auf die Menge an Quellcode durch Trends wie das automatisierte Fahren sind zwei weitere wesentliche Gründe für dieses Vorhaben:

1. Der schnellere Entwicklungszyklus der *Consumer Electronics (CE)-Branche* [Seite 33 ff.][40], welcher beispielsweise durch die Einbindung des Smartphones in das Fahrzeug einen Einfluss auf darin umgesetzte Software ausübt und 2. der steigende Umfang an vernetzten Fahrzeugen [2], wodurch Schnittstellen zwischen dem Fahrzeug und umgebenden Systemen kontinuierlich synchronisiert werden müssen.

Für die genannten Zusammenhänge bietet eine von einem physischen Netzwerk entkoppelte Softwarearchitektur (s. Bild 1.3) das Potential zeitlich flexibel und ohne *Bindung* an die Entwicklungszyklen der *Steuergeräte* Änderungen zu gestalten. Schlagwörter für diesen in Verbindung mit der Automobilindustrie neuen Entwicklungszusammenhang sind der Begriff eines Softwareökosystems [Seite 265][75] oder einer Architektur für ein Softwareökosystem [Seite 1478 f.][32].

1.3 Vorgehen

Das Vorgehen dieser Arbeit unterteilt sich in die Kapitel 1-8 (s. Bild 1.4).

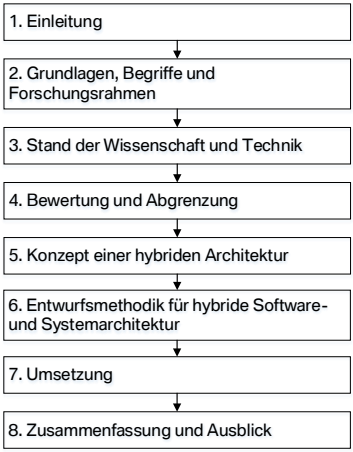

Bild 1.4: Vorgehen

Basierend auf dem skizzierten Begriff einer Software- und Systemarchitektur (s. Kapitel 1) werden zunächst *Grundlagen*, *Begriffe* und der *Forschungsrahmen* geordnet (s. Kapitel 2). Anschließend ist der *Stand der Wissenschaft und Technik* für den Entwurf einer heutigen Software- und Systemarchitektur dargelegt (s. Kapitel 3) und mittels einer *Bewertung* mit einem Potential

zur *Abgrenzung* belegt (s. Kapitel 4). Den Kern dieser Abgrenzung bildet ein *konzeptioneller* Rahmen für eine künftige Software- und Systemarchitektur durch Ergänzung des Signal-orientierten Kommunikationskonzepts um das der *Service-Orientierung* (s. Kapitel 5). Speziell für deren Entwurf wird eine *Methodik* als Ergebnis präsentiert (s. Kapitel 6) und im Rahmen eines modellbasierten Ansatzes zur Umsetzung gebracht (s. Kapitel 7). Abschließend ist der wissenschaftliche Beitrag der Arbeit zusammengefasst und mittels eines Ausblicks ergänzt (s. Kapitel 8).

2 Grundlagen, Begriffe und Forschungsrahmen

Als begriffliche Grundlagen werden das eingebettete System, das Automobile Elektrik- und Elektronik (E/E)-System und eine E/E-Architektur eingeführt. Als Kern des Kommunikationskonzepts einer E/E-Architektur gilt schlussendlich die Software- und Systemarchitektur, deren Realisierung über die Ansätze von Signal-Orientierung und Service-Orientierung betrachtet wird.

2.1 Eingebettete Systeme

Nach Steusloff [Seite 7][105] stellt ein eingebettetes System[1] ein informationsverarbeitendes System dar, welches unter Nutzung informationstechnologischer Mittel in seine Anwendungsumgebungen physisch integriert ist und dessen Eigenschaften durch diese Anwendungsumgebungen bestimmt sind.

Eine Möglichkeit ein eingebettetes System im Zusammenspiel mit seiner Anwendungsumgebung zu beschreiben, ist über das Konzept eines Wirkzusammenhangs zwischen System und Systemumgebung gezeigt (s. Bild 2.1).

[1] Ein eingebettetes System wird auch als reaktives System bezeichnet [Seite 17][78].

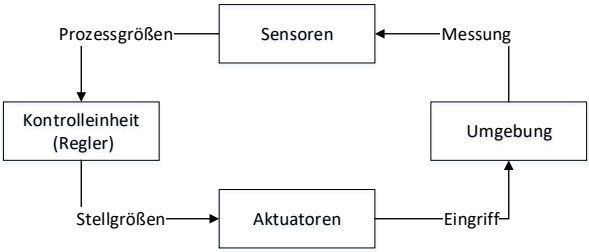

Bild 2.1: Eingebettetes System nach Siemers [Seite 9][101]

Die Pfeile im gezeigten Wirkzusammenhang stellen Kommunikationskanäle zur Regelung oder Steuerung eines Prozesses, welcher über Sensoren und Aktuatoren an eine Kontrolleinheit gekoppelt ist, dar. Eine Kontrolleinheit ist im Regelfall ein programmierbares System, das aus einer Kombination von Hardware und Software besteht [Seite 10][101]. In diesem Zusammenhang wird auch von einem Steuergerät (engl. Electronic Control Unit (ECU)) [Seite 7][101] gesprochen.

2.1.1 Eingebettete echtzeitfähige Systeme

Eine spezifische Ausprägung eines eingebetteten Systems ist das eingebettete echtzeitfähige System. Eingebettete echtzeitfähige Systeme können gemäß Wörn [Seite 1][117] von allgemeinen Computern (z.B. Büro-Computern) abgegrenzt werden. Dabei ist das hauptsächliche Unterscheidungsmerkmal die Definition von zeitlichen Bedingungen, welche neben der korrekten Ausführung einer Rechenoperation erfüllt sein müssen.

Die Deutsche Industrienorm (DIN) 44300 [111] beschreibt den Begriff Echtzeit wie folgt.

Definition 1. *Echtzeit beschreibt den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit und*

die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können einer zeitlich zufälligen Verteilung folgen oder zu vorherbestimmten Zeitpunkten anfallen.

Echtzeitfähige Systeme können in harte und weiche echtzeitfähige Systeme unterschieden werden. Harte echtzeitfähige Systeme beschreiben Systeme, die bei Nichteinhaltung von Echtzeitanforderungen einen Schaden für Menschen oder Maschinen hervorrufen können [Seite 16][101]. Weiche echtzeitfähige Systeme weisen diese Eigenschaft in Bezug auf mögliche Gefährdungen von Menschen oder Maschinen bei einer Nichteinhaltung von Echtzeitanforderungen nicht auf [Seite 16][101].

2.1.2 E/E-Systeme in Automobilen

E/E-Systeme in Automobilen können auch als eingebettete Systeme bezeichnet werden [Seite 16][61]. Die Regelungs- oder Steuerungsaufgabe, die von einem oder mehreren eingebetteten System ausgeführt wird, lässt sich bei E/E-Systemen jedoch über den allgemeineren Begriff einer Funktion beschreiben. Der Begriff einer Funktion ist laut Jaensch [Seite 14 f.][61] wie folgt definiert:

Definition 2. *Eine Funktion beschreibt ein technisches Ein-/ Ausgabeverhalten auf ein Ereignis*[2] *oder auf kontinuierliche Eingangsgrößen*[3] *und ist unabhängig von ihrer physikalischen Implementierung. Dabei stellt eine Funktion eine abstrakte Beschreibung dar.*

Jedes E/E-System setzt mindestens eine Funktion um. Diese kann im Weiteren in einzelne Funktionsbeiträge unterteilt werden, die bestimmte Anteile

[2] Ein Ereignis ist eine diskrete Eingangsgröße, welche einen Zustandswechsel eines Systems hervorruft [Seite 3 f.][73].

[3] Ein kontinuierliches Eingangssignal ruft eine zeitkontinuierliche Änderung des Zustands hervor [Seite 3 f.][73].

einer Funktion darstellen. In diesem Zusammenhang wird auch von der Dekomposition einer Funktion gesprochen [Seite 49][37]. Der Übergang einer Funktion zu einem E/E-System ergibt sich durch Zuordnung der aus der Funktion abgeleiteten Funktionsbeiträge zu E/E-Komponenten. In Übereinstimmung mit Jaensch [Seite 14][61] ist eine E/E-Komponente wie folgt definiert.

Definition 3. *Eine E/E-Komponente ist eine elektrische und/ oder elektronische Komponente (auch als mechatronische Komponente in Verbindung mit der Mechanik), welche eine bestimmte Funktionalität im Fahrzeug umsetzt.*

2.2 E/E-Architekturen

Der Fachbegriff um die Vernetzung aller E/E-Komponenten eines Fahrzeugs zu strukturieren ist durch die E/E-Architektur gegeben.

2.2.1 Beschreibung einer E/E-Architektur

Zur Beschreibung einer E/E-Architektur werden unterschiedliche Ebenen herangezogen [Seite 16][106]. Die Ebenen sind nach ihrem Abstraktionsgrad geordnet und finden in einer hierarchischen Darstellung ihre Auflösung (s. Bild 2.2).

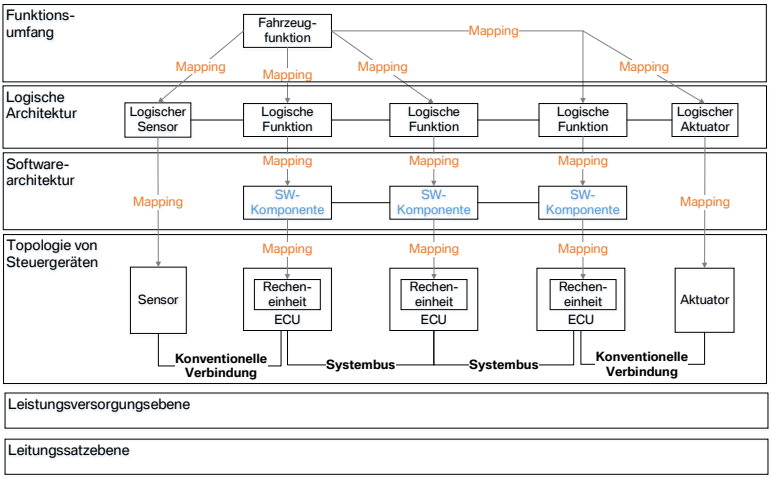

Bild 2.2: Artefakte und Ebenen einer E/E-Architektur in Anlehnung an Streichert und Traub [Seite 16][106]

Funktionsumfang

Der Funktionsumfang enthält die Menge der umzusetzenden Fahrzeugfunktionen einer E/E-Architektur. Als Funktion werden dabei Merkmale (engl.

Features) eines Fahrzeugs verstanden, die vom Kunden direkt oder indirekt wahrgenommen werden [Seite 2][99].

Logische Architektur

Um die Realisierung von Fahrzeugfunktionen zu beschreiben werden logische Funktionen, Sensoren und Aktuatoren herangezogen. Die Beziehungen zwischen diesen Elementen können Informationsflüsse, Energieflüsse und Stoffflüsse umfassen. Gemäß Pahl et al. [Seite 43][45] kann in diesem Zusammenhang auch von einer Funktionsstruktur gesprochen werden.

Softwarearchitektur

Die Realisierung von logischen Funktionen wird über Softwarekomponenten erreicht. Der Informationsfluss zwischen Softwarekomponenten realisiert dabei den Informationsfluss zwischen logischen Funktionen.

Topologie von Steuergeräten

Die Topologie[4] von Steuergeräten kennzeichnet die Partitionierung von Softwarekomponenten sowie logischen Sensoren und Aktuatoren auf Hardwarekomponenten. Die Kommunikation erfolgt dabei in digitalisierten Nachrichten über Bussysteme oder durch konventionelle Verbindungen.
Im Automobilbereich kommen heutzutage im Wesentlichen Busprotokolle wie Controller Area Network (CAN) [47], FlexRay [12], Local Interconnect Network (LIN) [15] oder Media Oriented Systems Transport (MOST) [33] zum Einsatz.

Leistungsversorgungsebene

Neben der Kommunikation durch Bussysteme wird die elektrische Infrastruktur im Rahmen des elektrischen Energiekonzepts und des Massekonzepts berücksichtigt. Hierbei werden unterschiedliche Realisierungen im

[4] Lehre von der Lage und Anordnung geometrischer Gebilde im Raum [4].

Rahmen von 12 Volt, 48 Volt oder 60 Volt (Hochspannung) Gleichspannungsbordnetzen betrachtet.

Leitungssatzebene

In der Leitungssatzebene als unterste Ebene wird die Konkretisierung der Bussysteme und der konventionellen Verbindungen als physikalischer Leitungssatz dargelegt.

Bezüge zwischen Architekturebenen

Die Bezüge zwischen Artefakten unterschiedlicher Architekturebenen werden über Mappings ausgedrückt. Im Folgenden sind die für die Darstellung einer E/E-Architektur grundlegenden Mappings zusammengefasst (s. Bild 2.2).

Dekompositionen Artefakte des Funktionsumfangs werden durch Artefakte der logischen Funktionsarchitektur dekomponiert.

Realisierungen Artefakte der logischen Funktionsarchitektur werden durch Softwarekomponenten und Hardwarekomponenten in Form von Sensoren und Aktuatoren realisiert.

Partitionierungen Artefakte der Softwarearchitektur in Form von Softwarekomponenten werden auf Recheneinheiten eines Steuergeräts partitioniert.

2.2.2 E/E-Domänenarchitektur

Über den Begriff einer E/E-Architektur ist die technische Realisierung von Fahrzeugfunktionen durch die Betrachtung unterschiedlicher Abstraktionsebenen festgelegt. Im Folgenden wird zunächst dargelegt, wie diese Funktionen heute im Rahmen der Fahrzeugproduktentwicklung fachlich kategorisiert sind. Anschließend wird der Ansatz der E/E-Domänenarchitektur eingeführt, welcher für diese fachliche Kategorisierung von Funktionen eine entsprechende Topologie von Steuergeräten vorgibt.

Fahrzeugdomänen

Eine Menge von fachlich zusammengehörenden Funktionen eines Fahrzeugs stellt gemäß Weber [Seite 54 f.][114] eine Fahrzeugdomäne dar. Die strukturierte Darstellung von Fahrzeugdomänen wird typischerweise über eine Baumstruktur erreicht (s. Bild 2.3).

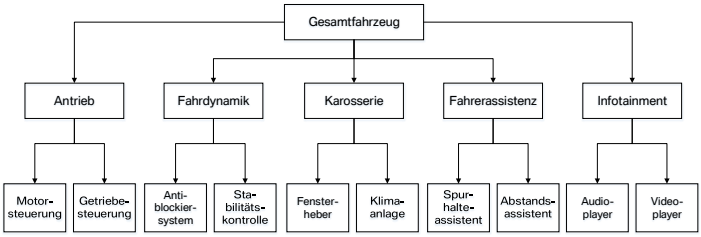

Bild 2.3: Aufteilung des Gesamtfahrzeugs in fachspezifische Fahrzeugdomänen in Anlehnung an Broy [Seite 32][30]

Klassische Domänen gemäß der gezeigten Baumstruktur sind die Antriebsdomäne, die Fahrdynamikdomäne und die Karosseriedomäne. Charakteristisch für Funktionen dieser Domänen ist die Wechselwirkung mit den mechanischen Anteilen des Fahrzeugs.

Vergleichsweise neue Domänen sind die Domäne für Funktionen aus dem Bereich der *Fahrerassistenz* als auch die Infotainmentdomäne. Charakteristisch für diese Domänen ist die noch stärkere Ausrichtung an der Softwaretechnik als es bei den klassischen Domänen der Fall ist.

Domänenspezifische Strukturierung

Die Steuergerätetopologie heutiger Fahrzeuge ist strukturiert nach Fahrzeugdomänen [Seite 9][28] und wird auch als E/E-Domänenarchitektur bezeichnet. Die E/E-Domänenarchitektur ordnet Steuergeräte durch deren Aufteilung in Teilnetze.

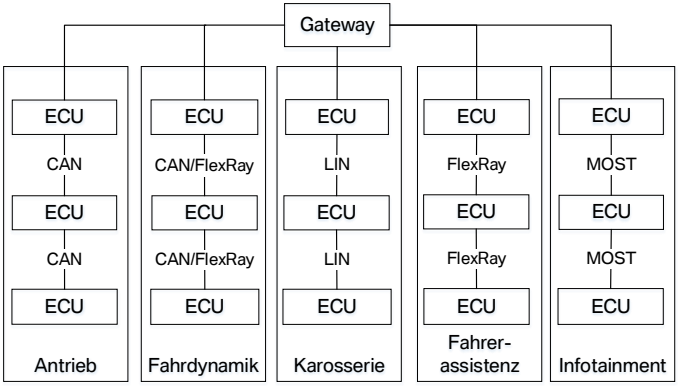

Bild 2.4: Beispielhafte E/E-Domänenarchitektur

Jedes Teilnetz ist wiederum über eine spezifische Vernetzungstechnologie umgesetzt. In diesem Zusammenhang kann nach Bach [Seite 12][22] auch von einem Bussegment gesprochen werden. In Bild 2.4 ist eine beispielhafte E/E-Domänenarchitektur mit Bussegmenten auf Basis der Technologien CAN, FlexRay, LIN und MOST dargestellt. Durch die Umsetzung spezifischer Protokolle in den einzelnen Bussegmenten muss im Weiteren für

eine domänenübergreifende Kommunikation ein Medienwandler in Form eines Gateways eingesetzt werden.

Beispiel Fahrerassistenzdomäne

Im Folgenden wird ein beispielhaftes Bussegment für die Realisierung von Funktionen der Fahrerassistenzdomäne durch die FlexRay Technologie gezeigt. FlexRay Bussegmente realisieren Netzwerke über ein zeitgesteuertes, serielles Bussystem mit einer vergleichsweise hohen Datenrate von bis zu 10 Mbit/s (s. Tabelle 2.1).

Technologie	Zeitgesteuert	Seriell	Datenrate
CAN	Nein	Ja	1 Mbit/s
FlexRay	Ja	Ja	10 Mbit/s
LIN	Ja	Ja	20 Kbit/s
MOST	Ja	Ja	150 Mbit/s

Tabelle 2.1: Gegenüberstellung von Bustechnologien

Als beispielhafte Funktion, die heute in einem solchen Netzwerk realisiert ist, wird ein Spurhalteassistent betrachtet.

Funktionsrealisierung Die Funktion des Spurhalteassistenten ist als Regelung über das Bussegment in Bild 2.5 verteilt. Initial werden hierzu *Kameradaten*, welche den Abstand zu den Fahrspurbegrenzungen beschreiben, von einem Steuergerät zur *Umfelderkennung* an ein Steuergerät zur *Fahrdynamikregelung* übertragen.

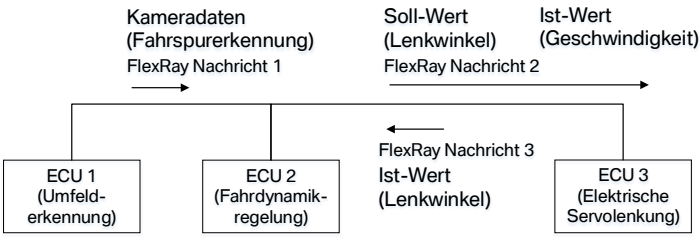

Bild 2.5: Spurhalteassistent realisiert in einem FlexRay Bussegment

Im Falle der Unterschreitung eines Mindestabstands wird von diesem Steuergerät ein Korrekturwert für die Fahrzeugquerbewegung in Richtung der Fahrspurmitte berechnet.

Anschließend wird dieser Korrekturwert als *Lenkwinkel-Soll-Wert* mit dem *Ist-Wert der Fahrzeuggeschwindigkeit* über den FlexRay Bus an ein Steuergerät zur Realisierung der *Elektrischen Servolenkung* versendet. Dieses Steuergerät ist für die Ansteuerung eines Lenkzylinders als Aktuator verantwortlich. Zur Festlegung des Regelkreises wird abschließend die Rückkopplung des *Lenkwinkel-Ist-Werts* an das Steuergerät zur *Fahrdynamikregelung* betrachtet.

Kommunikationskonzept

Die im vorangegangen Abschnitt eingeführte E/E-Domänenarchitektur beschreibt sich über Bussysteme, welche Funktionen über den Austausch von Nachrichten zwischen Steuergeräten realisieren. Der Aufbau einer Nachricht legt neben einer Identifikationsnummer (ID) sowie Status-, Steuer- und Prüfinformationen[5] Signale als Nutzdaten fest (s. Bild 2.6).

[5] Die Status-, Steuer- und Prüfinformationen werden eingesetzt um Fehler bei der Übertragung anzuzeigen.

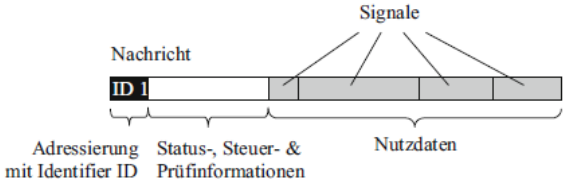

Bild 2.6: Nachrichten als Träger von Signalen nach Schäuffele und Zurawka [Seite 89][99]

Das übergeordnete Konzept für die Kommunikation zwischen Steuergeräten über Bussysteme und Signale wird dabei als *Signal-orientierte Architektur* bezeichnet.

2.3 Signal-orientierte Architekturen

Definition 4. *Signal-orientierte Architekturen beschreiben den Informationsfluss zwischen Softwarekomponenten über Signale, die von einem Sender zu einem Receiver versendet werden (s. Bild 2.7).*

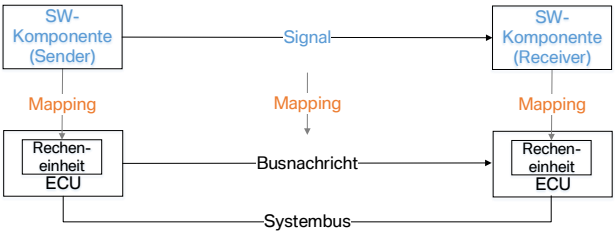

Bild 2.7: Software- und Systemarchitektur einer Signal-orientierten Architektur

Einordnung in OSI Modell

Das Signal-orientierte Kommunikationskonzept ist in das Open Systems Interconnection (OSI) Modell [118] wie folgt eingeordnet (s. Bild 2.8).

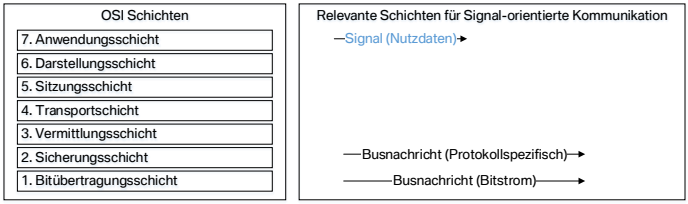

Bild 2.8: Relevante Schichten des OSI Modells für Signal-orientierte Architektur

Dabei wird der Informationsfluss zwischen Softwarekomponenten über Signale auf Schicht 7 (Anwendungsschicht) festgelegt. Die Schichten 2-1 (Sicherungsschicht und Bitübertragungsschicht) betrachten den Informations-

fluss zwischen Steuergeräten über Busnachrichten. Alle 3 Schichten des OSI Modells beziehen sich auf die Ebenen der Softwarearchitektur und der Topologie von Steuergeräten gemäß Bild 2.2.

Basissoftware im Kontext der Signal-orientierten Architektur

Im Rahmen einer Signal-orientierten Architektur werden feste Abbildungen von Signalen auf Nachrichten eines Bussystems erreicht. Während der Laufzeit des Systems werden diese Vorgaben von der Basissoftware eines Steuergeräts umgesetzt. Hierdurch wird es ermöglicht, dass Softwarekomponenten Nutzdaten (Signale) in eine zuvor festgelegte Busnachricht schreiben oder von einer zuvor festgelegten Busnachricht extrahieren können (s. Bild 2.9).

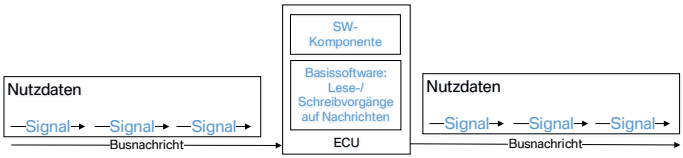

Bild 2.9: Basissoftware als Bindeglied zwischen Softwarekomponenten und Nachrichten

Der AUTomotive Open System ARchitecture (AUTOSAR) Standard [18] als maßgebender Standard für die Automobile Softwareentwicklung spricht im Zusammenhang mit den Nutzdaten einer Nachricht von der sogenannten Protocol Data Unit (PDU). Im Folgenden ist beispielhaft gezeigt, wie die Extraktion einer PDU aus einer Nachricht über die Basissoftware eines Steuergeräts umgesetzt wird.

Lesezugriff auf eine PDU

Für den Lesezugriff auf eine PDU werden mit Bezug auf den AUTOSAR Standard die in Bild 2.10 dargestellten Module der Basissoftware im Rahmen des PDU Routers und des Communication (COM) Modul betrachtet (s. Han et al. [Seite 2][72]).

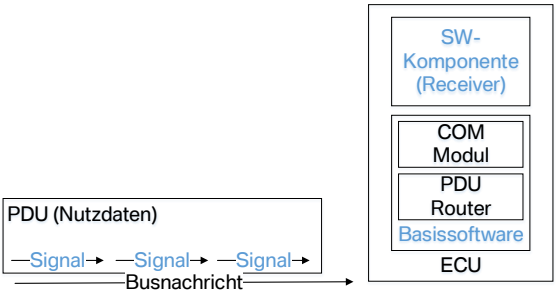

Bild 2.10: Lesezugriff auf eine PDU

Das Ziel des Lesevorgangs ist die Bereitstellung eines Signals an eine Softwarekomponente in der Rolle des Receivers. Die einzelnen Transformationen zwischen den Modulen der Basissoftware können dabei wie folgt dargestellt werden (s. Bild 2.11).

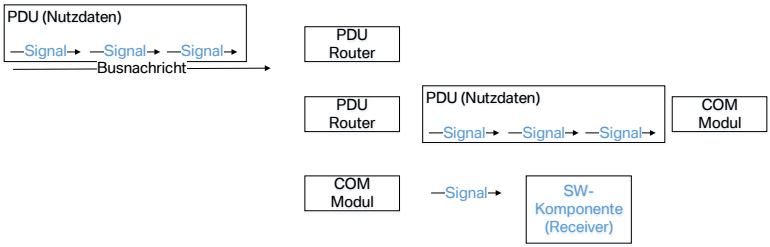

Bild 2.11: Transformationen beim Lesezugriff auf eine PDU

Zunächst wird eine Busnachricht mit einer PDU als Zusammenfassung von Signalen durch den PDU Router eingelesen. Anschließend gibt dieser die PDU als Nutzdaten an das COM Modul weiter, welches das für eine Softwarekomponente relevante Signal extrahiert. Abschließend kann auf Basis dieser Extraktion eine Verarbeitung des Signals durch den einer Softwarekomponente zugeordneten Programmcode stattfinden. Für einen Schreibvorgang verhält sich der Vorgang entsprechend aber in umgekehrter Weise.

Statische Kommunikationsfestlegung

Der Zusammenhang zwischen Signalen und Nachrichten ist heute während der Entwicklungszeit durch die Abbildung zwischen Informationsflüssen auf Schicht 7 und Informationsflüssen der Schichten 2-1 des OSI Modells fest vorgegeben. Als Beschreibungskonzept für alle Steuergeräte und Bussysteme der gezeigten E/E-Domänenarchitektur (s. Bild 2.4) kommt hierfür die sogenannte Kommunikationsmatrix zur Anwendung. In Bild 2.12 ist ein Ausschnitt einer Kommunikationsmatrix dargestellt.

Kommunikationsmatrix ('K-Matrix')

Netzknoten	Nachricht	Signal	ABS Steuergerät		Motorsteuergerät		Getriebesteuergerät	
ABS Steuergerät	ABS_1	Raddrehzahl vorne links	S		E	...		
		Raddrehzahl vorne rechts	S		E			
	ABS_2	Raddrehzahl hinten links	S		E			
		Raddrehzahl hinten rechts	S		E			
Motor-steuergerät	MS_1	Fahrpedalwert		S	E	...		
		Motordrehzahl	E	S	E			
	MS_2	Motortemperatur		S	E			
Getriebe-steuergerät	GS_1	Motorsollmoment		E	S	...		
...	...	...	...	...	...	...	...	...

Bild 2.12: Kommunikationsmatrix nach Schäuffele und Zurawka [Seite 90][99]

Dabei werden drei Steuergeräte als Netzknoten eines Bussystems betrachtet und in ihre Rollen als Sender (S) oder Empfänger (E) von Nachrichten mit

darin enthaltenen Signalen eingeordnet. Neben ihrer Aufgabe als Beschreibungsstruktur dient die Kommunikationsmatrix als Eingangsgröße, um die für die statische Kommunikationsfestlegung notwendigen Module der Basissoftware eines Steuergeräts zu konfigurieren.

2.4 Service-orientierte Architekturen

Als Alternative zu Signal-orientierten Architekturen und Möglichkeit zur Realisierung von Informationsflüssen in verteilten Systemen gelten Service-orientierte Architekturen[6].

Definition 5. *Service-orientierte Architekturen beschreiben den Informationsfluss zwischen Softwarekomponenten über Services, die bereitgestellt und genutzt werden. Anbieter eines Services werden als Server und Nutzer eines Services als Clients bezeichnet (s. Bild 2.13).*

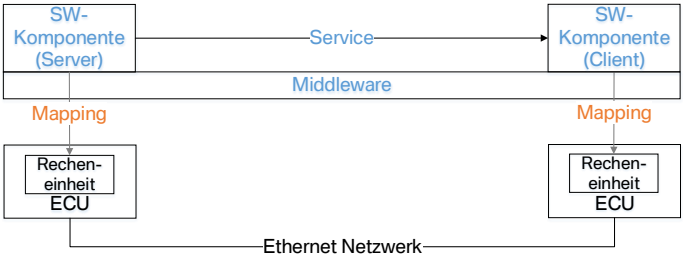

Bild 2.13: Software- und Systemarchitektur einer Service-orientierten Architektur

Der grundsätzliche Unterschied zwischen einer Service-orientierten Architektur und einer Signal-orientierten Architektur ist der Zusammenhang zwischen Softwarearchitektur und physischem Kommunikationsnetzwerk[7]. Für eine Service-orientierte Architektur besteht die Möglichkeit erst während der Laufzeit des Systems einen Informationsfluss zwischen Softwarekomponenten festzulegen, während dies in einer Signal-orientierten Architektur

[6] Für weitere Definitionen einer Service-orientierten Architektur sei auf Käbisch [Seite 12][64] und Göb [Seite 17][43] verwiesen.

[7] Hierbei kommen für eine Service-orientierte Architektur keine Bussysteme, sondern ein Kommunikationsnetzwerk auf Basis der Ethernet Technologie zum Einsatz. Eine aktuell verwendete Lösung ist über vier Standards [17], [16], [14], [13] im Rahmen des Audio Video Bridging (AVB) Ansatzes verfasst.

bereits in der Entwicklungszeit geschieht. Als Lösungskonzept kommt eine Service-orientierte Middleware zum Einsatz, deren Beitrag über die Einordnung in das OSI Modell aufgezeigt werden soll.

Einordnung in OSI Modell

Eine Service-orientierte Architektur implementiert alle Schichten des OSI Modells (s. Bild 2.14). Im Folgenden wird zunächst der Aufbau von Kommunikationsbeziehungen über die Service-orientierte Middleware auf den Schichten 7-5 dargestellt. Anschließend wird die sich daraus ableitende Kommunikation auf den Schichten 4-1 betrachtet.

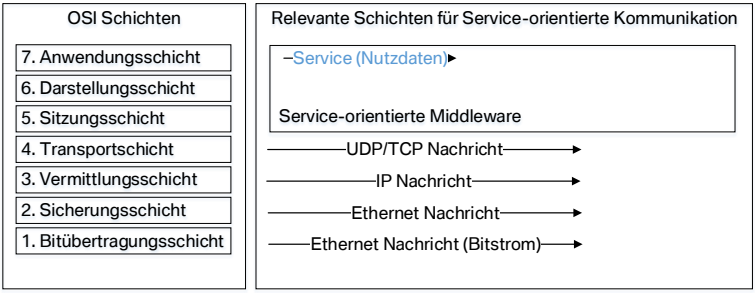

Bild 2.14: Service-orientierte Architektur im OSI Modell

Alle 7 Schichten detaillieren die Informationsflüsse auf Ebene der Softwarearchitektur und der Topologie von Steuergeräten gemäß Bild 2.2.

Service-orientierte Middleware

Eine Service-orientierte Middleware beschreibt eine Systemsoftware[8] zum Aufbau von Kommunikationsbeziehungen zwischen Softwarekomponenten in einem verteilten Rechnernetz [Seite 12][115]. Al-Jaroodi und Moha-

[8] Eine Systemsoftware kann in diesem Zusammenhang mit dem Begriff der Basissoftware aus Abschnitt 2.3 gleichgesetzt werden.

med [Seite 213 f.][19] beschreiben Anforderungen an eine Service-orientierte Middleware. Für die Abgrenzung zu Signal-orientierten Architekturen spielt dabei die Anwendung spezifischer Protokolle zum dynamischen Kommunikationsaufbau die zentrale Rolle.

Über die Zeit haben sich für diese Anforderung unterschiedliche Implementierungen herauskristallisiert, die als Service Discovery Protokolle bezeichnet werden. Konzepte aller Service Discovery Protokolle sind durch Perianu et al. [Seite 1 ff.][76] dargelegt und im Folgenden auf den Rahmen dieser Arbeit angepasst wiedergegeben.

Ziel der Service Discovery Ein Service Discovery Protokoll verbindet Softwarekomponenten in der Rolle des Servers und des Clients während der Laufzeit des Systems. Die Umsetzung der Service Discovery kann dabei über Varianten eines Nachrichtenprotokolls erreicht werden (s. Bild 2.15).

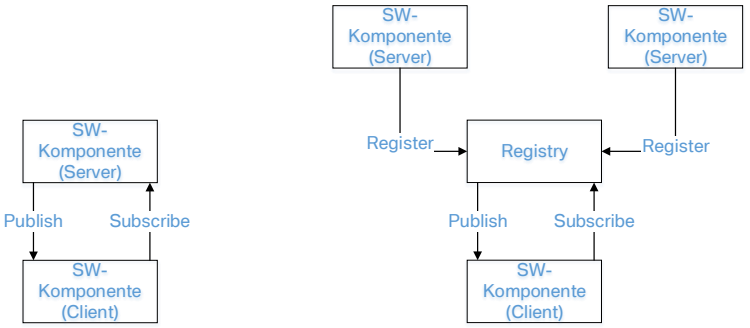

Bild 2.15: Verteilte Service Discovery (links) und Zentralisierte Service Discovery (rechts)

Verteilte Service Discovery Bei diesem Ansatz übermitteln Softwarekomponenten in der Rolle des Servers direkt ihren Service im Rahmen einer

Publish Nachricht. Anschließend wird die Service Discovery über eine *Subscribe* Nachricht des Clients abgeschlossen. Hierdurch ist dieser in der Lage, die von einem Service bereitgestellte Information in Form eines Fahrzeugparameters zu nutzen. Ein Fahrzeugparameter kann dabei das Fahrzeug oder seine Umwelt beschreiben.

Zentralisierte Service Discovery Dieser Ansatz ist über eine Vermittlungskomponente als *Registry* umgesetzt. Die *Registry* zentralisiert zunächst angebotene Services durch entsprechende Nachrichten aller Server. Anschließend werden die registrierten Services über *Publish* Nachrichten an Clients weitervermittelt. Der Abschluss der Service Discovery ist wiederum über den Versand einer *Subscribe* Nachricht umgesetzt.

Service Nutzung Auf Basis einer erfolgreichen Service Discovery kann ein Informationsfluss zwischen einem Server und einem Client stattfinden. Für die eingeführten Protokollvarianten (s. Bild 2.15) beginnt die Service Nutzung mit dem Erhalt der *Subscribe* Nachricht durch den Server. Bild 2.16 stellt dies für einen Client dar, welcher einen Service in Form

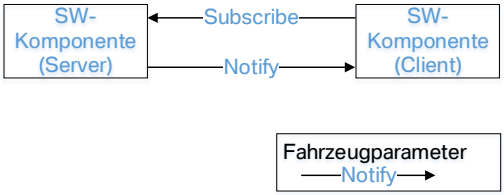

Bild 2.16: Service Nutzung im Rahmen des Publish-Subscribe Paradigmas

von Benachrichtigungen (engl. Notifications) nutzt. Die Übertragung dieser Benachrichtigungen kann ausschließlich im Falle von Änderungen eines Fahrzeugparameters oder zyklisch erfolgen [Seite 129][83].

Aufbau und Übertragung von Ethernet Nachrichten

Über das Konzept einer Service-orientierten Middleware kann die Initiierung von Kommunikationsbeziehungen zwischen Softwarekomponenten während der Laufzeit des Systems erreicht werden. Hierfür werden die Schichten 7-5 im Rahmen des OSI Modells realisiert.

Im Weiteren wird die aus der Kommunikation auf den Schichten 7-5 abgeleitete Kommunikation über Ethernet Nachrichten auf den Schichten 4-1 betrachtet. Hierdurch werden Informationsflüsse in einem physisch verteilten System ermöglicht.

Schichten 4 und 3 Die Kommunikation zwischen einem Server und einem Client auf den Schichten 7-5 wird zunächst auf ein Datenfeld einer User Datagram Protocol (UDP) [10] oder Transmission Control Protocol (TCP) [11] Nachricht umgesetzt (s. Bild 2.17). Der UDP oder TCP Header beschreibt die Port Adresse von Server und Client.

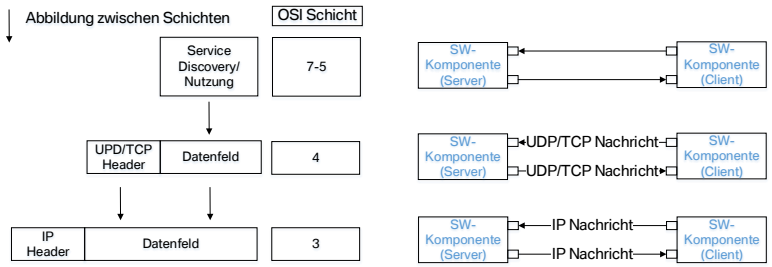

Bild 2.17: Ableitung von Ethernet Nachrichten auf Schichten 4 und 3

Aufgrund der Zugehörigkeit beider Protokolle zur Internet Protocol (IP) Protokollfamilie findet im Weiteren eine Abbildung auf das Datenfeld ei-

ner IP Nachricht statt. Der zusätzliche Header einer IP Nachricht beschreibt die IP Adressen von Server und Client.

Schichten 2 und 1 Für den Versand über ein physikalisches Netzwerk werden die Schichten 2 und 1 herangezogen (s. Bild 2.18). Hierbei werden zunächst IP Nachrichten in Ethernet Nachrichten eingefügt. Anschließend werden diese Nachrichten als Bitstrom serialisiert und über ein physikalisches Medium übertragen.

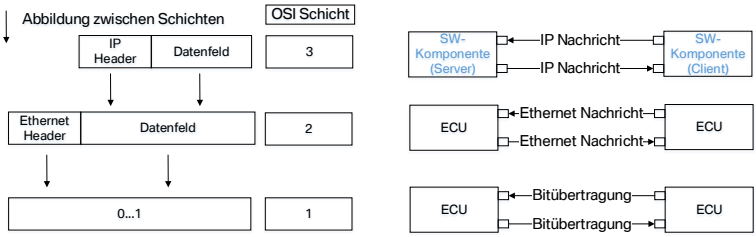

Bild 2.18: Ableitung von Ethernet Nachrichten auf Schichten 2 und 1

Technologien und Projekte

Technologien für eine Service-orientierte Architektur sind für die Automobile Softwareentwicklung heute verfügbar. Tabelle 2.2 kategorisiert diese nach ihrer Zugehörigkeit zum AUTOSAR Standard. Zusätzlich wird die Frage beantwortet, welche fachlichen Kategorien und somit Domänen der Produktentwicklung unterstützt werden.

Technologie	AUTOSAR	Domäne	Erscheinungsjahr
SOME/IP	Ja	Übergreifend	2013
DDS	Ja	Übergreifend	2018
GENIVI Platform	Nein	Infotainment	2009
Android Automotive	Nein	Infotainment	2017

Tabelle 2.2: Vermarktete Technologien für Automobile Service-orientierte Architekturen

In Bezug auf AUTOSAR ist dabei Scalable service-Oriented MiddlewarE over IP (SOME/IP) [113] seit dem Jahre 2013 als standardisierte Lösung spezifiziert. Neuer ist der Data Distribution Service (DDS) Ansatz [88], welcher seit 2018 Teil der AUTOSAR Standardisierung ist. Außerhalb von AUTOSAR können die GENIVI Platform [20] und Android Automotive [1] als Lösungen identifiziert werden. Beide Ansätze stellen jedoch einen Zuschnitt auf die Domäne *Infotainment* dar. Dem entgegen sind die AUTOSAR-konformen Lösungen über die Grenze eine Domäne hinweg einsetzbar.

Forschungsprojekte Außerhalb vermarkteter Lösungen ist der Ansatz einer Service-orientierten Architektur im Automobilen Kontext durch Forschungsprojekte betrachtet. Hierbei sind die öffentlich geförderten Projekte Sicherheit in Eingebetteten IP-basierten Systemen (SEIS) [46], das Projekt RACE [26], das Projekt UNICARagil [3] und das European Processor

Initiative (EPI) Projekt [5] zu nennen. Alle Projekte sind als Verbundprojekte zwischen Industrie und Forschungseinrichtungen umgesetzt. Der Fokus der Projekte liegt in der technischen Umsetzung einer Service-orientierten Architektur über Fahrzeugprototypen.

Projekt	Laufzeit	Öffentlich	Verbundprojekt
SEIS	2009-2012	Ja	Ja
RACE	2012-2015	Ja	Ja
UNICARagil	2018-2022	Ja	Ja
EPI	2018-2021	Ja	Ja

Tabelle 2.3: Forschungsprojekte für Automobile Service-orientierte Architekturen

2.5 Potentiale einer Service-orientierten Architektur

Eine Service-orientierte Architektur etabliert Informationsflüsse zwischen Softwarekomponenten während der Laufzeit des Systems. Ein zentrales Systemmodell im Rahmen einer Kommunikationsmatrix (s. Abschnitt 2.3), welches während der Entwicklungszeit definiert wird, existiert nicht. Für die Automobile Softwareentwicklung ergeben sich daraus drei wesentliche Potentiale.

Software-definierte Beschreibung

In der IT-Industrie existiert seit den 1990er Jahren der Begriff der sogenannten Software-definierten Netzwerke (engl. Software-defined Networking (SDN)) [68]. Diese beschreiben ein verteiltes vernetztes System ausschließlich über ihre Softwareanteile, während sich die Kommunikation auf Ebene der Hardware dynamisch ausbildet. Für eine Automobile Service-orientierte Architektur ergibt sich das Potential die bisherige Beschrei-

bung des verteilten vernetzten Systems als Signal-orientierte Architektur in Richtung einer Software-definierten Beschreibung zu ändern. Diese Anpassung stellt aktuell für Automobilhersteller eine wesentliche Transferleistung dar [Seite 1203][110], [Seite 201 f.][70].

Entwicklungszyklus

Durch die Beschreibung des Systems im Rahmen der Kommunikationsmatrix existieren heute feste Beziehungen zwischen den ausgetauschten Signalen von Softwarekomponenten und den Informationsflüssen zwischen Steuergeräten.

Um diese Abhängigkeit in der Produktentstehung zu berücksichtigen wird die Entwicklung von Software mit der Entwicklung von Hardware in Bezug auf den Entwicklungszyklus synchronisiert.

In einer Service-orientierten Architektur werden die Kommunikationswege für Steuergeräte während der Laufzeit des Systems abgeleitet. Als zu betrachtendes Potential ergibt sich hieraus die Idee Software unabhängiger von Hardware zu gestalten und damit Software-basierte Innovationen schneller zu vermarkten.

Spezifische Vorgehensmodelle für einen solchen Entwicklungskontext werden in Software-dominierten Branchen bereits angewendet und sind durch agile Vorgehensmodelle, wie Scrum [100], gegenwärtig. Das Ziel eines solchen Ansatzes ist es die Aktivitäten der Produktentstehung mehrfach zu durchlaufen und mit jedem Durchlauf ein Produktinkrement, das den bereits bestehenden Produktumfang erweitert, auszuliefern (s. Bild 2.19).

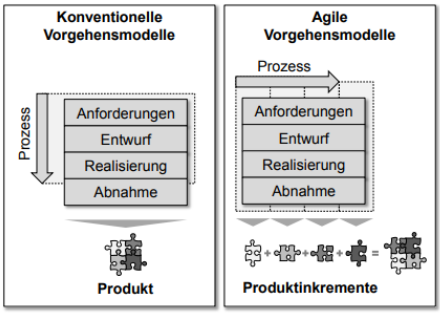

Bild 2.19: Schematische Gegenüberstellung des Prozesses bei Verwendung von konventionellen und agilen Vorgehensmodellen nach Klein [Seite 47][65] in Anlehnung an Johnson [63].

Updates Over The Air

Durch die langen Lebenszyklen eines Fahrzeugs ergibt sich die Notwendigkeit unter Umständen Änderungen an bereits ausgelieferten Softwaresystemen vorzunehmen. Diese betreffen beispielsweise Mängel in Bezug auf die Informationssicherheit, was durch nachträgliche Updates behoben werden kann. Eine Service-orientierte Architektur kann in diesem Zusammenhang durch die zeitlich flexible Integration von Updates unterstützend wirken.

Forschungsrahmen

Die Umsetzung einer Service-orientierter Architektur umfasst die beschriebenen Potentiale. Um diese zu heben, muss jedoch neben vorhandenen technischen Lösungen (s. Abschnitt 2.4) auch ein entsprechendes Vorgehen im Entwurf einer Software- und Systemarchitektur gegeben sein. Für den weiteren Verlauf dieser Arbeit werden in diesem Zusammenhang entsprechende Forschungsfragen aufgeworfen.

1. Forschungsfrage: Inwieweit muss der heutige Entwurf einer Software- und Systemarchitektur für den Ansatz einer Service-orientierten Architektur angepasst werden?

2. Wie können diese Anpassungen in einem entsprechenden Prozess zusammengefasst werden?

3 Stand der Wissenschaft und Technik

Um Anhaltspunkte für die Beantwortung der ersten Forschungsfrage zu erhalten soll zunächst der Stand der Wissenschaft und Technik analysiert werden. Dabei steht der Begriff des *Automotive Software Engineerings* als Rahmen für den Entwurf und die Realisierung einer heutigen Software- und Systemarchitektur im Vordergrund.

3.1 Automotive Software Engineering

Der Begriff *Automotive Software Engineering* als spezifische Form des Software Engineerings[1] beschreibt die systematische Entwicklung und Absicherung von verteilten Softwarekomponenten im Fahrzeug über ein Vorgehensmodell[2]. Ein heute etabliertes Vorgehensmodell für die Umsetzung des Automotive Software Engineerings ist das V-Modell (s. Bild 3.1).

Die Vorstellung der Phasen des Vorgehensmodells wird im Folgenden durch eine Unterteilung in einen Top-down-Prozess und einen Bottom-up-Prozess vorgenommen [Seite 70][94]. Der Top-down-Prozess kennzeichnet dabei den linken Ast als *Entwicklungsphasen* und der Bottom-Up-Prozess den rechten Ast als *Absicherungsphasen* des V-Modells.

[1] Im Deutschen wird das Software Engineering auch als Softwaretechnik bezeichnet. Eine inhaltliche Übersicht des Begriffs der Softwaretechnik kann bei Balzert [Seite 5][24] gefunden werden.

[2] Ein Vorgehensmodell setzt die Lösung eines Problems im Rahmen einer zeitlichen Gliederung von Lösungsschritten um.

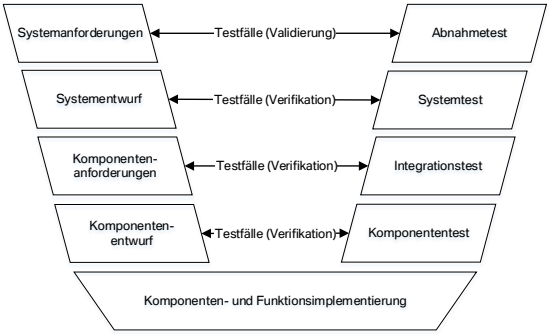

Bild 3.1: V-Modell gemäß Braun [Seite 29][28]

3.1.1 Entwicklungsphasen des V-Modells

Die Phasen erstrecken sich auf dem linken Ast von den Systemanforderungen bis zur Komponenten- und Funktionsimplementierung.

Systemanforderungen

Systemanforderungen umfassen abstrakte, funktionale Anforderungen an die Fahrzeugentwicklung im Rahmen neuer Fahrzeugfunktionen und ihrer Einteilung in Domänen (s. Bild 3.2).

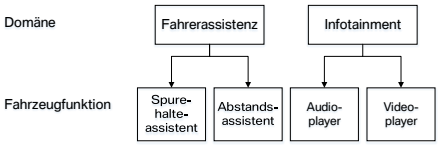

Bild 3.2: Fahrzeugfunktionen als abstrakte funktionale Systemanforderungen

Diese bilden dabei für den nachfolgenden Entwurf einer E/E-Architektur die oberste Architekturebene ab (s. Abschnitt 2.2.1). Neben funktionalen Anforderungen werden in dieser Phase nicht-funktionale Anforderungen (bspw.

Emissionsstandards oder Entwicklungskosten) analysiert sowie auf deren Konsistenz bewertet. Ein wichtiges Gütekriterium in diesem Zusammenhang ist durch die Konfliktfreiheit gegeben. Als Ziel der Anforderungsanalyse steht die Festlegung eines lösungsneutralen Anforderungsgerüsts [Seite 74][94].

Systementwurf

Im Rahmen des Systementwurfs wird die Entwicklung des Konzepts einer E/E-Architektur erarbeitet. Das E/E-Architekturkonzept beschreibt alle Architekturebenen (s. Bild 2.2) als übergreifendes Modell für logische Funktionen, Software- und Hardwareumfänge. Das Ziel des Konzepts ist eine zweckmäßige und kostengünstige Verteilung von Funktionen auf elektronischen Komponenten inklusive deren Vernetzung [Seite 17][29]. Zur Beschreibung dieser Vernetzung soll im Folgenden die hierfür im Fokus stehende Software- und Systemarchitektur näher betrachtet werden.

Software- und Systemarchitektur Die Software- und Systemarchitektur wird gemäß der dargestellten Beziehungen über Mappings als Teil der E/E-Architektur festgelegt (s. Bild 3.3). Dabei wird zunächst die Ausleitung von Softwarekomponenten aus der Dekomposition einzelner Fahrzeugfunktionen in logische Funktionen erreicht. Anschließend findet die Zuordnung dieser Softwarekomponenten auf Steuergeräte als Partitionierung statt und dient als Eingangsgröße für die Festlegung von Kommunikationsbeziehungen auf einer Topologie von Steuergeräten[3].

Für die logischen Funktionen umfassen diese Kommunikationsbeziehungen logische Informationsflüsse, die losgelöst von einer technischen Implementierung beschrieben sind[4] und anschließend durch die Softwarearchitektur

[3] Hierdurch werden die Ebenen der Softwarearchitektur und der Topologie von Steuergeräten (s. Abschnitt 2.2) miteinander verknüpft.

[4] Neben Informationen können logische Flüsse den Fluss von Energie oder Stoffen umfassen (s. Abschnitt 2.2.1). Im Zusammenhang mit der Software- und Systemarchitektur sind diese

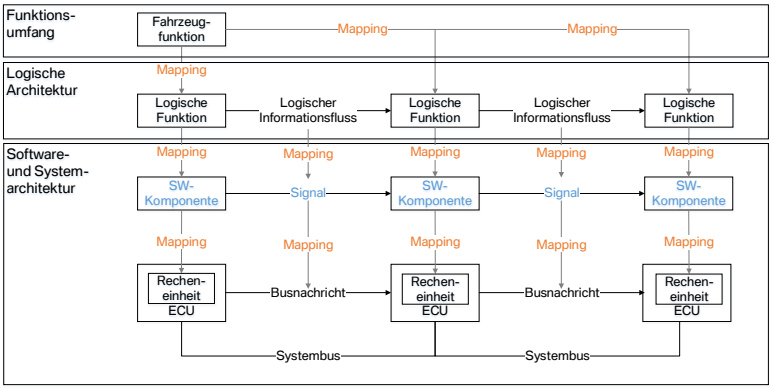

Bild 3.3: Festlegung der Software-und Systemarchitektur

in Signale als Informationsflüsse zwischen Softwarekomponenten übersetzt werden. Die Kommunikationsfestlegung wird vervollständigt durch die Abbildung dieser Signale auf Nachrichten eines Systembusses (s. Bild 2.7). Für alle Systembusse der heutigen E/E-Domänenarchitektur ist eine Beschreibung dieses Zusammenhangs durch die Kommunikationsmatrix erreicht (s. Abschnitt 2.3).

Komponentenanforderungen, Komponentenentwurf, Komponenten- und Funktionsimplementierung

Auf Basis der Software- und Systemarchitektur werden die darin enthaltenen Softwarekomponenten, Steuergeräte und Bussysteme verfeinert durch Spezifikationen. Für die Entwicklung von Softwarekomponenten umfasst diese Spezifikation die Beschreibung des Verhaltens[5] und die Festlegung von Echtzeitanforderungen [Seite 79][94]. Aus der Verhaltensbeschreibung

Flüsse jedoch nicht relevant, weil dafür keine Implementierung in Software erreicht werden kann.

[5] Das Verhalten einer Softwarekomponente wird mit dem Verhalten einer Funktion gleichgesetzt (s. Definition 2).

wird für die spätere Integration die Funktionsimplementierung einer Softwarekomponente als Quellcode gewonnen (s. Bild 3.4).

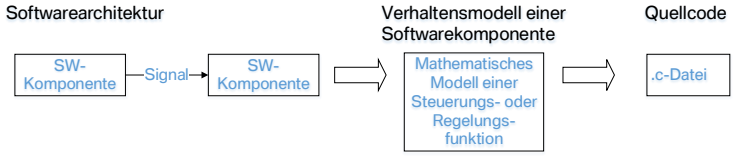

Bild 3.4: Ableitung von Funktionsimplementierung für Softwarekomponenten

Neben der Funktionsimplementierung werden Softwarekomponenten um den internen Aufbau im Rahmen von Schnittstellen zur Basissoftware eines Steuergeräts ergänzt (s. Bild 3.5).

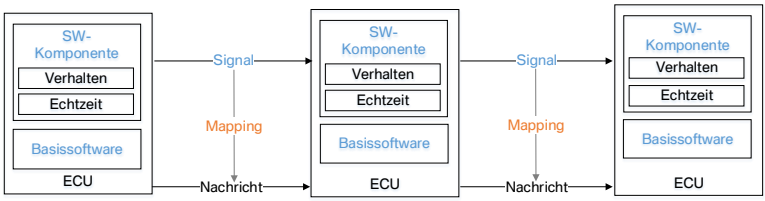

Bild 3.5: Interner Aufbau der Softwarearchitektur

Die Basissoftware schafft in diesem Zusammenhang den Bezug zu Anteilen des Betriebssystems, hardwareabhängigen Treibern und Diagnosemodulen [Seite 16][42]. Ein für die Kommunikationsfestlegung zwischen Signalen und Busnachrichten relevanter Entwicklungsschritt ist hierbei die Konfiguration des PDU Routers und des COM Moduls (s. Abschnitt 2.3), die in Kombination mit der Funktionsimplementierung für ein ausführbares System auf Basis eines Systembusses beide notwendig sind.

3.1.2 Absicherungsphasen des V-Modells

Die Phasen des V-Modells auf dem rechten Ast zeigen die Verifikation und Validierung der E/E-Architektur und der darin enthaltenen Software- und Systemarchitektur. Beide Begriffe sind nach Schäuffele und Zurawka [Seite 33][99] wie folgt definierbar.

Definition 6. *Verifikation ist der Prozess zur Beurteilung eines Systems oder einer Komponente mit dem Ziel festzustellen, ob die Resultate einer gegebenen Entwicklungsphase den Vorgaben für diese Phase entsprechen. Software-Verifikation ist demnach die Prüfung, ob eine Implementierung der für den betreffenden Entwicklungsschritt vorgegebenen Spezifikation genügt.*

Definition 7. *Validierung ist der Prozess zur Beurteilung eines Systems oder einer Komponente mit dem Ziel festzustellen, ob der Einsatzzweck oder die Benutzererwartungen erfüllt werden. Funktionsvalidierung ist demnach die Prüfung, ob die Spezifikation die Benutzeranforderungen erfüllt, ob überhaupt die Benutzerakzeptanz durch eine Funktion erreicht wird.*

Als Zusammenspiel von Verifikation und Validierung wird ein Durchlauf von der Komponentenebene in die Systemebene erreicht.

Komponententest

Der Test von Komponenten betrachtet die Ableitung von Testfällen aus der Spezifikation der Softwarekomponenten. Hierbei werden die variierenden Eingangsparameter einer Softwarekomponente, die einzuhaltenden Randbedingungen und das geforderte Verhalten der Ausgangsgrößen umfasst [Seite 80 f.][94]. Ein Verfahren für den Test von Softwarekomponenten wird über den Software-in-the-Loop (SiL)-Ansatz [Seite 46 f.][41] dargestellt.

Integrationstest

Im Integrationstest werden mehrere Softwarekomponenten zusammengeführt, die auf einem Steuergerät integriert sind. Dieses Zusammenführen wird durch automatisierte, statische Prüfungen begleitet, welche prüfen, ob Schnittstellenspezifikationen, der Namensraum für Variablen und die Verwendung eines einheitlichen Speicherlayouts eingehalten wurden [Seite 196][99].

Systemtest und Abnahmetest

In der Phase der Systemtests wird Software auf Hardware integriert und abgesichert. Für die Software umfasst dies zuerst eine Integration auf der jeweiligen Zielhardware, bevor in einer darauffolgenden Integration mehrere Steuergeräte inklusive der Sollwertgeber, Sensoren und Aktuatoren integriert werden, um das Zusammenwirken auf Systemebene abzusichern [Seite 197][99]. Ein etabliertes Testverfahren wird beim Test auf Systemebene unter dem Begriff Hardware-in-the-Loop (HiL) erfasst[6].

Die abschließende Validierung als Abnahmetest des Systems erreicht eine nochmals höhere Systemebene. Hierbei wird der Verbund aus Software und Hardware im Kontext des Gesamtfahrzeugs und damit der realen Betriebsumgebung abgesichert.

[6] Für eine tiefergehende Betrachtung zu diesem Verfahren sei auf die Publikation von Sax [98] verwiesen.

3.1.3 Modelle, Methoden, Werkzeuge des Automotive Software Engineerings

Um das Automotive Software Engineering im Rahmen des V-Modells umzusetzen werden Modelle, Methoden und Werkzeuge eingesetzt. Ein Überbegriff hierfür ist der Begriff der Methodik.

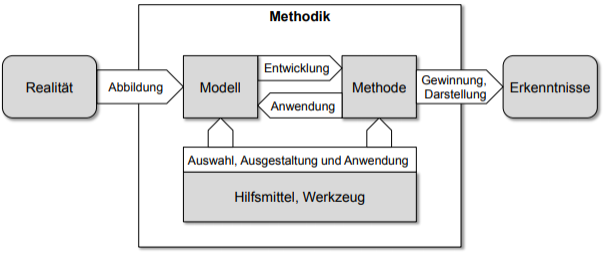

Bild 3.6: Vorgehensweisen der Planung nach Laufenberg [Seite 7][71] und Heyn [Seite 5 f.][57]

Eine Methodik ist nach Neuhausen [Seite 6][82] ein “System von zusammengehörigen Modellen, Methoden und Hilfsmitteln zur Lösung einer theoretischen und/ oder praktischen Aufgabenstellung”. Methoden werden dabei aus Modellen entwickelt bzw. auf Modelle angewandt (s. Bild 3.6).

Definition 8. *Ein Modell ist ein Abbild von etwas oder Vorbild für etwas bzw. Repräsentation eines bestimmten Originals [Seite 129][102].*

Definition 9. *Eine Methode ist ein Verfahren zur systematischen Erkenntnisgewinnung und Erkenntnisdarstellung [Seite 7][82].*

Zur Umsetzung des Zusammenspiels zwischen Modellen und Methoden im Rahmen einer Methodik stehen nach Hammerspiel [Seite 28][53] "Hilfekonzepte wie Checklisten, Werkzeuge oder Vorlagen zur Verfügung".

3.2 Modellbasierter Systementwurf

Speziell zur methodischen Entwicklung der Software- und Systemarchitektur stellt die modellbasierte Entwicklung heute einen etablierten Ansatz dar. Hierbei wird eine deterministische und damit maschinell auswertbare Methode, die auf Modellierungssprachen basiert, angewandt. Eine Modellierungssprache ist ein Synonym für eine Domain Specific Language (DSL)[7].

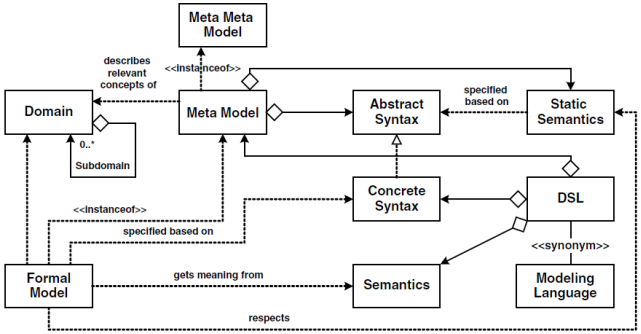

Bild 3.7: Konzeptbildung: Modellierung und DSLs nach Stahl et al. [Seite 56][103]

Für die Festlegung einer DSL sind die Begriffe von *Syntax* und *Semantik* im Folgenden detailliert.

[7] Eine DSL befähigt zur Modellierung von Sachverhalten aus einem bestimmten Problemraum [79][Seite 316 ff.].

Syntax

Die Syntax einer Modellierungssprache beschreibt die Regeln für ihre Anwendung. Damit sind Festlegungen für die Kombination von Elementen einer Sprache (Wörter, Zeichen, Symbole) zusammengefasst [Seite 66][50].

Abstrakte und konkrete Syntax Die abstrakte Syntax beschreibt wie die Elemente einer Sprache in struktureller Hinsicht in Beziehung zueinander stehen. Die konkrete Syntax beschreibt wie die Elemente einer Sprache tatsächlich (textuell bzw. grafisch) umgesetzt werden können [Seite 68][50]. Eine Möglichkeit, den Zusammenhang zwischen abstrakter und konkreter Syntax zu beschreiben, ist durch den Begriff des Metamodells gegeben.

Metamodell Ein Metamodell beschreibt die abstrakte Syntax eines Modells [Seite 84][50]. Die zu einem Metamodell entsprechenden Modelle, Instanzen eines Metamodells, beschreiben die konkrete Anwendung der vom Metamodell beschriebenen abstrakten Syntax.

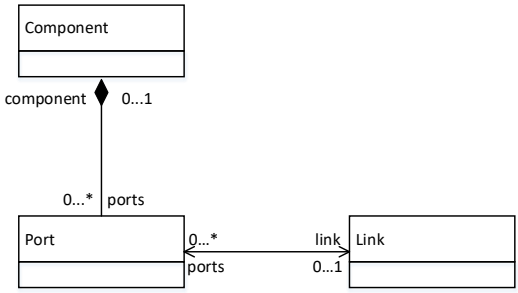

Bild 3.8: Metamodell des Port-Konzepts zur Beschreibung einer Komponente in Anlehnung an Gebauer [Seite 36][35]

Als Beispiel ist ein Metamodell in Bild 3.8 dargelegt, welches die Elemente zur Modellierung einer Komponente im Rahmen des Port-Konzepts als auch

deren wechselseitige Beziehungen durch Regeln bestimmt. Zur Darstellung des Metamodells wird ein Klassendiagramm, in dem einzelne Klassen über Assoziationen[8] miteinander in Beziehung gesetzt werden, verwendet. Im Weiteren trägt jede Assoziation an ihren Enden Multiplizitäten.

Eine Multiplizität koordiniert die Anzahl an Beziehungen, welche auf die Instanz einer bestimmten Klasse angewandt werden können. Aus dem gezeigten Metamodell ist dabei zu erkennen, dass Instanzen der Klasse *Component* mit mehreren Instanzen der Klasse *Port* assoziierbar sind. Gleichermaßen wird gefordert, dass jede Instanz der Klasse *Port* nur mit einer Instanz der Klasse *Link* verknüpft werden kann.

Für die Entwicklung von Metamodellen wird ein Meta-Metamodell eingesetzt. Der führende Standard zur Beschreibung dieses Zusammenhangs ist durch die Meta Object Facility (MOF) der Object Management Group (OMG) im Folgenden eingeführt.

Meta Object Facility (MOF) Die MOF [90] wurde von der OMG entwickelt und beschreibt ein Meta-Metamodell über Konstrukte der objektorientierten Softwareentwicklung in Form von Klassen, Attributen und Assoziationen [Seite 100][25]. Ausgehend von diesem Meta-Metamodell wird die Ableitung von Metamodellen und Modellen erreicht. In Bild 3.9 ist dies am Beispiel eines Leitungssatzes und einer Strukturierung über vier Ebenen dargestellt.

[8] Die spezifische Assoziation zwischen den Klassen *Component* und *Port* ist dabei eine Komposition, welche anzeigt, dass eine Komponente aus Ports besteht.

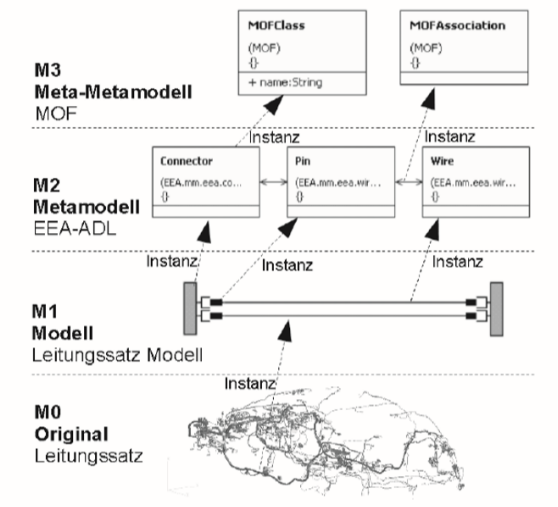

Bild 3.9: Darstellung der Abstraktionsebenen des Vier-Schichten-Modells der MOF am Beispiel der E/E-Architekturmodellierung eines Leitungssatzes nach Matheis[Seite 16][62]

Auf oberster Ebene (M3) steht dabei die Festlegung des Meta-Metamodells über die bereits eingeführten Konstrukte (Klassen, Attribute und Assoziationen). Auf der darunterliegenden Ebene (M2) ist ein zum Meta-Metamodell entsprechendes Metamodell als abstrakte Syntax eines Leitungssatzes festgelegt. Auf Basis dieses Metamodells kann schlussendlich die Modellierung eines Leitungssatzes (M1) und damit die Abstraktion des eigentlichen, physischen Leitungssatzes (M0) erreicht werden.

Semantik

Die Semantik einer Modellierungssprache beschreibt die Bedeutung der Syntax. Hierbei wird zwischen der statischen und der dynamischen Semantik unterschieden.

Statische Semantik Die statische Semantik wird als ein Satz von Regeln begriffen, die Eigenschaften eines Modellelements als auch dessen Beziehungen zu anderen Modellelementen einschränken [Seite 267][112]. Beispiele hierfür können der Wertebereich für die Beschreibung von Eigenschaften eines Modellelements oder die Anzahl an Beziehungen, die ein Modellelement mit anderen Modellelementen teilt, sein. Die statische Semantik verfeinert somit die abstrakte Syntax einer Sprache, was am Beispiel des Metamodells einer Komponente im Rahmen des Port-Konzepts bereits gezeigt wurde (s. Bild 3.8).

Dynamische Semantik Die dynamische Semantik bildet die abstrakte Syntax auf ein mathematisches Modell (denotationelle Semantik) oder ein ausführbares Modell (operationelle Semantik) ab [Seite 87][81]. Ein ausführbares Modell kann dabei durch den Quellcode einer Programmiersprache dargestellt werden.

3.3 Architekturbeschreibung

Durch die eingeführten Konzepte von Syntax und Semantik lassen sich Modellierungssprachen und wiederum daraus formale Modelle entwickeln. Eine spezifische Form der Modellierungssprache ist die Architekturbeschreibungssprache (engl. Architecture Description Language (ADL)), deren Einsatz sich zur Formalisierung eines Systementwurfs etabliert hat.

3.3.1 Architekturbeschreibung und -sichten

Gemäß der ISO 42010 [60] wird eine Architekturbeschreibungssprache angewendet, um die Beschreibung einer Architektur über Sichten zu erreichen.

Definition 10. *Eine Architektursicht ist eine spezifische Perspektive auf eine Architektur zur Abdeckung eines bestimmten Anliegens.*

Eine Möglichkeit unterschiedliche Sichten auf eine E/E-Architektur anzuwenden ist über die in Abschnitt 2.2.1 eingeführten Abstraktionsebenen gegeben (s. Bild 3.10).

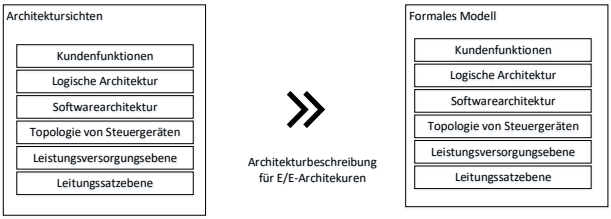

Bild 3.10: Architekturbeschreibung für eine E/E-Architektur in Anlehnung an Freess [Seite 46][42]

Im Folgenden sind mögliche Sprachen vorgestellt, die spezifisch für diese Sichten eingesetzt werden. Neben ihrer Einführung wird die Abgrenzung zu Beschreibungsansätzen für allgemeinere Anwendungsbereiche vorgenommen.

3.3.2 EEA-ADL

Die Electric Electronic Architecture - Analysis Design Language (EEA-ADL) ist eine Architekturbeschreibungssprache für Automobile E/E-Architekturen, die auf dem vorgestellten MOF Ansatz basiert. Umgesetzt ist die EEA-ADL in dem Werkzeug PREEvision [8].

Systemmodellierung über Abstraktionsebenen Die EEA-ADL setzt über ein entsprechendes Metamodell die Abstraktionsebenen einer E/E-Architektur um. Hierbei können für jede Abstraktionsebene die spezifischen Artefakte, wie logische Funktionen, Softwarekomponenten oder Steuergeräte, abgeleitet werden.

Zusätzlich werden diese Artefakte über grafische Notationen dargestellt. Beispielhaft für die Softwarearchitektur werden bei der grafischen Notation Elemente eines Blockschaltdiagramms eingesetzt. In Bild 3.11 ist eine spezifische Softwarearchitektur entlang des AUTOSAR Standards im Rahmen eines solchen Blockschaltdiagramms gezeigt.

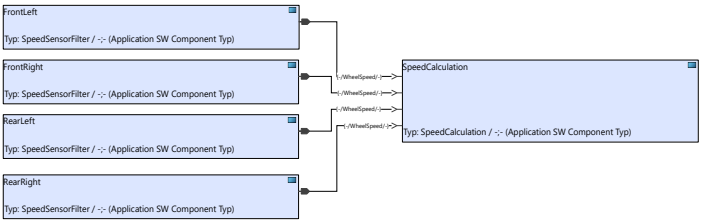

Bild 3.11: Softwarekomponenten im Rahmen eines Blockschaltdiagramms

Die Elemente des Blockschaltdiagramms umfassen Blöcke zur Darstellung von AUTOSAR Softwarekomponenten, welche in diesem Fall zur

Beschreibung einer *Applikation* dienen[9]. Zur Darstellung des Informationsaustauschs über Signale zwischen jeweils zwei Softwarekomponenten werden Ports und Links angewandt.

Verknüpfung von Abstraktionsebenen Die Modellierung einer Abstraktionsebene mit den dafür vorgesehenen Artefakten kann im Rahmen der EEA-ADL unabhängig von einer anderen Abstraktionsebene vorgenommen werden. Gemäß der Abstraktionsebenen einer E/E-Architektur (s. Bild 2.2) sind jedoch an bestimmten Stellen Verknüpfungen zwischen Artefakten unterschiedlicher Abstraktionsebenen notwendig. Ein allgemeines Konzept, um derartige Beziehungen über ein Metamodell zu realisieren, ist über das Mapping-Konzept gegeben (s. Bild 3.12).

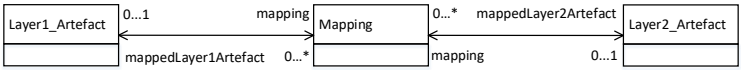

Bild 3.12: Allgemeines Mapping-Konzept gemäß Gebauer [Seite 35][35]

Beispielhaft für die Partitionierung von Softwarekomponenten auf Steuergeräten ist das Mapping-Konzept gezeigt (s. Bild 3.13). Das beschriebene Topologiemodell umfasst Steuergeräte, die über ein Bussegment auf Basis der CAN Technologie miteinander verknüpft sind.

[9] Durch die entsprechende Bezeichnung *Application SW Component Typ* sind diese im Rahmen des AUTOSAR Standards begrifflich von Anteilen der Basissoftware (s. Abschnitt 2.3), die auch über das Modell einer Softwarekomponente dargestellt werden können, abgegrenzt.

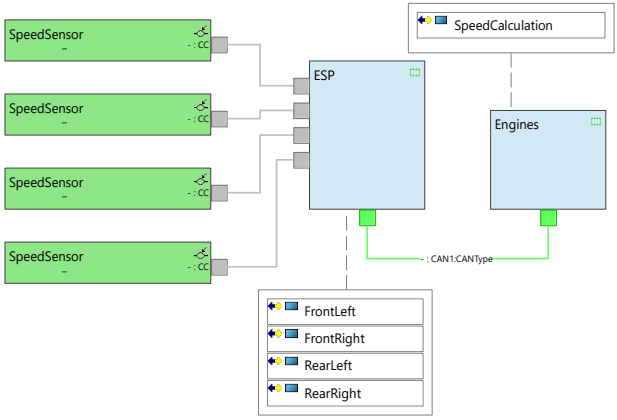

Bild 3.13: Beispielhaftes Mapping von Softwarekomponenten auf Steuergeräte im Rahmen eines Topologiemodells

Die Information, welche Softwarekomponente auf welchem Steuergerät partitioniert ist, wird modellhaft über Mappingobjekte erreicht. Ihre grafische Repräsentation findet sich über Labels wieder, welche an den Steuergeräten *ESP* und *Engines* angebracht sind.

Das Mappingobjekt ist somit ein Stellvertreter für eine Softwarekomponente im Rahmen des Topologiemodells. Die Softwarekomponente selbst tritt in diesem nicht auf, sondern existiert in einem spezifischen Modell zur Beschreibung der Softwarearchitektur. Das Ziel des Mapping-Konzepts ist der Bezug von Artefakten unterschiedlicher Abstraktionsebenen durch Stellvertreter, wodurch die eigentlichen Modelle weiterhin voneinander entkoppelt existieren.

3.3.3 EAST-ADL

Die Electronics Architecture and Software Technology-Architecture Description Language (EAST-ADL)[10] ist eine weitere Architekturbeschreibungssprache für Automobile E/E-Architekturen. Die Sprache basiert auf der Unified Modeling Language (UML)[11] und ist über Abstraktionsebenen strukturiert.

Systemmodellierung Die Ebenen setzen gemäß Bild 3.14 einen Top-down Systementwurf, der sich über das Vehicle Level, das Analysis Level, das Design Level und das Implementation Level erstreckt, um.

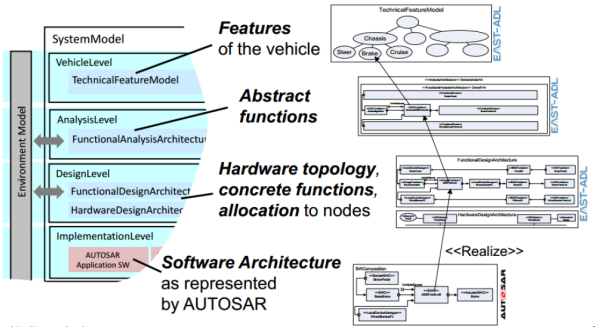

Bild 3.14: Ebenen der EAST-ADL gemäß Lönn [Seite 4][56]

Innerhalb dieser Ebenen können wie bei der EEA-ADL die unterschiedlichen Artefakte einer E/E-Architektur modelliert werden.

Verknüpfung von Abstraktionsebenen Die Verknüpfung von Abstraktionsebenen wird bei der EAST-ADL über unterschiedliche Typen von Beziehungen erreicht. Zur Umsetzung dieser Beziehungen werden Stereoty-

[10] Die EAST-ADL ist unter anderem in dem Werkzeug EAST-ADL Open Tool Platform [93] umgesetzt.

[11] Die EAST-ADL stellt in diesem Zusammenhang eine Erweiterung als Profil dar. Ein Profil ermöglicht die Erweiterung der UML auf Basis des bestehenden Metamodells [Seite 95][50].

pen angewandt. Ein Stereotyp ergänzt Elemente der UML bzw. einer UML-basierten Sprache um eine kontextspezifische Bedeutung. In Bild 3.15 ist dies durch die Notation ≪ADLRealize≫, welche die Beziehung zwischen logischen Funktionen und Softwarekomponenten als Realisierung deklariert, dargelegt.

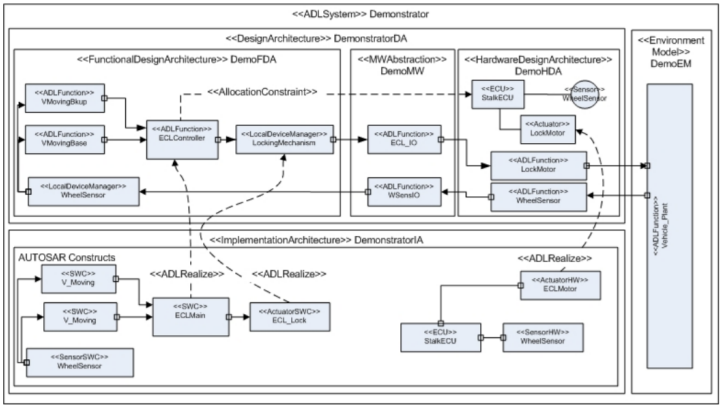

Bild 3.15: Realisierungsbeziehung zwischen logischen Funktionen und Softwarekomponenten gemäß Cuenot et al. [Seite 303][34]

Als Unterscheidung zum Mapping-Konzept gemäß der EEA-ADL stellen die Beziehungen direkte Verbindungen zwischen Artefakten unterschiedlicher Abstraktionsebenen her. Die Modellierung der verknüpften Artefakte in unterschiedlichen Modellen wird jedoch über beide Ansätze gleichermaßen erreicht.

3.3.4 MARTE

Ein OMG Ansatz für das allgemeinere Anwendungsfeld eingebetteter Systeme ist die Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [87] Language. Die Modellierungssprache basiert wie die EAST-ADL auf der UML und findet sich als Umsetzung in dem Werkzeug Papyrus [44] wieder.

Systemmodellierung Für die MARTE Language bildet die Modellierung der Softwarearchitektur und deren Partitionierung auf Hardwarekomponenten das zentrale Systemmodell. Als Abgrenzung zur UML werden hierbei spezifische Sprachkonstrukte zur Beschreibung von Echtzeiteigenschaften angewandt. Diese umfassen beispielsweise zeitliche Aspekte im Rahmen physikalischer Einheiten, die von der UML nicht unterstützt werden.

Verknüpfung von Abstraktionsebenen Im Rahmen von MARTE werden alle Artefakte einer Architektur über ein gemeinsames Modell erfasst. In diesem Modell können dann zwischen Software und Hardware Allokationsbezüge festgelegt werden. Dabei werden beide Artefakte zunächst über die Stereotypen ≪SwResource≫ und ≪HwResource≫ voneinander unterschieden (s. Bild 3.16). Die eigentliche Allokation wird dann über den Stereotyp ≪Allocate≫ realisiert. Einen mit der MARTE Language vergleichbaren Ansatz verfolgt die Systems Modeling Language (SysML) [89]. Diese unterscheidet sich jedoch dadurch, dass sie einen noch weiteren Bereich als den eingebetteter Systeme erfasst.

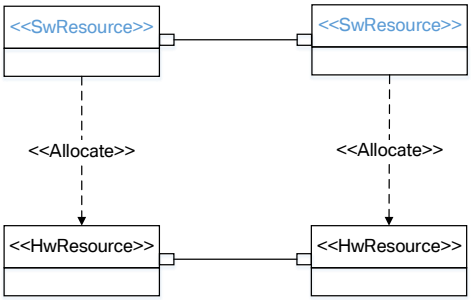

Bild 3.16: Allokation von Hardwareressourcen durch Softwareressourcen

3.3.5 Zusammenfassung

Die Idee der Architekturbeschreibung wird angewandt um einen Systementwurf im Rahmen des V-Modells durch eine formale Sprache zu verfassen. Für E/E-Architekturen finden sich durch die EEA-ADL und die EAST-ADL spezifische Umsetzungen (s. Tabelle 3.1).

Sprache	**Umfeld**	**Mögliches Werkzeug**	**Erweiterung der UML**
EEA-ADL	E/E-Architektur	PREEvision	Nein
EAST-ADL	E/E-Architektur	EAST-ADL Open Tool Platform	Ja
MARTE	Eingebettetes System	Papyrus	Ja
SysML	Allgemeines System	Enterprise Architect	Ja

Tabelle 3.1: Gegenüberstellung von Architekturbeschreibungssprachen

Ergänzend zu den Konzepten einer formalen Sprache werden Abstraktionsebenen für beide Ansätze zur Modellstrukturierung berücksichtigt. Weitergehend besteht durch die allgemeineren Ansätze der MARTE Language und

der SysML die Möglichkeit zur Modellierung eines Systementwurfs. Als Unterscheidung zur EEA-ADL und zur EAST-ADL findet keine Trennung nach Ebenen einer Architektur statt.

3.4 Architekturanalyse

Neben der Entwurfsformalisierung werden Architekturmodelle einer entsprechenden Sprache zur Architekturanalyse verwendet.

3.4.1 Statische Architekturanalyse

Die statische Architekturanalyse entspricht der Bewertung einer Architektur über ein spezifisches Kriterium. Speziell für eine E/E-Architektur kann dieses nach Gebauer [Seite 100][35] wie folgt definiert werden.

Definition 11. *Ein E/E-Architektur-Bewertungskriterium definiert als Eingangsgröße eine Menge von Artefakten einer E/E-Architektur. Als Ausgangsgröße definiert es eine oder mehrere Kennzahlen, die für eine Entscheidung relevant sind. Ein Sonderfall eines Bewertungskriteriums ist das Ausschlusskriterium.*

Beispielhaft für die Lastberechnung eines Bussegments setzt sich eine statische Modellanalyse über ein Bewertungskriterium wie folgt zusammen:

Als Eingangsgröße wird eine Kommunikationsmatrix für ein Bussegment herangezogen. Für den Bus steht dabei als relevante Modelleigenschaft die Übertragungsrate a. Für die Menge $\{\tau_i\}_{i=1}^{|T|}$ von Nachrichten findet eine Modellfestlegung durch ihre Nachrichtengröße b_i, ihre Wiederholrate im Rahmen einer Zykluszeit t_i und ihrer Laufzeit c_i statt[12]. Die Bewertung der Laufzeit c_i wird dabei als Verhältnis der Nachrichtengröße b_i und der Übertragungsgeschwindigkeit a des Busses gebildet.

$$c_\mathrm{i} = \frac{b_\mathrm{i}}{a} \tag{3.1}$$

Die Berechnung der Last des Busses wird programmatisch erreicht. Für die Buslastanalyse ist eine entsprechende Berechnung wie folgt gegeben und

[12] Die Modellfestlegung betrachtet alle Nachrichten als zyklisch versendete Nachrichten.

ermittelt einen numerischen Wert zwischen 0 und 1 als Auslastung (engl. Utilisation) U[13].

$$\sum_{i=1}^{n} U = \frac{c_i}{t_i} \tag{3.2}$$

Diese Kennzahl kann weitergehend genutzt werden, um die Einhaltung von Obergrenzen für die Buslast im Rahmen des Systementwurfs (s. Abschnitt 3.1.1) zu bewerten.

Bedeutung der Buslastbewertung

Durch die Realisierung einer heutigen E/E-Architektur als Verbund von Bussystemen (s. Bild 2.4) spielt die Last einzelner Segmente bzw. die entsprechende Kennzahl eine zentrale Rolle. Dabei wird vorrangig die Realisierbarkeit eines Kommunikationsnetzwerks für Nachrichten bewertet.

Durch die feste Verknüpfung von Nachrichten eines Systembusses mit den zwischen Softwarekomponenten ausgetauschten Signalen (s. Abschnitt 2.3) ist jedoch weitergehend ein Bezug zur Softwarearchitektur hergestellt. Speziell im Zusammenhang mit Änderungen einzelner Signale muss dabei eine negative Beeinflussung durch überschrittene Grenzlasten ausgeschlossen werden.

[13] Die Berechnung der Auslastung ist in Übereinstimmung mit der Arbeit von Liu und Layland [Seite 47 ff.][74] dargelegt.

3.4.2 Dynamische Architekturanalyse

Die statische Architekturanalyse analysiert strukturelle Eigenschaften eines Modells. Eine Stimulation mit spezifischen Eingangsdaten und damit eine Simulation[14] und Bewertung des Laufzeitverhaltens einer Architektur wird nicht erreicht. Bereits während des Systementwurfs ist diese jedoch notwendig, um Echtzeiteigenschaften eines verteilten Systems zu bewerten. Speziell für heutige Software- und Systemarchitekturen, die statisch über mehrere Bussysteme festgelegt sind, ist die Bewertung maximaler Laufzeiten über einen Netzwerkpfad relevant.

Zur entsprechenden Umsetzung haben sich Methoden als Teil einer dynamischen Architekturanalyse herausgebildet. Als Eingangsgröße für die Anwendung dieser Methoden werden zwei Arten eines Modells betrachtet:

1. Das Ausführungsmodell einzelner Rechenressourcen worunter ein Steuergerät oder ein Bussystem verstanden wird und

2. das Ausführungsmodell, was durch die Verknüpfung einzelner Rechenressourcen als ausführbares Systemmodell festgelegt ist und somit die Bewertung von verteilten Softwarekomponenten ermöglicht.

Ausführungsmodell von Rechenressourcen

Ein Ausführungsmodell einer Rechenressource kann sich auf einen Prozessor eines Steuergeräts oder auf ein Bussystem beziehen. Für einen Prozessor als mögliche Realisierung einer Recheneinheit besteht dabei die Aufgabe Tasks[15] unter Einhaltung von Echtzeitanforderungen auszuführen.

[14] Nach Hedtstück [Seite 3][54] ist eine Simulation ein Verfahren, bei dem für ein reales oder imaginäres System ein Modell erstellt wird, das experimentell untersucht werden kann mit dem Ziel, neue Erkenntnisse über das System zu gewinnen und daraus Handlungsanweisungen abzuleiten.

[15] Ein Task beschreibt Software, die auf einem Steuergeräte ausgeführt wird [Seite 27][97].

Für ein Bussystem besteht die Aufgabe Nachrichten unter Einhaltung von Echtzeitanforderungen zwischen zwei Steuergeräten d.h. einem Sender und einem oder mehreren Empfänger(n) zu übertragen. In Bild 3.17 ist beispielhaft ein Ausführungsmodell für einen Prozessor als eine Rechenressource, welcher ein Task *w* zugeordnet ist, dargestellt. Der Zusammenhang gilt gleichermaßen für ein Bussystem, welchem Nachrichten zugeordnet sind.

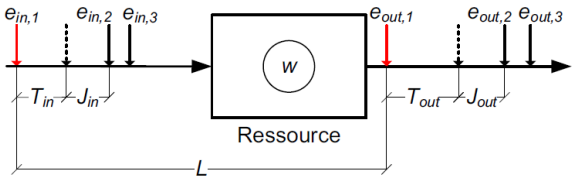

Bild 3.17: Beispiel für einen auf eine Ressource gemappten Task sowie einen Eingangsereignisstrom E_{in} und den resultierenden Ausgangsereignisstrom E_{out} in Übereinstimmung mit Traub [Seite 14] [109]

Die Modellierung des Tasks *w* wird über ein Ein-/ Ausgabeverhalten, das einen Eingangsereignisstrom bzw. einen Ausgangsereignisstrom darstellt, erreicht. Die Eingangsereignisse modellieren dabei die Aktivierungszeitpunkte des Tasks während der Ausgangsereignisstrom die entsprechenden Beendigungszeitpunkte modelliert. Zusätzlich werden die Ereignisströme mit den Jittereigenschaften J_{in} und J_{out} zur Beschreibung von Taktschwankungen beschrieben. Die Abstände zwischen zwei Eingangsereignissen bzw. zwei Ausgangsereignissen sind abschließend über die Zeitintervalle T_{in} und T_{out} festgelegt.

Analysefokus

Im Hinblick auf Echtzeitanforderungen ist die zentrale Eigenschaft des Modells die Latenzzeit *L* als Differenz zwischen zwei bestimmten Aktivierungs- und Beendigungszeitpunkten eines Tasks. In Bild 3.17 ist dies beispielhaft

für den Aktivierungszeitpunkt $e_{in,1}$ und den Beendigungszeitpunkt $e_{out,1}$ gezeigt.

Scheduling

Im Normalfall ist die Latenzzeit eines Tasks durch eine Menge anderer Tasks, die auf derselben Ressource ausgeführt werden, beeinflusst. Um diesen Zusammenhang zu koordinieren können Prioritäten vergeben werden, welche von einem Prozessor berücksichtigt sind und zu einer Zuweisungsstrategie[16] von Tasks zur Rechenressource führen.

Generell gilt, dass es für einen Task mit einer niedrigen Priorität durch ein Scheduling dazukommen kann, dass er trotz seiner Aktivierung durch ein Eingangsereignis warten muss bis höherpriore Tasks ausgeführt wurden. Dieser Zusammenhang ist in Bild 3.18 dargestellt.

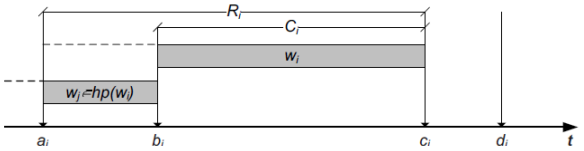

Bild 3.18: Timing-Eigenschaften eines Tasks w_i, welcher durch höherpriore Tasks w_j als Elemente von $hp(w_i)$ verzögert wird in Übereinstimmung mit Traub [Seite 11][109]

Die höherprioren Tasks (w_j als Elemente von $hp(w_i)$) verzögern dabei die Bearbeitung des Tasks w_i, obwohl der entsprechende Aktivierungszeitpunkt a_i durch ein Eingangsereignis bereits eingetroffen ist. Der Task wird resultierend daraus erst zum Zeitpunkt b_i ausgeführt und terminiert zum Zeitpunkt c_i.

Das Zeitintervall zwischen den Zeitpunkten b_i und c_i wird als Ausführungszeit C_i und das Zeitintervall von dem Aktivierungszeitpunkt a_i bis zum Be-

[16] Diese Zuweisungsstrategie wird auch als Scheduling bezeichnet [Seite 70][99].

endigungszeitunkt c_i als Antwortzeit R_i bezeichnet. Der Zeitpunkt d_i beschreibt die zeitliche Grenze bis zu welcher der Task spätestens ausgeführt worden sein muss.

Analysefokus

Eine Überprüfung, ob die Einhaltung der zeitlichen Grenze für alle Tasks erreicht wird, kann über eine Schedulinganalyse für eine Rechenressource dargestellt werden. Die entsprechenden Analyseverfahren ermitteln dabei für alle Tasks bzw. Nachrichten einer Rechenressource Best-Case und Worst-Case Antwortzeiten. Ein entsprechender Ansatz für die Bewertung eines Schedulings über die Berechnung von Best-Case und Worst-Case Antwortzeiten kann bei Kollmann et al. [67][66] gefunden werden.

Timingbewertung auf Systemebene

Die Timingbewertung auf Systemebene umfasst die Modellierung eines Netzwerkpfads und nicht nur einer Rechenressource als Prozessor oder Bussystem. In Bild 3.19 ist in diesem Zusammenhang ein Pfad mit dem Task 1, der Nachricht 2, dem Task 6 als auch der entsprechenden Rechenressourcen im Rahmen der Prozessoren 1 und 2 sowie eines Bussystems gezeigt.

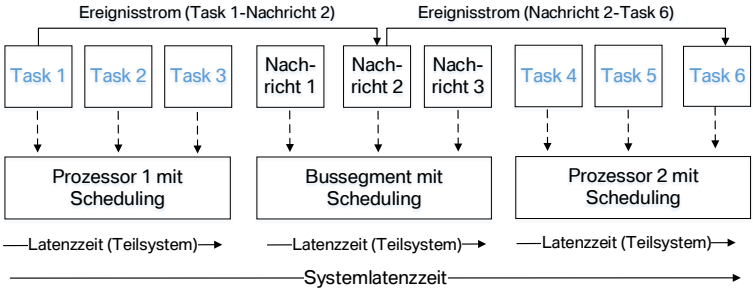

Bild 3.19: Timingbewertung auf Systemebene

Die beiden Tasks und die Nachricht sind durch Ereignisströme miteinander verknüpft. Zeitlich betrachtet weist der Pfad eine Systemlatenzzeit auf, welcher durch die Latenzzeiten der Teilsysteme festgelegt ist. Heute bekannte Ansätze zur Bestimmung dieser Systemlatenzzeiten arbeiten ein Modell sequentiell ab.

Hierbei wird für jedes Teilsystem eine Latenzzeit bestimmt und anschließend ein Ereignisstrom für das nachfolgende Teilsystem abgeleitet. Im Beispiel sind die Latenzzeiten der Teilsysteme durch das Scheduling der Tasks auf den Prozessoren 1 und 2 als auch das Scheduling der Nachricht auf dem Bussegment gegeben. Entsprechend einer Abarbeitung des Modells von links nach rechts entsteht hierdurch der Ereignisstrom zwischen Task 1 und Nachricht 2 und der Ereignisstrom zwischen Nachricht 2 und Task

6. Zwei mögliche Verfahren, um eine entsprechende dynamische Architekturanalyse des hierdurch festgelegten Wirkzusammenhangs zu bestimmen sind durch den Symbolische Laufzeitanalyse für Systeme (SymTA/S) Ansatz [96] und den Real-Time Calculus (RTC) Ansatz [107] gegeben.

3.4.3 Zusammenfassung

Im Entwurf heutiger Signal-orientierter Software- und Systemarchitekturen haben statische und dynamische Analysemethoden eine Bedeutung. Ihre grundlegende Aufgabe ist es zu analysieren, ob Anforderungen an die Auslastung einzelner Bussegmente eingehalten werden.

Ausgehend von eingehaltenen Grenzwerten für die Buslast sind weitergehend zeitliche Anforderungen für Wirkzusammenhänge, die über ein Bussegment oder mehrere Bussegmente verteilt sind, zu bewerten. Die statische Kommunikationsfestlegung heutiger Software- und Systemarchitekturen begünstigt diese Analysen, weil die entsprechenden Netzwerkpfade bereits während der Entwicklungszeit bekannt sind.

Angewandte Verfahren für die statische und die dynamische Architekturanalyse sind Metriken, Bewertungsskripte, der RTC Ansatz und der SymTA/S Ansatz. Tabelle 3.2 ordnet die Verfahren und zeigt mögliche Vertreter auf.

Architekturanalyse	**Ansatz**	**Vertreter**
Statisch	Metrik	[35], [48]
Statisch	Bewertungsskript	[42]
Dynamisch	RTC	[52], [31]
Dynamisch	SymTA/S	[109], [49]

Tabelle 3.2: Übersicht statischer und dynamischer Analysemethoden

4 Bewertung und Abgrenzung

Der Stand der Wissenschaft und Technik geht vom Entwurf der Software- und Systemarchitektur entlang eines Signal-orientierten Architekturstils aus. Für eine entsprechende *Bewertung* (s. Abschnitt 4.1) soll erörtert werden inwieweit das Vorgehen verändert werden muss, um Anforderungen an einen zukünftigen Entwicklungsprozess für Software umzusetzen.

Im Fokus dieser Veränderung steht die Betrachtung des Service-orientierten Architekturstils, wodurch unterschiedliche Potentiale im Zusammenhang mit einer verbesserten Entkopplung von Software und Hardware gehoben werden können (s. Abschnitt 2.5). Speziell zur Einbindung von *Service-Orientierung* in die frühe Phase der Entwicklung muss jedoch eine Abgrenzung von der heutigen Umsetzung des Systementwurfs im Rahmen des V-Modells erreicht werden (s. Abschnitt 4.2).

4.1 Bewertung des Stands der Wissenschaft und Technik

Der Entwurf der heutigen Software- und Systemarchitektur wird über Bezüge zwischen unterschiedlichen Ebenen einer Architektur ausgedrückt (s. Bild 3.3). Dabei werden zuerst Fahrzeugfunktionen durch eine Dekomposition in logische Funktionen verfeinert. Nachfolgend sind logische Funktionen und die entsprechenden Informationsflüsse in Softwarekomponenten und *Signale* im Rahmen des Modells einer Softwarearchitektur festgelegt.

Zur Abbildung der Softwarearchitektur auf eine Topologie wird anschließend die Partitionierung von Softwarekomponenten auf Recheneinheiten eines Steuergeräts vorgegeben. Der daraus resultierende Informationsfluss über Systembusse wird abschließend durch die Zuordnung von Signalen auf Busnachrichten festgelegt und in der Kommunikationsmatrix als zentrales Beschreibungskonzept zusammengefasst (vgl. Abschnitt 2.3).

Entwurfsmethodik

Zur Entwurfsunterstützung für eine Signal-orientierte Architektur wird heute ein modellbasierter Ansatz angewandt. Wesentlich hierfür ist der in Abschnitt 3.3 eingeführte Prozess der *Architekturbeschreibung* (s. Bild 4.1).

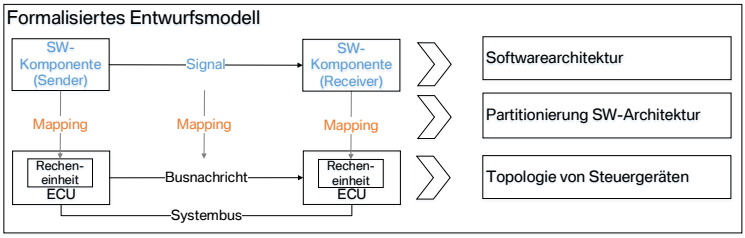

Bild 4.1: Architekturbeschreibung als Methode zur Entwurfsformalisierung Signal-orientierter Architekturen

Über die Architekturbeschreibung kann aus den eingeführten Bestandteilen und Beziehungen der Software- und Systemarchitektur ein *Entwurfsmodell* formalisiert werden. Das formale Entwurfsmodell beruht dabei auf den Konzepten einer *Beschreibungssyntax* und ihrer *Semantik*. Als Beiträge dieses Vorgehens finden sich zwei Aspekte:

1. Der Aufbau eines Entwurfs, welcher für die Beschreibung seiner Bestandteile und deren Beziehungen durch Syntax und Semantik festgelegten Regeln inklusive einer entsprechenden Bedeutung folgt und

2. die Möglichkeit Modellanalysen als frühzeitige Absicherung der Architektur anwenden zu können.

Hierfür wurden in Abschnitt 3.4 entsprechende Verfahren im Rahmen der statischen und der dynamischen Architekturanalyse eingeführt.

Einfluss auf Produktentwicklung

Der etablierte Stil einer Signal-orientierten Architektur folgt dem Aufbau aus Bild 4.1. Dabei beeinflusst die während der Entwicklungszeit festgelegte Zuordnung von *Signalen* auf *Busnachrichten* eines Kommunikationsnetzwerks die *Fahrzeugproduktentwicklung* an den folgenden Stellen grundlegend.

Funktionale Sicherheit Alle heutigen, auf dem Markt verfügbaren Fahrzeuge realisieren sicherheitskritische Fahrzeugfunktion, wie die Regelung des Antriebsstrangs oder der Fahrdynamik[1]. In Bezug auf die *rechtzeitige* und *korrekte* Ausführung dieser Funktionen sind hohe Anforderungen, zusammengefasst über den eingeführten Begriff eines *eingebetteten echtzeitfähigen Systems* (s. Abschnitt 2.1.1), gestellt.

[1] In Abschnitt 2.2.2 wurde ein entsprechendes Beispiel hierfür im Rahmen eines Spurhalteassistenten eingeführt.

Zur Entwicklung eines solchen Systems stellt die Signal-orientierte Architektur dahingehend eine Unterstützung bereit, dass die Kommunikation zwischen verteilten Softwarekomponenten über statische, während der *Entwicklung* festgelegte Netzwerkpfade verläuft. Diese sind somit zum Einen bekannt und können abgesichert werden, zum Anderen besteht während der Laufzeit des Systems keine Möglichkeit einen geänderten, potentiell nicht abgesicherten Pfad für die Übermittlung eines Signals zu nutzen.

Änderungsflexibilität von Software Die zweite grundlegende Beeinflussung, welche über das Konzept einer Signal-orientierten Architektur auf die Fahrzeugproduktentwicklung ausgeübt wird, betrifft die *Änderungsflexibilität von Software*. Hierbei wird aktuell seitens der Automobilindustrie die Anforderung gestellt, die Softwarearchitektur *unabhängig* von einem entsprechenden Kommunikationsnetzwerk ändern zu können. Das technische Konzept einer Signal-orientierten Architektur ist im Hinblick auf die Erfüllung dieser Anforderung beeinträchtigt, weil in ihr beide Bestandteile durch eine feste Abbildung miteinander verbunden sind.

Die *Änderung der Softwarearchitektur* für verteilte Softwarekomponenten[2] führt dabei grundsätzlich zu *Folgeänderungen* an der Ist-Konfiguration des Kommunikationsnetzwerks (s. Abschnitt 2.3). Erst wenn diese durchgeführt sind kann über entsprechende Implementierungs- und Absicherungsphasen im Rahmen des V-Modells eine Änderungsfreigabe als *Softwarerelease* erfolgen (s. Bild 4.2).

[2] Bei verteilten Softwarekomponenten wird gemeint, dass diese auf mindestens zwei unterschiedlichen Steuergeräten partitioniert sind und für die Kommunikation ein Systembus eingesetzt wird.

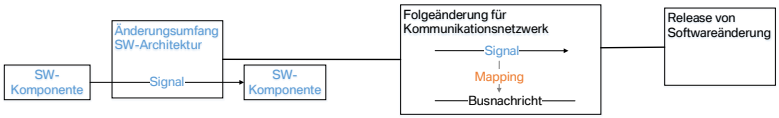

Bild 4.2: Änderung und Folgeänderung in Signal-orientierten Architekturen

Beitrag der Service-orientierten Architektur

Der gezeigte Zusammenhang für die Abhängigkeit zwischen Änderungen der Softwarearchitektur und Folgeänderungen an einem Kommunikationsnetzwerk (s. Bild 4.2) ist durch die technische *Kopplung* beider Bestandteile im Rahmen einer Signal-orientierten Architektur begründet. Für einen alternativen Architekturstil im Rahmen der Service-orientierten Architektur wurde eine entsprechende *Entkopplung* bereits als Kennzeichen erfasst (s. Abschnitt 2.4).

Das Potential für die Produktentwicklung liegt abgeleitet daraus in der Weiterentwicklung der Softwarearchitektur, die grundsätzlich ohne Folgeänderungen an einem physischen Kommunikationsnetzwerk erreicht wird (s. Bild 4.3). Hierbei ist im Speziellen auch für die nachträgliche Anpassung von Software bereits ausgelieferter Fahrzeuge ein Potential zu heben (s. Abschnitt 2.5).

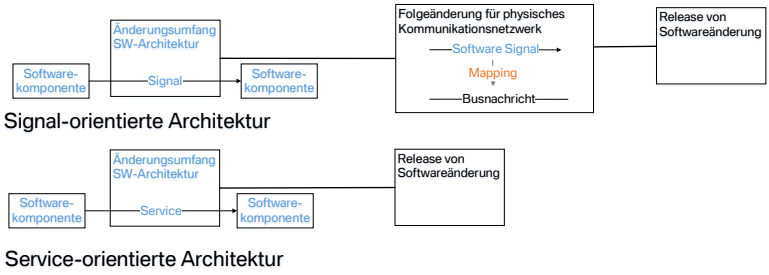

Bild 4.3: Vergleich von Änderungen der Softwarearchitektur

Als Vorbedingung für die Realisierung einer solchen Änderung der Softwarearchitektur über Implementierungs- und Absicherungsphasen des V-Modells muss jedoch eine festgelegte und abgesicherte Software- und Systemarchitektur vorliegen. Die Entwurfsmethodik dazu ist für Signal-orientierte Architekturen gegenwärtig und basiert auf der Annahme einer statischen Kommunikationsfestlegung über Systembusse.

Durch das technische Konzept einer Service-orientierten Architektur entfällt diese Annahme, weshalb mögliche *Anpassungen* der heutigen Entwurfsmethodik durch die Einführung einer Service-orientierten Architektur zu betrachten sind.

4.2 Abgrenzung zum Stand der Wissenschaft und Technik

Als Ausgangspunkt für die Ermittlung des *Anpassungsbedarfs* wird das etablierte Entwurfsmodell der Software- und Systemarchitektur für Signal-orientierte Architekturen herangezogen. Dabei wird für drei charakteristische Aspekte dieses Modells durch Deltabetrachtungen im Folgenden ermittelt, welche Anpassungen für ein entsprechendes Modell bei Service-orientierten Architekturen notwendig sind. Zur Umsetzung der Deltabetrachtungen kommen *Klassendiagramme* und *Sequenzdiagramme* als Spezifikationstechniken der UML zum Einsatz.

4.2.1 Deltabetrachtung 1: Softwarearchitektur

Die erste Deltabetrachtung betrifft die Gegenüberstellung des Modells einer Softwarearchitektur im Rahmen Signal-orientierter und Service-orientierter Architekturen. Dabei wird zwischen der *Struktur* von Softwarekomponenten und dem *Verhalten* von Softwarekomponenten differenziert.

Struktur von Softwarekomponenten

Zur Umsetzung der Strukturbetrachtung sind zunächst entsprechende Metamodelle für Softwarekomponenten in beiden Architekturstilen über Klassendiagramme formalisiert und einander gegenübergestellt.

Signal-orientierte Architektur Für Signal-orientierte Architekturen wird die Strukturfestlegung einer Softwarekomponente durch fünf Elemente mit den entsprechenden Klassen erreicht (s. Bild 4.4): 1. Die *Komponente* und ihre 2. entsprechenden *Ports*, 3. die Links zwischen Ports unterschiedlicher Komponenten als *Sender-Receiver Schnittstellen*, 4. die dabei betrachteten *Signale* zur Festlegung des Informationsaustauschs und 5. die Paradigmen der *zyklischen* und *ereignisorientierten* Kommunikation.

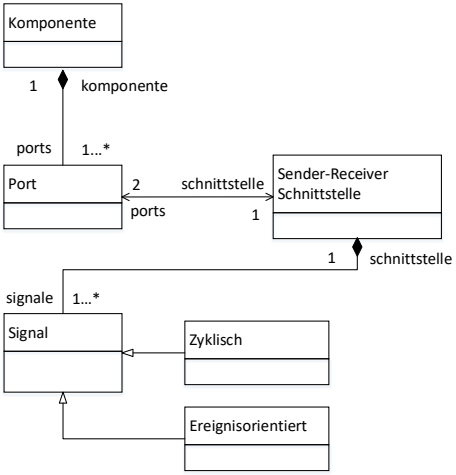

Bild 4.4: Struktur einer Softwarekomponente für Signal-orientierte Architekturen

Service-orientierte Architektur Demgegenüber steht die Struktur einer Softwarekomponente für Service-orientierte Architekturen (s. Bild 4.5). Die Festlegung ist hierbei auch durch fünf Elemente, von denen die ersten beiden die 1. *Komponente* und ihre entsprechenden 2. *Ports* beschreiben, erreicht. Die 3. *Links* zwischen *Ports* unterschiedlicher Komponenten als *Client-Server Schnittstellen* umfassen jedoch keine Signale, sondern bereitgestellte oder genutzte 4. *Services*. Diese können abschließend über das 5. *Publish-Subscribe Paradigma* und das *Request-Response Paradigma* umgesetzt werden[3].

[3] Das Request-Response Paradigma unterscheidet sich vom in Abschnitt 2.4 eingeführten Publish-Subscribe Paradigma durch einen Client der über Request Nachrichten anforderungsgetrieben einen Server aufruft und nicht passiv auf Benachrichtigungen eines Servers, wie beim Publish-Subscribe Paradigma, wartet. In der Softwaretechnik wird dieser Vorgang auch als Methode bezeichnet [Seite 30][22].

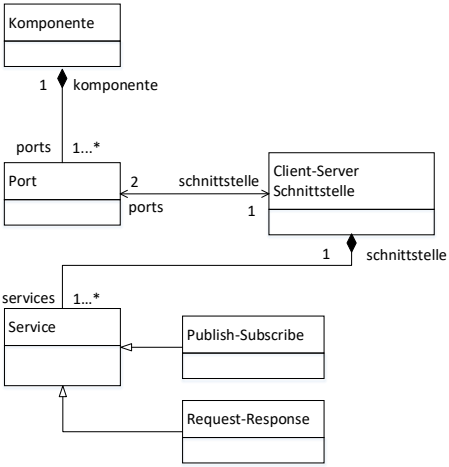

Bild 4.5: Struktur einer Softwarekomponente für Service-orientierte Architekturen

Verhalten von Softwarekomponenten

Aus der Strukturfestlegung kann die Ableitung von ausgetauschten Informationen zwischen Softwarekomponenten über Signale oder Services erreicht werden. Im Folgenden wird dies als Verhalten erachtet.

Als übergeordnete Konzepte zur Beschreibung der Informationsflüsse kommen dabei der Sender-Receiver Ansatz für Signal-orientierte Architekturen und der Client-Server Ansatz für Service-orientierte Architekturen zum Tragen. Zur Kenntlichmachung von Unterschieden zwischen beiden Konzepten werden im Folgenden Sequenzdiagramme als Spezifikationstechnik eingesetzt.

Sender-Receiver Ansatz Die Struktur einer Softwarekomponente im Rahmen Signal-orientierter Architekturen legt gesendete und empfangene Signale einer Komponente fest (s. Bild 4.4).

Jede Softwarekomponente verhält sich somit für eine spezifische Interaktion betrachtet entweder als Sender oder Receiver eines unidirektionalen Informationsflusses. In Bild 4.6 ist dies über die Modellierung von entsprechenden Sequenzdiagrammen dargestellt.

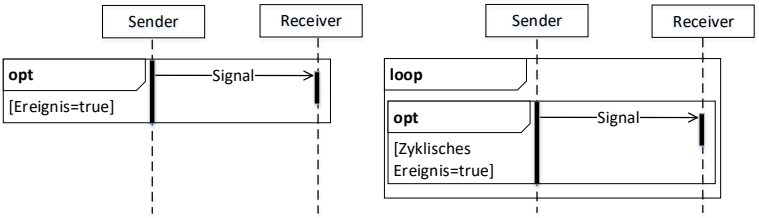

Bild 4.6: Ereignisorientierte Umsetzung (links) und zyklische Umsetzung (rechts)

Die Sequenzdiagramme in Bild 4.6 unterscheiden sich dahingehend, dass der Informationsfluss ereignisorientiert oder zyklisch umgesetzt werden kann. Bei der ereignisorientierten Umsetzung wird dabei vom Sender ein entsprechender Trigger, beispielsweise die Änderung eines Wertes wie der Fahrzeugtemperatur, als Initiator für den Versand eines Signals erkannt. Für eine zyklische Umsetzung steht ein Versand zu regelmäßigen, äquidistanten Zeitpunkten.

Client-Server Ansatz Die Struktur einer Softwarekomponente im Rahmen Service-orientierter Architekturen legt bereitgestellte und konsumierte Services einer Komponente fest (s. Bild 4.5).

Jede Softwarekomponente verhält sich somit für eine spezifische Interaktion betrachtet entweder als Client oder Server. Gemäß der für eine Interaktion anwendbaren Paradigmen, dem Publish-Subscribe und dem Request-Response Paradigma, bilden sich bidirektionale Informationsflüsse aus. In Bild 4.7 ist dies über die Modellierung von entsprechenden Sequenzdiagrammen dargestellt.

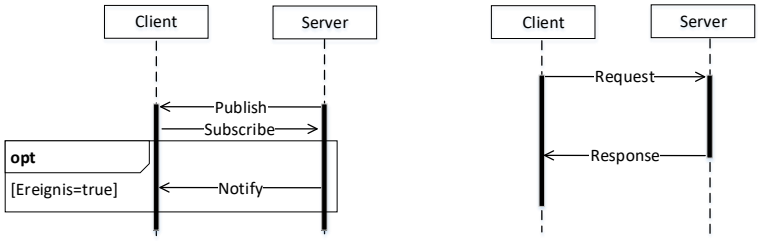

Bild 4.7: Publish-Subscribe Umsetzung (links) und Request-Response Umsetzung (rechts)

Der Unterschied kann hierbei in der Initiierung der Kommunikationsbeziehung, die für das Publish-Subscribe Paradigma vom Service und für das Request-Response Paradigma vom Client ausgeht, ausgemacht werden.

Weitergehend wird der für das Publish-Subscribe Paradigma dargestellte Informationsfluss im Rahmen der Notify Nachricht im Wesentlichen ereignisorientiert durch einen entsprechenden Trigger initiiert. Somit verhält sich diese Nachricht in Übereinstimmung mit dem ereignisorientierten Versand eines Signals im Rahmen des Sender-Receiver Ansatzes (s. Bild 4.6).

In Fällen, in denen ein Client kontinuierlich, zu äquidistanten Zeitpunkten benachrichtigt werden soll, ist jedoch auch eine zyklische Umsetzung möglich (s. Bild 4.8).

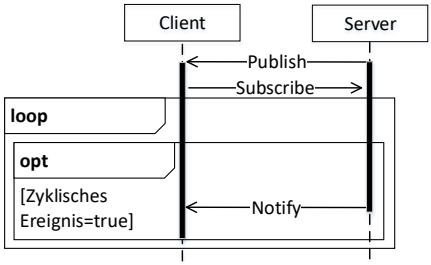

Bild 4.8: Zyklische Umsetzung des Publish-Subscribe Paradigmas

Eine mögliche Anwendung hierfür sind Fahrzeugfunktionen mit Echtzeitanforderungen, die heutzutage über ein entsprechendes Kommunikationsprinzip realisiert werden.

4.2.2 Deltabetrachtung 2: Partitionierung

Die Deltabetrachtung 2 stellt die Abbildungsbeziehungen der Softwarearchitektur auf Steuergeräte und damit die Partitionierung in beiden Architekturstilen einander gegenüber.

Signal-orientierte Architektur

Die Partitionierung der Softwarearchitektur für heutige Signal-orientierte Architekturen wird auf Steuergeräten, welche durch die in Abschnitt 2.2.2 eingeführte E/E-Domänenarchitektur entlang einer Fahrzeugdomäne kategorisiert sind, erreicht.

Um die Partitionierung von Softwarekomponenten auf einem nach Fahrzeugdomänen kategorisierten Steuergerät modellhaft auszudrücken, kann das Mapping-Konzept angewandt werden (vgl. Abschnitt 3.12).

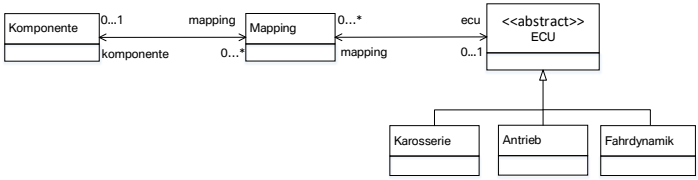

Bild 4.9: Partitionierung von Softwarekomponenten in heutigen Signal-orientierten Architekturen

In Bild 4.9 ist eine mögliche Darstellung hierfür erreicht. In dem gezeigten Klassendiagramm wird die Klasse *Komponente* zunächst über die Klasse *Mapping* mit der abstrakten Klasse *ECU* assoziiert. Hierdurch ist die Partitionierung einer Softwarekomponente grundsätzlich ausgedrückt. Für die Einordnung eines Steuergeräts in eine Fahrzeugdomäne werden jedoch weitere Klassen benötigt. In diesem Zusammenhang repräsentieren diese die

Domänen *Karosserie*, *Antrieb* und *Fahrdynamik*, wodurch die Instanziierung der abstrakten Klasse *ECU* entlang einer Domäne ermöglicht wird.

Service-orientierte Architektur

Für die Einführung eines entsprechenden Modells bei Softwarearchitekturen entlang des Service-orientierten Architekturstils wird ein Übergang der heutigen E/E-Domänenarchitektur zu einer zentralisierten E/E-Architektur angenommen [Seite 1 ff.][95]. Für die Beschreibung von Unterschieden bei der Partitionierung von Softwarekomponenten im Vergleich zur Signal-orientierten Architektur sollen deshalb zunächst die Auswirkungen dieses Übergangs betrachtet werden. Als Mittel wird hierzu eine Gegenüberstellung wesentlicher Eigenschaften einer E/E-Domänenarchitektur und einer zentralisierten E/E-Architektur erreicht (s. Tabelle 4.1).

Eigenschaft	**Domänenarchitektur**	**Zentralisierung**
Partitionierung	Domänenspezifische ECUs	Plattform ECUs
Vernetzung	Systembus	Middleware und Ethernet
Kommunikation	Statisch	Dynamisch

Tabelle 4.1: Gegenüberstellung von Domänenarchitektur und zentralisierter E/E-Architektur

Unterscheidungsmerkmale

Aus der Gegenüberstellung können unterschiedliche Typen eines Steuergeräts, welche bei der Partitionierung von Softwarekomponenten in beiden E/E-Architekturen betrachtet sind, erkannt werden.

Für die E/E-Domänenarchitektur stehen hierbei domänenspezifische Steuergeräte (s. Abschnitt 2.2.2), die Kommunikationsbeziehungen über Nachrichten auf einem Systembus statisch durch das Konzept der Kommunikationsmatrix festlegen (s. Abschnitt 2.3). In zentralisierten E/E-Architekturen

wird hingegen die Partitionierung von Softwarekomponenten domänenübergreifend auf Plattformsteuergeräten erreicht. Weitergehend unterscheiden sich diese Plattformsteuergeräte durch die Ableitung dynamischer Kommunikationsbeziehungen im Rahmen eines Netzwerks, das auf einer Service-orientierten Middleware kombiniert mit der Ethernet Technologie basiert (s. Abschnitt 2.4).

Modellierung der Partitionierung

Für Softwarekomponenten einer Service-orientierten Architektur folgt aus dem Übergang einer heutigen E/E-Domänenarchitektur zu einer zentralisierten E/E-Architektur die Notwendigkeit eine Partitionierung auf domänenübergreifenden Plattformsteuergeräten zu berücksichtigen. Zur Modellierung dieses Zusammenhangs kann wiederum das Konzept von Mappings angewandt werden (s. Bild 4.10).

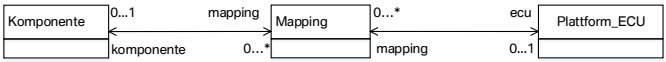

Bild 4.10: Partitionierung von Softwarekomponenten in Service-orientierten Architekturen

4.2.3 Deltabetrachtung 3: Laufzeitverhalten

Für die Deltabetrachtung 3 wird davon ausgegangen, dass die Softwarearchitektur bereits auf Steuergeräte partitioniert ist und sich nun Kommunikationsbeziehungen zwischen verteilten Softwarekomponenten als Laufzeitverhalten ausbilden können.

Statisches Laufzeitverhalten

Eine Signal-orientierte Architektur legt das Laufzeitverhalten in einem Kommunikationsnetzwerk statisch fest. Der Anbieter einer Information (Sender) und der Nutzer einer Information (Receiver) kommunizieren dabei immer über einen festgelegten Netzwerkpfad (s. Bild 4.11).

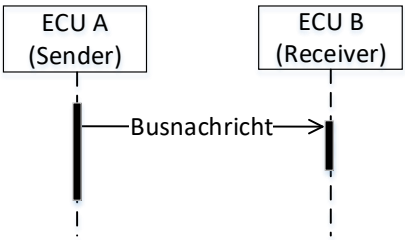

Bild 4.11: Statischer Netzwerkpfad für Sender-Receiver Ansatz

Auf diesem Netzwerkpfad dienen Busnachrichten, die zwischen den entsprechenden Steuergeräten des Senders und des Receivers übermittelt werden, als Träger von Signalen.

Dynamisches Laufzeitverhalten

In einer Service-orientierten Architektur können Netzwerkpfade während der Laufzeit zwischen dem Anbieter einer Information (Server) und dem Nutzer einer Information (Client) konfiguriert und rekonfiguriert werden. Das entsprechende Konzept der Laufzeitrekonfiguration stellt dabei eine

neue Anforderung im Rahmen des Entwurfs einer Software- und Systemarchitektur dar.

Architekturvoraussetzung

Die Laufzeitrekonfiguration setzt eine Softwarearchitektur, in der mindestens zwei Server denselben Service instanziieren, voraus. Ein Client kann dann während der Laufzeit zwischen der Nutzung unterschiedlicher Instanzen eine Rekonfiguration bewirken.

In Bild 4.12 ist ein Modell für eine Laufzeitrekonfiguration zwischen einem Client und zwei anbietenden Instanzen eines Services, dem Server 1 und dem Server 2, gegeben. Alle drei Komponenten sind auf unterschiedlichen Steuergeräten partitioniert und kommunizieren über Ethernet Nachrichten im Zusammenhang mit dem Publish-Subscribe Paradigma.

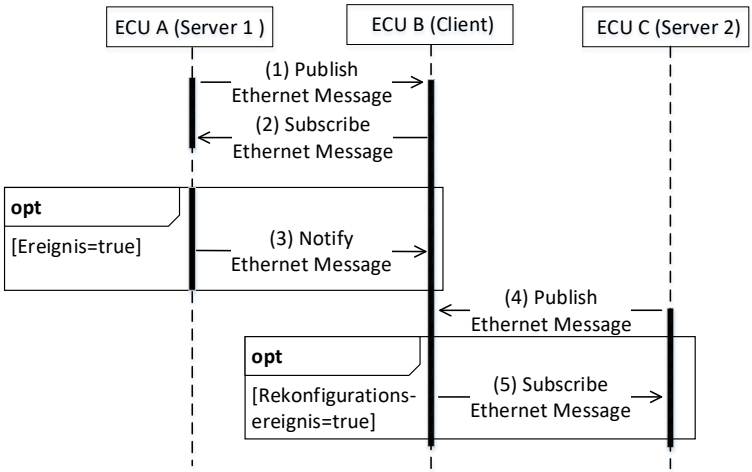

Bild 4.12: Laufzeitrekonfiguration

Dem Paradigma folgend werden zuerst die (1) Publish Nachricht und die (2) Subscribe Nachricht zur Initiierung einer Nutzungsbeziehung zwischen dem Client und dem Server 1 versendet. Mit dem Erhalt der (2) Subscribe Nachricht durch Server 1 beginnt die Servicenutzung. Diese ermöglicht es dem Client Benachrichtigungen über (3) Notifiy Nachrichten, die im gezeigten Fall ereignisorientiert versendet werden, zu empfangen.

Die Laufzeitrekonfiguration wird anschließend gezeigt. Dabei wird zunächst als Voraussetzung für die Rekonfiguration der Server 2, der sich im Rahmen einer (4) Publish Nachricht dem Client anbietet, betrachtet. Die Publish Nachricht kann dabei wie gezeigt *nach* der Publish Nachricht von Server 1 oder *zeitgleich* mit deren Übertragung gesendet werden. Dies kann beispielsweise im Rahmen eines gleichzeitigen Aufstartens der entsprechenden Steuergeräte geschehen. Die weitergehende Umsetzung der Rekonfiguration umfasst dann das Eintreten eines *Rekonfigurationsereignisses* und die entsprechende (5) Subscribe Nachricht, um die Initiierung einer Nutzungsbeziehung mit Server 2 und damit die Laufzeitrekonfiguration abzuschließen.

Eine mögliche Anwendung der Laufzeitrekonfiguration ist im Rahmen von Ausfällen eines Steuergeräts bzw. der Degradation darauf partitionierter Services gegeben. Hierbei kann der Ausfall als Rekonfigurationsereignis betrachtet werden, welches zum dynamischen Aufbau einer Kommunikationsbeziehung mit einer zweiten Instanz eines Services auf einem funktionsfähigen Steuergerät herangezogen wird [91].

4.2.4 Zusammenfassung

Die Deltabetrachtungen im vorangegangen Abschnitt beschreiben notwendige Anpassungen für das etablierte Entwurfsmodell einer Software- und Systemarchitektur entlang eines *Signal-orientierten Architekturstils*, welche durch die betrachtete Einführung einer *Service-orientierten Architektur* erzwungen werden. Für die *Deltabetrachtung 1* steht als Ergebnis die Berücksichtigung des Modells einer Softwarekomponente, das auf dem *Client-Server Ansatz* basiert. Dieses unterteilt sich weitergehend in eine spezifische *Strukturbeschreibung* und *Verhaltensbeschreibung* von Softwarekomponenten und ihrer Schnittstellen. Im Rahmen der *Deltabetrachtung 2* wurde zunächst die *Annahme* getroffen, dass die Einführung einer Service-orientierten Architektur von einem Übergang der heutigen E/E-Domänenarchitektur zu einer *zentralisierten E/E-Architektur* begleitet wird. Mit Bezug auf aktuelle Forschungsprojekte wie die EPI kann diese Annahme aufrechterhalten werden [Seite 2][95], erfordert jedoch gleichzeitig die Berücksichtigung von *Plattformsteuergeräten*.

Plattformsteuergeräte zielen darauf ab, die Ressourcenallokation von Softwarekomponenten unterschiedlicher Fahrzeugdomänen auf einer geteilten Ressource umzusetzen. Diese Veränderung im Vergleich zur heutigen Partitionierung von Softwarekomponenten auf domänenspezifischen Steuergeräten in einer Signal-orientierten Architektur wird als Resultat der *Deltabetrachtung 2* und relevante Randbedingung eines Entwurfsmodells Service-orientierter Architekturen aufgefasst.

Abschließend und im Zusammenhang mit der *Deltabetrachtung 3* wird das durch Service-Orientierung neue Konzept der *Laufzeitkonfiguration bzw. -rekonfiguration* eingeführt.

5 Konzept einer hybriden Architektur

In den bisherigen Betrachtungen stand nicht im Vordergrund in welchem Ausmaß der Stil einer *Service-orientierten Architektur* im Kontext der Fahrzeugproduktentwicklung eingesetzt werden soll. Im Folgenden wird deshalb nach einzelnen Fahrzeugdomänen (s. Abschnitt 2.2.2) betrachtet, inwieweit es sinnvoll ist die Signal-orientierte Architektur durch eine Service-orientierte Architektur abzulösen. Als Grundidee dieser Betrachtung steht das Konzept eines *hybriden Architekturstils*.

5.1 Hybrider Architekturstil

Zur möglichen Aufteilung von Domänen auf den Service-orientierten und den Signal-orientierten Architekturstil werden die *funktionale Sicherheit* und die Forderung nach *kontinuierlichen Softwareupdates* als Orientierungspunkte eingesetzt.

Fahrzeugdomänen Antrieb und Fahrdynamik

Fahrzeugdomänen mit *hohen* Anforderungen an die funktionale Sicherheit finden durch die statische Kommunikationsfestlegung einer Signal-orientierten Architektur Unterstützung. Hierzu zählen die von der *Antriebsdomäne* und der *Fahrdynamikdomäne* umfassten Funktionen.

In der Vergangenheit wurde deren Entwicklung und Vermarktung kontinuierlich vorangetrieben [Seite 493][55]. Heute wird die Differenzierung zwi-

schen Wettbewerbern vor allem durch die Kopplung von Antrieb und Fahrwerk mit Fahrassistenzsystemen[1] erreicht. Auch für das *autonome Fahren* als abschließende Ausbaustufe eines Fahrerassistenzsystems ändern sich die grundlegenden Anforderungen an die entsprechenden Regelungsfunktionen dabei aber nicht [Seite 274 ff.][116].

Die von beiden Domänen angezogenen Funktionen können folglich als etabliert und von Kunden als vorausgesetzt erachtet werden. Anforderungen an stetige Funktionserweiterungen durch Softwareupdates sind demnach geringfügig ausgeprägt.

Infotainment und Karosserie

Fahrzeugdomänen mit *hohen Anforderungen* an die kontinuierliche Aktualisierung durch Softwareupdates sind die *Infotainmentdomäne* und die *Karosseriedomäne*, welche über die Einbindung von *Consumer Electronics (CE)-Geräten* miteinander interagieren. Wahrnehmbar wird diese Interaktion beispielsweise durch die Berücksichtigung des *Smartphones* als neuartige Form zur Steuerung des Software-basierten Anteils der *Klimaelektronik* oder der *Umsetzung von Audio- und Videoanwendungen* [Seite 1 f.][6].

Um hierbei dann die stetige Synchronisation zwischen der Fahrzeugsoftware und den Anwendungen des Smartphones zu erreichen, müssen entsprechende Änderungen der Softwarearchitektur kontinuierlich durchgeführt werden. Für den fahrzeugbezogenen Anteil besteht aufgrund des aufwändigen Änderungskonzepts im Rahmen der Signal-orientierten Architektur eine geringe Motivation diese als Lösungsweg hierfür einzusetzen. Ergänzend und die Migration zur Service-orientierten Architektur unterstützend ist die Tatsache, dass beide Domänen vergleichsweise geringe Anforderungen an

[1] In diesem Zusammenhang sei auf den in Abschnitt 2.2.2 eingeführten Spurhalteassistenten verwiesen.

die funktionale Sicherheit tragen und somit ein wesentliches Argument für die Umsetzung einer Signal-orientierten Architektur entfällt.

Fahrerassistenz

Eine Sonderrolle nimmt die Fahrzeugdomäne der *Fahrerassistenz* ein, weil sie einerseits eine starke Vernetzung mit den klassischen Domänen des Antriebs und der Fahrdynamik ausweist und deshalb hohe Anforderungen an die funktionale Sicherheit hat. Zum anderen ist es für diese Domäne gleichzeitig notwendig eine bestehende Softwarearchitektur *kontinuierlich* weiterzuentwickeln.

Hierbei spielen im Besonderen *automatisierte Fahrfunktionen* und die dabei eingesetzten Sensortechnologien zur Erkennung des Fahrzeugumfelds (s. Gruyer et al. [Seite 329][51]) eine zentrale Rolle. Um von deren kontinuierlicher Weiterentwicklung, als Beitrag für einen immer höheren Automatisierungsgrad, zu profitieren, muss neben der Integration des Sensors als *Hardwarekomponente* auch die Einbindung einer entsprechenden Schnittstelle in die bestehende Softwarearchitektur eines Fahrzeugs gelingen. Der daraus resultierende Bedarf nach flexiblen Architekturänderungen wird in der Fahrerassistenzdomäne bereits heute durch Anwendung etablierter Vorgehensmodelle der Softwareindustrie, wie beispielsweise Scrum [100] oder Abwandlungen davon [Seite 859 ff.] [39], im Hinblick auf einen Prozess adressiert.

Neben der Prämisse kontinuierlicher Produktänderungen bedarf es für eine erfolgreiche Etablierung dieser Vorgehensmodelle aber auch eines entsprechenden technischen Rahmens. Die Motivation einer Migration der Fahrerassistenzdomäne in eine Service-orientierte Architektur ist dahingehend stark ausgeprägt, muss jedoch in Bezug auf die funktionale Sicherheit gestellte Anforderungen erfüllen.

Zuordnung von Domänen zu Architekturstil

Tabelle 5.1 fasst die Betrachtungen im Rahmen von Anforderungen an die *funktionale Sicherheit* und an *kontinuierliche Softwareupdates* zusammen.

Domäne	**Anforderungen (Funktionale Sicherheit)**	**Anforderungen (Software-updates)**	**Vorgeschlagener Architekturstil**
Antrieb	hoch	gering	Signal-Orientierung
Fahrdynamik	hoch	gering	Signal-Orientierung
Infotainment	gering	hoch	Service-Orientierung
Karosserie	gering	hoch	Service-Orientierung
Fahrerassistenz	hoch	hoch	Service-Orientierung

Tabelle 5.1: Abgeleiteter Vorschlag für die Zuordnung von Domäne zu Architekturstil

Als Ergebnis steht die Empfehlung einen *hybriden Architekturstil* aus Signal-Orientierung und Service-Orientierung zu entwerfen. Demnach können die *Antriebs- und Fahrdynamikdomäne* in einer bestehen *Signal-orientierten Architektur* weiter anforderungsgerecht umgesetzt werden. Für die Domänen *Karosserie und Infotainment* sowie die Domäne der *Fahrerassistenz* wird hingegen eine Migration in eine gemeinsame *Service-orientierte-Architektur* vorgeschlagen. Dies ist im Wesentlichen mit der Möglichkeit, eine höhere Flexibilität bei Softwareänderungen zu erreichen, begründet.

5.2 Entwurf hybrider Software- und Systemarchitektur

Für den Entwurf einer Software- und Systemarchitektur entlang des Service-orientierten Architekturstils wurden Modellbestandteile über Deltabetrachtungen zum etablierten Entwurfsmodell einer Signal-orientierten Architektur abgeleitet (s. Abschnitt 4.2).

Unter Berücksichtigung eines Bestands an Software-basierten Funktionen[2] für den es sinnvoll ist weiter in einer Signal-orientierten Architektur realisiert zu werden, müssen diese Bestandteile jedoch mit dem bestehenden Entwurfsmodell einer Signal-orientierten Architektur gekoppelt werden. Das Entwurfsmodell einer Service-orientierten Architektur wird hierdurch Teil eines *hybriden Modells* zur Abbildung zweier nach Architekturstilen unterscheidbarer Teilsysteme (s. Bild 5.1).

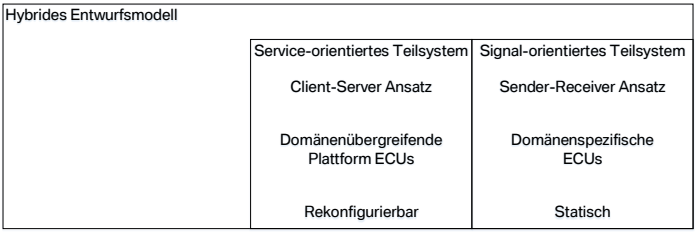

Bild 5.1: Hybrides Entwurfsmodell aus Service-orientiertem und Signal-orientiertem Teilsystem

[2] Ein Ansatz diesen Bestand zu identifizieren kann auf Ebene von Domänen gemäß Tabelle 5.1 und dem dabei vorgenommenen Bezug zu Anforderungen an die funktionale Sicherheit und der Updatefähigkeit gefunden werden. Eine Übersetzung von Domänen in einen konkreten Funktionsbestand wird dann durch Betrachtung der von einer Domäne heute angezogenen Steuergeräte und der darauf partitionierten Software möglich.

Resultierende Handlungsbedarfe

Mit Bezug auf das hybride Entwurfsmodell ergeben sich nachfolgende Handlungsbedarfe. Grundlegend hierfür ist die Frage nach notwendigen *Architektursichten* (s. Abschnitt 3.3.1), um die Kopplung der bisher getrennt betrachteten Teilsysteme beschreibbar zu machen[3].

Weitergehend muss die Entwicklung des Entwurfsmodells in ein übergeordnetes Vorgehensmodell eingebettet sein, um Änderungen einer hybriden Software- und Systemarchitektur implementieren und absichern zu können. Für eine heutige Software- und Systemarchitektur ist dieses Vorgehensmodell durch das V-Modell gegeben und stellt durch seine spezifische Ausgestaltung eine Lösung für das Signal-orientierte Teilsystem dar (s. Abschnitt 3.1).

[3] Ein generisches Konzept für die Ableitung von Architektursichten kann durch eine im Rahmen dieser Arbeit erreichte Publikation [84] eingesehen werden.

6 Entwurfsmethodik für hybride Software- und Systemarchitektur

Zur Anknüpfung an die aufgeworfenen Handlungsbedarfe wird die *Entwurfsmethodik* zur Entwicklung einer hybriden Software- und Systemarchitektur eingeführt.

Hybrides Entwurfsmodell		
	Service-orientiertes Teilsystem	Signal-orientiertes Teilsystem
Sicht 1: Hybride SW-Architektur	Client-Server Ansatz	Sender-Receiver Ansatz
Sicht 2: Partitionierung	Domänenübergreifende Plattform ECUs	Domänenspezifische ECUs
Sicht 3: Laufzeitverhalten	Rekonfigurierbar	Statisch

Bild 6.1: Sichten auf Entwurfsmodell einer hybriden Software- und Systemarchitektur

Hierbei wird das zur hybriden Architektur entsprechende *Entwurfsmodell* (s. Bild 5.1) zunächst über *Architektursichten* (s. Bild 6.1), welche das Service-orientierte und das Signal-orientierte Teilsystem in einen Zusammenhang stellen, erklärt.

6.1 Architektursichten auf das hybride Entwurfsmodell

Sicht 1: Hybride Softwarearchitektur

Die Architektursicht 1 beschreibt eine hybride Softwarearchitektur aus Softwarekomponenten, die über die Kommunikationsparadigmen des Client-Server und des Sender-Receiver Ansatzes miteinander interagieren. Mit Bezug auf den Vorschlag für eine Einteilung von Fahrzeugdomänen auf die beiden Architekturstile (s. Tabelle 5.1) ist diese Kopplung notwendig, wenn eine Fahrzeugfunktion über Softwarekomponenten aus mindestens zwei Domänen mit unterschiedlichem Stil realisiert werden soll[1].

Bild 6.2: Architektursicht 1

Weitergehend ist die Softwarearchitektur entlang der Schnittstellenmodelle in ein Service-orientiertes und ein Signal-orientiertes Teilsystem getrennt. Speziell für die Realisierung von Architekturänderungen im Rahmen eines V-Modells (s. Abschnitt 6.3) wird diese Trennung genutzt, um Unterschiede in Bezug auf den Durchlauf der entsprechenden Phasen für die Teilsysteme aufzuzeigen.

Sicht 2: Partitionierung

Die Architektursicht 2 beschreibt die Partitionierung der hybriden Softwarearchitektur. Das Service-orientierte Teilsystem ist hierbei auf Plattformsteuergeräten in einer Ethernet Topologie partitioniert.

[1] Ein Beispiel hierfür wird in Abschnitt 6.2.1 eingeführt.

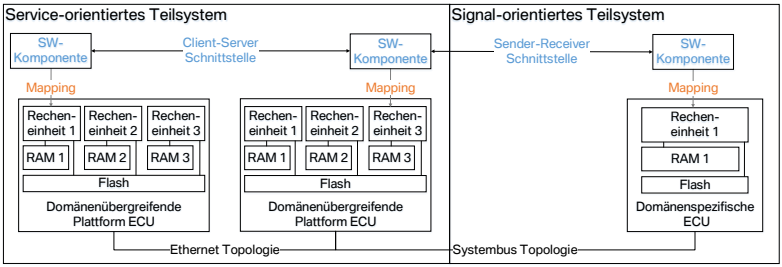

Bild 6.3: Architektursicht 2

Plattformsteuergeräte als neues Konzept für E/E-Architekturen können für die Partitionierung von Softwarekomponenten unterschiedlicher Fahrzeugdomänen ausgelegt sein. Ihre Komponentenarchitektur besteht dabei aus mehreren Recheneinheiten und entsprechenden Speicherkapazitäten.

Dem entgegenstehen Steuergeräte mit einer zentralen Recheneinheit. Dieser Typ eines Steuergeräts ist in heutigen E/E-Architekturen, in denen keine Zentralisierung von Softwarekomponenten unterschiedlicher Domänen erreicht wird und somit geringere Anforderungen an die Rechenleistung gestellt werden, umgesetzt. Für die Partitionierung des Signal-orientierten Teilsystems wird deshalb auf diese Steuergeräte, die spezifisch für eine Domäne oder eine Funktion ausgelegt sind, zurückgegriffen.

Sicht 3: Laufzeitverhalten

Die Architektursicht 3 beschreibt das Laufzeitverhalten der hybriden Softwarearchitektur in einem Kommunikationsnetzwerk. Die Interaktionen zwischen Softwarekomponenten setzten sich dabei aus einem rekonfigurierbaren Kommunikationskonzept für das Service-orientierte Teilsystem und ei-

nem statischen Kommunikationskonzept für das Signal-orientierte Teilsystem zusammen[2].

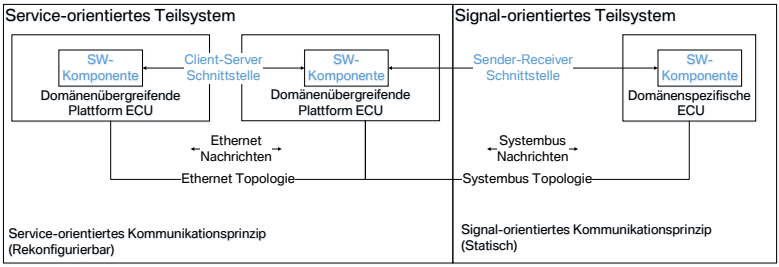

Bild 6.4: Architektursicht 3

Für die Modellierung und Bewertung des Verhaltens einer Fahrzeugfunktion, die sich aus Anteilen beider Kommunikationsprinzipien zusammensetzt, muss diese Separierung jedoch aufgelöst werden.

Eine entsprechende Methode hierzu ist in Abschnitt 6.2.3 eingeführt. Das Ziel der Methode ist die Modellierung des Laufzeitverhaltens von Fahrzeugfunktionen, die über ein Zusammenspiel von Client-Server und Sender-Receiver Schnittstellen realisiert sind, sowie eine sich daran anschließende *Simulation* zur Überprüfung von Laufzeitanforderungen.

Entwicklung der Sichten in einem Entwurfsprozess

Die Entwicklung der Architektursichten geschieht in einem Entwurfsprozess schrittweise und legt abschließend eine hybride Software- und Systemarchitektur fest. Die *Architektursicht 1*, die hybride Softwarearchitektur, stellt dabei den Ausgangspunkt für diese Festlegung dar. Die eigentlichen Entwurfs- und Analyseschritte sind dann in den *Architektursichten 2 und 3*

[2] In diesem Zusammenhang sei auf die in Abschnitt 4.2.3 erreichte Gegenüberstellung der beiden Kommunikationskonzepte verwiesen.

zu finden, in denen Softwarekomponenten der hybriden Softwarearchitektur partitioniert und in Bezug auf ihr Laufzeitverhalten hin bewertet werden.

6.2 Entwurfsprozess

Der Entwurfsprozess zur Umsetzung der Entwurfsmethodik für eine hybride Software- und Systemarchitektur umfasst die Entwicklung der Architektursichten 1-3 über die Prozessschritte 1-9. Die Schritte tragen wie folgt zur Entwicklung der Sichten bei und werden später als Teil des Systementwurfs in das V-Modell eingebunden (s. Abschnitt 6.3).

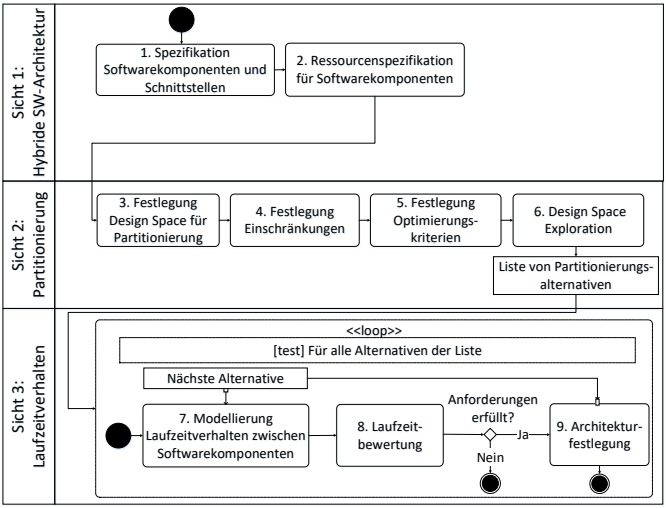

Bild 6.5: Entwurfsprozess

Die *Schritte 1-2* entwickeln die Modellierung der *hybriden Softwarearchitektur* und die Spezifikation von notwendigen Rechenressourcen als Vorbereitung für deren Partitionierung (*Architektursicht 1*), die *Schritte 3-6* setzen eine Design Space Exploration zur Generierung von Alternativen für die *Partitionierung* der Softwarearchitektur um (*Architektursicht 2*) und die *Schritte 7-9* bewerten jede Alternative im Hinblick auf das *Laufzeitverhalten*, wodurch abschließend eine Architekturfestlegung erreicht werden kann (*Architektursicht 3*).

6.2.1 Architektursicht 1: Hybride Softwarearchitektur

Die Architektursicht 1 ist die Vorbedingung, um die Ableitung der Architektursichten 2 und 3 und damit die Festlegung einer Software- und Systemarchitektur zu erreichen. Zur Entwicklung dieser Architektursicht werden die Prozessschritte 1-2 umgesetzt.

Prozessschritt 1: Spezifikation Softwarekomponenten/Schnittstellen

Die Abhängigkeiten zwischen Softwarekomponenten einer hybriden Softwarearchitektur sind durch Schnittstellen auf Basis des Client-Server und des Sender-Receiver Ansatzes festgelegt. In Bild 6.6 ist eine Spezifikation, die sich aus drei Softwarekomponenten mit jeweils einer Client-Server und einer Sender-Receiver Schnittstelle zusammensetzt, gezeigt.

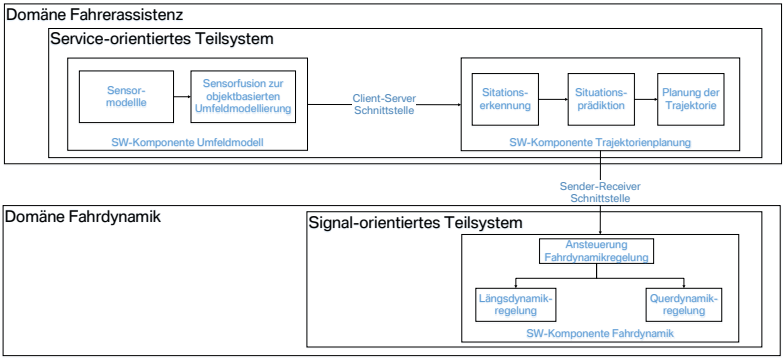

Bild 6.6: Hybride Softwarearchitektur aus Softwarekomponenten und Schnittstellen auf Basis des Client-Server und Sender-Receiver Ansatzes zur Verbindung der Domänen Fahrerassistenz und Fahrdynamik

Die Spezifikation stellt wesentliche Anteile einer automatisierten Fahrfunktion [Seite 420][38] über die Softwarekomponenten des *Umfeldmodells*, der *Trajektorienplanung* und der *Fahrdynamik* dar. Um den internen Aufbau der Softwarekomponenten zu erfassen ist im Weiteren eine Dekomposition in einzelne Funktionsbeiträge angewendet.

Informationsflüsse für automatisierte Fahrfunktion

Der Informationsfluss zwischen den für die Funktion eingeführten Softwarekomponenten wird von der *Umfeldmodellierung*, welche die Sensorik des Fahrzeugs, eine Lokalisierung der eigenen Position sowie das Straßennetz auf Basis digitalisierter Karten über entsprechende *Sensormodelle* fusioniert, initiiert. Die Fusion beschreibt heute im Wesentlichen eine *objektbasierte* Modellierung [Seite 328][51], die alle relevanten Akteure des Verkehrs und zu betrachtende Infrastrukturelemente in Ort und Zeit durch ein Bezugssystem ordnet.

Als softwaretechnische Strukturierung dieser Informationen dienen Listen, welche kontinuierlich über eine Client-Server Schnittstelle an die Softwarekomponente *Trajektorienplanung* bereitgestellt und von dieser als *Fahrsituationen* bzw. Fahrszenarien (s. Bagschik et al. [Seite 132][23]) interpretiert werden. Eine Fahrsituation beschreibt dabei zunächst den aktuellen Ist-Zustand des Fahrzeugs in seinem Umfeld und wird anschließend zu einem möglichen, folgenden Ist-Zustand prädiziert.

Basierend darauf ist die Planung eines Pfades als Trajektorie erreicht und durch Ermittlung von Soll-Werten für die Längs- und Querbewegung (Position und Geschwindigkeit) festgelegt. Die Bereitstellung dieser Werte wird abschließend an die Softwarekomponente *Fahrdynamik* über eine Sender-Receiver Schnittstelle zur Übersetzung in Soll-Werte für eine Längs- und Querdynamikregelung erbracht. Diese Schnittstelle koppelt die Domänen der *Fahrerassistenz* und der *Fahrdynamik*. Durch Einteilung beider Domänen auf unterschiedliche Architekturstile (s. Tabelle 5.1) wird dadurch gleichzeitig das Service-orientierte und das Signal-orientierte Teilsystem miteinander gekoppelt.

Schnittstellenspezifikation Um die relevanten Schnittstellen für den Informationsfluss zwischen den eingeführten Softwarekomponenten zu spezifizieren, wird im Folgenden ein entsprechender Ablauf eingeführt. Durch den hybriden Architekturstil untergliedert sich dieser in zwei Teilabläufe für Client-Server und Sender-Receiver Schnittstellen[3].

Spezifikation einer Client-Server Schnittstelle Für die Spezifikation einer Client-Server Schnittstelle (s. Bild 6.7) werden zunächst die (1) involvierten Softwarekomponenten, die sich auf die Rollen des *Servers* und des *Clients* verteilen, identifiziert und mögliche (2) Abhängigkeiten zu bestehenden Schnittstellen festgehalten.

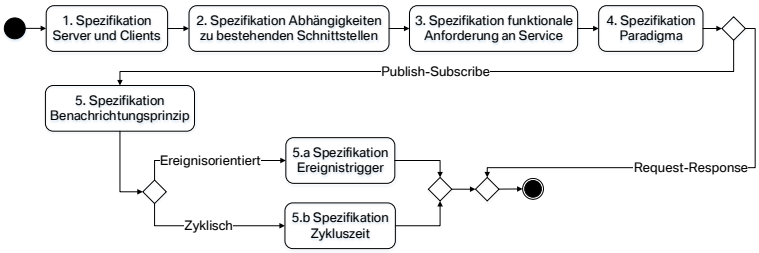

Bild 6.7: Ablauf bei der Spezifikation einer Client-Server Schnittstelle

Zur eigentlichen Entwicklung einer Client-Server Schnittstelle sind dann zunächst (3) *funktionale Anforderungen* an den vom Server bereitgestellten Service zu spezifizieren. Hierfür kann die dazu etablierte logische Funktionsarchitektur genutzt werden, welche funktionale Anforderungen im Rahmen neu zu entwickelnder Fahrzeugfunktionen in logische Informationsflüsse als Eingangsgröße für die Entwicklung einer Softwarearchitektur abbildet (s. Bild 3.3). Auf Basis funktionaler Anforderungen an einen Service

[3] Die zum Abschluss beider Abläufe eingeführten, beispielhaften Schnittstellen orientieren sich an einer Publikation [83], die im Rahmen dieser Arbeit erreicht wurde. Gleichermaßen sind die in Abschnitt 6.2.3 eingeführten zeitlichen Anforderungen an ein Zusammenspiel der Schnittstellen an dieser Publikation ausgerichtet.

wird das zu seiner Nutzung angewandte (4) *Paradigma* (s. Abschnitt 4.5) entwickelt und für den Fall einer Festlegung gemäß dem *Publish-Subscribe Paradigma* abschließend über ein *ereignisorientiertes* oder *zyklisches* (5) Benachrichtigungsprinzip detailliert.

In Tabelle 6.1 ist eine Beispielspezifikation für eine Client-Server Schnittstelle zwischen den Softwarekomponenten des *Umfeldmodells* und der *Trajektorienplanung* dargestellt.

Name	Schnittstelle Objektliste
SW-Komponenten	Umfeldmodell (Server), Trajektorienplanung (Client)
Abhängigkeit Bestandsschnittstellen	-
Funktionale Anforderung an Service	Bereitstellung von Objektliste zur 3-D Umfelderkennung des Fahrzeugs
Kommunikationsparadigma	Publish-Subscribe
Benachrichtigungsprinzip	Zyklisch
Zykluszeit	10 ms

Tabelle 6.1: Client-Server Schnittstelle zwischen den Softwarekomponenten Umfeldmodell und Trajektorienplanung

Spezifikation einer Sender-Receiver Schnittstelle Bei der Spezifikation einer Sender-Receiver Schnittstelle (s. Bild 6.8) werden wie für die Client-Server Schnittstelle zunächst die (1) involvierten Softwarekomponenten, verteilt auf die Rollen des *Senders* und *Receivers*, erfasst und mögliche (2) Abhängigkeiten zu bestehenden Schnittstellen festgehalten.

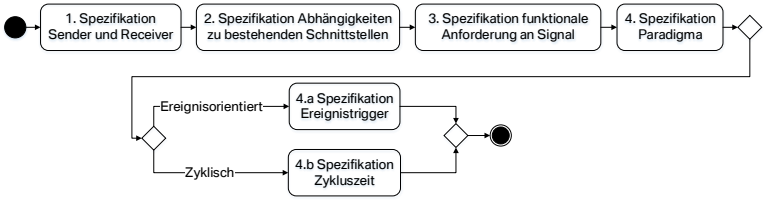

Bild 6.8: Ablauf bei der Spezifikation einer Sender-Receiver Schnittstelle

Anschließend ist die (3) *funktionale Anforderung* an das bereitgestellte Signal aus dem heute vorliegenden Zusammenhang zwischen Fahrzeugfunktionen, logischer Funktionsarchitektur und Softwarearchitektur (s. Bild 3.3) und das zum Versenden angewandte (4) *Paradigma* abgeleitet. In Tabelle 6.2 ist eine Beispielspezifikation für eine Sender-Receiver Schnittstelle zwischen den Softwarekomponenten *Trajektorienplanung* und *Fahrdynamik* dargestellt.

Name	Schnittstelle Trajektorie
SW-Komponenten	Trajektorienplanung (Sender), Fahrdynamik (Receiver)
Abhängigkeit Bestandsschnittstellen	Schnittstelle Objektliste
Funktionale Anforderung an Signal	Bereitstellung von Trajektorie zur Berechnung von Soll-Werten für Längs- und Querdynamik
Kommunikationsparadigma	Ereignisorientiert
Ereignistrigger	Erhalt von Mindestanzahl an Objektlisten zur Umgebungsauflösung

Tabelle 6.2: Sender-Receiver Schnittstelle zwischen den Softwarekomponenten Trajektorienplanung und Fahrdynamik

Einordnung Für die bereits in Prozessschritt 1 entwickelten Festlegungen ist das Potential gegeben Informationsflüsse zwischen Softwarekomponenten über zwei unterschiedliche Schnittstellenmodelle im Rahmen eines Modells einer Softwarearchitektur abzuleiten. Der heutige Entwurf einer Signal-orientierten Architektur über den Sender-Receiver Ansatz (s. Abschnitt 4.1) ist hierzu ergänzt um das Modell der Client-Server Schnittstelle.

Wichtig für die Abläufe zur Spezifikation der Schnittstellen ist die Möglichkeit beide unabhängig anwenden zu können. Besonders für die Entwicklung neuer Client-Server Schnittstellen im entsprechenden Service-orientierten Teilsystem wird hierdurch eine Möglichkeit zur Integration von Software losgelöst vom heutigen Vorgehen über Signale geschaffen.

Prozessschritt 2: Spezifikation von Rechenressourcen

Auf Basis des in Prozessschritt 1 entwickelten Modells einer Softwarearchitektur wird in Prozessschritt 2 die Partitionierung der entsprechenden Softwarekomponenten vorbereitet. Für das Signal-orientierte Teilsystem, das nicht auf Plattformsteuergeräten partitioniert werden soll, kann hierbei die bestehende Partitionierung aus einer heutigen E/E-Architektur auf domänen- oder funktionsspezifischen Steuergeräten herangezogen werden.

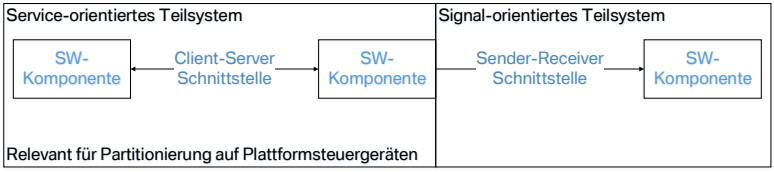

Bild 6.9: Betrachtungsumfang für die Spezifikation von Rechenressourcen

Für das Service-orientierte Teilsystem wird hingegen für jede Softwarekomponente zunächst eine Spezifikation von notwendigen Eigenschaften, die von Rechenressourcen auf einem Plattformsteuergerät bereitgestellt werden müssen, erreicht (s. Bild 6.9). Auf Basis dieser Ressourcenspezifikationen wird dann in den Prozessschritten 3-6 systematisch eine Partitionierung der Softwarekomponenten auf gegebenen Plattformsteuergeräten abgeleitet.

Strukturierung der Ressourcenspezifikation Für die Ressourcenspezifikation einer Softwarekomponente wird eine Unterteilung in angeforderte *Rechenkapazitäten* und die angeforderte *Qualifizierung* einer Rechenressource im Hinblick auf die funktionale Sicherheit vorgenommen.

Angeforderte Rechenkapazitäten Die angeforderten Rechenkapazitäten für eine Softwarekomponente, die auf einem Plattformsteuergerät partitioniert werden soll, sind in Tabelle 6.3 gezeigt.

Angeforderte Rechenkapazität	**Einheit**
Arbeitsspeicher (engl. RAM)	Kibibyte
Festwertspeicher (engl. Flash Memory)	Kibibyte
Prozessorverfügbarkeit	%

Tabelle 6.3: Angeforderte Rechenkapazitäten

Die Rechenkapazitäten beschreiben Anforderungen an den Arbeitsspeicher (engl. RAM) in Kibibyte (KiB), den Festwertspeicher (engl. Flash Memory) in KiB und die Prozessorverfügbarkeit in Prozent. Speziell die Prozessorverfügbarkeit bildet die Vorbedingung, um die von einer Softwarekomponente umfasste Funktionalität als ausführbare Task nach dem Architekturentwurf zu implementieren (s. Abschnitt 3.4.2). Die numerische Angabe der angeforderten Prozessorverfügbarkeit kann dabei nur erreicht werden, wenn ein Referenzmodell einer Recheneinheit, welches eine festgelegte Taktfrequenz und eine festgelegte Prozessorarchitektur besitzt, verfügbar ist.

In heutigen Steuergeräten in der Automobilindustrie ist die dominante Prozessorarchitektur basierend auf dem Reduced Instruction Set Computer (RISC) Konzept (s. Patterson und Sequin [Seite 443 ff.][92]).

Angeforderte Qualifizierung von Rechenressourcen Neben Rechenkapazitäten werden Anforderungen an die Qualifizierung von Rechenressourcen gestellt. Die wesentliche Anforderung drückt sich hierbei über die Spezifikation der Automotive Safety Integrity Level (ASIL) Bewertung und damit der Kritikalität einer Softwarekomponente in Bezug auf die funktionale Sicherheit aus (s. Bild 6.10).

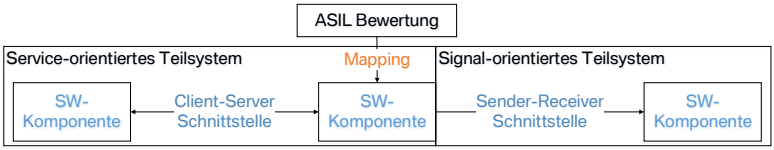

Bild 6.10: ASIL Bewertung einer Softwarekomponente

Für diese Spezifikation muss der Entwurf einer Software- und Systemarchitektur mit einem Entwicklungsvorgehen gemäß der ISO 26262 [59] synchronisiert werden. Als Synchronisationspunkt dient hierbei die Festlegung des funktionalen Sicherheitskonzepts.

Modell eines funktionalen Sicherheitskonzepts Ein funktionales Sicherheitskonzept für eine Fahrzeugfunktion ist über logische Funktionen modelliert. Jede logische Funktion zeigt dabei die Kritikalität der Fahrzeugfunktion über eine ASIL Bewertung, die einen Wert von A-D ausweisen kann, an (s. Bild 6.11).

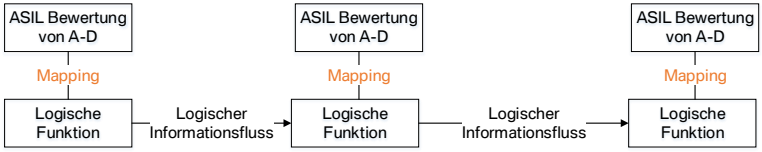

Bild 6.11: Logische Funktionsarchitektur mit zugeordneter ASIL Bewertung

Als Eingangsgrößen für die ASIL Bewertung der Fahrzeugfunktion werden Gefährdungen, die durch den Funktionsausfall auftreten können, über drei Kenngrößen betrachtet:

1. Die Schadensschwere (engl. Severity) einer Gefährdung, 2. Die Auftrittswahrscheinlichkeit (engl. Exposure) einer Gefährdung und 3. Die Kontrol-

lierbarkeit (engl. Controllability) des Gesamtsystems durch den Fahrer, um nach dem Eintreten einer Gefährdung einen Schaden abzuwenden. Aus den drei Kenngrößen wird schlussendlich das Risiko über die Abstufungen A-D festgelegt (s. Bild 6.12).

S0	S1	S2	S3
No injuries	Light, moderate injuries	Severe, life-threatining injuries	Life-threatining, fatal injuries
E0	E1	E2	E3
Very low probality	Low probality	Medium probability	High probability
C0	C1	C2	C3
Controllable in general	Simply Controllable	Normally Controllable	Difficult, uncontrollable

Severity Class	Exposure Class	Controllability Class C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	ASIL A
	E4	QM	ASIL A	ASIL B
S1	E1	QM	QM	QM
	E2	QM	QM	ASIL A
	E3	QM	ASIL A	ASIL B
	E4	ASIL A	ASIL B	ASIL C
S2	E1	QM	QM	ASIL A
	E2	QM	ASIL A	ASIL B
	E3	ASIL A	ASIL B	ASIL C
	E4	ASIL B	ASIL C	ASIL D

Bild 6.12: Ableitung von ASIL Einstufung aus Severity, Exposure und Controllability Classes (in Anlehnung an Messnarz et al. [Seite 327][80])

Zusätzlich kann ein Sonderfall gemäß der Bewertung QM vorliegen. Dies bedeutet, dass es sich bei der Fahrzeugfunktion um keine sicherheitskritische Funktion handelt und die weitergehende Entwicklung nicht mehr strikt an ein Vorgehen gemäß der ISO 26262 gebunden ist.

ASIL Bewertung und Softwarearchitektur Um von der Bewertung logischer Funktionen über ASIL Einstufungen auf die entsprechende Bewertung von Softwarekomponenten zu gelangen, werden im Folgenden zwei mögliche Szenarien eingeführt.

Szenario 1 Im Falle von Szenario 1 wird eine 1:1 Beziehung zwischen einer logischen Funktion und einer Softwarekomponente festgelegt. Hierdurch wird die Aussage getroffen, dass genau eine Softwarekomponente die Realisierung der logischen Funktion ausarbeitet.

Bild 6.13: Szenario 1 für ASIL Übertragung

Für die zugeordnete ASIL Bewertung findet in diesem Zusammenhang eine direkte Übertragung auf die Softwarekomponente statt.

Szenario 2 Im Falle von Szenario 2 wird eine 1:2 Beziehung zwischen einer logischen Funktion und zwei Softwarekomponenten erreicht. Hierdurch wird das Vorhandensein von zwei Softwarekomponenten, welche dieselbe logische Funktion realisieren, modelliert.

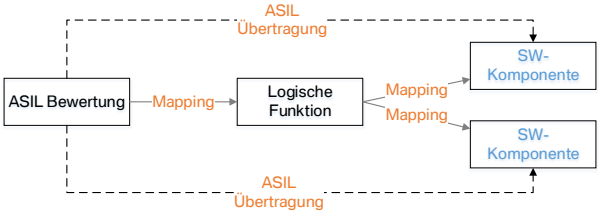

Bild 6.14: Szenario 2 für ASIL Übertragung

Sofern deren Unabhängigkeit[4] gegeben ist, wird die ASIL Bewertung der logischen Funktion über eine Arithmetik zur ASIL Dekomposition auf beide Softwarekomponenten übertragen. Die möglichen Übertragungen durch die Arithmetik sind wie folgt gegeben:

[4] Die Unabhängigkeit von zwei Softwarekomponenten, welche dieselbe logische Funktion realisieren, erfordert die diversitäre Implementierung des von den Softwarekomponenten gekapselten Programmcodes. Zudem muss die anschließende Partitionierung auf unterschiedlichen Steuergeräten mit jeweils unabhängigen Energieversorgungen erreicht werden.

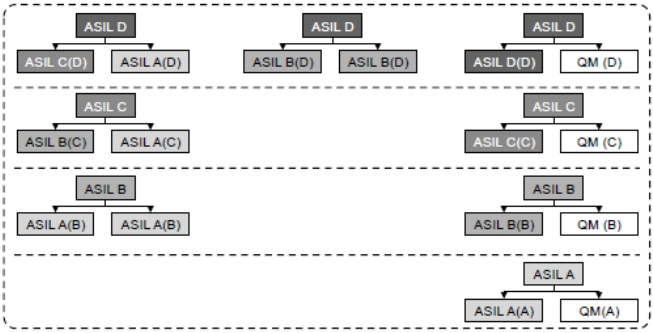

Bild 6.15: Klassifikationsschema von ASILs für die Dekomposition von Sicherheitsanforderungen nach ISO 26262 gemäß Hillenbrand [Seite 176][58]

Zusammenfassung der Prozessschritte 1-2

Als Ziel des Prozessschritts 1 steht das Modell einer hybriden Softwarearchitektur mit dem Fokus auf der Spezifikation von Schnittstellen. Durch die eingeführten Abläufe wird der heutige Sender-Receiver Ansatz (s. Bild 6.8) um eine Alternative im Rahmen des Client-Server Ansatzes (s. Bild 6.7) ergänzt.

Fortführend und als notwendige Eingangsgröße für die Partitionierung von Softwarekomponenten ist in *Prozessschritt 2* die Festlegung entsprechender Anforderungen erreicht. Neben Rechenkapazitäten spielt die ASIL Einstufung eine zentrale Rolle, weshalb die Synchronisation zwischen dem Entwurf einer hybriden Software- und Systemarchitektur mit einem Entwicklungsvorgehen gemäß der ISO 26262 eingefordert wird.

6.2.2 Architektursicht 2: Partitionierung

Durch den vorangegangenen Prozessschritt 2 sind Softwarekomponenten mit einer Spezifikation von notwendigen Rechenressourcen versehen (s. Abschnitt 6.2.1). Für das Signal-orientierte Teilsystem ist durch die Übernahme von bestehenden Steuergeräten einer heutigen E/E-Architektur bereits eine Festlegung im Hinblick auf die Partitionierung erreicht. Im Rahmen der *Ableitung von Architektursicht 2* soll nun in den Prozessschritten 3-6 eine entsprechende Partitionierung für das Service-orientierte Teilsystem im Rahmen einer Design Space Exploration festgelegt werden.

Prozessschritt 3: Festlegung Design Space für Partitionierung

Zur Festlegung des Design Space für die Partitionierung werden Softwarekomponenten des Service-orientierten Teilsystems und Recheneinheiten, die sich auf unterschiedliche Plattformsteuergeräte verteilen können, betrachtet. Die Softwarekomponenten sind dabei durch die nichtleere Menge S von Softwarekomponenten $\{s_i\}_{i=1}^{|S|}$ und die Recheneinheiten durch die nichtleere Menge R von Recheneinheiten $\{r_j\}_{j=1}^{|R|}$ festgelegt. Die möglichen Abbildungen von Softwarekomponenten auf Recheneinheiten sind über Entscheidungsvariablen $a_{i,j}$ modelliert[5].

$$A = \begin{pmatrix} a_{1,1} & \dots & a_{1,|R|} \\ \vdots & \vdots & \vdots \\ a_{|S|,1} & \dots & a_{|S|,|R|} \end{pmatrix} \tag{6.1}$$

Jeder Entscheidungsvariable kann im Weiteren ein Element aus der Binärmenge $B = \{0, 1\}$ zugeordnet werden. Diese Zuordnung dient zur Anzeige, ob eine Softwarekomponente s_i auf einer Recheneinheit r_j partitioniert wird.

[5] Vergleichbare Ansätze für die mathematische Notation einer Design Space Exploration über Entscheidungsvariablen können bei Martins und Lambe [Seite 3 f.][77] und Kugele [Seite 139 ff.][69] gefunden werden.

$$a_{i,j} = 1, s_i \textit{ partitioniert auf } r_j \tag{6.2}$$

$$a_{i,j} = 0, s_i \textit{ nicht partitioniert auf } r_j \tag{6.3}$$

Durch die Betrachtung der Fallunterscheidung für alle Softwarekomponenten kann eine Recheneinheit r_j entweder vollständig ungenutzt oder durch mindestens eine Partitionierung einer Softwarekomponente s_i belegt sein. Zur Anzeige beider Fälle wird jeder Recheneinheit r_j ein Element $\xi_j\,(0 < j <= |R|)$ zugeordnet.

$$\xi_j = 0, \textit{wenn} \sum_{i=1}^{|S|} a_{i,j} = 0 \tag{6.4}$$

$$\xi_j = 1, \textit{wenn} \sum_{i=1}^{|S|} a_{i,j} > 0 \tag{6.5}$$

Prozessschritt 4: Einschränkungen bei der Design Space Exploration

In Prozessschritt 3 wurde ein theoretischer Design Space für die Partitionierung von Softwarekomponenten auf einer Menge von Recheneinheiten festgelegt. Im Folgenden sollen nun Einschränkungen berücksichtigt werden, die für eine praktische Umsetzung notwendig sind.

Grundlegende Einschränkung für Exploration Jede Softwarekomponente soll auf genau einer Recheneinheit partitioniert werden. Hieraus ergibt sich die Notwendigkeit, dass die Summe aller Entscheidungsvariablen für die Partitionierung einer Softwarekomponente s_i den Wert 1 ergeben muss.

$$\forall i, 0 < i \leq |S| : \sum_{j=1}^{|R|} a_{i,j} = 1 \tag{6.6}$$

Einschränkungen basierend auf Ressourcenanforderungen Im Weiteren werden Einschränkungen auf Basis der Ressourcenanforderungen von Softwarekomponenten und dem Ressourcenangebot von Recheneinheiten berücksichtigt. Die Eigenschaften, die von einer Softwarekomponente angefordert bzw. von einer Recheneinheit angeboten werden, sind in Tabelle 6.4 zusammengefasst[6].

Eigenschaften	**Einheit**
Qualifizierung hinsichtlich funktionaler Sicherheit	ASIL
Prozessorverfügbarkeit	%
Arbeitsspeicherkapazität (engl. RAM)	Kibibyte
Festwertspeicherkapazität (engl. Flash Memory)	Kibibyte

Tabelle 6.4: Eigenschaften zur Beschreibung von Ressourcenanforderungen und Ressourcenangebot

Für alle betrachteten Eigenschaften werden Einschränkungen formalisiert. Diese stellen sicher, dass durch die Design Space Exploration nur Partitionierungsalternativen errechnet werden, bei denen keine Recheneinheit über ihr maximales Ressourcenangebot in Form einer maximalen Prozessorverfügbarkeit, maximaler Arbeitsspeicher- und Festwertspeicherkapazitäten sowie einer maximalen ASIL Qualifizierung belastet wird.

Für die Formalisierung dieser Einschränkungen über arithmetische Ausdrücke können die folgenden Schritte 1-3 durchlaufen werden. Als Beispiel wird die ASIL Qualifizierung, die eine Softwarekomponente als Anforde-

[6] Bei der Festlegung des Arbeitsspeichers ist die Unterscheidung in einen statischen Arbeitsspeicher (Static Random Access Memory (SRAM)) und dynamischen Arbeitsspeicher (Dynamic Random Access Memory (DRAM)) möglich. Speziell für den Festwertspeicher wird jedoch eine Einschränkung auf die Flash-Speichertechnologie, welche sich für die Aktualisierung von Software durch die Eigenschaft der Wiederbeschreibbarkeit im Vergleich zu anderen Technologien eignet, vorgenommen (s. Tietze und Schenk [Seite 751 f.][108] und Brandt [Seite 81][27]).

rung bei der Partitionierung auf Recheneinheiten und damit als Einschränkung im Rahmen der Design Space Exploration stellt, betrachtet.

Schritt 1 Zunächst wird jeder Softwarekomponente $s_i \in S$ und jeder Recheneinheit $r_j \in R$ eine in Tabelle 6.4 eingeführte Eigenschaft als Anforderung bzw. Angebot zugeordnet. Für das Beispiel der ASIL Bewertung sind dies Elemente aus der Menge $ASIL = \{QM, A, B, C, D\}$ (s. Bild 6.12), deren Zuordnung auf Softwarekomponenten und Recheneinheiten über die Abbildung γ: $S \cup R \rightarrow ASIL$ erreicht wird.

Schritt 2 Als Vorbedingung für die Entwicklung eines arithmetischen Ausdrucks wird in einem nächsten Schritt jedem Element der Menge *ASIL* ein Wert aus der Menge $\mathbb{N}_0$ zugeordnet. Die Zuordnung ist über die Abbildung $\delta : ASIL \rightarrow \mathbb{N}_0$ definiert:

$$\delta(QM) = 0, \delta(A) = 1, \delta(B) = 2, \delta(C) = 3, \delta(D) = 4 \tag{6.7}$$

Schritt 3 Für die eigentliche Einschränkung soll nun gelten, dass bei der Design Space Exploration die ASIL Qualifizierung einer beliebigen Recheneinheit r_j die ASIL Anforderung einer Softwarekomponente s_i immer erfüllt oder übererfüllt. Eine entsprechende Formalisierung ist über die folgende Ungleichung erreicht.

$$\forall i, 0 < i \leq |S|, \forall j, 0 < j \leq |R| : a_{i,j}(\delta(\gamma(r_j)) - \delta(\gamma(s_i))) \geq 0 \tag{6.8}$$

Prozessschritt 5: Festlegung von Optimierungskriterien

Die Design Space Exploration wird unter Anwendung von Optimierungskriterien durchgeführt. Im Rahmen dieser Arbeit werden zwei Optimierungskriterien betrachtet, die eine kosteneffiziente Partitionierung der Softwarearchitektur ermöglichen sollen.

1. Optimierungskriterium Die erste Optimierung minimiert die Fälle bei denen eine Partitionierung von Softwarekomponenten auf einer Recheneinheit mit einer ASIL Qualifizierung erreicht wird, die höher ist als angefordert.

Hierzu wird für jede Recheneinheit die Abweichung zwischen den angeforderten ASIL Qualifizierungen von Softwarekomponenten und der bereitgestellten ASIL Qualifizierung einer Recheneinheit bewertet. Anschließend werden alle Abweichungen für eine betrachtete Recheneinheit r_j aufsummiert und einem Element aus der Menge $L = \{l_j\}_{j=1}^{|R|}$ zugeordnet.

$$\forall j, 0 < j \leq |R| : l_j = \sum_{i=1}^{|S|} a_{i,j}(\delta(\gamma(r_j)) - \delta(\gamma(s_i))) \tag{6.9}$$

Je höher dieser Wert l_j ist, desto stärker ist eine Übererfüllung der ASIL Anforderungen von Softwarekomponenten durch die entsprechende Recheneinheit r_j gegeben. Für die Minimierung des Wertes über alle Recheneinheiten und damit der Summe $\sum_{j=1}^{|R|} l_j$ kann im besten Fall eine Bewertung von 0 erzielt werden.

$$\sum_{j=1}^{|R|} l_j = 0 \tag{6.10}$$

Dies bedeutet, dass keine Übererfüllungen von ASIL Anforderungen existieren.

2. Optimierungskriterium Die zweite Optimierung umfasst die Anzahl von benötigten Recheneinheiten r_j. Bei dieser Minimierung kann im besten Fall für die Summe $\sum_{j=1}^{|R|} \xi_j$ ein Wert von 1 erzielt werden.

$$\sum_{j=1}^{|R|} \xi_j = 1 \tag{6.11}$$

Dieser Extremfall bedeutet, dass alle Softwarekomponenten auf derselben Recheneinheit partitioniert sind.

Prozessschritt 6: Festlegung von Plattformpartitionen

Die Design Space Exploration unter Miteinbezug der Optimierungskriterien 1 und 2 wird über eine Pareto-Optimierung umgesetzt. Als Ergebnis der Optimierung wird eine Pareto-Front generiert. Jede Lösung auf der Pareto-Front stellt eine Konfiguration dar, die allen Softwarekomponenten des Service-orientierten Teilsystems eine Abbildung auf eine Recheneinheit eines Plattformsteuergeräts zuordnet (s. Bild 6.16).

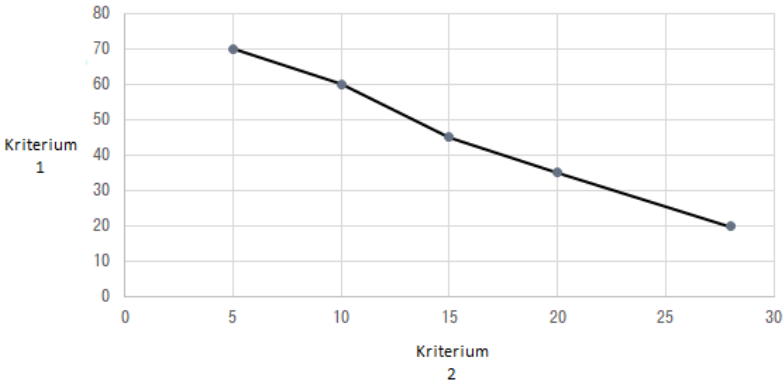

Bild 6.16: Beispielhafte Pareto-Front mit fünf Lösungen

Das Optimierungskriterium 1 ist auf der y-Achse aufgetragen und beschreibt die Summe der Abweichungen zwischen angefordertem ASIL einer Softwarekomponente und bereitgestelltem ASIL der entsprechenden Recheneinheit als Partitionierungsziel.

Das Optimierungskriterium 2 ist auf der x-Achse aufgetragen und beschreibt die Anzahl an verwendeten Recheneinheiten für eine spezifische Lösung.

Zusammenfassung der Prozessschritte 3-6

Die *Prozessschritte 3-6* betrachten die optimierte Allokation von Ressourcen eines Plattformsteuergeräts. Dabei werden in den *Prozessschritten 3-4* zunächst unterschiedliche Anforderungen von Softwarekomponenten im Hinblick auf die Prozessorverfügbarkeit, den Speicher sowie die notwendige ASIL Bewertung einer Hardwarekomponente formalisiert.

Anschließend sind in *Prozessschritt 5* Optimierungsziele gesetzt. Hierbei spielen die Minimierung der Komponentenanzahl und die Minimierung von hochqualifizierten Komponenten im Hinblick auf die ASIL Bewertung die zentrale Rolle.

Beide Ziele dienen der Abbildung von Kosten und sind Eingangsgröße für eine Pareto-Optimierung, die abschließend in *Prozessschritt 6* Alternativen für die Partitionierung einer Softwarearchitektur und somit Vorschläge für eine optimierte Ressourcenallokation ermittelt.

6.2.3 Architektursicht 3: Laufzeitverhalten

Über die Prozessschritte 1-6 wurde die Modellierung einer hybriden Softwarearchitektur und die Ableitung von Partitionierungen für die entsprechenden Softwarekomponenten erreicht. Für das Service-orientierte Teilsystem stehen zudem Alternativen wie diese Partitionierung von Softwarekomponenten umgesetzt werden kann, weil sie keine Vorfestlegung auf Steuergeräten besitzen wie die Softwarekomponenten des Signal-orientierten Teilsystems.

Um nun für ermittelte Alternativen die Interaktion der Softwarekomponenten in einem Kommunikationsnetzwerk zu beschreiben, werden entsprechende Laufzeitmodelle in den Prozessschritten 7-8 entwickelt und mit Anforderungen an ihre Bewertung ergänzt.

Auf Basis einer umgesetzten Bewertung soll abschließend in Prozessschritt 9 entschieden werden, ob eine Architekturalternative als Entwurf für eine nachfolgende Implementierung und Absicherung freigegeben werden kann.

Prozessschritt 7: Laufzeitmodellierung

Für die Modellierung des Laufzeitverhaltens wird zunächst eine Menge von Softwarekomponenten, die in einen gemeinsamen Wirkzusammenhang eingebettet sind, festgelegt. In Bild 6.17 sind die eingeführten Softwarekomponenten zur Realisierung eines Teilausschnitts einer automatisierten Fahrfunktion als Wirkzusammenhang begriffen (s. Abschnitt 6.2.1).

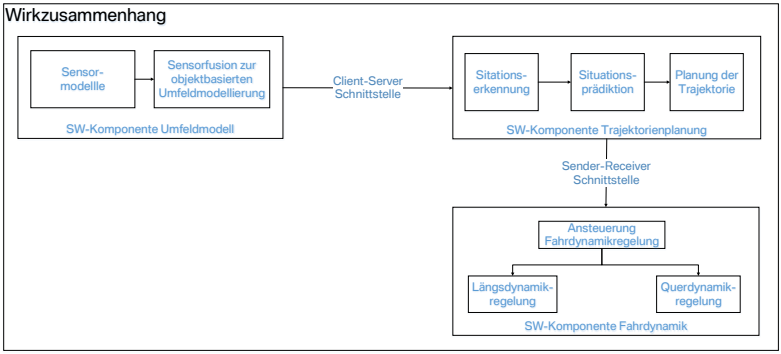

Bild 6.17: Wirkzusammenhang im Rahmen einer hybriden Softwarearchitektur

Schnittstellenspezifikationen Um die wechselseitigen Informationsflüsse zwischen den Softwarekomponenten zu beschreiben, müssen zeitliche Eigenschaften berücksichtigt werden. Diese sind in den Schnittstellenspezifikationen der entsprechenden Softwarekomponenten als Ergebnis von Prozessschritt 1 beschrieben[7].

[7] Für den Wirkzusammenhang aus Bild 6.17 können mögliche Schnittstellenspezifikationen in Tabelle 6.1 und 6.2 gefunden werden.

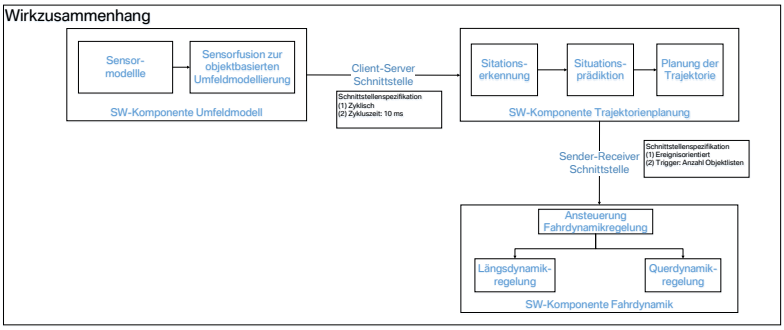

Bild 6.18: Einbindung von Schnittstellenspezifikationen in Wirkzusammenhang

Softwarepartitionen Für die Modellierung des Laufzeitverhaltens in einem Kommunikationsnetzwerk werden die relevanten Netzwerkteilnehmer (Steuergeräte) ergänzt. Hierzu wird jede Softwarekomponente mit einem möglichen Partitionierungsziel als Ergebnis der vorherigen Prozessschritte in Beziehung gesetzt (s. Bild 6.19).

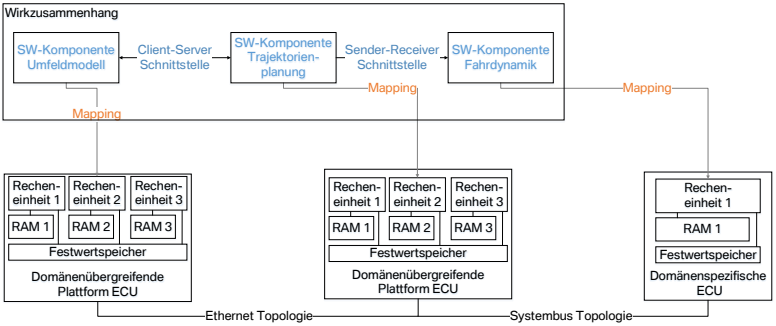

Bild 6.19: Einbindung von Partitionierungen in Wirkzusammenhang

Laufzeitmodell Auf Basis des um zeitliche Eigenschaften und Partitionierungsbeziehungen ergänzten Wirkzusammenhangs wird das *Laufzeitmodell* entwickelt. Das Modell beschreibt die zur Kommunikation zwi-

schen Softwarekomponenten notwendigen Netzwerknachrichten. Hierdurch wird die Ausführung einer Funktion beschrieben, die sich auf ein Service-orientiertes Teilsystem und ein Signal-orientiertes Teilsystem verteilt.

Modellierung Als Modellierungstechnik kommen Sequenzdiagramme zum Einsatz. Für jeden Wirkzusammenhang können bei einer ausführlichen Betrachtung und Herleitung die Sequenzdiagramme A-C modelliert werden.

Das Sequenzdiagramm A umfasst dabei eine Initiierungsphase, in der alle Softwarekomponenten (Server) zunächst ihre angebotenen Services veröffentlichen und von Softwarekomponenten (Clients) einen potentiellen Nutzungswunsch vermittelt bekommen. Dieses Konzept wurde über den Begriff der *Service Discovery* eingeführt (s. Abschnitt 2.4).

Im Sequenzdiagramm B wird dann auf Basis einer durchgeführten Service Discovery die *Service Nutzung* im Rahmen des Service-orientierten Teilsystems umfasst, bevor abschließend im Rahmen des Sequenzdiagramms C der Miteinbezug einer noch aufzulösenden Abhängigkeit zu einem Signal-orientierten Teilsystem und somit die *Ausführung einer Funktion*, die sich auf beide Teilsysteme verteilt, dargestellt wird.

Über einen Zustandsautomaten kann der Zusammenhang abstrahiert zusammengefasst werden (s. Bild 6.20). Die Bezeichnung *Teilsystem 1* beschreibt dabei das Service-orientierte Teilsystem und die Bezeichnung *Teilsystem 2* das Signal-orientierte Teilsystem[8].

[8] Die im Automaten beschriebene *Verfügbarkeit des Netzwerks* ist gegeben, wenn alle relevanten Steuergeräte eines Wirkzusammenhangs gestartet wurden.

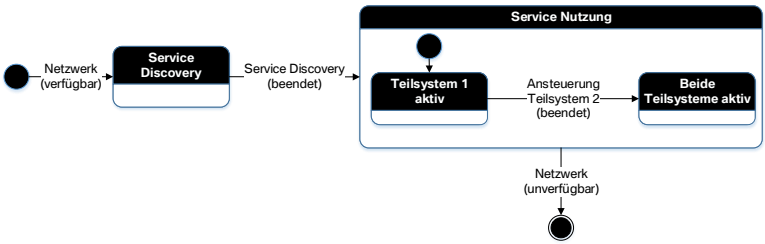

Bild 6.20: Darstellung von Zuständen für die Nutzung eines Services durch eine auf ein Service-orientiertes und Signal-orientiertes Teilsystem verteilte Funktion

Sequenzdiagramm A Die im Rahmen von Sequenzdiagramm A abzubildende Service Discovery wird durch einen Server initiiert. Bild 6.21 stellt dies für den eingeführten Wirkzusammenhang und den darin abgebildeten Softwarekomponenten des *Umfeldmodells* (Server) und der *Trajektorienplanung* (Client) dar.

Die für die Partitionierung der Softwarekomponenten relevanten Steuergeräte sind im gezeigten Szenario als ECU 1 (*Umfeldmodell*) und ECU 2 (*Trajektorienplanung*) bezeichnet. Dabei wird zunächst ECU 1 betrachtet, welche auf Basis einer versendeten (1) Publish Nachricht den Service des *Umfeldmodells* veröffentlicht. Für die Weiterleitung dieser Nachricht an die *Trajektorienplanung* ist anschließend ein Ethernet Switch als Übertragungskomponente genutzt, bevor der entsprechende Antwortpfad eingeleitet wird: Hierbei versendet zunächst ECU 2 eine (3) Subscribe Nachricht an den Ethernet Switch. Abschließend wird ihre Weiterleitung an ECU 1 umgesetzt, wodurch die Service Discovery ihren Abschluss findet.

Sequenzdiagramm B Das Sequenzdiagramm B beschreibt die Nutzung von Services. Als Beispiel hierfür soll die Nutzung des vom *Umfeldmodell* bereitgestellten Services beschrieben werden. Gemäß der Spezifikation der

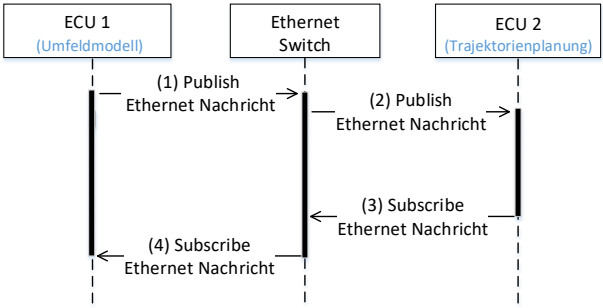

Bild 6.21: Sequenzdiagramm A im Rahmen der Service Discovery

entsprechenden Schnittstelle ist sein Zweck die zyklische Bereitstellung einer Objektliste zur 3-D Umfelderkennung (s. Tabelle 6.1). Um das Verhalten durch ein Sequenzdiagramm abzubilden werden alle relevanten Netzwerkkomponenten in einer *loop* zusammengefasst.

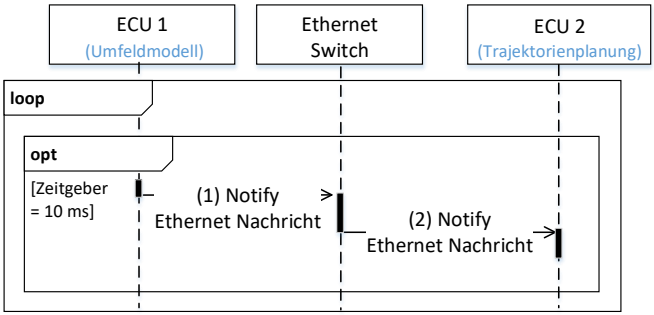

Bild 6.22: Sequenzdiagramm B im Rahmen der Service Nutzung

Das Verhalten des *Services* ist dann über entsprechende Nachrichten wie folgt beschrieben (s. Bild 6.22): Innerhalb der loop wird zunächst geprüft, ob der Eintritt eines Ereignisses, das sich in festen zeitlichen Abständen wiederholt, gegeben ist. Im Beispiel ist dieses Ereignis die Belegung ei-

nes Zeitgebers mit dem Wert 10 ms (s. Tabelle 6.1). Anschließend wird der Service über eine Benachrichtigung des Clients erbracht. Die dafür relevanten Netzwerknachrichten umfassen zunächst den Versand einer (1) Notifiy Nachricht von ECU 1 an den Ethernet Switch. Anschließend findet die Weiterleitung an ECU 2 statt, wodurch die *Trajektorienplanung* als Client über eine aktualisierte Objektliste benachrichtigt wurde.

Sequenzdiagramm C Die Softwarekomponenten *Umfeldmodell* und *Trajektorienplanung* sind Teil eines Wirkzusammenhangs, der eine Abhängigkeit zu einem Signal-orientierten Teilsystem aufweist (s. Bild 6.18). Die Abhängigkeit ist dabei als Sender-Receiver Schnittstelle zwischen der *Trajektorienplanung* und der *Fahrdynamik* modelliert und weist gemäß ihrer Spezifikation ein ereignisorientiertes Verhalten auf (s. Tabelle 6.2): Demnach wird die Ansteuerung der Softwarekomponente *Fahrdynamik* nur vorgenommen, sofern ein entsprechender Trigger ausgelöst ist. Die Bedingung für diese Auslösung ist wiederum abhängig vom Verhalten des *Umfeldmodell* Services.

Hierbei wird gefordert, dass die *Trajektorienplanung* zunächst eine Mindestanzahl an Aktualisierungen der Objektliste erhalten muss, um ihre Berechnung und die anschließende Stimulation der *Fahrdynamik* anzustoßen. Zur Darstellung des Zusammenhangs wird das Sequenzdiagramm B um ein Steuergerät ECU 3 (*Fahrdynamik*) zunächst ergänzt (s. Bild 6.22).

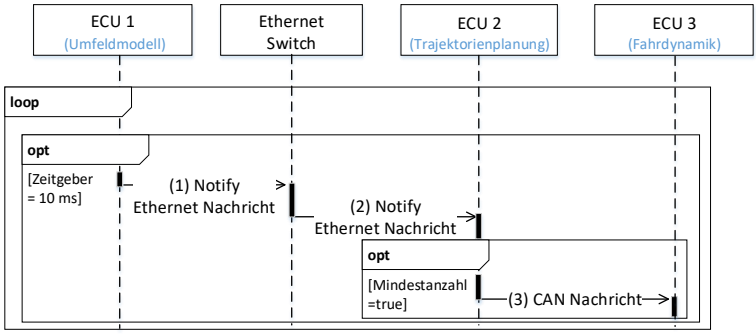

Bild 6.23: Sequenzdiagramm C im Rahmen der Auflösung von Abhängigkeiten zu Signal-orientiertem Teilsystem

Anschließend werden neuerlich alle Netzwerkteilnehmer in einer *loop* betrachtet. Dies begründet sich durch das weiter *primär zyklische* Verhalten im Rahmen der Stimulation durch den Umfeldmodell Service.

Prozessschritt 8: Laufzeitbewertung

Die Modellierung des Laufzeitverhaltens dient neben der Beschreibung vorrangig als Eingangsgröße zur Bewertung einer verteilten Funktion im Hinblick auf die Einhaltung zeitlicher Anforderungen (s. Bild 6.24).

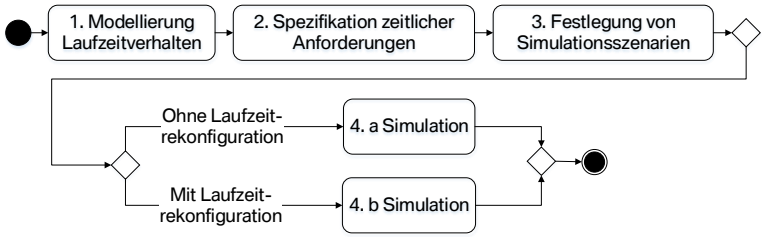

Bild 6.24: Vorgehen für Laufzeitbewertung

Für ein Bewertungsszenario werden dazu zunächst zeitliche Anforderungen spezifiziert (s. Bild 6.25).

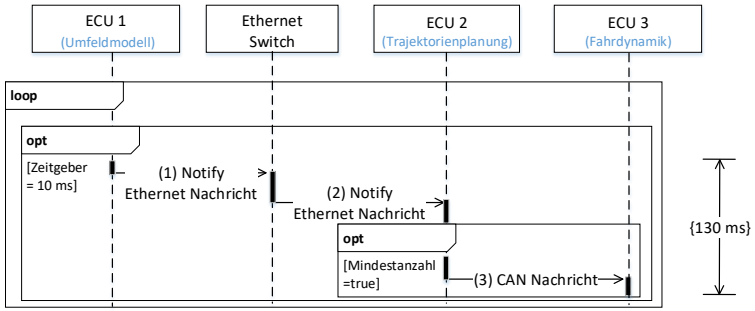

Bild 6.25: Szenario für simulative Laufzeitbewertung unter Berücksichtigung einer maximal tolerierbaren Latenzzeit für die Ansteuerung der Fahrdynamik

Beispielhaft für das Sequenzdiagramm C ist eine maximal tolerierbare Latenzzeit von 130 ms für die Ansteuerung der Softwarekomponente Fahrdynamik als Anforderung für eine Simulation festgelegt.

Berücksichtigung von Laufzeitrekonfiguration Neben dem bereits eingeführten Bewertungsszenario besteht für eine Service-orientierte Architektur bzw. das Service-orientierte Teilsystem die Möglichkeit der *Laufzeitrekonfiguration* (s. Abschnitt 4.2.3). Hierbei wird davon ausgegangen, dass nach einer bereits mindestens einmal erfolgten *Service Discovery* diese neuerlich umgesetzt wird.

Zur Modellierung der Laufzeitrekonfiguration kann das entsprechende Verhalten über einen Zustandsautomaten (s. Bild 6.26) als Alternative zu einem Sequenzdiagramm (s. Bild 4.12) vereinfacht beschrieben werden.

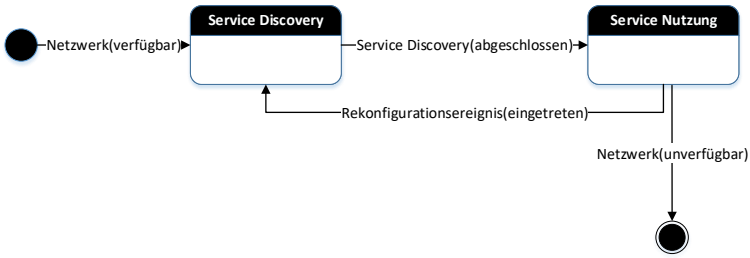

Bild 6.26: Darstellung der *Laufzeitrekonfiguration* durch einen Zustandsautomaten

Zeitliche Anforderung für Laufzeitbewertung Der eingeführte Automat zur Beschreibung der Laufzeitrekonfiguration wird weitergehend um einen zeitlichen Aspekt ergänzt. Hierdurch wird die Festlegung einer Rekonfigurationszeit als Zeitintervall Δt_1 für die Dauer der *Service Discovery* erreicht (s. Bild 6.27). Zur Absicherung eines Architekturentwurfs, der

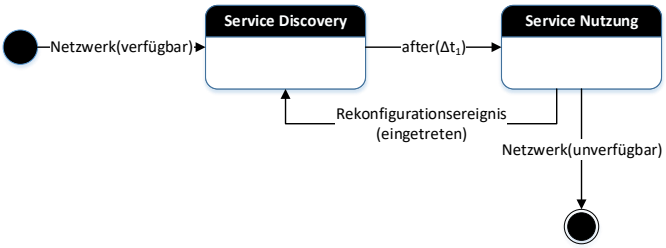

Bild 6.27: Darstellung der *Laufzeitrekonfiguration* durch einen Zustandsautomaten mit Zeitintervall Δt_1 als Rekonfigurationszeit

einen rekonfigurierbaren Service[9] trägt, muss durch ein entsprechendes Bewertungsszenario im Rahmen einer Simulation sichergestellt werden, dass dieses Zeitintervall zwischen dem Rekonfigurationsereignis und der neuerlichen Verfügbarkeit eines Service im Rahmen der Service Nutzung eine maximal tolerierbare Rekonfigurationszeit $t_{service}$ ($t_{service} >= \Delta t_1$) erfüllt.

Laufzeitrekonfiguration im Kontext einer Funktion Sofern die Rekonfiguration eines Services im Kontext einer übergeordneten Funktion betrachtet wird (s. Bild 6.25), ist eine zusätzliche Randbedingung für die Laufzeitrekonfiguration nicht nur, dass die Verfügbarkeit eines Services wieder hergestellt ist, sondern die übergeordnete Funktion wieder zur Verfügung steht. In diesem abgewandelten Szenario der Laufzeitrekonfiguration ist die Bewertung in Bezug auf die Einhaltung einer maximalen Rekonfigurationszeit $t_{function}$ zu bewerten. Im Normalfall ist diese Zeit mit der Anforderung für die maximal tolerierbare Latenzzeit der Funktion gleichgestellt.

[9] Hierbei kann beispielsweise eine zweite Instanz des eingeführten Umfeldmodell Services aus Bild 6.21 betrachtet werden.

Die für die Laufzeitbewertung betrachteten Szenarien sind in Tabelle 6.5 über eine Beschreibung zusammengefasst.

Szenario	Beschreibung
Szenario 1	Latenzzeitbewertung ohne rekonfigurierbaren Anteil: Einhaltung von maximal tolerierbarer Latenzzeit einer Funktion
Szenario 2	Latenzzeitbewertung mit rekonfigurierbaren Anteil: Einhaltung von maximal tolerierbarer Rekonfigurationszeit für Service bzw. Funktion

Tabelle 6.5: Szenarien für die Laufzeitbewertung

Zusammenfassung der Prozessschritte 7-9

Die Prozessschritte 7-9 dienen zur Beschreibung und Bewertung des Laufzeitverhaltens von verteilten Softwarekomponenten. Dabei werden in Prozessschritt 7 zunächst die unterschiedlichen Laufzeitphasen einer Funktion, die aus Service-orientierten und Signal-orientierten Anteilen aufgebaut ist, beschrieben.

In Prozessschritt 8 sind die für eine Bewertung des Laufzeitverhaltens betrachteten Szenarien festgelegt (s. Tabelle 6.5). Als Unterscheidung zum heutigen Ansatz für die Analyse des Laufzeitverhaltens im Rahmen statisch festgelegter Netzwerkpfade über Systembusse (s. Abschnitt 3.4.2) wird das Konzept der Laufzeitrekonfiguration verfolgt.

Abschließend in einem Prozessschritt 9 wird entschieden, ob nach Durchführung der Laufzeitbewertung, die dabei berücksichtigten Softwarekomponenten für einen nachfolgenden Implementierungs- und Absicherungsprozess freigegeben werden können.

6.3 Einbindung der Entwurfsmethodik in V-Modell

Die Entwurfsmethodik für eine *hybride Software- und Systemarchitektur* beschreibt die Abbildung funktionaler Anforderungen auf Softwarekomponenten und Schnittstellen sowie deren anschließende Integration auf eine Topologie von Steuergeräten über einen entsprechenden Entwurfsprozess (s. Abschnitt 6.2). Eine Betrachtung des Vorgehens für die Implementierung und Absicherung der in der Softwarearchitektur zusammengefassten Teilsysteme steht jedoch noch aus und soll über das *V-Modell* als heute etabliertes Vorgehensmodell in der Automobilindustrie erreicht werden (s. Abschnitt 3.1).

Als Vorschlag steht die Idee das Signal-orientierte und das Service-orientierte Teilsystem zunächst separat im V-Modell zu betrachten. Unterstützung erfährt dieser Ansatz durch die Tatsache, dass die Implementierung von Änderungen der Softwarearchitektur basierend auf einem Signal-orientierten Ansatz einen Zusatzaufwand durch Folgeänderungen an einem physischen Kommunikationsnetzwerk birgt (s. Bild 6.28).

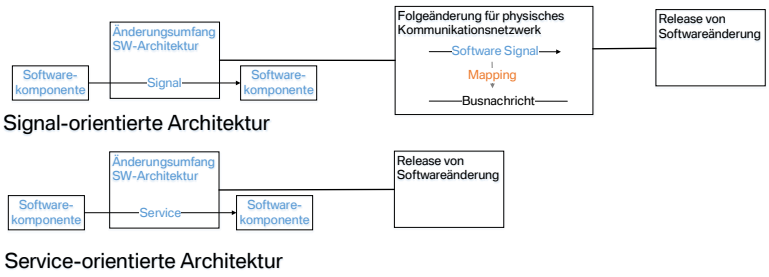

Bild 6.28: Vergleich von Änderungen der Softwarearchitektur

Speziell in den *Entwicklungsphasen* des V-Modells und der darin umgesetzten Realisierung dieser Folgeänderungen durch die Anpassung der *Basissoftware* eines Steuergeräts (s. Abschnitt 2.3) nehmen beide Teilsysteme somit nicht denselben Durchlauf.

6.3.1 Signal-orientiertes Teilsystem im V-Modell

Das V-Modell als generischer Ablauf von Phasen wird für das Signal-orientierte Teilsystem zuerst betrachtet. Die Einbindung der Entwurfsmethodik für eine hybride Software- und Systemarchitektur betrifft dabei die *Systemanforderungen*[10] und den *Systementwurf* über die heute eine Architekturfestlegung erreicht wird. Diese Phasen werden als *Architekturphasen* zusammengefasst (s. Bild 6.29).

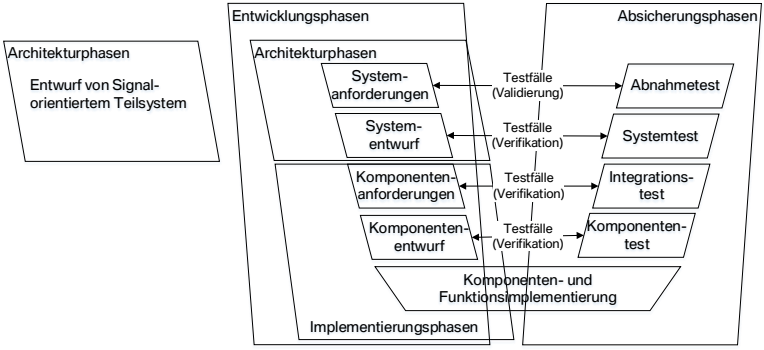

Bild 6.29: Signal-orientiertes Teilsystem im Rahmen der Architekturphasen des V-Modells

Die darauffolgenden Phasen können mit Bezug auf ihren Zusammenhang zur Architektur in *Implementierungsphasen* und *Absicherungsphasen* logisch gruppiert werden.

[10] Im Rahmen der Systemanforderungen stehen funktionale Systemanforderungen im Vordergrund.

Umsetzung der Implementierungs- und Absicherungsphasen

Innerhalb der *Implementierungsphasen* stehen *Technologien* zur Verfügung, die ausgehend von einer Änderung der Softwarearchitektur das entsprechende Kommunikationskonzept zwischen Steuergeräten über Signale und Nachrichten realisieren.

Als heute etablierte Technologien werden die durch AUTOSAR standardisierten Module der Basissoftware eines Steuergeräts verstanden. Um Änderungen einer Softwarearchitektur in Form von Signalen über die Basissoftware zu implementieren werden dabei speziell der PDU Router und das COM Modul konfiguriert (s. Abschnitt 2.3). Beide Anteile sind notwendig, um die Funktionalität einer Softwarekomponente in einem Kommunikationsnetzwerk zu realisieren die Signale eingangsseitig von einem *Systembus* nutzt oder ausgangsseitig über einen Systembus bereitstellt (s. Abschnitt 3.1.1).

Absicherungsphasen Die Absicherungsphasen des V-Modells tragen zum Änderungsvorgehen für eine Softwarearchitektur zunächst über die Betrachtung einer Softwarekomponente als abgeschlossene Einheit bei. In diesem Zusammenhang sei auf die Software-in-the-Loop Methodik (s. Abschnitt 3.1.1) als etabliertes Verfahren für deren Test verwiesen [Seite 46 f.][41]. Darauffolgend wird über statische Prüfungen im Rahmen des *Integrationstests* das Einhalten von Schnittstellenspezifikationen, der Namensraum für Variablen und die Verwendung eines einheitlichen Speicherlayouts für Softwarekomponenten auf einem Steuergerät geprüft. Hierdurch wird die interne Softwarearchitektur für ein Steuergerät abgesichert.

Die eigentliche Absicherung des Signal-orientierten Kommunikationskonzepts über einen Systembus erreicht der darauffolgende *Systemtest*. Hierbei wird die Interaktion von Softwarekomponenten über die Grenze eines

Steuergeräts hinweg durch Anwendung der Hardware-in-the-Loop (s. Abschnitt 3.1.2) Methodik verifiziert, bevor im abschließenden Abnahmetest das Fahrzeug im Kontext seiner realen Betriebsumgebung validiert wird.

Prozessuale Darstellung Aus den dem V-Modell entnommenen Phasengruppierungen und den für deren Umsetzung angewandten Technologien und Methodiken wird eine prozessuale Darstellung gewonnen (s. Bild 6.30).

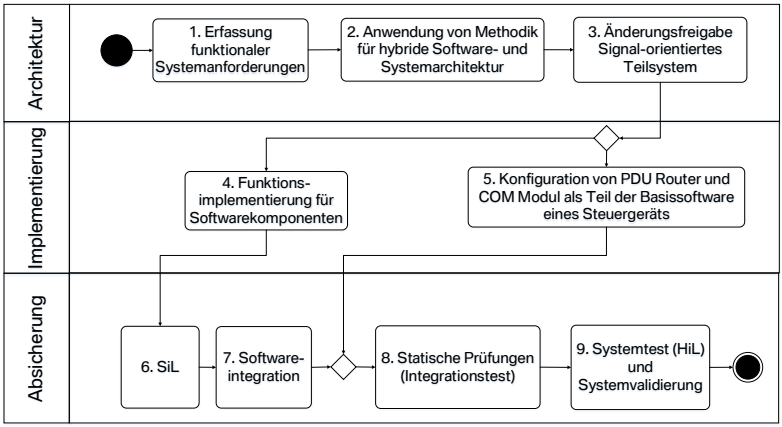

Bild 6.30: Architektur, Implementierung und Absicherung für das Signal-orientierte Teilsystem

Die *Prozessschritte 1-3* kennzeichnen dabei die Abbildung funktionaler Systemanforderungen über Software durch Anwendung der Entwurfsmethodik für eine hybride Software- und Systemarchitektur und die Freigabe daraus resultierender Änderungen des Signal-orientierten Teilsystems[11]. Die *Prozessschritte 4-5* umfassen die auf die Funktionalität von Software bezogene Implementierung dieser Änderungen und die notwendigen An-

[11] Die Prozessschritte 1-3 umfassen den vollständigen Durchlauf des Entwurfsprozess für eine hybride Software- und Systemarchitektur aus Bild 6.5 für den Fall, dass ausschließlich das Signal-orientierte Teilsystem betrachtet wird.

passungen für ein physisches Kommunikationsnetzwerk inklusive der hierdurch umfassten Steuergeräte durch Konfiguration des PDU Routers und des COM Moduls. Die abschließenden *Prozessschritte 6-9* werden durch heute etablierte Methodiken der Absicherung erreicht.

6.3.2 Service-orientiertes Teilsystem im V-Modell

Das Service-orientierte Teilsystem übernimmt dieselbe Einordnung in das V-Modell und wird somit auf die *Systemanforderungen* und den *Systementwurf* im Rahmen der *Architekturphasen* bezogen.

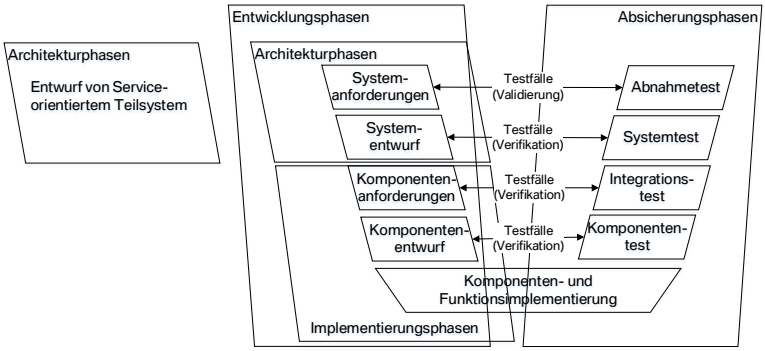

Bild 6.31: Service-orientiertes Teilsystem im Rahmen der Architekturphasen des V-Modells

Die darauffolgenden Phasen sind mit Bezug auf ihren Zusammenhang zur Architektur demnach wiederum in *Implementierungsphasen* und *Absicherungsphasen* logisch gruppiert.

Umsetzung der Implementierungs- und Absicherungsphasen

Für die Implementierung des Service-orientierten Teilsystems stehen Technologien zur Verfügung, die in Teilen durch die AUTOSAR Standardisierung berücksichtigt werden (s. Abschnitt 2.4). Im Gegensatz zu den Festlegungen für die AUTOSAR Module bei Signal-orientierten Architekturen

berührt deren Konfiguration die Kommunikation in einem physischen Netzwerk jedoch nicht.

Logische Netzwerkkonfiguration Die Netzwerkkonfiguration für das Service-orientierte Teilsystem ist entkoppelt vom physischen Netzwerk. Als Lösungselemente hierfür kommen die Protokolle UDP und TCP zum Einsatz (s. Bild 6.32).

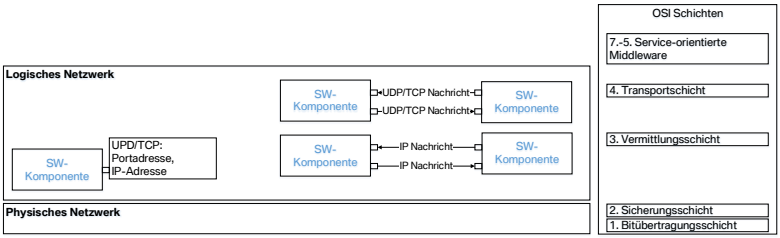

Bild 6.32: UDP und TCP als Teil der Implementierung des Service-orientierten Teilsystems

Durch Mitgliedschaft in der IP Protokollfamilie fassen die notwendigen Konfigurationsschritte für beide Protokolle die Festlegung von Port- und IP Adressen. Der resultierende Adressraum stellt somit die Implementierung eines Netzwerks auf den Schichten 4 und 3 des OSI Modells dar. Durch eine Service-orientierte Middleware auf den Schichten 7-5 wird diese Konfiguration genutzt, um die Kommunikation auf einem darunterliegenden physischen Netzwerk (Schichten 2 und 1) automatisch abzuleiten.

Absicherungsphasen Die Absicherung des Service-orientierten Teilsystems umfasst durch seine Einbindung in das V-Modell dieselben Phasen wie das Signal-orientierte Teilsystem. Die spezifisch für eine Phase angewandten Methodiken erfahren jedoch Anpassungen. Dabei werden zwei wesentliche Einflüsse, die durch Service-Orientierung in der Absicherung zum Tragen kommen, betrachtet: (1) Die Veränderung in der Schnittstellenspezifi-

kation von Softwarekomponenten durch den Client-Server Ansatz (s. Abschnitt 4.2.1) und (2) die Umstellung auf ein sich während der Laufzeit rekonfigurierbares Netzwerk (s. Abschnitt 4.2.3). Tabelle 6.6 ordnet beide Einflüsse im Hinblick auf die Phasen der Absicherung und fasst Anpassungsbedarfe.

Absicherungsphase	**Komponente**	**Integration**	**System**
Heutige Methodik	SiL	Statische Prüfungen	HiL
Einfluss	Schnittstellenspezifikation	Schnittstellenspezifikation	Laufzeitrekonfiguration
Anpassungsbedarf	Testfälle entsprechend Publish-Subscribe/Request-Response Paradigma	Prüfkriterien für Schnittstellenspezifikationen	Berücksichtigung von Rekonfigurationszenario

Tabelle 6.6: Einfluss und resultierender Anpassungsbedarf durch Service-Orientierung für Absicherungsmethodiken

Prozessuale Darstellung Gleichermaßen zum Signal-orientierten Teilsystem wird durch die aus dem V-Modell entnommenen Phasengruppierungen eine prozessuale Darstellung gewonnen (s. Bild 6.33). Die *Prozessschritte 1-3* beschreiben dabei die Abbildung funktionaler Systemanforderungen über Software durch Anwendung der Entwurfsmethodik für eine hybride Software- und Systemarchitektur und die Freigabe daraus resultierender Änderungen des Service-orientierten Teilsystems[12]. Im Rahmen des *Prozessschritts 5* ist die Konfiguration des PDU Routers und des COM Moduls ersetzt durch die Festlegung von Port- und IP Adressen.

[12] Die Prozessschritte 1-3 umfassen den vollständigen Durchlauf des Entwurfsprozess für eine hybride Software- und Systemarchitektur aus Bild 6.5 für den Fall, dass ausschließlich das Service-orientierte Teilsystem betrachtet wird.

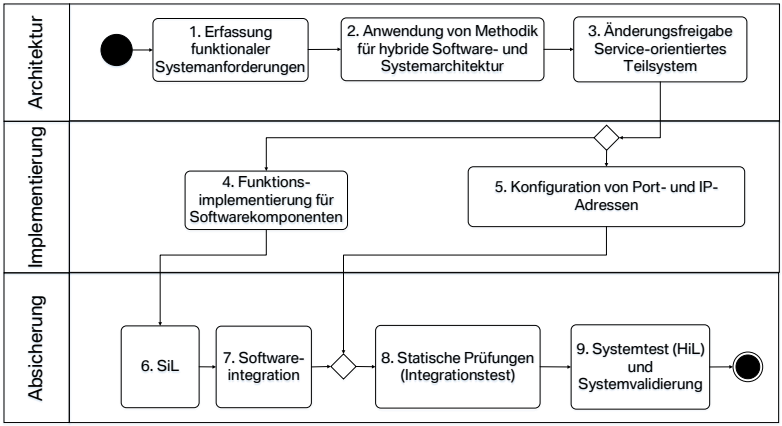

Bild 6.33: Architektur, Implementierung und Absicherung für das Service-orientierte Teilsystem

Die abschließenden *Prozessschritte 6-9* werden durch heute etablierte Methodiken der Absicherung unter Berücksichtigung der in Tabelle 6.6 zusammengefassten Anpassungen erreicht.

6.3.3 Zusammenfassung und Einordnung

Für ein Vorgehen zur Implementierung und Absicherung von Änderungen der Softwarearchitektur wurde das V-Modell für das Service-orientierte und das Signal-orientierte Teilsystem betrachtet. Speziell für die Implementierung eines Kommunikationsnetzwerks zur Realisierung von Informationsflüssen zwischen Softwarekomponenten sind beide Teilsysteme entlang der Konzepte eines *logischen Netzwerks* für das Service-orientierte Teilsystem und eines *physischen Netzwerks* für das Signal-orientierte Teilsystem trennbar.

Durch das logische Netzwerk (s. Bild 6.32) stellt das Service-orientierte Teilsystem separat betrachtet eine Systemgrenze zwischen Software und Hardware dar. Hierdurch wird es ermöglicht funktionale Anforderungen an eine Softwarearchitektur ausschließlich über Änderung von Softwarekomponenten zu realisieren (s. Bild 6.34).

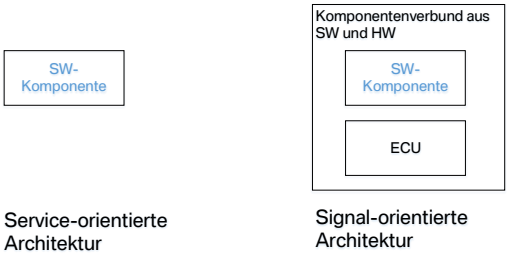

Bild 6.34: Vergleich von relevanten Komponenten bei einer Änderung der Softwarearchitektur

Eine entsprechende prozessuale Darstellung wurde hierfür gewonnen (s. Bild 6.33). Sofern die betrachteten Änderungen jedoch eine Mitbeeinflussung des Signal-orientierten Teilsystems während des Entwurfs der Software- und Systemarchitektur ausweisen, ist die Kopplung mit dem dafür vorgese-

henen Prozess (s. Bild 6.30) weiterhin notwendig. Die dabei relevanten Teile des V-Modells beziehen sich auf die zur Architektur entsprechende Absicherungsphase im Rahmen des *Systemtests* als auch der sich daran anschließende *Abnahmetest*.

7 Umsetzung

Neben der Einbindung der Entwurfsmethodik in das V-Modell soll im Folgenden ihre Umsetzung durch eine Werkzeugkette dargestellt werden.

7.1 Werkzeuggestützte Umsetzung

Im Rahmen der Entwurfsmethodik sind unterschiedliche Beschreibungselemente einer hybriden Software- und Systemarchitektur betrachtet. Diese werden bei der Entwicklung der Architektursichten 1-3 (s. Kapitel 6) spezifiziert und sollen für eine Werkzeugumsetzung zunächst über Metamodelle formalisiert werden.

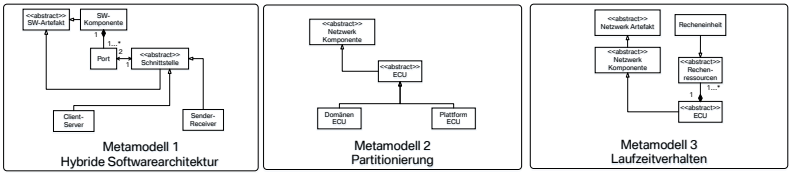

Bild 7.1: Formalisierung von Beschreibungselementen der Architektursichten 1-3 über Metamodelle

In Bild 7.1 sind Ausschnitte aus den zu den Architektursichten entsprechenden Metamodellen als Klassendiagramme gezeigt[1].

[1] Für die vollständige Einsicht in die Metamodelle sei auf den Anhang A verwiesen.

7.1.1 Modellierungswerkzeug

Die Basis der Werkzeugumsetzung bildet eine Modellierungsumgebung. In diesem Zusammenhang werden Elemente der Metamodelle 1-3 auf grafische Beschreibungselemente eines Modellierungswerkzeugs abgebildet. Beispielhaft für das Metamodell 1 (Hybride Softwarearchitektur) und die darin verankerten Klassen zur Beschreibung einer Softwarekomponente ist dies in Bild 7.2 gezeigt.

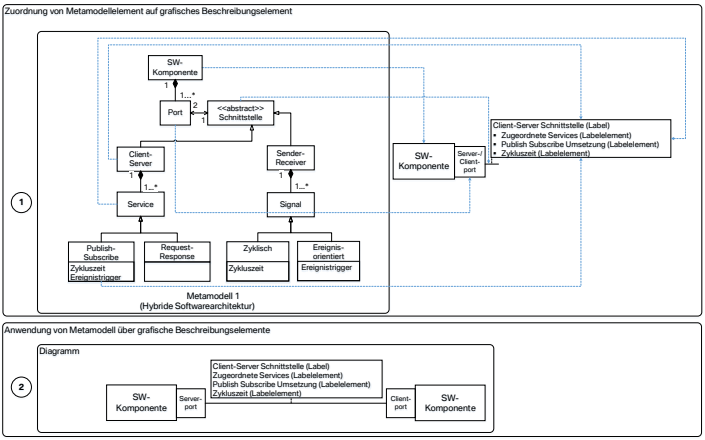

Bild 7.2: Anwendung der Metamodelle in einem Modellierungswerkzeug

Als Ziel der Zuordnung von Klassen zu grafischen Beschreibungselementen sollen die Metamodelle im Rahmen von Diagrammen nutzbar gemacht werden. Die Auswahl eines geeigneten Modellierungswerkzeugs wird weitergehend an Kriterien ausgerichtet.

Auswahlkriterien für ein Modellierungswerkzeug

Kriterium 1: Die Beschreibungselemente des Werkzeugs sollen die notwendigen Artefakte einer Automobilen E/E-Architektur abdecken (s. Bild 2.2).

Kriterium 2: Es soll eine Trennung nach Architekturebenen (s. Bild 2.2) für die Softwarearchitektur und die Topologie von Steuergeräten zur verbesserten Übersichtlichkeit erreicht werden.

Kriterium 3: Es sollen Bezüge zwischen Elementen unterschiedlicher Abstraktionsebenen einer Architektur über Mappings (s. Bild 3.12) unterstützt werden, um sowohl die Zuordnung von Anforderungen auf Softwarekomponenten als auch die Partitionierung von Softwarekomponenten auf Recheneinheiten zu beschreiben.

Kriterium 4: Es soll eine Programmierschnittstelle bereitgestellt werden, um auf Modelle programmatisch zuzugreifen. Durch den programmatischen Zugriff sollen weitergehend Austauschformate als Kopplung zwischen dem Modellierungswerkzeug und einem oder mehreren Analysewerkzeugen ermöglicht werden.

Auswahl eines Modellierungswerkzeugs

Zur Auswahl eines Werkzeugs wird eine Bewertung pro Kriterium vorgenommen. Tabelle 7.1 ordnet in diesem Zusammenhang die im Stand der Wissenschaft und Technik eingeführten Werkzeuge (s. Abschnitt 3.3).

Werkzeug	Erfüllung Kriterium 1	Erfüllung Kriterium 2	Erfüllung Kriterium 3	Erfüllung Kriterium 4
EAST-ADL Open Tool Platform	Ja	Ja	Nein	Ja
PREEvision	Ja	Ja	Ja	Ja
Papyrus	Nein	Nein	Nein	Ja
Enterprise Architect	Nein	Nein	Nein	Ja

Tabelle 7.1: Bewertung von Modellierungswerkzeugen

Es ist zu erkennen, dass die spezifisch für den Automobilen Kontext ausgelegten Werkzeuge PREEvision und EAST-ADL Open Tool Platform mindestens drei Kriterien erfüllen. Die Unterscheidung ist die fehlendende Unterstützung für das Mapping-Konzept für EAST-ADL Open Tool Platform, weshalb die Wahl auf PREEvision als Modellierungswerkzeug fällt. Die Abbildung der Metamodelle 1-3 in PREEvision folgt hierfür dem Prinzip aus Bild 7.2 und wird über eine Zuordnung von mindestens einer Metamodellklasse auf ein grafisches Beschreibungselement des Werkzeugs vorgenommen[2].

7.1.2 Kopplung des Modellierungswerkzeugs mit Analysewerkzeugen

Basierend auf der Abbildung der Metamodelle 1-3 in das Modellierungswerkzeug PREEvision werden für die Umsetzung der Methodik eine Design Space Exploration zur Berechnung von Softwarepartitionen (s. Abschnitt 6.2.2) und eine anschließende Laufzeitbewertung (s. Abschnitt 6.2.3) als Analysemethoden betrachtet.

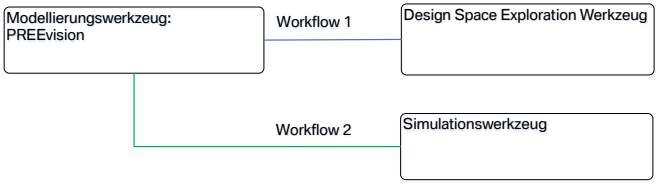

Bild 7.3: Werkzeugkopplung zwischen PREEvision, einem Werkzeug zur Design Space Exploration und Simulationswerkzeug

Das Modellierungswerkzeug muss hierfür mit entsprechenden Werkzeugen im Rahmen der Workflows 1 und 2 gekoppelt werden.

[2] Für die Einsicht zu den Abbildungen zwischen Elementen der Metamodelle 1-3 und grafischen Beschreibungselementen in PREEvision sei auf den Anhang A verwiesen.

Werkzeugauswahl für Workflow 1

Der Workflow 1 betrachtet die Werkzeugkopplung zwischen PREEvision und einem Werkzeug zur Umsetzung der Design Space Exploration (s. Bild 7.3). Hierbei kommen grundsätzlich zwei unterschiedliche Ansätze in Frage. Tabelle 7.2 ordnet in diesem Zusammenhang den Ansatz über einen Solver zur linearen Programmierung und den Ansatz über einen Satisfiability Modulo Theories (SMT) Solver.

Solver	Lineare Programmierung	SMT
Pareto-Optimierung	Ja	Ja
Möglichkeiten für Syntax	Arithmetische Ausdrücke	Kombinationen aus booleschen und arithmetischen Ausdrücken
Mögliches Werkzeug	Matlab	Z3

Tabelle 7.2: Ansätze zur Design Space Exploration

Beide Ansätze unterstützen eine Pareto-Optimierung, wodurch die eingeführten Optimierungskriterien (s. Abschnitt 6.2.2) berücksichtigbar sind. Für den Rahmen dieser Arbeit wird dennoch die höhere Flexibilität des SMT Solvers durch eine Syntax, welche für die Kombination boolescher Operatoren mit arithmetischen Ausdrücken genutzt werden kann, als Vorteil gesehen[3].

In Abschnitt 7.2 wird in diesem Zusammenhang gezeigt, wie die Syntax des gewählten SMT Solvers Z3 [36] zur Kodierung dieser kombinierten Ausdrücke angewendet wird.

[3] Die allgemeine Funktionsweise eines SMT Solvers ist im Anhang A dargestellt.

Werkzeugkopplung im Rahmen von Workflow 1

Für die eigentliche Werkzeugkopplung zwischen PREEvision und dem ausgewählten SMT Solver Z3 werden zunächst *Softwarekomponenten* mit Ressourcenanforderungen sowie die für die Partitionierung betrachteten *Steuergeräte* mit Ressourcenkapazitäten von einem in PREEvision umgesetzten *Codegenerator* verarbeitet (s. Bild 7.4).

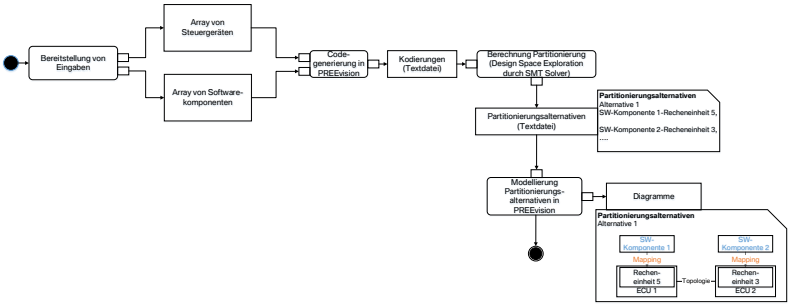

Bild 7.4: Workflow 1

Der Codegenerator leitet aus den beiden Eingangsgrößen in Form von Arrays die notwendigen Elemente der Design Space Exploration ab. Diese umfassen die Entscheidungsvariablen als Kombinationen von Softwarekomponenten und Recheneinheiten, die Einschränkungen für die Partitionierung (bspw. durch begrenzte Rechenkapazitäten) und die beiden Optimierungskriterien (s. Abschnitt 6.2.2). Die Umsetzung über die Syntax des Solvers Z3 ist in Abschnitt 7.2.2 über Ableitungsregeln dargelegt. Nach der Berechnung durch den Solver werden die ermittelten Partitionierungsalternativen als Rückkopplung in PREEvision über *Diagramme* visualisiert.

Werkzeugauswahl für Workflow 2

Für jede *Partitionierungsalternative* als Ergebnis des Workflows 1 werden durch den Workflow 2 Wirkzusammenhänge mit zeitlichen Anforderungen

spezifiziert und anschließend simulativ bewertet (s. Abschnitt 6.2.3). Zur Implementierung der Simulation wird das Werkzeug chronSIM [21] für die Analyse von Latenzzeiten genutzt. Ein Simulationsmodell in chronSIM umfasst hierfür alle Softwarekomponenten eines Wirkzusammenhangs im Rahmen einer sogenannten Ereigniskette (engl. Event Chain) (s. Bild 7.5).

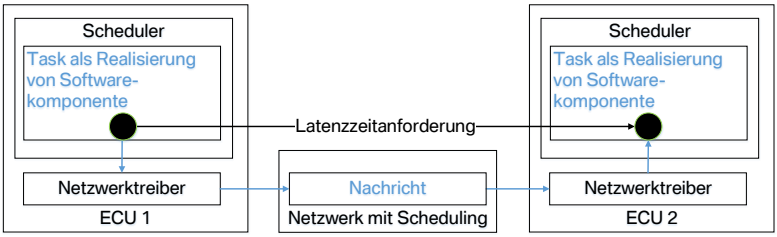

Bild 7.5: Ereigniskette zwischen zwei Tasks und einer Nachricht in chronSIM

Eine Ereigniskette legt das Zusammenspiel von Steuergeräten über Tasks (s. Abschnitt 3.4.2) und Nachrichten mit einer zugeordneten Latenzzeit als Anforderung fest. Die darin sich ausbildenden Ereignisse beschreiben die abgeschlossene Ausführung eines Tasks bzw. die Übertragung einer Nachricht, welche zur Stimulation des nächsten Elements der Ereigniskette und somit zum Aufbau eines gerichteten Graphen führt. Zur eigentlichen Laufzeitprüfung wird dann die Zeit bis zur vollständigen Ausführung des letzten Anteils einer Ereigniskette gemessen und mit der Latenzzeitanforderung verglichen.

Werkzeugkopplung im Rahmen von Workflow 2

Für die Werkzeugkopplung zwischen PREEvision und chronSIM werden zunächst die im Laufzeitmodell betrachteten Softwarekomponenten und die relevante Topologie aus PREEvision in zwei *proprietäre Tabellenformate*[4]

[4] Für die Einsicht in die Formate sei auf den Anhang A verwiesen.

überführt (s. Bild 7.6). Diese Formate umfassen die grundlegenden Komponenten für ein Simulationsmodell und werden durch das Simulationswerkzeug eingelesen. Zur Verknüpfung der Komponenten als ausführbare Simulation wird anschließend das Konzept einer Ereigniskette (s. Bild 7.5) genutzt.

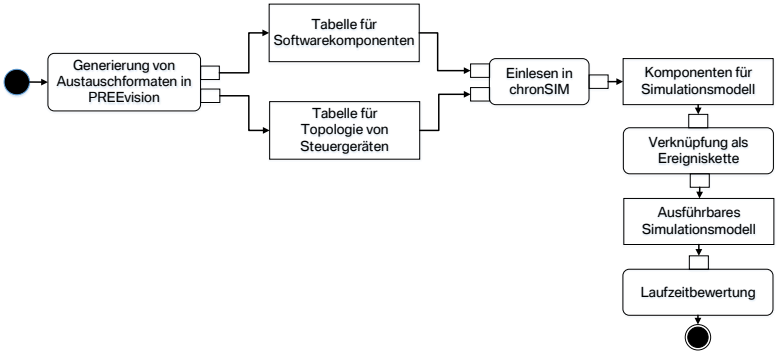

Bild 7.6: Workflow 2

Hierdurch kann neben der Partitionierung von Softwarekomponenten einer hybriden Softwarearchitektur im Rahmen von Workflow 1 auch deren funktionales Zusammenspiel unter Berücksichtigung zeitlicher Anforderung als Abschluss von Workflow 2 bewertet werden.

7.2 Anwendung der Werkzeugkette

Die Werkzeugkette konnte im Rahmen eines Forschungsprojekts für eine künftige E/E-Architektur bei der BMW Group eingesetzt werden. Hierzu wurde zunächst eine hybride Softwarearchitektur aus einem Service-orientierten und einem Signal-orientierten Teilsystem mit Softwarekomponenten der Domänen *Fahrerassistenz*, *Karosserie*, *Infotainment* und der *Fahrdynamikdomäne* in PREEvision modelliert.

Softwarepartitionierung

Für die Partitionierung wurde die Randbedingung berücksichtigt, dass die Softwarekomponenten des Service-orientierten Teilsystems keine Vorfestlegung besitzen (s. Abschnitt 6.2.2). Diese Softwarekomponenten wurden somit als Eingangsgröße für eine Design Space Exploration im Rahmen von Workflow 1 betrachtet. Dementgegen wurde für Softwarekomponenten des Signal-orientierten Teilsystems die bestehende Partitionierung auf domänenspezifischen Steuergeräten genutzt.

Laufzeitbewertung

Für die Laufzeitbewertung im Rahmen von Workflow 2 wurden anschließend Softwarekomponenten der *Fahrerassistenzdomäne* und der *Fahrdynamikdomäne* über die ermittelten Partitionierungsalternativen betrachtet. Die Analyse umfasst zwei Wirkzusammenhänge, die sich durch die Softwarekomponenten *Umfeldmodell 1, 2*, *Trajektorienplanung* und *Fahrdynamik* abbilden lassen (s. Bild 7.7).

Die Spezifikation und Funktionsweise einer Softwarekomponente für ein Umfeldmodell, eine Trajektorienplanung und eine Fahrdynamikregelung wurde beginnend mit Abschnitt 6.2.1 über Schnittstellen und die sich daraus ableitbaren eingangs bzw. ausgangsseitigen Informationsflüsse der Komponenten eingeführt.

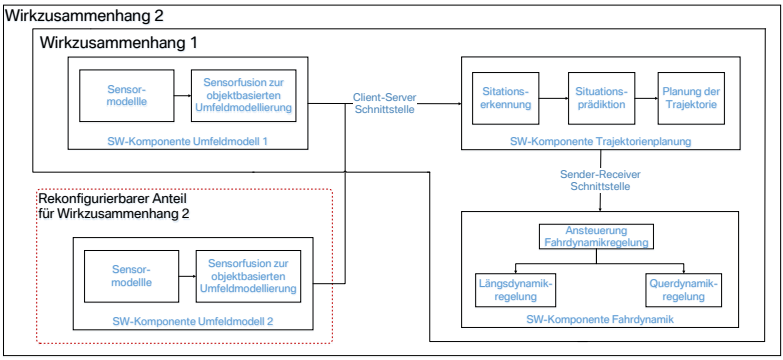

Bild 7.7: Softwarekomponenten *Umfeldmodell 1, 2*, *Trajektorienplanung* und *Fahrdynamik* im Rahmen der Wirkzusammenhänge 1 und 2

Wirkzusammenhang 1

Der erste Wirkzusammenhang berücksichtigt ausschließlich die Softwarekomponenten *Umfeldmodell 1*, *Trajektorienplanung* und *Fahrdynamik* ohne den rekonfigurierbaren Anteil im Rahmen von *Umfeldmodell 2*. Dieser entspricht dem eingeführten Wirkzusammenhang aus Abschnitt 6.2.3.

Latenzzeitanforderung Für die Bewertung des Wirkzusammenhangs 1 wird vorgegeben, dass die Zeit von der Bereitstellung des Umfeldmodells über die Berechnung der Trajektorie bis zur Ansteuerung der Softwarekomponente *Fahrdynamik* eine maximale Latenzzeit von 130 ms nicht überschreiten darf. Dies entspricht dem Bewertungsszenario gemäß Bild 6.25 und wird in Abschnitt 7.2.3 über ergänzende Anforderungen, die für eine Bewertung erfüllt sein müssen, betrachtet.

Wirkzusammenhang 2

Für den zweiten Wirkzusammenhang wurde zusätzlich der rekonfigurierbare Anteil im Rahmen von Umfeldmodell 2 berücksichtigt. In diesem Fall

ist das allgemeine Konzept der Laufzeitrekonfiguration aus Abschnitt 4.2.3 konkret für die Softwarekomponenten Umfeldmodell 1 und 2 als anbietende Instanzen des gleichen Services betrachtet.

Latenzzeitanforderung Für die Bewertung dieses Wirkzusammenhangs muss gelten, dass die Zeit von der Bereitstellung des Umfeldmodells über die Berechnung der Trajektorie bis zur Ansteuerung der Softwarekomponente *Fahrdynamik* auch im Falle eines Wechsels während der Laufzeit von Umfeldmodell 1 zu Umfeldmodell 2 eine maximale Latenzzeit von 130 ms nicht überschreitet. Dieses Bewertungsszenario wird gleichermaßen in Abschnitt 7.2.3 über zusätzliche Anforderungen detailliert.

Simulationsmodelle

Für die Analyse von Wirkzusammenhang 1 und 2 wurde jeweils ein grundsätzliches Simulationsmodell entwickelt. Durch die errechneten Partitionierungsalternativen für die Softwarekomponenten des Service-orientierten Teilsystems im Rahmen von Workflow 1 wurde anschließend pro Alternative jedes dieser Simulationsmodelle nochmals mit spezifischen Eigenschaften der Alternativen weiterentwickelt. Somit wurden insgesamt 6 Modellvarianten für die Laufzeitbewertung genutzt. Bild 7.8 fasst die einzelnen Schritte bei der Anwendung der Werkzeugkette zusammen.

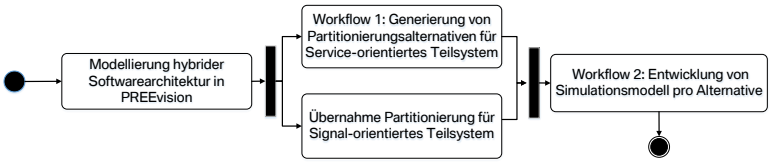

Bild 7.8: Anwendung der Werkzeugkette

Die Modellierung in PREEvision umfasst dabei die Prozessschritte 1-2 gemäß des eingeführten Entwurfsprozesses aus Abschnitt 6.2. Speziell für die

Partitionierung des Service-orientierten Teilsystems werden die darauffolgenden Schritte 3-6 für die Design Space Exploration umgesetzt, bevor abschließend die Simulation zur Umsetzung der Laufzeitbewertung im Rahmen der Prozessschritte 7-9 erreicht wird.

7.2.1 Modellierung hybrider Softwarearchitektur

Das Modell der hybriden Softwarearchitektur für die eingeführten Softwarekomponenten (s. Bild 7.7) ist über ein entsprechendes Komponentendiagramm (s. Bild 7.9) in PREEvision spezifiziert.

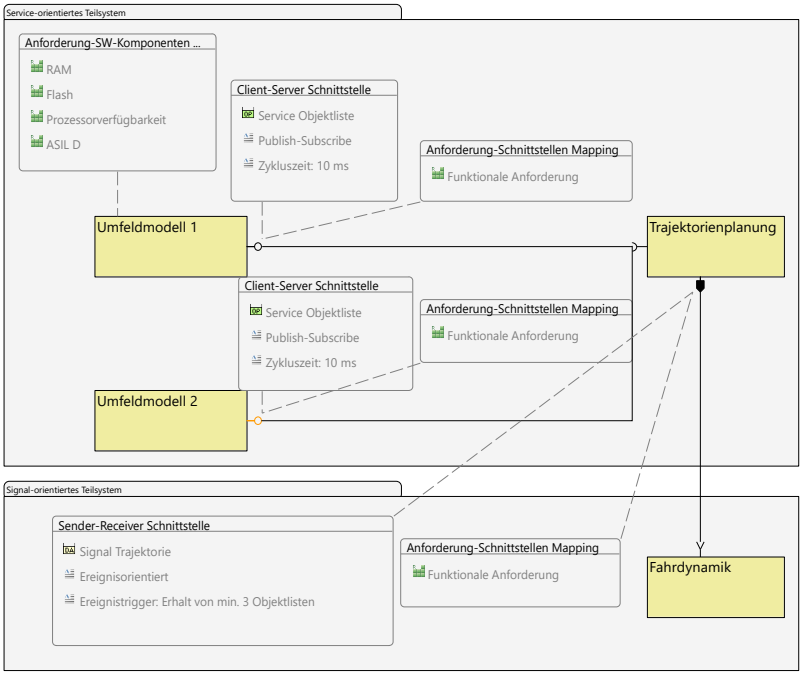

Bild 7.9: Beschreibung einer Kopplung zwischen Service-orientiertem und Signal-orientiertem Teilsystem

Die Softwarekomponenten *Umfeldmodell 1* und *Umfeldmodell 2* werden dabei als funktional identisch deklariert, um sie später für eine Laufzeitbewertung im Rahmen eines Rekonfigurationsszenarios berücksichtigen zu können. Als Ausdrucksmittel hierfür wird für beide Komponenten dieselbe Client-Server Schnittstelle mit einem angebotenen Service spezifiziert. Zur Festlegung des Umsetzungsparadigmas wird diesem Service weiterge-

hend eine *zyklische* Umsetzung mit einer entsprechenden Zeit von 10 ms zugeordnet. Die *Trajektorienplanung* als Nutzer des Services ist als dritte Softwarekomponente des Service-orientierten Teilsystems eingeführt.

Kopplung zwischen Teilsystemen

Für die Kopplung zwischen beiden Teilsystemen stellt die Softwarekomponente *Trajektorienplanung* das zentrale Element dar. Diese realisiert einerseits ein Service-orientiertes Kommunikationsprinzip über die eigeführten Client-Server Schnittstellen mit den Softwarekomponenten Umfeldmodell 1 und 2 und andererseits ein Signal-orientiertes Kommunikationsprinzip über eine entsprechende Sender-Receiver Schnittstelle mit der Softwarekomponente Fahrdynamik.

Entlang dieser Sender-Receiver Schnittstelle ist ein beispielhafter Übergang des Service-orientierten Teilsystems in das Signal-orientierte Teilsystem festgelegt. Der zur Schnittstelle gehörende Informationsfluss wird über ein Label deklariert und umfasst neben einem Signal das zur Umsetzung des Signals spezifizierte ereignisorientierte Kommunikationsparadigma.

Mapping von Anforderungen

Für die hybride Softwarearchitektur sind drei Kategorien von Anforderungen berücksichtigt, die Softwarekomponenten und ihren Schnittstellen zugeordnet werden. Für Softwarekomponenten sind (1) ressourcenbezogene Anforderungen in Form von Speicher- und Prozessoranforderungen festgelegt. Zusätzlich wird die (2) ASIL Bewertung als Ergebnis einer funktionalen Sicherheitsanalyse bestimmt. Beide Kategorien sind über das Label *Anforderung-SW-Komponenten Mapping* visualisiert. Den Schnittstellen zwischen Softwarekomponenten sind (3) funktionale Anforderungen als dritte Kategorie über das Label *Anforderung-Schnittstellen Mapping* zugeordnet.

7.2.2 Workflow 1: Softwarepartitionierung

Für die Design Space Exploration über die Werkzeugkopplung zwischen PREEvision und dem SMT Solver Z3 wurden 25 Softwarekomponenten aus den Domänen *Fahrerassistenz* (4), *Karosserie* (15) und *Infotainment* (6) berücksichtigt. Alle 25 Softwarekomponenten gehören dem Service-orientierten Teilsystem an und stellen das aktuelle Potential bei der BMW Group zur Partitionierung auf Plattformsteuergeräten dar. Im Rahmen der Fahrerassistenzdomäne sind dabei die Softwarekomponenten *Umfeldmodell 1* und *Umfeldmodell 2* sowie die *Trajektorienplanung* mit einbezogen (s. Bild 7.9).

Softwarekomponenten und Ressourcenanforderungen

Die von den Softwarekomponenten umfassten Ressourcenanforderungen beziehen sich auf die eingeführten Kategorien (s. Abschnitt 6.2.1). Ihre Bewertung ist abstrahiert durch Verteilungen aufgeschlüsselt (s. Tabelle 7.3).

ASIL	QM	B	D
Anzahl Softwarekomponenten	14	8	3
RAM [KiB]	[0-100]	(100-1000]	(1000-15000]
Anzahl Softwarekomponenten	8	17	0
Flash [KiB]	[0-100]	(100-1000]	(1000-15000]
Anzahl Softwarekomponenten	1	15	9
Prozessorverfügbarkeit [%]	[0-10]	(10-100]	(100-1000]
Anzahl Softwarekomponenten	3	21	1

Tabelle 7.3: Ressourcenanforderungen von 25 Softwarekomponenten durch Angabe von Verteilungen

Die angeforderte Prozessorverfügbarkeit ist dabei auf eine Recheneinheit mit einer Taktfrequenz von 200 MHz inklusive festgelegter Prozessorarchi-

tektur bezogen[5]. Hierdurch kann eine Prozessorverfügbarkeit von über 100 % als Anforderung einer Softwarekomponente spezifiziert sein.

Steuergeräte und Ressourcenangebot

Als zweite Eingangsgröße für die Exploration sind drei Plattformsteuergeräte festgelegt. Diese sind Teil einer Ethernet Topologie und kommunizieren über ein für das Service-orientierte Teilsystem entsprechendes Kommunikationsprinzip (s. Abschnitt 4.2.2).

Das Ressourcenangebot für ein Steuergerät umfasst zunächst Recheneinheiten, welche über die Taktfrequenz sowie die ASIL Qualifizierung variieren. Jede Recheneinheit verfügt weitergehend über einen exklusiven Arbeitsspeicher (RAM) (s. Bild 7.10). Der Festwertspeicher (Flash) ist ein unter allen Recheneinheiten geteilter Speicher des Plattformsteuergeräts. Tabelle 7.4 fasst die Anzahl an Recheneinheiten eines Plattformsteuergeräts zusammen. Dabei sind pro Steuergerät die Verteilungen der Recheneinheiten auf Taktfrequenzen als Intervalle angegeben. Die Angabe des Arbeitsspeichers pro Steuergerät stellt die Summe über alle Recheneinheiten dar. Der Festwertspeicher ist eine geteilte Ressource aller Recheneinheiten.

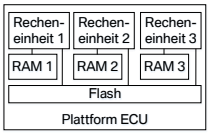

Bild 7.10: Steuergerätearchitektur

ECU	Rechen-einheiten	Taktfrequenz [GHz]	RAM [MiB]	Flash [MiB]
ECU 1	9	0.2 - 2	2.506	512
ECU 2	10	0.2 - 2	4.002	256
ECU 3	6	0.2 - 2	8	512

Tabelle 7.4: Ressourcenangebot von Plattformsteuergeräten

[5] Diese Spezifikation wurde im Rahmen des Projekts als Vorgabe festgelegt.

Generierung des Design Space Exploration Problems

Um die Kodierungen für die Exploration abzuleiten, wurde ein prototypisch umgesetzter Generator[6] in PREEvision genutzt (s. Bild 7.4). Die abgeleiteten Kodierungen umfassen die *(1) Entscheidungsvariablen* für die Partitionierung, die *(2) Einschränkungen* für die Partitionierung und die beiden *(3-4) Optimierungskriterien* 1 und 2 (s. Abschnitt 6.2.2). Zur Realisierung der Bestandteile *(1)* bis *(4)* werden die folgenden Elemente des gewählten Solvers Z3 genutzt:

(1) Entscheidungsvariablen: Realisierung über Integervariablen. Zusätzlich wird der Befehl *assert* bei der Initialisierung genutzt, um sicherzustellen, dass eine Variable für eine binäre Entscheidung nur die Belegungen 0 und 1 annehmen kann. *(2) Einschränkungen*: Anwendung von Befehl *assert* auf arithmetische Ausdrücke, um sicherzustellen, dass für jedes berechnete Design des Solvers Einschränkungen (bspw. durch begrenzte Rechenkapazitäten) erfüllt sind. *(3-4) Optimierungskriterien 1 und 2*: Anwendung von Befehl *minimize* auf die pro Kriterium zu minimierenden Ausdrücke: Für das Kriterium 1 soll dabei sichergestellt werden, dass bei der Berechnung eines Designs die Abweichungen zwischen angefordertem ASIL einer Softwarekomponente und bereitgestelltem ASIL einer Recheneinheit minimal wird. Für das Kriterium 2 soll sichergestellt werden, das bei der Berechnung eines Designs eine minimale Anzahl an Recheneinheiten für die Partitionierung von Softwarekomponenten festgelegt wird.

Ableitungsregeln des Generators Im Folgenden ist für die eingeführten Bestandteile der Exploration gezeigt, wie der Generator in PREEvision diese über Ableitungsregeln als Eingabe für den Solver erzeugt. Zusätzlich wird pro Regel eine konkrete Ausgabe entlang der Syntax des Solvers dargestellt und erläutert.

[6] Der Generator ist über die Programmiersprache Java implementiert.

Entscheidungsvariablen

Zur Generierung von Entscheidungsvariablen werden die entsprechenden Entitäten, Softwarekomponenten und Recheneinheiten, zunächst in zwei Arrays überführt. Anschließend werden alle Kombinationsmöglichkeiten zwischen den eingangsseitigen Elementen der zwei Arrays genutzt und als Ausgabearray von Entscheidungsvariablen bereitgestellt (s. Bild 7.11).

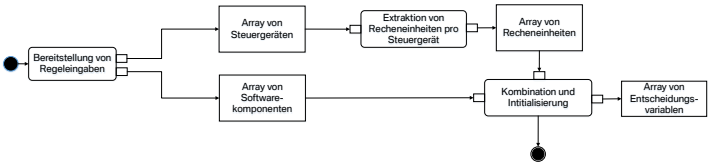

Bild 7.11: Ableitungsregel 1: Entscheidungsvariablen

Eine beispielhafte Entscheidungsvariable des Ausgabearrays ist im Folgenden gezeigt (s. Bild 7.12).

```
(declare-const x_0_0 Int)

// Deklaration einer Entscheidungsvariable
// Modellierte Entscheidung: Partitionierung von SW-Komponente 0 auf Recheneinheit 0

(assert(<= x_0_0 1))
(assert(>= x_0_0 0))

// Initialisierung der Entscheidungsvariable
```

Bild 7.12: Kodierung einer Entscheidungsvariable

Der erste Index beschreibt dabei die von der Entscheidungsvariable betrachtete Softwarekomponente. Der zweite Index beschreibt die dazugehörige Recheneinheit. Neben ihrer Deklaration darf jede Variable nur die Belegungen 0 (Softwarekomponente nicht partitioniert auf Recheneinheit) und 1 (Softwarekomponente partitioniert auf Recheneinheit) erreichen, weshalb

zwei ergänzende arithmetische Ausdrücke im Rahmen des Befehls *assert* als Initialisierung festgelegt sind.

Grundlegende Einschränkung für Exploration

Die grundlegende Einschränkung für die Exploration trifft die Aussage, dass jede Softwarekomponenten nur auf einer Recheneinheit partitioniert werden darf (s. Abschnitt 6.2.2). Hierfür wird das Ausgabearray von Ableitungsregel 1 durch eine Funktion, welche alle relevanten Entscheidungsvariablen für eine Softwarekomponente addiert und einfordert, dass die Summe immer kleiner oder gleich 1 ist, weiterverarbeitet (s. Bild 7.13).

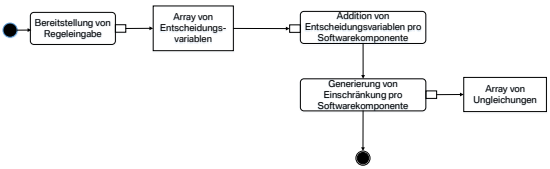

Bild 7.13: Ableitungsregel 2: Generierung der grundlegenden Einschränkung

Beispielhaft für die Softwarekomponente mit dem Index 0 und dem betrachteten Umfang von 25 Recheneinheiten (s. Tabelle 7.4) ergibt sich hierdurch folgende Ungleichung (s. Bild 7.14).

```
(assert(<=(+( x_0_0)( x_0_1)...(x_0_24)) 1) )

// SW-Komponente 0 kann nur auf einer der 25 Recheneinheiten partitioniert werden
```

Bild 7.14: Grundlegende Einschränkung für Softwarekomponente 0

Einschränkungen bezüglich Ressourcenanforderungen

Die Ressourcenanforderungen von Softwarekomponenten und das Ressourcenangebot von Steuergeräten schränken die Exploration weiter ein (s. Ab-

schnitt 6.2.2) und werden durch entsprechende Ableitungsregeln berücksichtigt.

Einschränkungen bezüglich ASIL Für Einschränkungen durch die ASIL Anforderung einer Softwarekomponente und die ASIL Qualifikation einer Recheneinheit ist die Ableitungsregel in Bild 7.15 gezeigt.

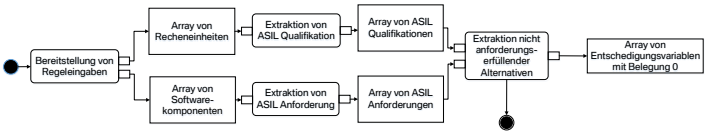

Bild 7.15: Ableitungsregel 3: Einschränkungen durch ASIL

Die Ableitungsregel überführt für jede Recheneinheit bzw. Softwarekomponente die ASIL Qualifikation bzw. ASIL Anforderung in ein Array. Anschließend wird geprüft, welche Recheneinheiten ungeeignet sind für eine Partitionierung aufgrund eines zu geringen ASIL und somit die Entscheidungsvariable vorab mit dem Wert 0 zum Ausschluss für den Solver zu belegen ist. In Bild 7.16 ist beispielhaft eine ableitbare Kodierung gezeigt.

```
(assert(= x_0_1 0))

// Einschränkung auf Basis von ASIL Bewertung
// SW-Komponente 0 wird nie auf Recheneinheit 1 partitioniert
```

Bild 7.16: Einschränkung in Bezug auf die ASIL Anforderung für Softwarekomponente 0 und die ASIL Qualifikation für Recheneinheit 1

Durch die Kodierung wird sichergestellt, dass die Softwarekomponente mit dem Index 0 nie auf der Recheneinheit mit den Index 1 aufgrund eines zu geringen ASIL partitioniert wird.

Einschränkungen bezüglich Arbeitsspeicher und Prozessorverfügbarkeit Für Einschränkungen bezüglich des begrenzten Arbeitsspeichers und der begrenzten Prozessorverfügbarkeit einer Recheneinheit wird die Ableitungsregel in Bild 7.17 genutzt.

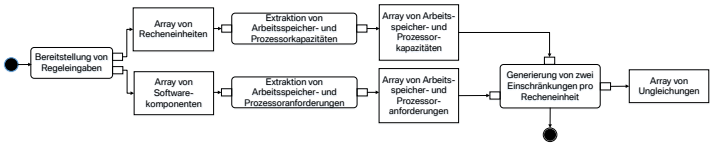

Bild 7.17: Ableitungsregel 4: Einschränkungen durch Arbeitsspeicher und Prozessorverfügbarkeit

Die Ableitungsregel überführt für jede Recheneinheit bzw. Softwarekomponente die Kapazität bzw. Anforderung in ein Array. Anschließend werden zwei Einschränkungen, für den Arbeitsspeicher und die Prozessorverfügbarkeit, pro Recheneinheit als Ungleichung generiert. Diese stellen sicher, dass eine Recheneinheit für den Extremfall, dass alle Softwarekomponenten auf ihr partitioniert sind, nicht überlastet wird. In Bild 7.18 ist beispielhaft eine durch die Regel ableitbare Kodierung in Bezug auf den Arbeitsspeicher gezeigt.

```
(assert(<=(+(* a_0_0 300)...(* a_24_0 500)) 1500))

// Die Summe der Arbeitsspeicheranforderungen (300 KiB +...+ 500 KiB) der 25
Softwarekomponenten darf die Kapazität von 1500 KiB der Recheneinheit 0 nicht
überschreiten
```

Bild 7.18: Einschränkung in Bezug auf den Arbeitsspeicher für Recheneinheit 0

Dabei wird die mögliche Partitionierung aller 25 Softwarekomponenten auf der Recheneinheit mit dem Index 0 betrachtet, wobei jede Softwarekomponente über ihre entsprechende Entscheidungsvariable kodiert ist. Für den

Fall der Partitionierung muss die Summe der Integerwerte als Arbeitsspeicheranforderungen kleiner oder gleich des Integerwerts als Kapazität sein.

Einschränkungen bezüglich Festwertspeicher Für Einschränkungen in Bezug auf den Festwertspeicher wird die Generierung der Kodierungen pro Steuergerät vorgenommen, weil sich mehrere Recheneinheiten eines Steuergeräts einen Festwertspeicher teilen (s. Bild 7.10). Die entsprechende Ableitungsregel hierfür ist in Bild 7.19 gezeigt.

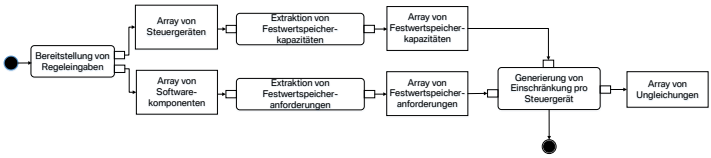

Bild 7.19: Ableitungsregel 5: Einschränkungen durch Festwertspeicher

Die Regel überführt für jedes Steuergerät bzw. jede Softwarekomponente die Festwertspeicherkapazität bzw. Festwertspeicheranforderung in ein Array. Im Unterschied zur Ableitungsregel 4 für Einschränkungen in Bezug auf den Arbeitsspeicher und die Prozessorverfügbarkeit (s. Bild 7.17) werden die Ungleichungen jedoch durch Entscheidungsvariablen mehrerer Recheneinheiten, die den gemeinsamen Festwertspeicher eines Steuergeräts nutzen, gebildet.

In Bild 7.20 ist beispielhaft eine durch die Regel ableitbare Kodierung gezeigt. Dabei wird die Partitionierung der 25 Softwarekomponenten auf den neun Recheneinheiten, berücksichtigt über die Indizes 0 bis 8, des Steuergeräts 1 betrachtet (s. Tabelle 7.4). Jede der 25 Softwarekomponenten ist durch Entscheidungsvariablen kodiert und trägt zur Belastung des Steuergeräts im Falle der Partitionierung durch eine Anforderung als Integerwert bei.

```
(assert(<=(+(* a_0_0 1500)...(* a_24_0 6000)...(* a_0_8 1500)...(* a_24_8 6000)) 512000))

// Die Summe der Festwertspeicheranforderungen (1500 KiB +... + 6000 KiB) der 25
Softwarekomponenten darf die Kapazität von 512000 KiB des Steuergeräts nicht überschreiten
```

Bild 7.20: Einschränkung in Bezug auf den Festwertspeicher

Optimierungskriterien

Die Kriterien für die Exploration beschreiben beide eine Minimierung (s. Abschnitt 6.2.2).

Kriterium 1 Das Kriterium 1 minimiert die Überfüllung von ASIL Anforderungen durch Recheneinheiten bei der Partitionierung. Zur Generierung der notwendigen Kodierungen wird die Ableitungsregel 6 genutzt (s. Bild 7.21).

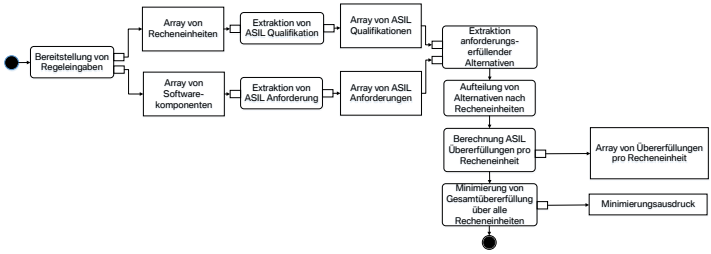

Bild 7.21: Ableitungsregel 6: Kodierungen für Optimierungskriterium 1

Dabei wird gleichermaßen zu Ableitungsregel 3 zunächst für jede Recheneinheit bzw. Softwarekomponente die ASIL Qualifikation bzw. ASIL Anforderung in ein Array überführt. Im Gegensatz zu Ableitungsregel 3 findet danach der komplementäre Filter Anwendung, wodurch ausschließlich die anforderungserfüllenden Partitionierungsalternativen zur weiteren Verarbeitung extrahiert werden.

Der Regel folgend werden diese Alternativen nach Recheneinheiten aufgeteilt und anschließend pro Recheneinheit ein Ausdruck für die ASIL Überfüllung abgeleitet (s. Bild 7.22).

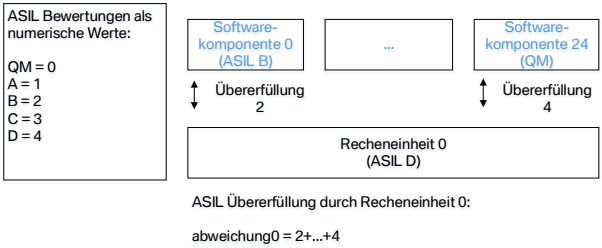

Bild 7.22: Berechnung der Übererfüllung von ASIL Anforderungen durch eine Recheneinheit

Der eigentliche Minimierungsausdruck wird dann über die Übererfüllungen aller Recheneinheiten und die Anwendung des Befehls *minimize* gebildet. In Bild 7.23 ist eine beispielhafte Ausgabe der Regel gezeigt. Dabei wird zunächst für die Recheneinheit 0 die Übererfüllung berechnet und über den Befehl *assert* sichergestellt, dass diese der Integervariable *abweichung0* zugewiesen ist.

```
(assert (= abweichung0 (+ (* a_0_0 2)...(* a_24_0 4))))

// Summation von Abweichung zwischen angebotenem ASIL der Recheneinheit 0 zu
gefordertem ASIL der 25 SW-Komponenten

(minimize (+ abweichung0 abweichung1...))
// Minimierung von Gesamtabweichung für alle Recheneinheiten
```

Bild 7.23: Optimierungskriterium 1: Minimierung

Anschließend wird diese Variable mit den entsprechenden Variablen der übrigen Recheneinheiten als Summe verrechnet, bevor abschließend die Funktion *minimize* auf die Gesamtabweichung angewendet wird.

Kriterium 2 Das Kriterium 2 minimiert die genutzten Recheneinheiten bei der Partitionierung. Zur Generierung der notwendigen Kodierungen wird die Ableitungsregel 7 genutzt (s. Bild 7.24). Demnach werden zunächst die zu minimierenden Recheneinheiten in ein Array überführt.

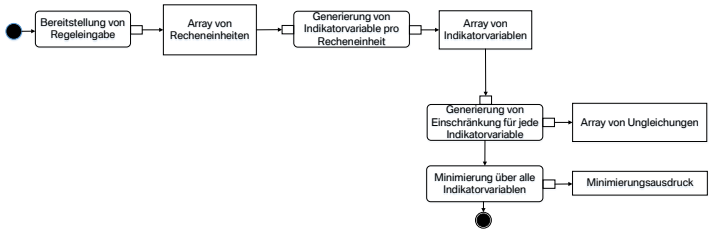

Bild 7.24: Ableitungsregel 7: Kodierungen für Optimierungskriterium 2

Anschließend wird für jede Recheneinheit eine Indikatorvariable generiert, welche anzeigen soll, ob eine Recheneinheit bei der Partitionierung durch mindestens eine Softwarekomponente genutzt wird. In Bild 7.25 ist beispielhaft eine Indikatorvariable *r0* für die Recheneinheit mit dem Index 0 als Integervariable eingeführt.

```
(declare-const r0 Int)

// Indikatorvariable für Recheneinheit 0
```

Bild 7.25: Indikatorvariable für Recheneinheit 0

Für die Belegung der Indikatorvariable wird gefordert, dass sie mit dem Wert 0 belegt werden soll, falls keine Softwarekomponente auf der entsprechenden Recheneinheit partitioniert wird. Für den Fall, dass mindestens eine Softwarekomponente für die Partitionierung auf der Recheneinheit betrachtet wird, soll die Belegung mit dem Wert 1 erfolgen. Um dies zu erreichen wird für jede Indikatorvariable eine Einschränkung generiert, welche

sicherstellt, dass im Falle mindestens einer Partitionierung durch eine Softwarekomponente die Belegung entsprechend erfolgt. In der Kodierung (s. Bild 7.26) wird dies über eine Ungleichung erreicht, welche beispielhaft für die Indikatorvariable der Recheneinheit 0 einfordert, dass sie größer oder gleich ist als alle 25 für die entsprechende Recheneinheit relevanten Entscheidungsvariablen.

```
(assert(and(>= r0 a_0_0 )...(>= r0 a_24_0)))
// Einschränkung für Recheneinheit 0

(minimize (+ r0 r1...))
// Minimierung über Indikatorvariablen aller Recheneinheiten
```

Bild 7.26: Optimierungskriterium 2: Minimierung

Anschließend wird die Summe über alle Indikatorvariablen und somit Recheneinheiten gebildet und mit dem Befehl *minimize* zur Optimierung ausgeführt. Hierdurch wird sichergestellt, dass die Indikatorvariable nur den Wert 1 trägt, wenn mindestens eine Softwarekomponente auf der entsprechenden Recheneinheit platziert wird. Ansonsten trägt die Indikatorvariable den Wert 0 und ist für das Optimierungskriterium somit im Idealzustand.

Zusätzliche Anforderungen und Lösungen

Tabelle 7.3 umfasst 100 Anforderungen von 25 Softwarekomponenten. Neben diesen 100 Anforderungen wurden für die Softwarekomponenten *Umfeldmodell 1* und *Umfeldmodell 2* zusätzliche Anforderungen bezüglich der Verfügbarkeit von Sensorik, die zur Umsetzung der Sensormodelle als Teil des internen Aufbaues der SW-Komponenten notwendig sind, betrachtet. Es wurde dabei über entsprechende Kodierungen festgelegt, dass die Softwarekomponente *Umfeldmodell 1* nur auf Recheneinheiten von *ECU 1* und die Softwarekomponente *Umfeldmodell 2* nur auf Recheneinheiten von *ECU 3* partitioniert werden kann. Auf Basis von 102 Anforderungen an die Partitionierung konnten 3 Lösungen mit spezifischen Bewertungen der Optimierungskriterien ermittelt werden (s. Tabelle 7.5).

Lösung	1	2	3
Kriterium 1: Übererfüllungen ASIL	22	40	68
Kriterium 2: Anzahl Recheneinheiten	13	9	8

Tabelle 7.5: Lösungen der Design Space Exploration mit Bewertung von Optimierungskriterium 1 und 2

Modellierung von Partitionierungsalternativen

Jede Lösung der Exploration beschreibt neben der Bewertung der Optimierungskriterien eine spezifische Partitionierungsalternative.

Softwarekomponente	Recheneinheit	Zugehörige ECU
Umfeldmodell 1	3	1
Umfeldmodell 2	4	3
Trajektorienplanung	2	2

Tabelle 7.6: Partitionierungsalternative 1 für Softwarekomponenten *Umfeldmodell 1 und 2* sowie *Trajektorienplanung*

Beispielhaft für Lösung 1 ist die ermittelte Partitionierungsalternative für die Softwarekomponenten *Umfeldmodell 1 und 2* sowie *Trajektorienplanung* gezeigt (s. Tabelle 7.6). Zur Abbildung dieser Partitionierungsalternative in PREEvision sind die Softwarekomponenten aus Bild 7.9 mit den entsprechenden Mappings auf Recheneinheiten der für das Service-orientierte Teilsystem relevanten Plattformsteuergeräte festgelegt und über das Label *SW-Komponenten-Recheneinheiten Mapping* visualisiert (s. Bild 7.27)[7].

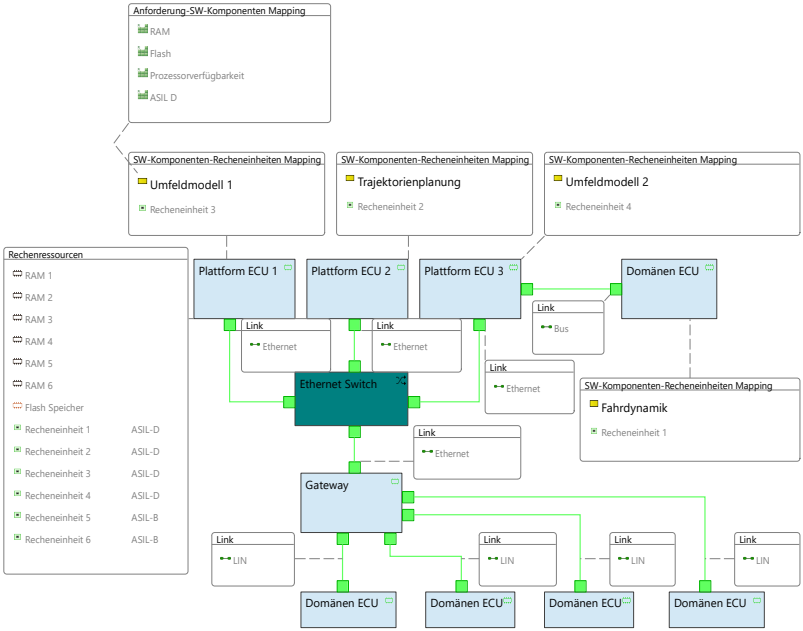

Bild 7.27: Beschreibung der Partitionierungsalternative 1 für Softwarekomponenten *Umfeldmodell 1 und 2* sowie *Trajektorienplanung* (gelb) in Ethernet Topologie (Service-orientiertes Teilsystem), welche mit klassischen Bussegmenten (Signal-orientiertes Teilsystem) gekoppelt ist.

Das entwickelte Modell dient neben der Abbildung von Partitionierungsalternativen für das Service-orientierte Teilsystem auf Plattformsteuergeräten

[7] Die Abbildung zeigt für Plattform ECU 1 eine Untermenge aller Rechenressourcen.

zur Synchronisation mit dem Signal-orientierten Teilsystem. Dieses wurde für die Design Space Exploration nicht betrachtet, weil die entsprechenden Partitionierungsziele der Softwarekomponenten durch domänenspezifische Steuergeräte vorab festgelegt wurden (s. Abschnitt 7.2). In Bild 7.27 ist dies für die Softwarekomponente *Fahrdynamik* mit ihrer Partitionierung auf einem Steuergerät der entsprechenden Domäne über ein Mapping explizit dargestellt.

Weitergehend berücksichtigt das Modell eine LIN Topologie mit vier domänenspezifischen Steuergeräten der Karosseriedomäne. Diese werden für 15 Softwarekomponenten dieser Domäne aus dem Signal-orientierten Teilsystem zur Partitionierung genutzt und stellen Übernahmekomponenten einer heutigen E/E-Architektur dar.

Zur Kommunikation mit den Softwarekomponenten des Service-orientierten Teilsystems im Rahmen der Ethernet Topologie wird ein LIN Gateway herangezogen. Aus Latenzgründen ist das Konzept eines Gateways für die Kommunikation zwischen der Ethernet Topologie und dem domänenspezifischen Steuergerät für die *Fahrdynamikdomäne* vernachlässigt und durch eine direkte Busverbindung festgelegt.

7.2.3 Workflow 2: Laufzeitbewertung

Für die in Workflow 1 errechneten Partitionierungsalternativen wurden im Rahmen des Workflows 2 zeitliche Aspekte für die Kommunikation zwischen Softwarekomponenten betrachtet. Den Betrachtungsumfang stellen die eingeführten Wirkzusammenhänge 1-2 (s. Bild 7.7) im Rahmen der Domänen *Fahrerassistenz* und *Fahrdynamik* dar.

Simulationsmodelle

Die für die Laufzeitbewertung genutzten Simulationsmodelle sind in Teilen aus Modellen in PREEvision generierbar. Hierzu können die zwei Exporte aus PREEvision nach chronSIM genutzt werden (s. Bild 7.6). Der erste Export umfasst die für die Laufzeitbewertung relevanten Softwarekomponenten. Der zweite Export erfasst die notwendige Topologie der Steuergeräte[8] auf der die Softwarekomponenten partitioniert sind.

Nach dem Export in chronSIM werden Softwarekomponenten als Tasks weiterentwickelt. Die Weiterentwicklung umfasst die Implementierung von Funktionen, welche die von einer Softwarekomponente spezifizierten eingangsseitigen und/oder ausgangsseitigen Informationsflüsse realisieren (s. Bild 7.28). Die zeitlichen Eigenschaften dieser Verarbeitung folgen den im Rahmen des Modells der Softwarearchitektur spezifizierten Festlegungen eines zyklischen bzw. ereignisorientierten Verhaltens (s. Bild 7.9).

[8] Für eine prototypische Umsetzung ist dabei ausschließlich die Ethernet Topologie, welche aktuell in der BMW Group neu konzipiert wird, betrachtet. Die für die Simulation notwendigen Anteile einer Systembus Topologie werden als vorhanden im Simulationswerkzeug vorausgesetzt und sind nicht Teil der Exporte.

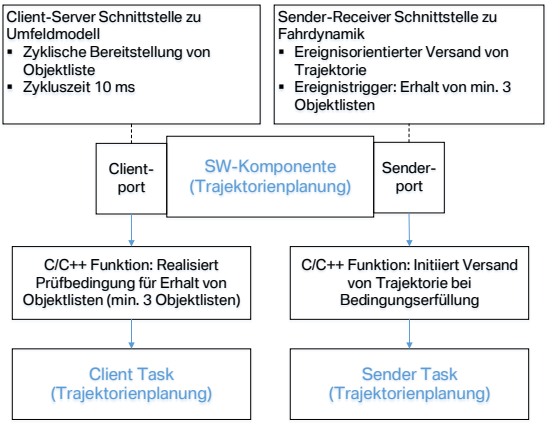

Bild 7.28: Softwarekomponente Trajektorienplanung und entsprechende Tasks

Neben der Weiterentwicklung von Softwarekomponenten zu Tasks werden die für die Ausführung der Tasks notwendigen Steuergeräte in chronSIM betrachtet und über das Konzept einer Ereigniskette verknüpft (s. Bild 7.5). Durch Berücksichtigung von zeitlichen Anforderungen an diese Ereigniskette kann anschließend ein Wirkzusammenhang als verteiltes System in Bezug auf die Laufzeit zwischen dem Start der ersten Task und der Beendigung der letzten Task der Ereigniskette bewertet werden.

Speziell für die im Rahmen dieser Arbeit betrachtete hybride Architektur ist der Aufbau der notwendigen Steuergeräte für eine Ereigniskette voneinander zu unterscheiden. Der Grund hierfür ist, dass sowohl Steuergeräte mit einem Service-orientierten Kommunikationsprinzip (Plattformsteuergeräte) als auch Steuergeräte mit einem ausschließlich Signal-orientierten Kommunikationsprinzip (domänenspezifische Steuergeräte) herangezogen werden.

Simulationsansatz für Plattformsteuergeräte

Plattformsteuergeräte realisieren zunächst jede auf einer Recheneinheit eines Plattformsteuergeräts partitionierte Softwarekomponente über Tasks in Übereinstimmung mit dem Vorgehen aus Bild 7.28. Das für alle Plattformsteuergeräte angewendete Service-orientierte Kommunikationsprinzip zur Initiierung eines Informationsaustauschs zwischen einem Task als Service bzw. Client ist über das Konzept einer Service-orientierten Middleware in C/C++ implementiert (s. Bild 7.29).

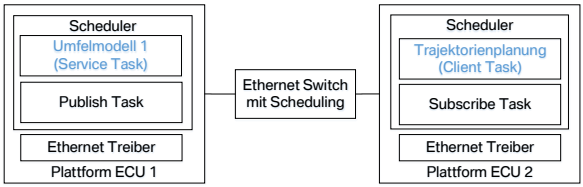

Bild 7.29: Plattformsteuergerät mit Service-orientiertem Kommunikationsprinzip in chronSIM

Die konkrete Realisierung adressiert einen Publish-Subscribe Mechanismus. Hierdurch werden die Services als angebotene Funktionen zunächst über Publish Tasks (s. Bild 7.29 links) im Netzwerk veröffentlicht. Zur Übermittlung eines Nutzungswunschs wird anschließend ein Subscribe Task (s. Bild 7.29 rechts) ausgeführt, bevor der Client Informationen bereitgestellt durch einen Service nutzen kann.

Alle Kategorien eines Tasks werden durch einen Scheduler organisiert. Dieser stellt sicher, dass zuerst der Publish-Subscribe Mechanismus und anschließend die Service bzw. Client Tasks ausgeführt werden. Durch Betrachtung eines in einer Ethernet Topologie verteilten Systems werden weiterhin entsprechende Treiber für den Netzwerkzugriff realisiert (s. Bild 7.29). Die-

se sind in Übereinstimmung mit den aktuell in der Automobilindustrie verfolgten Standards für die Ethernet Technologie [17], [16], [14], [13].

Simulationsansatz für Domänenspezifische Steuergeräte

Domänenspezifische Steuergeräte realisieren gleichermaßen wie Plattformsteuergeräte jede darauf partitionierte Softwarekomponente als Task in Übereinstimmung mit dem Vorgehen aus Bild 7.28. Im Unterschied zu Plattformsteuergeräten können domänenspezifische Steuergeräte jedoch ausschließlich ein Signal-orientiertes Kommunikationsprinzip für den Informationsaustausch zwischen Tasks umsetzen. Hierbei realisieren dem Prinzip entsprechende Sender bzw. Receiver Tasks eine statische Kommunikationsfestlegung. In Bild 7.30 ist ein solcher Zusammenhang für einen CAN Bus gezeigt und berücksichtigt einen Spezialfall der Signal-orientierten Kommunikation.

Dieser ist dadurch gegeben, dass der über das Signal-orientierte Kommunikationsprinzip dargestellte Informationsfluss auf dem Bus exakt an der Grenze zur Ethernet Topologie mit Service-orientiertem Kommunikationsprinzip realisiert wird.

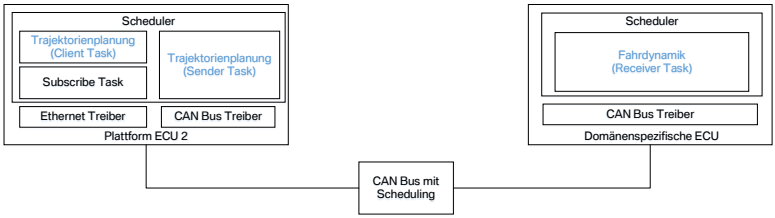

Bild 7.30: Signal-orientiertes Kommunikationsprinzip in chronSIM

Der für den Informationsfluss notwendige Sender Task (s. Bild 7.30 links) verteilt sich in diesem Zusammenhang auf das bereits in Bild 7.29 eingeführte Plattformsteuergerät. Dieses Steuergerät dient als Übersetzer von

Service-orientierter in Signal-orientierte Kommunikation. Der entsprechende Receiver Task hingegen ist auf einem domänenspezifischen Steuergerät der *Fahrdynamikdomäne* umgesetzt. Dieses Steuergerät setzt ausschließlich das Signal-orientierte Kommunikationsprinzip um. Für den entsprechenden Task des Steuergeräts ist weitergehend ein Scheduler eingeführt. Dieser wird nach erfolgreicher Übertragung von Nachrichten über den CAN Bus aktiv und führt die Receiver Task aus.

Im Unterschied zu einem Plattformsteuergerät entfallen für das Scheduling die zur Umsetzung des Publish-Subscribe Mechanismus notwendigen Tasks. Die Zugriffe auf das Bussystem als Übertragungsmedium werden gleichermaßen zu einem Service-orientierten Kommunikationsprinzip über entsprechende Treiber realisiert.

Simulationsansatz für Gateways

Neben Plattformsteuergeräten und domänenspezifischen Steuergeräten ist für die eingeführte Topologie ein LIN Gateway betrachtet (s. Bild 7.27). Dieses agiert zwischen einer Ethernet Topologie von Plattformsteuergeräten und einer LIN Topologie von domänenspezifischen Steuergeräten. Die für die Laufzeitbewertung betrachteten Wirkzusammenhänge 1 und 2 im Rahmen der Fahrerassistenzdomäne und der Fahrdynamikdomäne (s. Bild 7.7) haben keine funktionale Abhängigkeit zu den Steuergeräten der LIN Topologie.

Der Grund ist deren ausschließliche Nutzung als Partitionierungsziele für Komponenten der Karosseriedomäne im Rahmen des Signal-orientierten Anteils. Gleichzeitig sind Softwarekomponenten der Karosseriedomäne im Rahmen des Service-orientierten Anteils auf Recheneinheiten von Plattformsteuergeräten der Ethernet Topologie partitioniert.

Speziell durch den Ethernet Switch als geteiltes Medium ist somit wiederum eine Abhängigkeit zwischen allen Domänen erzeugt. Um diese Abhängigkeit zu berücksichtigen wird die notwendige Kommunikation zwischen dem Service-orientierten Anteil der Karosseriedomäne auf Plattformsteuergeräten und dem Signal-orientierten Anteil in der LIN Topologie berücksichtigt. Hierbei werden für jede Partitionierungsalternative die maximalen Kommunikationsaufwände zwischen den Plattformsteuergeräten und dem LIN Gateway über den Switch berechnet und im Simulationsmodell fest konfiguriert[9]. Bei der Simulation wird somit dieser Switch durch Ethernet Nachrichten aller Domänen mit Abhängigkeiten zum Signal-orientierten Teilsystem belastet. Zur Bevorzugung der Fahrerassistenzdomäne sind die entsprechenden Nachrichten jedoch mit der höchsten Priorität versehen.

[9] Dasselbe Vorgehen kann gewählt werden, wenn Abhängigkeiten zwischen Softwarepartitionen auf Plattformsteuergeräten in Bezug auf ihre maximalen Aufwände für das Kommunikationsnetzwerk berücksichtigt werden sollen.

Laufzeitbewertung

Zur Anwendung der Simulationsansätze im Rahmen einer Laufzeitbewertung werden zunächst die eingeführten Wirkzusammenhänge 1 und 2 (s. Bild 7.7) in PREEvision modelliert. Die Modellierung berücksichtigt dabei die bereits in Bild 7.9 eingeführten Softwarekomponenten, welche immer in Bezug auf eine spezifische Partitionierungsalternative bezüglich der Ergebnisse der Design Space Exploration festgelegt sind (s. Bild 7.31).

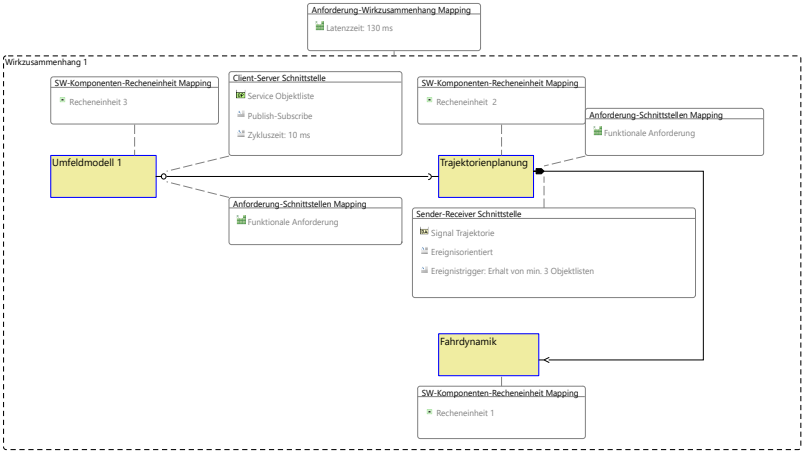

Bild 7.31: Wirkzusammenhang 1 mit zugeordneter Latenzzeitanforderung in PREEvision

Im konkreten Fall aus Bild 7.31 für Wirkzusammenhang 1 entspricht diese der Partitionierungsalternative 1 für die Softwarekomponenten Umfeldmodell 1 und Trajektorienplanung (s. Tabelle 7.6). Die Softwarekomponente Fahrdynamik als Anteil aus dem Signal-orientierten Teilsystem ist weiterhin fest auf einer Recheneinheit eines domänenspezifischen Steuergeräts betrachtet (s. Bild 7.27).

Spezifikation von Laufzeitverhalten Durch die Spezifikation des Laufzeitverhaltens für Wirkzusammenhang 1 sollen zu berücksichtigende Anfor-

derungen an eine nachfolgende Simulation in chronSIM festgelegt werden (s. Tabelle 7.7).

Anforderung	Beschreibung
1. Verhalten von Umfeldmodell Service 1	Aktualisierung von Objektliste in 10 ms Zyklen
2. Verhalten von Trajektorienplanung	Prüfung, ob mindestens drei aktualisierte Objektlisten erfolgreich übertragen wurden und Fahrdynamik angesteuert werden kann
3. Latenzzeitanforderung für Wirkzusammenhang	Max. 130 ms bis zur Ansteuerung und Ausführung der Fahrdynamik

Tabelle 7.7: Laufzeitanforderungen an Wirkzusammenhang 1

Zur Modellierung dieser Laufzeitanforderungen wird ein Sequenzdiagramm abgeleitet (s. Bild 7.32).

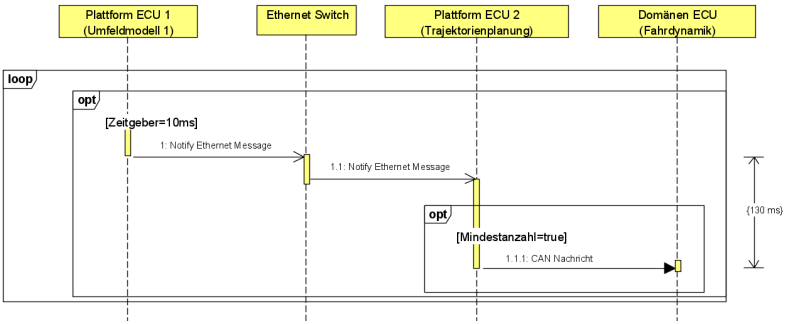

Bild 7.32: Wirkzusammenhang 1 im Rahmen eines Laufzeitmodells

Für das Sequenzdiagramm wird davon ausgegangen, dass zunächst eine Service Discovery (s. Bild 6.21) zwischen den Softwarekomponenten Umfeld-

modell 1 und Trajektorienplanung ausgeführt wurde. Anschließend ist die zyklische Nutzung des von Umfeldmodell 1 erbrachten Service über Ethernet Nachrichten und die anschließende Ansteuerung der Fahrdynamik über eine CAN Nachricht im Diagramm modelliert.

Entwickeltes Simulationsmodell Das zur Simulation des Wirkzusammenhangs entwickelte Modell ist in Bild 7.33 gezeigt und stellt den Übergang von der Modellierung zu dem Werkzeug chronSIM dar.

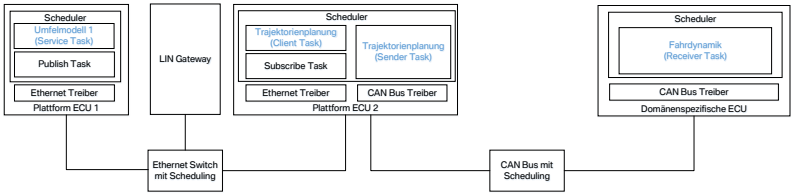

Bild 7.33: Hybrider Simulationsansatz für Wirkzusammenhang 1

Für den Service bzw. den Client Task als Realisierung des zyklischen Informationsflusses zwischen dem Umfeldmodell 1 und der Trajektorienplanung wird der Simulationsansatz für Plattformsteuergeräte genutzt (s. Abschnitt 7.2.3). Zur Initiierung der Kommunikation zwischen diesen Tasks kommt die vorgestellte Realisierung eines Publish-Subscribe Mechanismus zum Tragen.

Die Verbindung der Steuergeräte wird weitergehend über eine Ethernet Topologie mit einem zentralen Ethernet Switch erreicht[10]. Innerhalb dieses Verbunds ist auch das LIN Gateway über den entsprechenden Simulationsansatz (s. Abschnitt 7.2.3) berücksichtigt. Für das Signal-orientierte Kommunikationsprinzip zur Realisierung des ereignisorientierten Informa-

[10] Das Plattformsteuergerät ECU 3 aus Bild 7.27 ist Teil dieser Topologie. Für den Wirkzusammenhang 1 ist es jedoch funktional ohne Bedeutung und deshalb in diesem Abschnitt nicht explizit aufgeführt.

tionsflusses zwischen dem Sender Task der Trajektorienplanung und dem Receiver Task der Fahrdynamik wird der Simulationsansatz für domänenspezifische Steuergeräte (s. Bild 7.30) genutzt.

Die sich während der Simulation ausbildende Ereigniskette mit der zu prüfenden Latenzzeitanforderung von 130 ms ist in Bild 7.34 gezeigt.

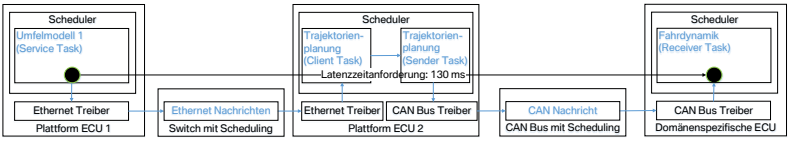

Bild 7.34: Ereigniskette für Wirkzusammenhang 1 als gerichteter Graph in blau

Das Ereignis für den Übergang zwischen dem *Client Task* und dem *Sender Task* auf *Plattform ECU 2* wird in Übereinstimmung mit den für die Simulation spezifizierten Anforderungen umgesetzt (s. Tabelle 7.7). Im konkreten Fall prüft hierbei der Client Task den Erhalt von mindestens drei Objektlisten, die als Ethernet Nachrichten ausgehend von Umfeldmodell 1 übertragen werden.

Rekonfigurationsszenario Durch Erweiterung des Wirkzusammenhangs 1 um die Softwarekomponente Umfeldmodell 2 soll ein rekonfigurierbarer Anteil analysiert werden. In Bild 7.35 ist der dazu entsprechende Wirkzusammenhang 2 modelliert.

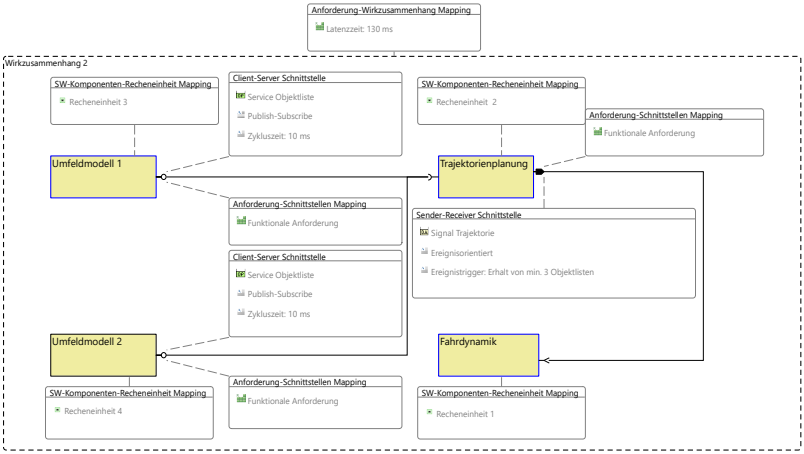

Bild 7.35: Wirkzusammenhang 2 mit zugeordneter Latenzzeitanforderung in PREEvision

Das Modell ist in Bezug auf die Partitionierung von Softwarekomponenten auf Recheneinheiten festgelegt. Im konkreten Fall ist die entsprechende Partitionierungsalternative wiederum durch Partitionierungsalternative 1 gegeben (s. Tabelle 7.6). Die Softwarekomponente Fahrdynamik hat weiterhin eine feste Partitionierung, welche nicht durch die Exploration errechnet wurde (s. Bild 7.27).

Spezifikation von Laufzeitverhalten Durch den rekonfigurierbaren Anteil im Rahmen von Umfeldmodell 2 kann sich ein Rekonfigurationsszenario während der Laufzeit ausbilden. Als Vorbedingung in diesem Fall wird festgelegt, dass eine Nutzungsbeziehung mit dem von Umfeldmodell 1 angebotenen Service zuerst stattfindet. In Bezug auf Anforderungen an das Rekonfigurationsszenario dient die folgende Übersicht (s. Tabelle 7.8).

Anforderung	Beschreibung
1. Verhalten von Umfeldmodell Service 1/2	Aktualisierung von Objektliste in 10 ms Zyklen
2. Verhalten von Trajektorienplanung	Prüfung, ob mindestens drei aktualisierte Objektlisten in 30 ms erfolgreich übertragen wurden und Fahrdynamik angesteuert werden kann
3. Auslösung von Rekonfigurationsereignis	Service von Umfeldmodell 1 verletzt zeitliche Anforderung an Aktualisierungen
4. Rekonfiguration zu Umfeldmodell 2	Subscription zu Umfeldmodell 2 durch Trajektorienplanung, Nutzung des entsprechenden Service und Ansteuerung und Ausführung der Fahrdynamik dauern max. 130 ms

Tabelle 7.8: Laufzeitanforderungen an Wirkzusammenhang 2

Dabei ist zu erkennen, dass die Anforderung an das Verhalten der Trajektorienplanung angepasst wurde und neben der erfolgreichen Übertragung von mindestens drei Objektlisten auch ein spezifisches Zeitintervall hierfür im Rahmen von 30 ms vorgeschrieben wird. Die Verletzung dieser Anforderung wird weitergehend als Auslöser für das Rekonfigurationsereignis herangezogen. Zur Modellierung der Laufzeitanforderungen ist gleichermaßen zu Wirkzusammenhang 1 ein Sequenzdiagramm eingeführt (s.

Bild 7.36). Dabei ist zunächst die Auslösung des Rekonfigurationsereignis dargestellt. Nachfolgend wird die Rekonfiguration durch den Versand einer *Subscription* von der Trajektorienplanung zu Umfeldmodell 2 eingeleitet (s. Bild 7.36).

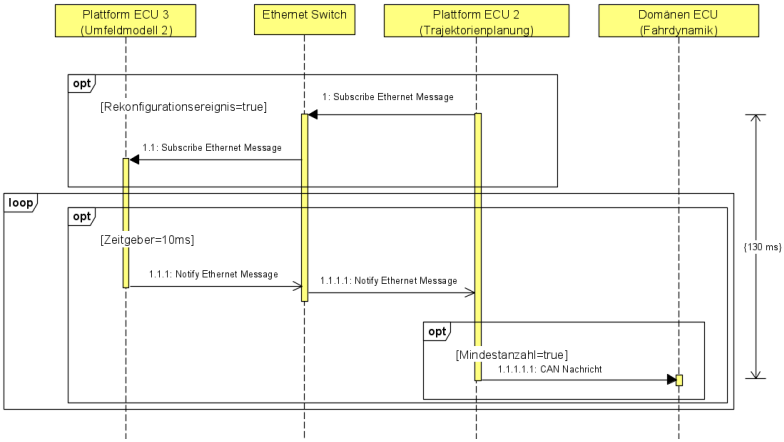

Bild 7.36: Laufzeitmodell des Wirkzusammenhangs 2 im Rahmen eines Rekonfigurationsszenarios

Für den Fortgang des Protokolls wird die nach der erfolgreichen Subscription ermöglichte Nutzung des Services von Umfeldmodell 2 betrachtet. Den Abschluss der Rekonfiguration von Umfeldmodell 1 zu Umfeldmodell 2 beschreibt die Ansteuerung und Ausführung der Fahrdynamik.

Entwickeltes Simulationsmodell Das nach der abgeschlossenen Modellierung entwickelte Simulationsmodell zur Laufzeitbewertung des Wirkzusammenhangs 2 in chronSIM ist in Bild 7.37 gezeigt. Bei der Simulation wird in Übereinstimmung mit dem modellierten Sequenzdiagramm die Rekonfiguration durch die Trajektorienplanung zu Umfeldmodell 2 im Hinblick auf die Latenzzeitanforderung von 130 ms bewertet. Zur Umsetzung dieses Ablaufs veröffentlichen sowohl das Steuergerät *Plattform ECU 1* als

auch das Steuergerät *Plattform ECU 3* zunächst einen funktional identischen Service zur zyklischen Bereitstellung einer Objektliste über einen Publish Task. Anschließend wird durch den Subscribe Task der Trajektorienplanung zuerst eine Kommunikationsbeziehung mit dem von Plattform ECU 1 angebotenen Service eingeleitet.

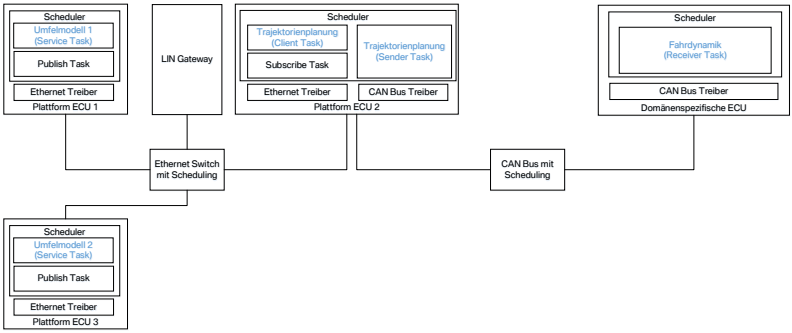

Bild 7.37: Hybrider Simulationsansatz für Wirkzusammenhang 2

Dies ist die Vorbedingung, um anschließend eine Rekonfiguration zu Umfeldmodell 2 zu analysieren und ermöglicht einen zyklischen Informationsfluss im Rahmen einer Aktualisierungszeit von 10 ms zwischen der Service Task des Umfeldmodells 1 und der Client Task der Trajektorienplanung.

Prüfbedingung für Eintritt von Rekonfigurationsereignis Speziell für das Rekonfigurationsszenario prüft der Client Task anschließend nicht nur den Erhalt von einer Mindestanzahl an Aktualisierungen von Umfeldmodell 1, bevor eine Trajektorie berechnet wird, sondern auch das zeitliche Verhalten dieser Aktualisierungen (s. Tabelle 7.8). Bei Verletzung der zeitlichen Bedingung (s. Bild 7.38) wird ein Rekonfigurationsereignis ausgelöst und eine Kommunikationsbeziehung mit dem von Umfeldmodell 2 angebotenen Service als Rekonfiguration initiiert.

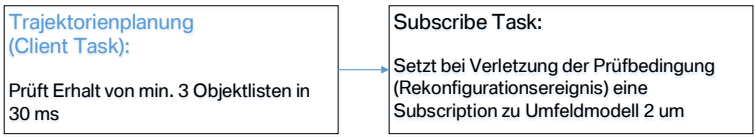

Bild 7.38: Prüfbedingung zur Auslösung der Rekonfiguration

Um während der Simulation die Verletzung der Prüfbedingung hervorzurufen, wird das Scheduling des Umfeldmodells 1 mit Verzögerungen belegt. Diese Verzögerungen sind zyklisch ausgeführt und resultieren darin, dass für drei Aktualisierungen immer ein größeres Zeitintervall als 30 ms benötigt wird.

Die Trajektorienplanung bewertet dies als nicht erfüllte Eingangsbedingung für ihre Funktionsausführung und löst das Rekonfigurationsereignis und somit die Rekonfiguration zu dem bereits veröffentlichten und funktional identischen Service von Umfeldmodell 2 aus. Dieser wird anschließend ohne Beeinträchtigung durch eine gesteuerte Verzögerung ausgeführt. Hierdurch erhält die Trajektorienplanung als Client im geforderten Zeitintervall eingangsseitig Aktualisierungen durch das Umfeldmodell 2 und kann ausgangsseitig die Fahrdynamik als abschließenden Anteil der Ereigniskette ansteuern.

Die sich während der Simulation ausbildende Ereigniskette mit der zu prüfenden Latenzzeitanforderung von 130 ms ist in Bild 7.39 gezeigt.

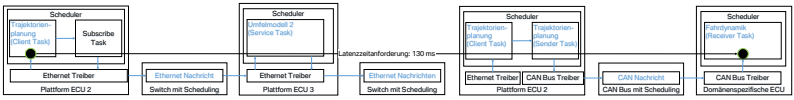

Bild 7.39: Ereigniskette für Wirkzusammenhang 2 als gerichteter Graph in blau

Den Beginn der Ereigniskette stellt der Client Task dar, welcher eine Verletzung des zeitlichen Verhaltens von Umfeldmodell 1 erkennt und durch Stimulation der Subscription Task eine Rekonfiguration zu Umfeldmodell 2 vornimmt.

Simulationsausführung

Für die Simulation wurden die Wirkzusammenhänge 1 und 2 (s. Bild 7.32 und 7.36) durch die entsprechenden Simulationsmodelle separat ausgeführt und somit die Bewertung einer verteilten Funktion mit und ohne rekonfigurierbare Anteile umgesetzt (s. Abschnitt 6.2.3).

Simulation pro Partitionierungsalternative Weitergehend wurde die aus den Lösungen der Exploration entstehende Varianz berücksichtigt.

Link	Lösung 1	Lösung 2	Lösung 3
ECU 1-Gateway	141.75	141.75	222.75
Gateway-ECU 1	141.75	141.75	222.75
ECU 2-Gateway	60.75	121.50	0.00
Gateway-ECU 2	60.75	121.50	0.00
ECU 3-Gateway	101.25	40.50	81.00
Gateway-ECU 3	101.25	40.50	81.00
ECU 1-ECU 3	1.20	0.00	0.00
ECU 3-ECU 1	1.20	0.00	0.00
ECU 1-ECU 2	150.30	151.50	150.30
ECU 2-ECU 1	0.30	1.50	0.30
ECU 3-ECU 2	150.30	150.30	150.30
ECU 2-ECU 3	0.30	0.30	0.30

Tabelle 7.9: Auslastung von Ethernet Links für Lösungen 1-3 in Kbit/s

Zur Darstellung dieser aus der unterschiedlichen Softwarepartitionierung erzeugten Varianz sind die ableitbaren, maximalen Auslastungsraten[11] einzelner Ethernet Links gezeigt (s. Tabelle 7.9). Durch die Betrachtung der Lösungen 1-3 müssen die zwei eingeführten Modelle zur Simulation von Wirkzusammenhang 1 und 2 (s. Bild 7.33 und 7.37) somit jeweils als *Modellvarianten 1-3* konfiguriert werden. Die Datenrate für die Ethernet Topologie wurde für alle sechs Modelle auf 100 Mbit/s festgelegt.

Simulationsergebnisse Die Ergebnisse der Laufzeitbewertung für die Wirkzusammenhänge 1 und 2 unter Berücksichtigung der Modellvarianten 1-3 sind in Tabelle 7.10 und 7.11 dargestellt.

Wirkzusammenhang	Modell	Erfolgreiche Durchläufe	Anforderung
1	Variante 1	100 %	130 ms
1	Variante 2	100 %	130 ms
1	Variante 3	100 %	130 ms

Tabelle 7.10: Ergebnisse der Laufzeitbewertung für eine Latenzzeitanforderung von 130 ms für Wirkzusammenhang 1

Wirkzusammenhang	Modell	Erfolgreiche Durchläufe	Anforderung
2	Variante 1	100 %	130 ms
2	Variante 2	100 %	130 ms
2	Variante 3	100 %	130 ms

Tabelle 7.11: Ergebnisse der Laufzeitbewertung für eine Latenzzeitanforderung von 130 ms für Wirkzusammenhang 2 unter Berücksichtigung eines Rekonfigurationsszenarios

[11] Die Berechnung der Auslastungsraten pro Lösung wurde manuell erreicht. Hierbei wurde für alle Client-Server Beziehungen zwischen Softwarepartitionen einer Lösung festgelegt, wie hoch die maximal erlaubte Übertragungsrate für die Service Discovery und die Service Nutzung sein darf. Anschließend wurden pro Link die entsprechenden Werte summiert.

Dabei stellen sich die drei Partitionierungsalternativen als indifferent zueinander heraus und können für eine mögliche Implementierung gleichermaßen betrachtet werden. Für jeden Simulationsdurchlauf wurde eine Simulationszeit von einer Sekunde festgelegt. Der Start jedes Durchlaufs ist dabei durch den Publish-Subscribe Mechanismus zwischen den für Wirkzusammenhang 1 und 2 relevanten Softwarekomponenten gegeben. Für den Mechanismus wird ein Jitter mit einer Taktschwankung von bis zu einer Sekunde festgelegt und pro Durchlauf durch den Simulationsengine von chronSIM zufällig festgelegt.

Hierdurch unterscheiden sich einzelne Simulationsdurchläufe, weil der Startzeitpunkt für die Ausführung des Publish-Subscribe Mechanismus nicht immer zum gleichen Zeitpunkt ausgelöst wird. Speziell bei der Anzahl von 500 Durchläufen wurde erkannt, dass sich die durch den Jitter verursachten Variationen wiederholten. Dieser Befund wurde als Indikator dafür herangezogen, dass die erzeugbare Varianz vollständig ausgereizt wurde. Entsprechend wurde die Anzahl an Simulationsdurchläufen nicht über den Wert 500 erhöht.

7.3 Handlungsbedarf und Potential

Die Werkzeugkette zur Umsetzung der Entwurfsmethodik für eine hybride Software- und Systemarchitektur wurde über ein Zusammenspiel aus Modellierungs- und Analyseschritten umgesetzt und in einem Forschungsprojekt für eine künftige E/E-Architektur eingesetzt. Speziell für das Werkzeug PREEvision zur Architekturmodellierung (s. Abschnitt 7.1.1) steht abschließend ein Handlungsbedarf als auch ein weitergehendes Potential für dessen Nutzung.

Handlungsbedarf

Die Integration der Metamodelle in das Modellierungswerkzeug wurde über die Abbildung auf grafische Beschreibungselemente in PREEvision vorgenommen (s. Tabelle A.1, A.2 und A.3). PREEvision kann dabei bis auf die im Rahmen des Metamodells für Architektursicht 3 geforderten Sequenzdiagramme für alle Metamodellelemente geeignete grafische Beschreibungselemente bereitstellen. Für die Modellierung der Sequenzdiagramme (s. Bild 7.32 und 7.36) wird jedoch die Einbindung eines externen UML Werkzeugs benötigt. Im Rahmen dieser Arbeit wird hierfür das Werkzeug Visual Paradigm [9] eingesetzt. Zur Vermeidung des entsprechenden Werkzeugbruchs empfiehlt sich die Integration des Metamodells für Sequenzdiagramme entlang der UML in zukünftige Versionen von PREEvision.

Weitergehende Nutzung

Durch PREEvision kann für die als Projektergebnis stehenden Architekturmodelle der Übergang zwischen Entwurf und Implementierung im Rahmen des V-Modells als Potential gehoben werden. Dies begründet sich durch die erreichte Erweiterung des Metamodells um die notwendigen Beschreibungsinhalte der Technologie SOME/IP (s. Tabelle 2.2). Hierdurch können Softwarekomponenten eines Service-orientierten Teilsystems (s.

Bild 7.9) im Rahmen der Entwurfsmethodik mit einem möglichen Implementierungsmodell für eine Service-orientierte Architektur verknüpft werden (s. Bild 7.40).

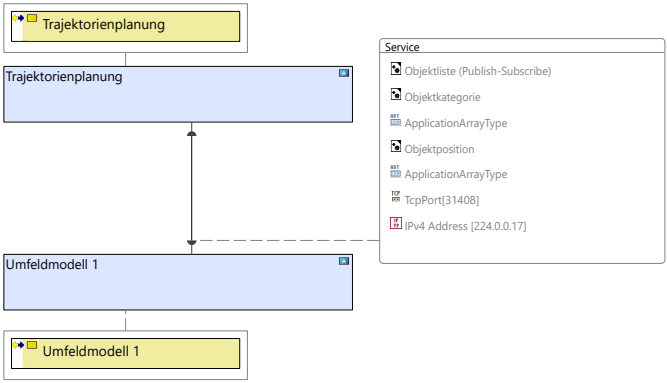

Bild 7.40: Mapping von Softwarekomponenten im Rahmen der Entwurfsmethodik (gelb) mit Implementierungsmodell (blau) für Technologie SOME/IP

Dies betrifft für die Entwicklung von Quellcode als funktionale Implementierung zunächst die Festlegung von Datenstrukturen. Für die Implementierung der Software als Teil eines Kommunikationsnetzwerks ist eine Zuweisung von Port- und IP Adressen zu betrachten.

Die weitergehende Verarbeitung des Implementierungsmodells wird durch ein zur textuellen Beschreibung der Softwarekomponente genutztes Austauschformat im Rahmen der AUTOSAR Standardisierung erreicht. Dieses kann von Generatoren eingelesen und als Eingangsgröße für die Entwicklung von Quellcode über generierte Quellcoderümpfe genutzt werden. Hierdurch ist eine Kopplung zwischen Architekturentwicklung über ein Entwurfs- und ein Implementierungsmodell von Softwarekomponenten und einem nachgelagerten Softwareentwicklungsprozess abschließend möglich (s. Bild 7.41).

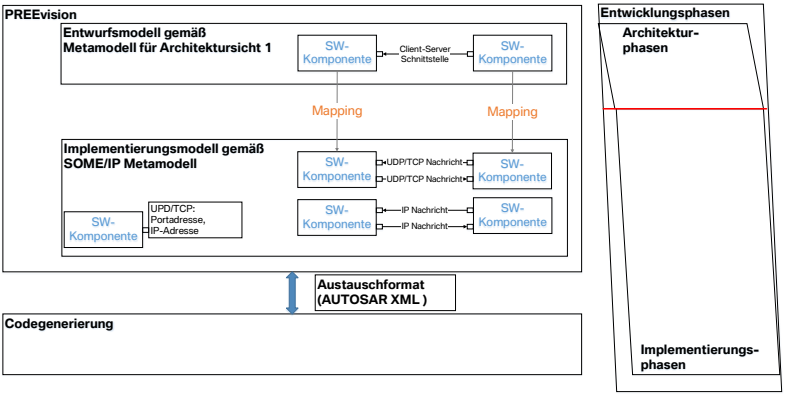

Bild 7.41: Nutzung von PREEvision für den Übergang zwischen Architektur und Implementierung

8 Zusammenfassung und Ausblick

Durch die weiter zunehmende Bedeutung von Fahrzeugsoftware spielt die abstrahierte Beschreibung von verteilten Funktionen eine wichtige Rolle, um ein System mit in Zukunft bis zu 300 Millionen Zeilen Quellcode zu verstehen (s. Abschnitt 1.1). Ein zentrales Modell für diese Beschreibung ist die Software- und Systemarchitektur, welche Informationsflüsse zwischen auf unterschiedlichen Steuergeräten verteilten Softwarekomponenten festlegt (s. Bild 3.3).

Als Alternative zum Signal-orientierten Stil für den Entwurf dieser Beziehungen stellt Service-Orientierung einen aktuellen Forschungsgegenstand dar (s. Tabelle 2.3), welcher Ergebnisse im Rahmen entsprechender Technologien in die Industrie transferiert (s. Tabelle 2.2).

Im Gegensatz zu bisherigen Forschungsschwerpunkten betrachtet diese Arbeit die Einbindung des Service-orientierten Kommunikationskonzepts in den Systementwurf. Hierdurch wird es ermöglicht Anteile einer Softwarearchitektur gemäß des Client-Server Ansatzes als Ergänzung zum bestehenden Sender-Receiver Ansatz zu entwerfen und über ein angepasstes Vorgehen für die Implementierung und Absicherung von Software entlang des V-Modells zu realisieren (s. Abschnitt 6.3.2).

Die Hauptmotivation für diesen angepassten Entwurf ist die Etablierung einer Softwarearchitektur, die Kommunikationsbeziehungen in einem logischen, von Hardware getrennten Netzwerk, ausbildet (s. Abschnitt 6.3.2).

Speziell bei Änderungen dieser Beziehungen sind ausschließlich Softwarekomponenten als Entwicklungsartefakte und nicht mehr Steuergeräte bzw. ihre Kommunikationsschnittstellen über Nachrichten betroffen (s. Bild 6.34).

Für die Einbindung des Ansatzes in das V-Modell wurde zunächst der Entwurf einer Software- und Systemarchitektur entlang des Signal-orientierten Stils mit den relevanten Methoden zur Entwurfsunterstützung als Stand der Wissenschaft und Technik begriffen (s. Abschnitt 3). Anschließend konnte das für diesen Entwurf entsprechende Modell mit Anpassungsbedarfen, die für die Abbildung eines Service-orientierten Kommunikationskonzepts notwendig sind, belegt werden (s. Abschnitt 4.2).

Als Konsequenz dieser Anpassungsbedarfe ergibt sich der Ansatz eines hybriden Modells (s. Abschnitt 5.2), welches über das Konzept einer Architektursicht strukturiert wird (s. Definition 10). Die drei dabei abgeleiteten Sichten können durch eine entsprechende Methodik als wesentliches Ergebnis dieser Arbeit in einen Zusammenhang gestellt werden (s. Kapitel 6) und wurden abschließend im Rahmen eines modellbasierten Vorgehens implementiert (s. Kapitel 7).

Ausblick

Das entwickelte hybride Entwurfsmodell gewinnt seinen Charakter aus der Berücksichtigung von Client-Server und Sender-Receiver Schnittstellen. Ein Verhalten dieser Schnittstellen während der Laufzeit ist im Rahmen der vorgestellten Methodik berücksichtigt (s. Abschnitt 6.2.3). Die Unterscheidung zur heutigen, statischen Festlegung von Kommunikation ist das Konzept der Service Discovery zur Laufzeitkonfiguration- und rekonfiguration von Kommunikationsbeziehungen.

Im Rahmen dieser Arbeit wurde das Service Discovery Konzept immer nach der Verfügbarkeit eines Kommunikationsnetzwerks und somit dem Aufstarten der entsprechenden Steuergeräte betrachtet. Anschließend wird die Annahme getroffen, dass ein Service genutzt wird, solange die Netzwerkverfügbarkeit nicht beeinträchtigt ist (s. Bild 6.20).

Unberücksichtigt ist die Frage, ob die übergeordnete Funktion, zu der ein Service beiträgt, während der vollständigen Zeitspanne zwischen Netzwerkverfügbarkeit und -abschaltung durch einen Kunden verwendet wird. Speziell über den Einsatz *maschineller Lernverfahren* ist es in Zukunft denkbar, den Kontext bei der Verwendung einer Funktion durch Fahrzeug- und Umgebungsparameter zu lernen und somit abzuleiten, unter welchen Bedingungen die für die Funktion relevanten Services tatsächlich gebraucht werden.

Durch Bereitstellung dieses Wissens kann sich eine Service-orientierte Architektur nicht nur abhängig von der Netzwerkverfügbarkeit konfigurieren, sondern auch in Abhängigkeit des Nutzungskontexts durch den Kunden. Ein abgeleitetes Potential daraus ist die Optimierung der Ressourcenauslastung, da Services, die aus Sicht des Kunden nicht gebraucht werden, auch nicht durch die Service Discovery aktiviert und von einem Steuergerät ausgeführt werden müssen. Ein erster Ansatz zur Ableitung dieses Kontextwissens durch ein *maschinelles Lernverfahren* wurde bereits im Rahmen dieser Arbeit erarbeitet [86] und kann als Grundlage für weitergehende Arbeiten dienen.

A Anhang

A.1 Metamodell für Architektursicht 1

Die Architektursicht 1 umfasst die Beschreibung für eine aus zwei Teilsystemen bestehende hybride Softwarearchitektur. Das entsprechende Metamodell strukturiert die hierfür notwendigen Beschreibungselemente und deren Beziehungen als Klassendiagramm (s. Bild A.1).

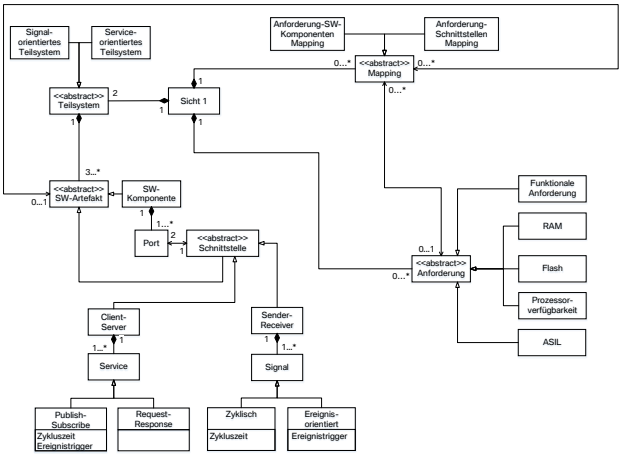

Bild A.1: Metamodell für Architektursicht 1: Hybride Softwarearchitektur

Die Integration des Metamodells für Architektursicht 1 in PREEvision ist über die Zuordnung von *Klassen* zu *grafischen Beschreibungsmitteln* des Werkzeugs umgesetzt.

Klasse(n)	Beschreibungsmittel
Sicht 1	Diagramm
Service-orientiertes Teilsystem	Komponentenpaket
Signal-orientiertes Teilsystem	Komponentenpaket
SW-Komponente	Block
Port	Serverport: Geöffneter Kreis Clientport: Geschlossener Kreis Senderport: Ausgefüllte Pfeilspitze Receiverport: Geöffnete Pfeilspitze
Schnittstelle, Client-Server	Verbindungslinie + Label
Service	Labelelement
Request-Response	Labelelement
Publish-Subscribe	Labelelement
Schnittstelle, Sender-Receiver	Verbindungslinie + Label
Signal	Labelelement
Zyklisch	Labelelement
Ereignisorientiert	Labelelement
Anforderung-SW-Komponenten Mapping	Label
RAM	Labelelement
Flash	Labelelement
Prozessorverfügbarkeit	Labelelement
ASIL	Labelelement
Anforderung-Schnittstellen Mapping	Label
Funktionale Anforderung	Labelelement

Tabelle A.1: Zuordnung: Klasse und grafisches Beschreibungsmittel für Architektursicht 1

A.2 Metamodell für Architektursicht 2

Die Architektursicht 2 umfasst die Partitionierung von Softwarekomponenten auf einer Topologie von Steuergeräten. Ein entsprechendes Metamodell formalisiert die notwendigen Beschreibungselemente und deren Beziehungen (s. Bild A.2).

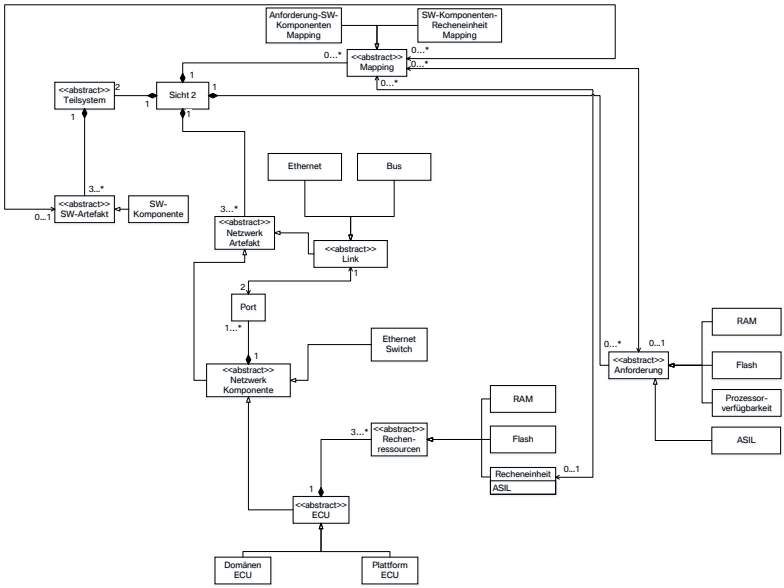

Bild A.2: Metamodell für Architektursicht 2: Partitionierung

Die Zuordnung von Klassen des Metamodells für Architektursicht 2 und grafischen Beschreibungsmitteln in PREEvision ist in Tabelle A.2 dargelegt.

Klasse	Beschreibungsmittel
Sicht 2	Diagramm
Domänen ECU	Block
Plattform ECU	Block
Switch	Block
Port	Quadrat
Link	Verbindungslinie + Label
Ethernet	Labelelement
Bus	Labelelement
Rechenressourcen	Label
RAM	Labelelement
Flash	Labelelement
Recheneinheit	Labelelement
SW-Komponenten-Recheneinheit Mapping	Label
SW-Komponente	Labelelement
Anforderung-SW-Komponenten Mapping	Label
SW-Komponente	Labelelement
RAM	Labelelement
Flash	Labelelement
Prozessorverfügbarkeit	Labelelement
ASIL	Labelelement

Tabelle A.2: Zuordnung: Klasse und grafisches Beschreibungsmittel für Architektursicht 2

A.3 Metamodell für Architektursicht 3

Die Architektursicht 3 umfasst die Beschreibung des Laufzeitverhaltens für partitionierte Softwarekomponenten. Ein entsprechendes Metamodell strukturiert die hierfür notwendigen Beschreibungselemente und deren Beziehungen (s. Bild A.3).

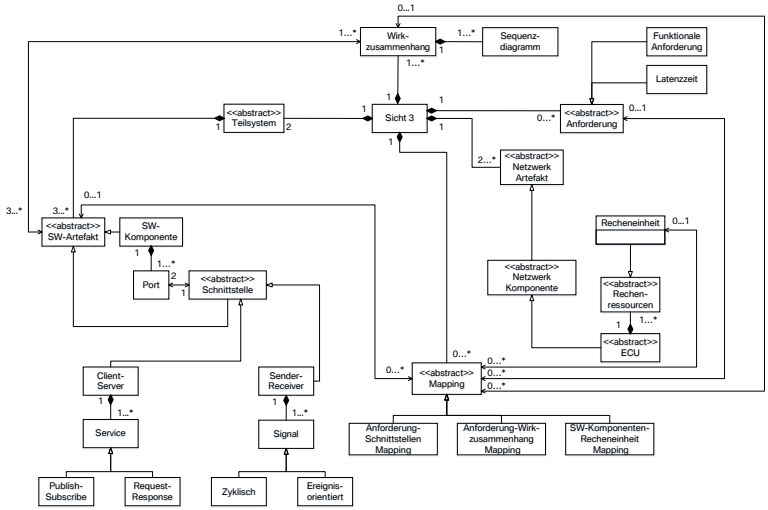

Bild A.3: Metamodell für Architektursicht 3: Laufzeitverhalten

Die Klassen sind wiederum in PREEvision über grafische Beschreibungsmittel umgesetzt (s. Tabelle A.3).

Klasse(n)	Beschreibungsmittel
Sicht 3	Diagramm
SW-Komponente	Block
Port	Serverport: Geöffneter Kreis Clientport: Geschlossener Kreis Senderport: Ausgefüllte Pfeilspitze Receiverport: Geöffnete Pfeilspitze
Schnittstelle, Client-Server	Verbindungslinie + Label
Service	Labelelement
Publish-Subscribe	Labelelement
Request-Response	Labelelement
Schnittstelle, Sender-Receiver	Verbindungslinie + Label
Signal	Labelelement
Zyklisch	Labelelement
Ereignisorientiert	Labelelement
Anforderung-Schnittstellen Mapping	Label
Funktionale Anforderung	Labelelement
SW-Komponenten-Recheneinheit Mapping	Label
Recheneinheit	Labelelement
Anforderung-Wirkzusammenhang Mapping	Label
Latenzzeit	Labelelement
Wirkzusammenhang	Markierung
Sequenzdiagramm	Grafische Notationen der UML

Tabelle A.3: Zuordnung: Klasse und grafisches Beschreibungsmittel für Architektursicht 3

A.4 Ansatz eines SMT Solvers

Ein SMT Solver stellt eine Möglichkeit bereit, um die Erfüllbarkeit eines formal beschriebenen Problems zu entscheiden.

Problemformalisierung und Funktionsweise

Ein SMT Problem ist formalisiert über logische Prädikate, die entweder wahr oder falsch sind. Die im Rahmen eines SMT Ausdrucks genutzten Variablen sind im Unterschied zum Prädikat nicht rein binär, sondern können in den Rahmen unterschiedlicher mathematischer Theorien [Seite 337] [36]) eingebettet sein. Der SMT Solver stellt für einen solchen Ausdruck (oder eine Menge solcher Ausdrücke) als Problem die Aussage bereit, ob eine Problemerfüllbarkeit gegeben ist (s. Bild A.4).

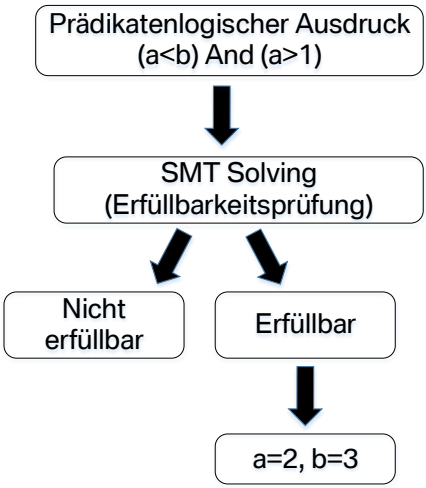

Bild A.4: Funktionsweise eines SMT Solvers

Sofern dies der Fall ist, wird weitergehend eine mögliche Problemlösung über die Bewertung der verwendeten Variablen bereitgestellt.

Pareto-Optimierung

Speziell für den im Rahmen dieser Arbeit genutzten Solver Z3 (s. Abschnitt 7.1.2) werden für das Lösen eines SMT Problems Funktionen zur Pareto-Optimierung unterstützt. Im Hinblick darauf wird für die Erfüllbarkeit eines Problems nicht nur eine mögliche Lösung ausgegeben, sondern eine Menge als Pareto-Front.

A.5 Austauschformate: PREEvision/chronSIM

Austauschformat 1

Das Austauschformat 1 beschreibt eine aus PREEvision generierte Excel Tabelle. Die Tabelle wird über einen Generator in der Programmiersprache Java aus PREEvision generiert und kann von einem Skript in der Programmiersprache Python in chronSIM eingelesen werden. Jede Softwarekomponente wird durch das Skript in chronSIM zunächst über einen Task repräsentiert, wodurch die Struktur der Softwarearchitektur in Bezug auf die Anzahl an Komponenten in chronSIM beibehalten wird. Für ein ausführbares Simulationsmodell können einzelne Tasks nach dem Import weiter unterteilt werden, weil nicht zwangsläufig eine Softwarekomponente durch genau einen Task repräsentiert wird (s. Abschnitt 7.2.3).

Aufbau der Tabelle

In der Tabelle sind Softwarekomponenten als *Items* beschrieben.

Item	Type	ECU	Core	Prio	Period	BCET	AVET	WCET
SW-C1	Basic	ECU1	1	1	50	10	15	20

Tabelle A.4: Austauschformat 1: Softwarekomponenten

Die weiteren Attribute beschreiben den *Type*[1] der *Task* und seine Partitionierung auf einer Recheneinheit eines Steuergeräts als *ECU* und *Core*. Zusätzlich kann eine Priorität und eine Zykluszeit in *ms* für den Task vergeben werden. Für eine mögliche Vergabe einer Ausführungszeit kann abschließend die Festlegung einer Best Case Execution Time (BCET), Average Execution Time (AVET) und einer Worst Case Execution Time (WCET) ebenfalls in *ms* erreicht werden.

Austauschformat 2

Das Austauschformat 2 beschreibt eine aus PREEvision generierte Excel Tabelle zur Beschreibung einer Ethernet Topologie. Die Tabelle ist wiederum über einen prototypischen Generator in der Programmiersprache Java umgesetzt und kann von einem Skript in der Programmiersprache Python in chronSIM eingelesen werden.

Aufbau der Tabelle

In der Tabelle sind Steuergeräte einer Ethernet Topologie aus PREEvision über die entsprechende Spalte *ECU* gekennzeichnet.

ECU	Controller	Src-MAC	Connection Target
ECU1	ctrlEth	46:24:76:60:00:01	Switch

Tabelle A.5: Austauschformat 2: Ethernet Topologie

Die Festlegung der Topologie wird weitergehend über einzelne Links festgelegt. Ein Link umfasst einen Controller, welcher standardmäßig einen Ethernet Controller *ctrlEth* als Quelle des Links beschreibt, die Quell Media Access Control (MAC) Adresse als *Src-MAC* und das Ziel des Links als *Connection Target*. Das Connection Target ist im Regelfall durch einen zentralen Ethernet Switch gegeben.

[1] Der Type kann als zyklisch (Type: Basic) oder ereignisorientiert deklariert sein.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] *Android Automotive*. https://source.android.com/devices/automotive, . – Letzter Zugriff: 2020-02-11

[2] *Digital Auto Report 2017*. https://www.strategyand.pwc.com/de/de/presse/2017/digital-auto-report.html, . – Letzter Zugriff: 2021-02-09

[3] *Disruptive modulare Architektur für agile, automatisierte Fahrzeugkonzepte*. https://www.unicaragil.de, . – Letzter Zugriff: 2020-05-01

[4] *Duden Online Wörterbuch*. https://www.duden.de/rechtschreibung/Topologie, . – Letzter Zugriff: 2018-06-01

[5] *European Processor Initiative (EPI)*. https://www.european-processor-initiative.eu, . – Letzter Zugriff: 2020-05-01

[6] *FAZ*. https://www.faz.net/aktuell/technik-motor/digital/tesla-app-smartphone-uebernimmt-die-kontrolle-ueber-das-auto-16691737.html, . – Letzter Zugriff: 2020-02-03

[7] *NXP Semiconductors*. https://www.nxp.com/docs/en/brochure/S32-Automotive-Processing-Platform.pdf, . – Letzter Zugriff: 2021-01-29

[8] *PREEvision*. https://vector.com/vi_preevision_de.html, . – Letzter Zugriff: 2018-10-29

[9] *Visual Paradigm*. https://www.visual-paradigm.com/, . – Letzter Zugriff: 2020-06-01

[10] *User Datagram Protocol*. RFC 768. https://rfc-editor.org/rfc/rfc768.txt. Version: August 1980

[11] *Transmission Control Protocol*. RFC 793. https://rfc-editor.org/rfc/rfc793.txt. Version: September 1981

[12] *FlexRay Communication System - Protocol Specification*. 2010

[13] *IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams*. 2010

[14] *IEEE Standard for Local and metropolitan area networks–Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP)*. 2010

[15] *LIN Specification Package, Revision 2.2A*. 2010

[16] *IEEE Standard for Local and metropolitan area networks– Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks– Corrigendum 1: Technical and Editorial Corrections*. 2013

[17] *IEEE Std 802.1BA-2011/Cor 1-2016 (Corrigendum to IEEE Std 802.1BA-2011): IEEE Standard for Local and metropolitan area networks– Audio Video Bridging (AVB) Systems– Corrigendum 1: Technical and Editorial Corrections*. 2016

[18] *AUTOSAR: Automotive Open System Architecture*. http://www.autosar.org, 2020

[19] AL-JAROODI, Jameela ; MOHAMED, Nader: Service-oriented middleware: A survey. In: *J. Network and Computer Applications*, 2012

[20] ALLIANCE, GENIVI: *GENIVI*. https://www.genivi.org/, . – Letzter Zugriff: 2020-02-11

[21] ANSSI, Saoussen ; ALBERS, Karsten ; DÖRFEL, Matthias ; GÉRARD, Sébastien: chronVAL/chronSIM: A Tool Suite for Timing Verification of Automotive Applications. In: *Embedded Real Time Software and Systems (ERTS2012)*, 2012

[22] BACH, J.: *Methoden und Ansätze für die Entwicklung und den Test prädiktiver Fahrzeugregelungsfunktionen*, Karlsruher Institut für Technologie, Diss., 2018

[23] BAGSCHIK, Gerrit ; MENZEL, Till ; RESCHKA, Andreas ; MAURER, Markus: Szenarien für Entwicklung, Absicherung und Test von automatisierten Fahrzeugen. In: *11. Workshop Fahrerassistenzsysteme und automatisiertes Fahren*, 2017

[24] BALZERT, Helmut: *Lehrbuch der Softwaretechnik: Bd. 2: Entwurf, Implementierung, Installation und Betrieb*. Heidelberg Berlin : Spektrum Akademischer Verlag, 2011

[25] BARTSCH, C.: *Modellierung und Simulation von IT-Dienstleistungsprozessen*. KIT Scientific Publ., 2010 https://books.google.de/books?id=gWisNh9PJu8C

[26] BECKER, Klaus ; FRTUNIKJ, Jelena ; FELSER, Meik ; FIEGE, Ludger ; BUCKL, Christian ; ROTHBAUER, Stefan ; ZHANG, Licong ; KLEIN, Cornel: RACE RTE: A Runtime Environment for Robust Fault-Tolerant Vehicle Functions. In: *3rd Workshop on Critical Automotive applications - Robustness & Safety (CARS)*, 2015

[27] BRANDT, Laura S.: *Architekturgesteuerte Elektrik/Elektronik Baukastenentwicklung im Automobil*, Technische Universität München, Dissertation, 2016

[28] BRAUN, Lisa: *Modellbasierte Design-Space-Exploration nichtfunktionaler Auslegungskriterien des Fahrzeugenergiebordnetzes*, Karlsruher Institut für Technologie (KIT), Diss., 2018

[29] BRAUN, Lisa ; SAX, Eric ; GAUTERIN, Frank: Abschlussbericht: Experteninterview zur Anforderungsanalyse heutiger und zukünftiger E/E Architekturen im Kraftfahrzeug, Karlsruher Institut für Technologie (KIT), 2016

[30] BROY, Julian: *Modellbasierte Entwicklung und Optimierung flexibler zeitgesteuerter Architekturen im Fahrzeugserienbereich*, KIT, Karlsruhe, Diss., 2010

[31] CHOKSHI, D. B. ; BHADURI, P.: Modeling Fixed Priority Non-Preemptive Scheduling with Real-Time Calculus. In: *14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2008

[32] CHRISTENSEN, Henrik B. ; HANSEN, Klaus M. ; KYNG, Morten ; MANIKAS, Konstantinos: Analysis and design of software ecosystem architectures Towards the 4S telemedicine ecosystem. In: *Information and Software Technology*, 2014

[33] COOPERATION, MOST: *MOST Specification*. 2010

[34] CUENOT, Philippe ; FREY, Patrick ; JOHANSSON, Rolf ; LÖNN, Henrik ; PAPADOPOULOS, Yiannis ; REISER, Mark-Oliver ; SANDBERG, Anders ; SERVAT, David ; TAVAKOLI KOLAGARI, Ramin ; TÖRNGREN, Martin ; WEBER, Matthias: 11 The EAST-ADL Architecture Description Language for Automotive Embedded Software. In: GIESE, Holger (Hrsg.) ; KARSAI, Gabor (Hrsg.) ; LEE, Edward (Hrsg.) ; RUMPE, Bernhard (Hrsg.) ; SCHÄTZ, Bernhard (Hrsg.): *Model-Based Engineering of Embedded Real-Time Systems: International Dagstuhl Workshop, Dagstuhl Castle, Germany, November 4-9, 2007. Revised Selected Papers*, Springer Berlin Heidelberg, 2010

[35] DANIEL JOSEF GEBAUER: *Ein modellbasiertes, graphisch notiertes, integriertes Verfahren zur Bewertung und zum Vergleich von Elektrik/Elektronik-Architekturen*, Karlsruher Insitut für Technologie, Dissertation, 2016

[36] DE MOURA, Leonardo ; BJØRNER, Nikolaj: Z3: An Efficient SMT Solver. In: *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer-Verlag, 2008

[37] DEFENSE SYSTEMS MANAGEMENT COLLEGE PRESS, Virginia Fort B. Fort Belvoir (Hrsg.): *Systems Engineering Fundamentals*. 2001 https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-885j-aircraft-systems-engineering-fall-2005/readings/sefguide_01_01.pdf

[38] DIETMAYER, Klaus ; MAURER, Markus (Hrsg.) ; GERDES, J. C. (Hrsg.) ; LENZ, Barbara (Hrsg.) ; WINNER, Hermann (Hrsg.): *Prädiktion von maschineller Wahrnehmungsleistung beim automatisierten Fahren*. Springer Berlin Heidelberg, 2015

[39] DORRER, Claus: Automated driving at BMW – Solutions for today and tomorrow. In: *18. Internationales Stuttgarter Symposium*, Springer Fachmedien Wiesbaden, 2018

[40] EBERT, Christof ; FAVARO, John M.: Automotive Software. In: *IEEE Software*, 2017

[41] FISCHER, Till: *Eine Technologie für das durchgängige und automatisierte Testen eingebetteter Software*, Karlsruher Institut für Technologie (KIT), Diss., 2017

[42] FREESS, Jascha: *Modelle zur Beschreibung und Evaluierung von Architekturkonzepten der Elektrik und Elektronik in Kraftfahrzeugen*, Universität Karlsruhe, Karlsruhe, Diss., 2007

[43] GÖB, Andreas: *SOA und Softwarequalität*. München, Technische Universität München, Dissertation, 2013

[44] GÉRARD, Sébastien ; DUMOULIN, Cédric ; TESSIER, Patrick ; SELIC, Bran: Papyrus: A UML2 Tool for Domain-specific Language Modeling. In: *Proceedings of the 2007 International Dagstuhl Conference on Model-based Engineering of Embedded Real-time Systems*, Springer-Verlag, 2007

[45] GERHARD PAHL, Wolfgang B.: *Konstruktionslehre: Grundlagen erfolgreicher Produktentwicklung Methoden und Anwendung*. Springer Berlin Heidelberg, 2007

[46] GLASS, Michael ; HERRSCHER, Daniel ; PIASTOWSKI, Martin ; MEIER, Herbert ; SCHOO, Peter: SEIS - Sicherheit in Eingebetteten IP-Basierten Systemen. In: *ATZ - Automobiltechnische Zeitschrift*, 2010

[47] GMBH, Robert B.: *Controller Area Network Specification (CAN) - Version 2.0*. 1991

[48] GROSSE-ROHDE, Martin ; EURINGER, Simon ; KLEINOD, Ekkart ; MANN, Stefan: Grobentwurf des VEIA-Referenzprozesses, Fraunhofer Institut Software- und Systemtechnik, 2007

[49] GRÖNNIGER, Hans: *Formale Analyse eines automotive Bussystems mit SymTA/S auf der Grundlage von K-Matrizen*. Diplomarbeit, Institut für Datentechnik IDA, TUBS, 2005

[50] GRUHN, Volker ; PIEPER, Daniel ; RÖTTGERS, Carsten: *MDA: Effektives Software-Engineering mit UML 2 und Eclipse*. Springer Berlin Heidelberg New York, 2006

[51] GRUYER, Dominique ; MAGNIER, Valentin ; HAMDI, Karima ; CLAUSSMANN, Laurene ; ORFILA, Olivier ; RAKOTONIRAINY, Andry: Perception, information processing and modeling: Critical stages

for autonomous driving applications. In: *Annual Reviews in Control*, 2017

[52] HAGIESCU, A. ; BORDOLOI, U. D. ; CHAKRABORTY, S. ; SAMPATH, P. ; GANESAN, P. V. V. ; RAMESH, S.: Performance Analysis of FlexRay-based ECU Networks. In: *44th ACM/IEEE Design Automation Conference*, 2007

[53] HAMMERSCHALL, Ulrike: *Flexible Methodenintegration in anpassbare Vorgehensmodelle*, Technical University Munich, Germany, Diss., 2008. http://mediatum2.ub.tum.de/doc/645115/document.pdf

[54] HEDTSTÜCK, Ulrich: *Simulation diskreter Prozesse*. Springer Vieweg, 2013

[55] HEISSING, Bernd (Hrsg.) ; ERSOY, Metin (Hrsg.): *Systeme im Fahrwerk*. Wiesbaden : Vieweg+Teubner, 2008

[56] HENRIK, Lönn: *Model-based Analysis & Engineering of Novel Architectures for Dependable Electric Vehicles*. https://egvi.eu/wp-content/uploads/2013/10/1_4-MAENAD.pdf. Version: July 2018

[57] HEYN, Markus: *Methodik zur schnittstellenorientierten Gestaltung von Entwicklungskooperationen; Als Ms. gedr.* Aachen, Diss., 1999. http://publications.rwth-aachen.de/record/58175. – Zugl.: Aachen, Techn. Hochsch., Diss., 1998

[58] HILLENBRAND, Martin: *Funktionale Sicherheit nach ISO 26262 in der Konzeptphase der Entwicklung von Elektrik/Elektronik Architekturen von Fahrzeugen*, KIT, Karlsruhe, Diss., 2012

[59] ISO: *ISO 26262: Road vehicles – Functional safety*. 2011

[60] ISO/IEC/IEEE: Systems and software engineering – Architecture description, 2011

[61] JAENSCH, M.: *Modulorientiertes Produktlinien Engineering für den modellbasierten Elektrik/Elektronik-Architekturentwurf*, Karlsruher Institut für Technologie, Diss., 2014

[62] JOHANNES MATHEIS: *Abstraktionsebenenübergreifende Darstellung von Elektrik/Elektronik-Architekturen in Kraftfahrzeugen zur Ableitung von Sicherheitszielen nach ISO 26262 von Sicherheitszielen nach ISO 26262*. Karlsruhe, Karlsruher Insitut für Technologie, Diss., 2009

[63] JOHNSON, Neil: *Agile hardware development ? nonsense or necessity?* `https://www.eetimes.com/agile-hardware-development-nonsense-or-necessity/`. – Letzter Zugriff: 2021-02-10

[64] KÄBISCH, Sebastian: *Resource Optimization of SOA-Technologies in Embedded Networks*, Universität Passau, Diss., 2014

[65] KLEIN, T.: *Agiles Engineering im Maschinen- und Anlagenbau*. Utz Verlag GmbH, 2016 (Forschungsberichte IWB). `https://books.google.de/books?id=z0hJDQAAQBAJ`

[66] KOLLMANN, Steffen ; POLLEX, Victor ; SLOMKA, Frank: Global Best-Case Response Time for Improving the Worst-Case Response Times in Distributed Real-Time Systems. In: *DIPES/BICC*, Springer, 2010

[67] KOLLMANN, Steffen ; POLLEX, Victor ; SLOMKA, Frank: Holistic Real-Time Analysis with an Expressive Event Model. In: *MBMV*, 2010

[68] KREUTZ, Diego ; RAMOS, Fernando M. V. ; VERÍSSIMO, Paulo ; ROTHENBERG, Christian E. ; AZODOLMOLKY, Siamak ; UHLIG, Steve: Software-Defined Networking: A Comprehensive Survey. In: *Proceedings of the IEEE*, 2015

[69] KUGELE, Stefan: *Model-Based Development of Software-intensive Automotive Systems*, Technische Universität München, Dissertation, 2012

[70] KUGELE, Stefan ; OBERGFELL, Philipp ; BROY, Manfred ; CREIGHTON, Oliver ; TRAUB, Matthias ; HOPFENSITZ, Wolfgang: On Service-Orientation for Automotive Software. In: *International Conference on Software Architecture, ICSA*, 2017

[71] LAUFENBERG, L.: *Methodik zur integrierten Projektgestaltung für die situative Umsetzung des simultaneous engineering*. Shaker, 1996 (Berichte aus der Produktionstechnik)

[72] LEE, Jeong-Hwan ; HWANG, Hyun Y. ; HAN, Tae M. ; AHN, Yong H.: A Study on Signal Group Processing of AUTOSAR COM Module. In: *Journal of Physics: Conference Series*, 2013

[73] LEÓN, F.P. ; KIENCKE, U.: *Ereignisdiskrete Systeme: Modellierung und Steuerung verteilter Systeme*. De Gruyter, 2009 `https://books.google.de/books?id=M8TnBQAAQBAJ`

[74] LIU, C. L. ; LAYLAND, James W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. In: *J. ACM*, ACM, 1973

[75] LUNGU, Mircea ; LANZA, Michele ; GIRBA, Tudor ; ROBBES, Romain: The Small Project Observatory: Visualizing software ecosystems. In: *Science of Computer Programming*, 2010. – Experimental Software and Toolkits (EST 3): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT 2008)

[76] MARIN PERIANU, Raluca ; HARTEL, Pieter H. ; SCHOLTEN, Johan: *A Classification of Service Discovery Protocols*. Netherlands : Centre for Telematics and Information Technology (CTIT), 2005 (CTIT Technical Report Series)

[77] MARTINS, Joaquim R. R. A. ; LAMBE, Andrew B.: Multidisciplinary design optimization: A Survey of architectures. In: *AIAA Journal*, 2013

[78] MARWEDEL, P.: *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*. Springer International Publishing, 2021

[79] MERNIK, Marjan ; HEERING, Jan ; SLOANE, Anthony M.: When and How to Develop Domain-Specific Languages. In: *Association for Computing Machinery*, 2005

[80] MESSNARZ, Richard ; KREINER, Christian ; BACHMANN, Ovi ; RIEL, Andreas ; DUSSA-ZIEGER, Klaudia ; NEVALAINEN, Risto ; TICHKIEWITCH, Serge: Implementing Functional Safety Standards - Experiences from the Trials about Required Knowledge and Competencies (SafEUr). In: *Communications in Computer and Information Science*. Germany : Springer Verlag, 2013

[81] NEBEL, M.: *Formale Grundlagen der Programmierung*. Vieweg+Teubner Verlag, 2012 (Studienbücher Informatik). https://books.google.de/books?id=1ZbNygAACAAJ

[82] NEUHAUSEN, Jörn: *Methodik zur Gestaltung modularer Produktionssysteme für Unternehmen der Serienproduktion*. Aachen, Diss., 2002. http://publications.rwth-aachen.de/record/58788. – Aachen, Techn. Hochsch., Diss., 2002

[83] OBERGFELL, P. ; KUGELE, S. ; SAX, E.: Model-Based Resource Analysis and Synthesis of Service-Oriented Automotive Software Architectures. In: *ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2019

[84] OBERGFELL, P. ; OSZWALD, F. ; TRAUB, M. ; SAX, E.: Viewpoint-Based Methodology for Adaption of Automotive E/E-Architectures.

In: *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*, 2018

[85] OBERGFELL, Philipp ; KUGELE, Stefan ; SEGLER, Christoph ; KNOLL, Alois ; SAX, Eric: Continuous Software Engineering of Innovative Automotive Functions: An Industrial Perspective. In: *IEEE International Conference on Software Architecture Companion, ICSA Companion*, 2019

[86] OBERGFELL, Philipp ; SEGLER, Christoph ; SAX, Eric ; KNOLL, Alois: Synchronization between Run-Time and Design-Time View of Context-Aware Automotive System Architectures. In: *IEEE International Systems Engineering Symposium (ISSE)*, IEEE, 2018

[87] OMG: *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*. 2011

[88] OMG (Hrsg.): *Data Distribution Service (DDS)*. 1.4. OMG, 2015. `http://www.omg.org/spec/DDS/1.4/`

[89] OMG: *OMG Systems Modeling Language (OMG SysML), Version 1.4*. `https://www.omg.org/spec/SysML/1.4/`. Version: 2015

[90] OMG: *Meta Object Facility*. `http://www.omg.org/spec/MOF/`. Version: 2016

[91] OSZWALD, Florian ; OBERGFELL, Philipp ; LIU, Bo B. ; VICTOR PAZMINO ; BECKER, Juergen: Model-Based Design of Service-Oriented Architectures for Reliable Dynamic Reconfiguration, 2020 (SAE technical paper series)

[92] PATTERSON, David A. ; SEQUIN, Carlo H.: Risc i: A reduced instruction set vlsi computer. In: *In ISCA 81: Proceedings of the 8th annual symposium on Computer Architecture*, IEEE Computer Society Press, 1981

[93] PROJECT, Eclipse M.: *EAST-ADL Open Tool Platform*. https://www.eclipse.org/eatop/. – Letzter Zugriff: 2018-9-05

[94] REIF, K.: *Automobilelektronik: Eine Einführung für Ingenieure*. Springer Fachmedien Wiesbaden, 2014 (ATZ/MTZ-Fachbuch)

[95] REINHARDT, D. ; DANNEBAUM, U. ; SCHEFFER, M. ; TRAUB, M.: High Performance Processor Architecture for Automotive Large Scaled Integrated Systems within the European Processor Initiative Research Project, 2019

[96] RICHTER, Kai: *Compositional Scheduling Analysis Using Standard Event Models*, Diss., Dec 2004. https://publikationsserver.tu-braunschweig.de/receive/dbbs_mods_00001765

[97] SAGSTETTER, Florian: *Schedule Synthesis for Time-Triggered Automotive Architectures*. München, Technische Universität München, Dissertation, 2016

[98] SAX, Eric.: *Automatisiertes Testen Eingebetteter Systeme in der Automobilindustrie*. Hanser, Carl, 2008

[99] SCHÄUFFELE, J. ; ZURAWKA, T.: *Automotive Software Engineering: Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen*. Springer Fachmedien Wiesbaden, 2016 (ATZ/MTZ-Fachbuch). https://books.google.de/books?id=-9VgvgAACAAJ

[100] SCHWABER, Ken ; SUTHERLAND, Jeff: Der Scrum Guide, 2014

[101] SIEMERS: *Handbuch Embedded Systems Engineering V 0.61a*. TU Clausthal, FH Nordhausen, 2012

[102] STACHOWIAK, H.: *Allgemeine Modelltheorie*. Springer-Verlag, 1973 https://books.google.de/books?id=DK-EAAAAIAAJ

[103] STAHL, Thomas ; VOELTER, Markus ; CZARNECKI, Krzysztof: *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006

[104] STARON, Miroslaw: *Automotive software architectures: An introduction*. Springer International Publishing, 2017

[105] STEUSLOFF, H.: Verteilte Echtzeitsysteme. In: HOLLECZEK, Peter (Hrsg.) ; VOGEL-HEUSER, Birgit (Hrsg.): *Verteilte Echtzeitsysteme*. Springer, 2003 (Informatik aktuell)

[106] STREICHERT, T. ; TRAUB, M.: *Elektrik/Elektronik-Architekturen im Kraftfahrzeug: Modellierung und Bewertung von Echtzeitsystemen*. Springer Berlin Heidelberg, 2012 (VDI-Buch)

[107] THIELE, L. ; CHAKRABORTY, S. ; NAEDELE, M.: Real-time calculus for scheduling hard real-time systems. In: *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353)*, 2000

[108] TIETZE, Christoph Ulrich und S. Ulrich und Schenk: *Halbleiter Schaltungstechnik: 11. Auflage*. Springer, 1999

[109] TRAUB, M.: *Durchgängige Timing-Bewertung von Vernetzungsarchitekturen und Gateway-Systemen im Kraftfahrzeug*, Karlsruher Institut für Technologie, Diss., 2010

[110] TRAUB, M. ; VÖGEL, H. ; SAX, E. ; STREICHERT, T. ; HÄRRI, J.: Digitalization in automotive and industrial systems. In: *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018

[111] V., DIN Deutsches I. e.: *DIN 44300-1 - Informationsverarbeitung - Begriffe - Allgemeine Begriffe*. Beuth, 1985

[112] VOGEL, Oliver ; ARNOLD, Ingo ; CHUGHTAI, Arif ; IHLER, Edmund ; KEHRER, Timo ; MEHLIG, Uwe ; ZDUN, Uwe: *Software-Architektur: Grundlagen – Konzepte – Praxis*. Springer Spektrum, 2005

[113] VÖLKER, Lars: *Scalable service-Oriented MiddlewarE over IP*. `http://some-ip.com/`. – Letzter Zugriff: 2020-05-15

[114] WEBER, J.: *Automotive Development Processes: Processes for Successful Customer Oriented Vehicle Development*. Springer Berlin Heidelberg, 2009

[115] WECKEMANN, Kay: *Domänenübergreifende Anwendungskommunikation im IP-basierten Fahrzeugbordnetz*, Universität München, Diss., 2014

[116] WINNER, Hermann ; WACHENFELD, Walther ; MAURER, Markus (Hrsg.) ; GERDES, J. C. (Hrsg.) ; LENZ, Barbara (Hrsg.) ; WINNER, Hermann (Hrsg.): *Auswirkungen des autonomen Fahrens auf das Fahrzeugkonzept*. Springer Berlin Heidelberg, 2015

[117] WÖRN, H.: *Echtzeitsysteme: Grundlagen, Funktionsweisen, Anwendungen*. Springer Berlin Heidelberg, 2006 `https://books.google.de/books?id=llghBAAAQBAJ`

[118] ZIMMERMANN, H.: OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. In: *IEEE Transactions on Communications*, 1980