STEINBUCH SERIES ON ADVANCES
IN INFORMATION TECHNOLOGY

11

FLORIAN OSZWALD

Dynamische Rekonfigurationsmethodik für zuverlässige, echtzeitfähige Eingebettete Systeme in Automotive

KIT Scientific Publishing

Florian Oszwald

Dynamische Rekonfigurationsmethodik für zuverlässige, echtzeitfähige Eingebettete Systeme in Automotive

Band 11
Steinbuch Series on Advances in Information Technology

Karlsruher Institut für Technologie (KIT)
Institut für Technik der Informationsverarbeitung

Dynamische Rekonfigurationsmethodik für zuverlässige, echtzeitfähige Eingebettete Systeme in Automotive

von
Florian Oszwald

KIT Scientific Publishing

Karlsruher Institut für Technologie
Institut für Technik der Informationsverarbeitung

Dynamische Rekonfigurationsmethodik für zuverlässige, echtzeitfähige Eingebettete Systeme in Automotive

Zur Erlangung des akademischen Grades eines Doktor-Ingenieurs von der KIT-Fakultät für Elektrotechnik und Informationstechnik des Karlsruher Instituts für Technologie (KIT) genehmigte Dissertation

von Dipl.-Ing. Florian Oszwald

Tag der mündlichen Prüfung: 29. Juli 2021
Referent: Prof. Dr.-Ing. Dr. h.c. Jürgen Becker
Korreferent: Prof. Dr. sc.techn. Andreas Herkersdorf

Impressum

Karlsruher Institut für Technologie (KIT)
KIT Scientific Publishing
Straße am Forum 2
D-76131 Karlsruhe

KIT Scientific Publishing is a registered trademark
of Karlsruhe Institute of Technology.
Reprint using the book cover is not allowed.

www.ksp.kit.edu

Print on Demand 2021 – Gedruckt auf FSC-zertifiziertem Papier

ISSN 2191-4737
ISBN 978-3-7315-1102-1
DOI 10.5445/KSP/1000132321

Kurzfassung

Die Elektrik/Elektronik(E/E)-Architektur in Kraftfahrzeugen ist eines der prägenden Merkmale in der Automobilindustrie. Aufgrund der historischen Entwicklung von Fahrzeugen ist sie ein vorrangig gewachsener Anteil in der intensiv durch den Maschinenbau geprägten Industrie. Die Bedeutung der E/E-Architektur wird jedoch zukünftig noch stärker wachsen, da aufgrund der fortschreitenden Digitalisierung sich die Anforderungen der Kunden an die Fahrzeuge bezüglich Ihrer Eigenschaften weiter ändern werden. Dafür muss die E/E-Architektur vorbereitet sein, um auf diese Änderungen flexibel reagieren zu können.

Besonders die Entwicklung im Bereich des autonomen Fahrens hat dabei einen enorm verändernden Einfluss. Der Paradigmenwechsel hin zu Funktionen mit Anforderung an ein fail-operational-Verhalten, hervorgerufen durch den Übergang von assistierten oder teilassistierten Fahrfunktionen zu voll- und hochautomatisierten Fahrfunktionen, stellt hohe Anforderungen an kommende E/E-Architekturen.

Heutige Lösungen für diese Art von Funktionen sind häufig kostenintensiv, bringen zusätzliches Gewicht ein, benötigen mehr Bauraum und erhöhen zudem die Komplexität der ohnehin schon sehr umfangreichen E/E-Architektur. Zudem sind die derzeitigen Mechanismen immer funktionsspezifisch entwickelt und nicht für komplexe Funktionen anwendbar. In dieser Arbeit wird eine modulare, parametrierbare Methodik- und Konzeptbibliothek erarbeitet, die es ermöglicht, zukünftige Funktionen umzusetzen, die Anforderungen an ein Fail-Operational-Verhalten stellen.

Der Fokus bei der Betrachtung liegt auf Funktionen innerhalb eines Fahrzeugs. Fahrzeuggrenzen überschreitende Funktionen, die es zukünftig vermehrt geben wird, werden dabei bereits antizipiert.

Eingeordnet wird die erarbeitete Bibliothek in den Standardentwicklungsprozess der Automobilindustrie. Dynamische Rekonfiguration wird in dieser Dissertation erstmalig in diesem Kontext gesamthaft angedacht und an einem prototypischen Aufbau werden

beispielhaft die grundlegenden Konzepte und Methoden herausgearbeitet, angewendet und umgesetzt. Diese Arbeit spannt den Bogen, angefangen bei der Analyse über den Stand der Technik und der Identifikation der Anforderungen über den Einsatz zukunftsweisender Prozessortechnologien, den Anforderungsvergleich und die Bewertung unterschiedlicher Middleware-Protokolle bis hin zur Berücksichtigung der Trade-offs von zentralisierten und dezentralisierten Architekturansätzen. Dabei kommt gezielt eine abstrahierte, ebenenübergreifende Rekonfiguration, die Verwendung einer serviceorientierten Architektur sowie die modellbasierte Entwicklung und Evaluierung des Gesamtsystems zum Einsatz.

In dieser Dissertation wurde erstmalig eine serviceorientierte Architektur basierend auf heterogenen Multicore-Plattformen erarbeitet, die mit Hilfe eines neuartigen Cross-Layer-Rekonfigurationsansatzes technische Fail-Operational-Pattern für eine Rekonfiguration sowohl auf HW- als auch SW-Ebene miteinander verbindet und für komplexe Funktionen bereitstellt. Die erarbeiteten Ansätze und Methoden wurden prototypisch aufgebaut, in einen Fahrzeugsimulator integriert und die Messergebnisse den Fail-Operational-Anforderungen gegenübergestellt mit dem Ergebnis, dass die Zielsetzung erfüllt wurde.

Die Arbeit hat auf zahlreichen Konferenzen das Interesse der Automobilbranche geweckt, sowohl bei Automobilherstellern als auch der gesamten Zuliefererkette. Des Weiteren wurden sieben Patentanträge gestellt und zum größten Teil bereits erteilt. Die erarbeiteten Konzepte, Methoden und Techniken werden bei der BMW Group weiterentwickelt.

Abstract

The E/E-architecture of motor vehicles is one of the defining features of the automotive industry. Due to the historical development of vehicles, it is a primarily growing share in the intensively influenced industry by mechanical engineering. However, the importance of the E/E-architecture will grow even more in the future, as the demands of customers on the vehicles regarding their characteristics will continue to change due to the advancing digitalization. To do this, you need to be prepared to respond flexibly to these changes.

The development in the field of autonomous driving in particular has a hugely changing influence. The paradigm shift to functions requiring fail-operational behavior, caused by the transition from assisted or semi-assisted driving functions to fully automated and highly automated driving functions, places high demands on upcoming E/E-architectures.

Today's solutions for this type of feature are often costly, add extra weight, require more space, and increase the complexity of the already very large E/E-architecture. In addition, they are always functionally developed and not applicable to complex functions. In this work, a modular, parameterizable methodological and concept library is developed, which makes it possible to implement future functions that place the requirements for fail-operational behavior.

The focus of the viewing is on functions within a vehicle. Vehicle-border functions, which will increasingly exist in the future, are already anticipated.

The developed library is integrated into the standard development process of the automotive industry. Dynamic reconfiguration is considered for the first time in this context and the basic concepts and methods are elaborated, applied, and implemented as an example in a prototypical structure. This work spans the spectrum, from analysis to state-of-the-art and requirements identification, to the use of forward-looking processor

technologies, requirement comparison and evaluation of different middleware protocols, to the consideration of the trade-offs of centralized and decentralized architectural approaches. In this context, an abstracted, cross-level reconfiguration, the use of a service-oriented architecture, and the model-based development and evaluation of the overall system are specifically employed.

In this dissertation, a service-oriented architecture based on heterogeneous multiprocessor platforms was developed for the first time, which uses a novel cross-layer reconfiguration approach to combine technical fail-operational patterns for reconfiguration at both the HW and SW levels and provide them for complex functions. The approaches and methods developed were prototyped, integrated into a vehicle simulator and the measurement results were compared with the fail-operational requirements with the result that the objective was met.

The work has aroused the interest of the automotive industry at numerous conferences, both among car manufacturers and the entire supply chain. In addition, seven patent applications have been submitted and most of them have already been granted. The concepts, methods and techniques developed will be further developed at the BMW Group.

Vorwort

Die vorliegende Dissertation entstand während meiner Tätigkeit als Mitarbeiter der BMW Group im Bereich der Forschung, neue Technologien und Innovationen. Das Institut für Technik der Informationsverarbeitung (ITIV) des Karlsruher Instituts für Technologie (KIT) betreute diese Arbeit mit ihrer wissenschaftlichen Kompetenz. Geprägt war die spannende Zeit in der ich nebenberuflich diese Promotion erstellt habe von einem umfangreichen und facettenreichen Qualifizierungsprozess und Wissensaufbau: Mir gelang es in dieser Phase, viele wertvolle Erfahrungen zu sammeln, Einblicke in verschiedene Forschungsthemen zu erhalten und dabei zugleich im engen wissenschaftlichen Austausch zahlreiche wundervolle Menschen kennenzulernen. Ich denke gern an diese nicht nur inhaltlich, sondern auch persönlich prägende Zeit der Dissertation zurück und möchte mich hiermit bei den vielen Menschen bedanken, die mich auf diesem Weg begleitet haben.

Mein erster besonderer Dank gilt meinem Doktorvater Prof. Jürgen Becker vom Karlsruher Institut für Technologie für seine exzellente Betreuung und Unterstützung. Die zahlreichen, fachlich fundierten Gespräche und die sich hieraus ergebenden Diskussionen haben mich bei der Entwicklung der Forschungsansätze stets vorangebracht und waren für mich von großem Wert. Mit einem immer offenen Ohr und fachlich-methodisch hinterfragenden Unterredungen hat er in den gemeinsamen Besprechungen maßgeblich zum Gelingen dieser Dissertation beigetragen. Auch möchte ich mich für sein forderndes, aber auch förderndes Engagement sowie das mir entgegengebrachte Vertrauen bedanken, das mir erlaubte, in der gemeinsamen Zeit Verantwortung für unterschiedliche Aufgaben zu übernehmen. Prof. Jürgen Beckers Wirken hat mich darüber hinaus persönlich inspiriert, wofür ich ihm ebenfalls danke.

Für die Übernahme des Korreferats danke ich Prof. Andreas Herkersdorf von der Technischen Universität München (TUM) herzlich: Die inhaltliche Unterstützung, sein Interesse und das kritische Hinterfragen haben mich in meiner Arbeit positiv beeinflusst. Seine Vorträge und seine Persönlichkeit waren für mich eine Bereicherung.

Mein weiterer Dank geht an meine Kolleginnen und Kollegen, die mich jederzeit mit Rat und Tat unterstützten. Dabei möchte ich Dr. Jürgen Gebert, Hans-Ulrich Michel, Dr. Mohamad Chamas und Dr. Martin Meseth hervorheben. Zusätzlich danke ich auch gerade meinem Mitstreiter Philipp Obergfell, mit dem ich auf dem Weg zum Doktortitel in unzähligen Stunden und bei manch einem abendlichen Bier stets die richtige Mischung aus Fokussierung auf das Promotionsthema und freundschaftlich-kreativer Zerstreuung finden konnte. Nur so konnten Ideen, Veröffentlichungen und Patente reifen.

Auch allen betreuten Studenten möchte ich einen Dank aussprechen, die mit ihren Abschlussarbeiten einen wichtigen Beitrag zu dieser Dissertation geleistet haben. Mein Dank gilt auch den stets unterstützenden Mitarbeiterinnen und Mitarbeitern des ITIVs und des Forschungszentrums Informatik (FZI).

Weiterhin danke ich meinen Freundinnen und Freunden, die mir durch ihre Anregungen, Diskussionen und Korrekturen die Augen öffneten, wenn ich beim Schreiben mal wieder zu sehr an den Inhalt dachte. Besonders möchte ich dabei Nils Frase, Jochen Kuckuck und Justus von Richthofen erwähnen. Ein weiterer Dank geht an Christine Müller, die den Mut in mir weckte, mich überhaupt für die Promotion zu entscheiden.

Insbesondere der BMW Group möchte ich dafür danken, mir die Promotion ermöglicht zu haben: Dr. Gerd Schuster, Bereichsleiter der Forschung, Neue Technologien und Innovationen bei der BMW Group, ebenso den ehemaligen Hauptabteilungsleitern Martin Arend und Dr. Michael Würtenberger sowie dem heutigen Hauptabteilungsleiter Dr. Peter Lehnert – Ihnen verdanke ich durch ihr persönliches, materielles und finanzielles Engagement die breite Unterstützung meiner Promotion. Ebenso gebührt mein besonderer Dank meinen ehemaligen Gruppenleitern Dr. Hans-Jörg Vögel, Dr. Matthias Traub und meinem heutigen Gruppenleiter Graham Smethurst für ihre fachliche Expertise und organisatorische Unterstützung.

Ein ganz besonderer Dank geht an meine Eltern Doris und Peter, die stets das liebevolle Vertrauen in meine Vorhaben und in mich setzen. Ihre fortwährende Unterstützung und Ermutigung sind für mich von unschätzbarem Wert. Ebenso danke ich meinem Bruder Markus dafür, dass er mir zu jeder Zeit ein Vorbild ist. Den größten Dank möchte ich meiner Freundin Katja aussprechen, die stets mit unermüdlicher Geduld und Verständnis hinter mir steht: Sie hat nie an mir gezweifelt, und für ihre unerschöpfliche Motivation bin ich ihr von ganzem Herzen dankbar! Abschließend möchte ich meinem

Sohn Nils danken, der fast immer Verständnis hatte, wenn ich mich in meine Arbeit vertiefte, der in letzter Zeit oft zu kurz kam und mir zugleich in den wenigen gemeinsamen Stunden viel Kraft gegeben hat. Meinem erst einjährigen Sohn Finn danke ich für den Motivationsschub zur Vollendung dieser Dissertation.

München, im Juli 2021
Florian Oszwald

Inhaltsverzeichnis

1 Einleitung

Das Automobil gibt es seit mehr als einhundert Jahren und es dient seitdem als Transport- und Fortbewegungsmittel. Das Grundprinzip ist für den größten Anteil an Fahrzeugen mit vier Rädern, Antrieb und Getriebe erhalten geblieben. Jedoch sind seit den Anfängen zahlreiche Veränderungen in das Automobil eingeflossen. Seit Beginn ist es einer wachsenden Komplexität ausgesetzt, so dass neben dem Produkt auch die Strukturen und Methoden ständigen Veränderungen unterworfen sind [106, S. 1].

In den Anfängen war das Automobil ein stark durch den Maschinenbau geprägtes Produkt, in das im Laufe der Zeit andere Disziplinen Einzug erhalten haben. Ende 1913 stammten von den 44 Firmen, die in Deutschland Personenkraftwagen bauten, ein Drittel vom Fahrradbau, ein Drittel vom Maschinenbau und fünf Firmen von der Elektrotechnik ab. Damals war noch nicht entschieden, ob der Elektromotor oder der Verbrennungsmotor der Antrieb für die nächsten Jahrzehnte sein wird [26, S. 65].

Elektrische Komponenten beim Fahrzeug waren zunächst der elektrische Anlasser, die elektrische Zündung, der Generator sowie die Beleuchtung [138, S. 167]. Sukzessive wurde die vorerst mechanisch umgesetzte Benzineinspritzung ab 1967 gegen eine elektronische Variante ausgetauscht [106, S. 21]. Bei der passiven Sicherheit wurden Verbesserungen hauptsächlich durch konstruktive Entwicklungen erreicht. Ab 1974 gehörten zu der passiven Sicherheit auch Airbag-Systeme dazu; auch hier war die Elektronik involviert. Bei der aktiven Sicherheit kamen elektronische System wie das ABS[1] und die elektrische Servolenkung (*englisch*: *Electric Power Steering*, EPS) zum Einsatz [28, S. 1 ff.].

Der Einzug von immer mehr Elektronik in das Automobil schreitet weiter voran [118, S. 18]. Gleichzeitig nimmt auch die Komplexität der Gesamtelektronik im Fahrzeug

[1] Antiblockiersystem

weiterhin zu [94], [126, S. 2 ff.]. In Abbildung 1.1 ist ein Überblick von E/E-Systemen im Fahrzeug dargestellt.

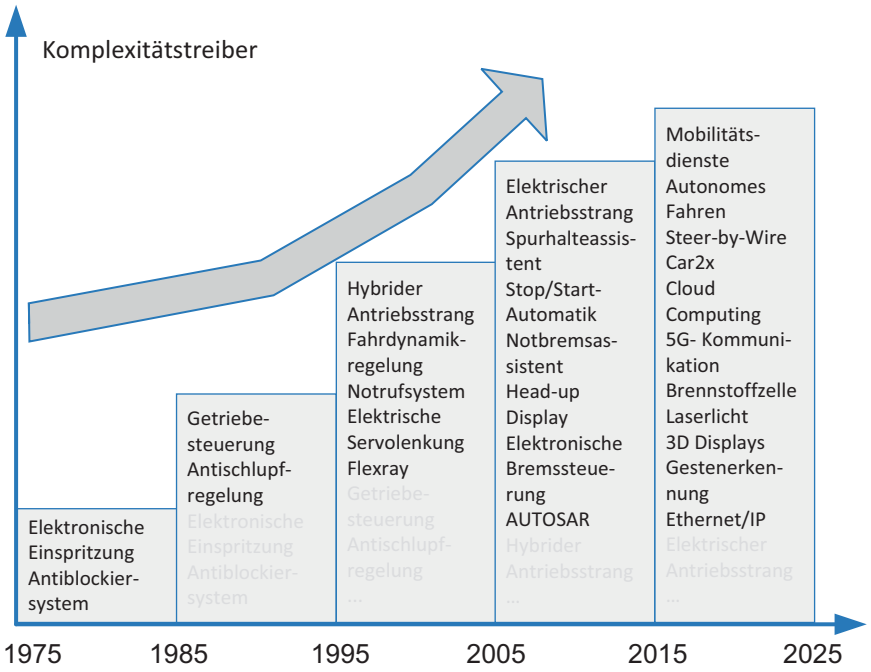

Abbildung 1.1: Die Entwicklung von E/E-Komplexitätstreibern im Fahrzeug nach [45]

Die Informationstechnik erhielt in den 1980er Jahren einen enormen Aufschwung. 1977 kam erstmals ein Mikrocontroller für die Zündung in einem Personenkraftwagen der Marke Oldsmobile zum Einsatz. Bereits 1981 wurde eine Mikrocontroller-gestützte Benzineinspritzung in einem General Motors Personenkraftwagen eingebaut, um die Anforderungen hinsichtlich Verbrauch und Emissionen zu erfüllen. Gleichzeitig wurde mit diesem Meilenstein der Beginn der Computerära im Automobil eingeläutet. In diesem Steuergerät für die Benzineinspritzung von 1981 waren circa 50.000 Zeilen Software-Code enthalten [21]. Dieser Wert beträgt in modernen Fahrzeugen mittlerweile bis zu 100.000 Code-Zeilen und kann als Maß der Komplexität herangezogen werden.

Bezogen auf die Herstellkosten des gesamten Fahrzeugs liegt der Anteil der Elektroniksysteme je nach Ausstattungsvariante bei ca. 25 % [106, S. 929]. Zudem wird in der

Automobilindustrie der Elektronikbereich zukünftig weiterhin als Innovationsmotor gesehen. Etwa 90 % aller Innovationen im Automobil werden in Zukunft auf die Elektronik entfallen. Von diesen wiederum werden ungefähr 80 % in Software umgesetzt [106, S. 930]. Neben dem Einzug von E/E-Systemen in das Fahrzeug passten sich die Strukturen und Methoden an [118, S. 21 ff.], wohingegen bei komplizierten Systemen weiterhin die klassischen Entwicklungsmethoden zum Einsatz kommen. Hier werden zunehmend agile Methoden, wie zum Beispiel Design Thinking, Scrum oder Large Scale Scrum bei komplexen Systemen eingesetzt. Diese sind auch vereinbar mit der Entwicklung von sicherheitskritischen Anwendungen, die konform zu der ISO[2] 26262 [77] entwickelt werden müssen [62], [137].

Der in Abbildung 1.1 dargestellte Zuwachs der Gesamtkomplexität im Automobil resultiert in gesteigertem Entwicklungsaufwand [94]. Die zusätzlichen Funktionen müssen jedoch weiterhin fehlerfrei realisiert werden. Darüber hinausreichende Anforderungen aus dem vollautomatisierten Fahren wie zum Beispiel die Anforderung an ein Fail-Operational-Verhalten, bringen zusätzliche Komplexität in das System mit ein. Dieses gilt ebenfalls für sicherheitskritische Funktionen.

1.1 Motivation und Umfeld

Ein gesetztes Ziel in der Automobilindustrie ist das autonome Fahren. Der Weg dorthin ist anhand der SAE[3]-Level in der Norm SAE J3016 beschrieben [110]. Die Norm beschreibt sechs Stufen, die von Stufe 0 bis Stufe 5 reichen:

Die Stufe 0 beschreibt das Level, in dem keine Automatisierung vorliegt. Systeme wie das ABS oder die EPS können den Fahrer bei seinen Aufgaben unterstützen. In dieser Stufe hat der Fahrer die Verantwortung für die Quer- und die Längsführung sowie die Umgebungsbeobachtung und übernimmt in einem Fehlerfall die Rückfallebene.

In Stufe 1 sind die Assistenzsysteme wie zum Beispiel der ACC[4] eingeordnet. Der Fahrer hat weiterhin die Verantwortung für die oben genannten Bereiche von Quer- und Längsführung, Umgebungsbeobachtung und Rückfallebene. Bei der Längsführung

[2] International Organization for Standardization
[3] Society of Automotive Engineers
[4] *englisch*: *Adaptive Cruise Control*, aktive, abstandsabhängige Geschwindigkeitsregelung

übernimmt der ACC jedoch nach Aktivierung die Verantwortung. Tritt ein Fehler auf, ist weiterhin der Fahrer in der Verantwortung.

In Stufe 2 sind mehrere Assistenzsysteme vorhanden; es wird von Teilautomatisierung gesprochen. Das System hat nun die Verantwortung für Quer- und Längsführung. Der Fahrer hat allerdings weiterhin die Verantwortung für die Umgebungsbeobachtung und er ist die Rückfallebene im Fehlerfall.

Ab Stufe 3 ist das System verantwortlich für die Umgebungsbeobachtung, diese Stufe wird als bedingte Automatisierung beschrieben. Treten während des Betriebs in den Stufen 0 bis 3 Fehler auf, so werden die entsprechenden Systeme abgeschaltet und der Fahrer muss übernehmen. Der Fahrer ist also die Rückfallebene.

Zwischen Stufe 3 und Stufe 4 findet ein gravierender Sprung mit erheblichen technischen Konsequenzen statt. Die Verantwortung für die Rückfallebene geht vom Fahrer auf das System über. Die Systeme, die sich im Fehlerfall abschalten, werden als fail-silent-Systeme bezeichnet.

Ab Stufe 4 übernimmt das System auch die Verantwortung für die Rückfallebene. Die Stufe 4 beschreibt das Level hochautomatisierten Fahrens. Ein Abschalten im Fehlerfall ist nicht mehr möglich. Das System muss die Funktion aufrechterhalten. Diese Systeme werden als Fail-Operational-Systeme bezeichnet. Der Unterschied zu Stufe 5 besteht darin, dass der Fahrer auf Anforderung des Fahrzeugs reagiert.

Stufe 5 hingegen beschreibt das Level vollständiger Automatisierung, es beinhaltet das fahrerlose oder auch vollautomatisierte Fahren. Die wesentlichen sicherheitskritischen Funktionen in jeder dieser sechs Stufen sind die Quer- und die Längsführung, also das Bremsen und das Lenken. Diese Systeme müssen jederzeit fehlerfrei funktionieren [110].

Eine vergleichbare Einteilung wurde auch von der BASt[5] [54] durchgeführt. Bei dieser fehlt jedoch das fahrerlose Fahren.

Eine weitere Interpretation dieser Stufen ist in Abbildung 1.2 zu sehen. In dieser sind die Stufen 1 bis 5 dargestellt. Die Definition für die Stufen 4 und 5 weichen von der Definition der SAE Level leicht ab. So wird in dieser Darstellung zwischen dem autonomen (Stufe

[5] Bundesanstalt für Straßenwesen

5) und dem vollautomatisierten Fahren (Stufe 4) unterschieden. Die Stufe 3 stellt in dieser Stufe das hochautomatisierte Fahren dar, eine bedingte Automatisierung gibt es in dieser Darstellung nicht. Die Stufe 0 findet sich hier gar nicht wieder.

Abbildung 1.2: Fünf Stufen des autonomen Fahrens, Quelle: [15]

1.2 Zielsetzung und Abgrenzung der Dissertation

In der Fahrzeugentwicklung sind die drei Größen Kosten, Gewicht und verfügbarer Bauraum entscheidende Stellhebel. Dadurch vollzieht sich in der Fahrzeugtechnik ein Wechsel weg von hardwareorientierten hin zu softwareorientierten Lösungen. Des Weiteren werden im Bereich des hoch- und vollautomatisierten Fahrens Systeme benötigt, die eine Anforderung an ein Fail-Operational-Verhalten erfüllen. Nachfolgend wird in der Zielsetzung erklärt, wie die dynamische Rekonfiguration dazu beitragen kann, diese beiden Anforderungen miteinander zu verbinden. Derzeit kommen DRS[6] im Automobil noch nicht zum Einsatz und es gibt auch kein Vorgehensmodell, wie diese Systeme zu entwickeln sind. Aus diesem Grund liegt der Schwerpunkt dieser

[6] dynamisch rekonfigurierbares System

Dissertation auf der Erforschung von Methoden und Ansätzen für die Entwicklung von dynamisch rekonfigurierbaren Systemen. Auf Basis einer Analyse des Stands der Technik wird daher auf folgende Forschungsfragen eingegangen:

- Welche Vorteile bietet dynamische Rekonfiguration und wie kann sie sinnvoll oberhalb der Virtualisierungsschicht eingesetzt werden?
- Welche Methoden werden benötigt, um dynamische Redundanz im Automobil einzusetzen?

Die Herausforderung dieser Arbeit ergibt sich aus der Kombination von zwei dominierenden Architekturtreibern. Ein erster Architekturtreiber ist dabei das autonome Fahren als ein gesetztes Ziel in der Automobilindustrie.

Ein weiterer Architekturtreiber sind neue Architekturansätze, die funktionale Hochintegration auf zentralen Rechner-Plattformen als einen zentralen Lösungsbaustein betrachten. Die Gründe für eine funktionale Hochintegration sind aus E/E-Sicht unter anderem, dass die Komplexität der Funktionen steigt. Zu nennen sind dabei Funktionen wie die Fahrerassistenzsysteme mit Quer-/Längs- und Vertikaldynamik, hoch- und vollautomatisiertes Fahren, sowie Steer-/Brake-by-Wire. Die intelligenten Anteile der Funktionen lassen sich nicht mehr kosteneffizient auf kleineren Steuergeräten darstellen und müssen auf leistungsfähigen Recheneinheiten, zum Teil mit funktionsspezifischer Hardware, partitioniert werden. Des Weiteren steigen bei den OEMs[7] die Eigenprogrammieranteile. Aufgrund eines klaren Verantwortungsmodells wird eine vom Lieferanten getrennte Entwicklungs- und Steuergerätepartitionierung bevorzugt. Weiterhin ist die Reduktion der Anzahl der Steuergeräte ein Architekturtreiber und die damit verbundene Vereinfachung des gesamten Kabelbaums, der ein zentraler Kostenfaktor ist.

Parallel ermöglicht die technologische Entwicklung auf Hardware(HW)-Ebene die Realisierung von Hochintegrationsplattformen, bei Bedarf mit spezifischen HW-Beschleunigern. Diese Entwicklung beruht zuletzt nicht nur auf der Situation, dass die Performance von Mehrprozessorsystemen ausgereizt ist und sich ein Übergang hin zu Mehrkernprozessorsystemen vollzieht. Bei der funktionalen Hochintegration gilt es, die an die einzelnen Softwarepfade gerichteten unterschiedlichen Anforderungen

[7] *englisch*: *Original Equipment Manufacturer*s, Originalausrüstungshersteller, Erstausrüster; dieser ist im Automobilbereich der Automobilhersteller selbst, z. B.: BMW, Daimler, Volkswagen, etc.

miteinander zu vereinen und zu einem funktionierenden Ganzen zu entwickeln. Die unterschiedlichen Anforderungen dabei sind zum Beispiel die Anforderungen an die funktionale Sicherheit, die zugrundeliegenden Betriebssysteme, die Entwicklungszyklen oder die Anforderungen an Echtzeitfähigkeit. Einzelne Softwarepfade haben unterschiedliche Entwicklungszyklen und müssen getrennt voneinander aktualisierbar sein. Es existieren darüber hinaus noch weitere, übergreifende Anforderungen.

Im Allgemeinen stellt sich im Kraftfahrzeug in vielen Bereichen die Herausforderung, ein Optimum zwischen Gewicht, zur Verfügung stehendem Bauraum und Kosten zu finden. Durch die Anforderung, ein Fail-Operational-Verhalten für bestimmte Funktionen zu erfüllen, werden derzeit Redundanzkonzepte im Fahrzeug entwickelt. Dynamische Rekonfiguration wurde dabei noch nicht umgesetzt. Unter Berücksichtigung der beiden oben genannten Architekturtreiber, der Anforderung von sicherheitskritischen Funktionen an ein Fail-Operational-Verhalten und der funktionalen Hochintegration, wird die dynamische Rekonfiguration eingeordnet und erforscht. Sie soll bei den Untersuchungen den Kern dieser Arbeit darstellen.

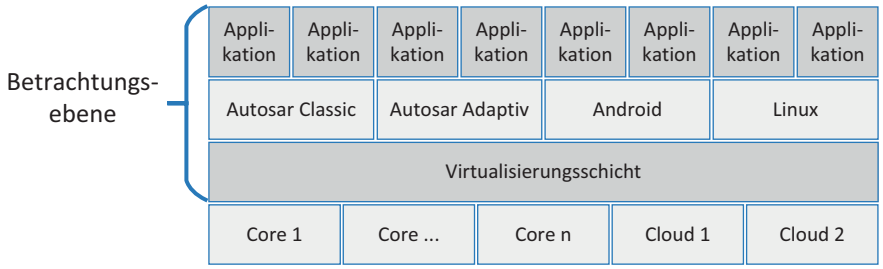

Abbildung 1.3: Problemstellung - Virtualisierungsebene

Die Anwendung von dynamischer Rekonfiguration im Rahmen von Fail-Operational im automobilen Sektor ist neu und wurde in dem Umfang noch nicht untersucht. Dynamische Rekonfiguration kann als Lösungsbaustein für Funktionen mit oder auch ohne eine Anforderung an ein Fail-Operational-Verhalten verwendet werden und soll in dieser Arbeit oberhalb der Virtualisierungsebene herausgearbeitet werden. Dieser Bereich ist in Abbildung 1.3 dargestellt. Somit wird die dynamische Rekonfiguration für eine Funktion mit Anforderungen an ein Fail-Operational-Verhalten untersucht, die gegebenenfalls auch über eine Cloud realisiert werden kann.

1.3 Gliederung der Arbeit

Im Folgenden wird der Aufbau der Arbeit dargelegt. Sie ist in acht Kapitel eingeteilt, beginnend mit Kapitel 2 werden die Inhalte der einzelnen Kapitel erläutert.

In Kapitel 2 werden zunächst relevante Definitionen und Begrifflichkeiten eingeführt. Dabei werden die notwendigen Grundlagen aufgezeigt und es wird auf den aktuellen Status im Umfeld der Thematik eingegangen.

In den nächsten beiden Abschnitten wird die Idee dieser Arbeit umrissen. Daraus stellen sich die Anforderungen an das Thema.

Kapitel 3 befasst sich mit der thematischen Einordnung der dynamischen Redundanz: Dabei wird zunächst ein Überblick über die unterschiedlichen Arten von Redundanz inklusive der dynamischen Redundanz gegeben. Im Anschluss werden relevante Vorarbeiten in einem Überblick dargestellt und in die erarbeiteten Kategorien eingeordnet. Abschließend werden die unterschiedlichen Ansätze gegenübergestellt und bewertet.

In Kapitel 4 werden sowohl eine Methodik als auch eine Systematisierung hergeleitet. Aus den in Kapitel 3 identifizierten Lücken werden strukturiert einzelne Anforderungen abgeleitet.

In Kapitel 5 wird ein neues, gesamthaftes Konzept zur dynamischen Rekonfiguration von Echtzeitsystemen für Automotive Embedded-Systeme im Fail-Operational-Kontext aufgezeigt. Weiterhin wird eine Bewertungsmöglichkeit dieser Systeme vorgestellt. Abschließend erfolgt eine Überleitung, wie das Konzept schließlich mit Hilfe von Active Replay simuliert und verifiziert werden kann.

Kapitel 6 befasst sich mit der detaillierten Implementierung des in Kapitel 5 beschriebenen Modells: Hierbei wird auf die Umsetzung in zwei verschiedenen Aufbauten eingegangen. Zusätzlich werden dabei notwendige Messaufbauten erklärt und die entstandenen Messungen beschrieben. Darauf aufbauend werden die Ergebnisse diskutiert und in den Zusammenhang zu einem Modellbasierten Ansatz gebracht. Abschließend werden die Ergebnisse ausgewertet.

Die in dieser Dissertation erreichten Umfänge werden im anschließenden Kapitel 7 in Korrelation zu den eingangs gesetzten Zielen gebracht: Es wird darauf eingegangen, in wie weit eine Übertragbarkeit der Ergebnisse möglich ist und welche Anforderungen

daraus resultieren. Ein abschließender Ausblick zeigt, welche Folgearbeiten sich aus dieser Arbeit unmittelbar ergeben.

2 Grundlagen

In diesem Kapitel wird auf Themen eingegangen, die als Grundlage für die weiteren Erkenntnisse dieser Arbeit dienen. Sie stellen eine Basis für das Verständnis der zugrundeliegenden Technologien und des Umfelds der Arbeit dar. Das Kapitel ist dabei in drei Abschnitte unterteilt. Der erste Abschnitt liefert Begriffe und Definitionen, um ein einheitliches Verständnis zu erzeugen. Anschließend werden Grundlagen aus dem Bereich der sicherheitskritischen Systeme als Treiber aus dem autonomen Fahren aufgezeigt. Abgeschlossen wird das Grundlagenkapitel mit einem Fazit und der Überleitung zum dritten Kapitel.

2.1 Entwicklung automobiler E/E-Architekturen

Diese Arbeit befasst sich mit dynamischer Rekonfiguration im Automobilbereich. Aus diesem Grund werden im Folgenden einige Begriffe und Definitionen aus diesem Themenbereich eingeführt. Auf diese Weise ist ein einheitliches Verständnis im Rahmen dieser Arbeit gewährleistet.

2.1.1 Systemebenen in der Fahrzeugelektronik

Um die Gesamtkomplexität von einem Fahrzeug zu beherrschen, werden zwei verschiedene Einteilungen vorgenommen: Zum einen erfolgt eine Einteilung in Domänen. Diese Einteilung wird in verschiedener Literatur beschrieben. Eine Domäne fasst verschiedene Funktionen im Fahrzeug zusammen. Üblicherweise gibt es dabei folgende Domänen: Fahrwerk, Antrieb, Chassis, Infotainment, Innenraum und Fahrerassistenz. Siehe dazu auch [7, S. 9 f.], [31, S. 16], [127, S. 45] und [136, S. 54 f.]. Die einzelnen Domänen haben sich in den letzten Jahren verändert. So ist zum Beispiel die Domäne für Fahrerassistenzsysteme aus der Domäne der Chassis-Funktionen bereits herausgelöst

und existiert eigenständig. Bei der Zulieferpyramide in der Automobilbranche finden sich diese Domänen beim OEM, bis hin zu den einzelnen Tiers[1] wieder. Dabei orientiert sich häufig die Ausrichtung der Unternehmensorganisation an diesen Domänen. Auch die Freigabeverantwortung der Komponenten gliedert sich entsprechend der Domänen. Zukünftig werden durch Hochintegration Domänen-übergreifende Funktionen in wenigen Steuergeräten zusammengeführt. Dadurch werden sich die Unternehmensorganisationen verändern. Die Hochintegration stellt die Basis dar, um die erforderlichen Fail-Operational-Anforderungen zu erfüllen.

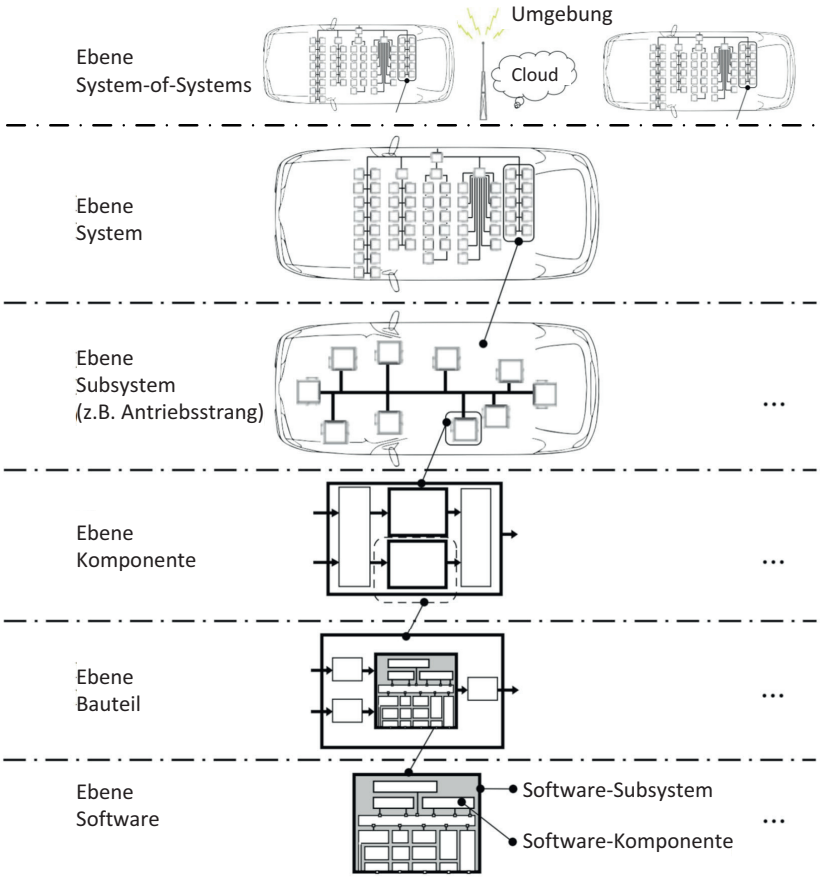

Abbildung 2.1: Systemebenen in der Fahrzeugelektronik nach [118, S. 130]

[1] *englisch*: *Tier*, Ebene; Bezeichnung als Tier-1, Tier-2 als Systemlieferant, Komponentenlieferant etc.

Zusätzlich zu dieser Kapselung der einzelnen Segmente des Gesamtsystems[2] in unterschiedliche Domänen ist eine Partitionierung oder auch Dekomposition[3] erforderlich. Dieses Vorgehen zur Verringerung der Komplexität ist in der E/E-Entwicklung von entscheidender Bedeutung. Das Gesamtsystem wird dabei in zusammenhängende Einheiten unterteilt, die weitestgehend eigenständig von den anderen umgesetzt werden können. Die Unterteilung umfasst dabei mehrere Hierarchieebenen [7, S. 13]. In dieser Arbeit werden Definitionen von Schäuffele [118, S. 130] und aus der ISO/IEC[4]/IEEE[5] 24765 [76] miteinander verbunden: konkret werden die Ebenen System-of-Systems, E/E-System, E/E-Subsystem, E/E-Komponente, E/E-Bauteil und Software verwendet.

In der Ebenendarstellung Abbildung 2.1(vgl. [118, S. 130]) wurde eine weitere Stufe eingeführt, die des System-of-Systems. Des weiteren wurden die einzelnen Stufen den Fahrzeugebenen zugeordnet. Auf der obersten Ebene befindet sich das Automotive-System-of-Systems. Es reicht über die Fahrzeuggrenzen hinaus. Auf dieser Ebene befinden sich Inter-Car und Car-to-Backend-Beziehungen. Die einzelnen Systeme sind die Fahrzeuge. Definiert ist diese Ebene wie folgt:

Definition 2.1 (Automotive-System-of-Systems) *Ein System-of-Systems ist eine Sammlung von Komponenten, bei der die Komponenten jeweils einzeln wiederum als System betrachtet werden können und die zwei zusätzliche Eigenschaften besitzen* [64], [97].

Operative Unabhängigkeit der Komponenten: Wenn das System-of-Systems in seine Komponentensysteme zerlegt wird, müssen die einzelnen Komponentensysteme wiederum in der Lage sein, eigenständig zu arbeiten. Das heißt, dass sie Kunden- und Betreiberzwecke eigenständig erfüllen.

Führungsunabhängigkeit der Komponenten: Die Komponentensysteme können nicht nur eigenständig vom System-of-Systems arbeiten, sie arbeiten tatsächlich eigenständig. Die Komponentensysteme werden separat integriert und halten eine kontinuierliche Unabhängigkeit vom System-of-Systems aufrecht.

[2] Kombination von interagierenden Komponenten, die organisiert sind, um eine oder mehrere festgelegte Bestimmungen zu erfüllen [7, S. 13], [76].
[3] Die Aufteilung oder Zerlegung eines Systems in Komponenten wird als Partitionierung oder Dekomposition bezeichnet [118, S. 129].
[4] International Electrotechnical Commission
[5] Institute of Electrical and Electronics Engineers

Die zweithöchste Ebene ist die Fahrzeugebene, die in diesem Zusammenhang als E/E-Systemebene bezeichnet wird. Auf dieser Ebene liegen alle E/E-Systeme innerhalb der Fahrzeuggrenzen. Für ein E/E-System wird in dieser Arbeit folgende Definition verwendet:

Definition 2.2 (E/E-System) *Ein E/E-System stellt eine oder mehrere Funktionen im Fahrzeug bereit. Die technische Umsetzung der einzelnen Funktionen erfolgt durch E/E-Komponenten und durch notwendige Schnittstellen* [*79, S. 15*]. *Somit besteht ein E/E-System aus Software* (*Funktionen*) *und Hardware* (*E/E-Komponenten*).

Die verschiedenen E/E-Komponenten sind im Fahrzeug verteilt und aufgrund der verteilten Funktionsbeiträge[6] miteinander vernetzt. Daher wird der Begriff des verteilten E/E-Systems ebenfalls für ein E/E-System verwendet. Das E/E-System bietet dem Fahrzeug eine oder mehrere Funktionen an. Es wird erst durch die Interaktion der einzelnen Funktionsbeiträge und nicht schon durch die Menge aller Funktionsbeiträge dargestellt [101, S. 148]. Neben der Bezeichnung des verteilten E/E-Systems kommt auch die Verwendung als eingebettetes System[7] vor.

Definition 2.3 (eingebettetes System) *Ein eingebettetes System ist eine aus Software- und Hardwareanteilen bestehende Einheit, die zur Überwachung, Steuerung oder Regelung bestimmter Aufgaben eingesetzt wird. Ein entscheidender Hardwareanteil ist dabei der Mikrocontroller. Eingebettete Systeme sind keine Rechner, die auf dem Arbeitsplatz stehen. Vielmehr sind sie in den häufigsten Fällen für den Anwender verborgen und kommen im Fahrzeug in Form jedes einzelnen Steuergerätes zum Einsatz* [*29, S. 113*], [*102*].

Im Fahrzeug sind die E/E-Systeme als Echtzeitsysteme umgesetzt. Dies trifft bis auf einzelne Ausnahmen in der Infotainment-Domäne auf alle E/E-Systeme zu. Ein Echtzeitsystem ist ein System, das seine Eingangs- und Zustandsgrößen garantiert innerhalb eines vorgegebenen Zeitintervalls verarbeitet und darin die korrespondierenden Ausgangs- und Zustandsgrößen erzeugt [7, S. 11], [116]. Eine weitere Form des Echtzeitsystems ist ein System, das zusätzlich eine Anforderung an harte Echtzeit hat.

[6] Durch eine Softwarekomponente umgesetzter und zur Erfüllung einer (verteilten) Funktion notwendiger Bestandteil [79, S. 15].
[7] *englisch*: *Embedded System*, eingebettetes System

Definition 2.4 (Echtzeit) *Von harter Echtzeit spricht man, wenn ein Echtzeitsystem die Einhaltung des Echtzeitkriteriums unter allen Umständen garantiert, bzw. Verstöße detektiert und geeignete Fehlermaßnahmen einleitet. Von weicher Echtzeit spricht man dagegen, wenn ein System nur in den meisten Fällen das Echtzeitkriterium erfüllt* [7, S. 11], [116].

Damit ist die zweithöchste Ebene beschrieben. Auf der dritten Ebene liegen die E/E-Subsysteme. Dabei besteht ein E/E-Subsystem zum Beispiel aus allen einer Domäne zugehörigen E/E-Komponenten. Es können jedoch E/E-Komponenten außerhalb der Domänenzugehörigkeit liegen, die einen Funktionsbeitrag für exakt diese Domäne leisten.

Auf der vierten Ebene liegen die E/E-Komponenten. Für die E/E-Komponenten wird folgende Definition verwendet:

Definition 2.5 (E/E-Komponenten) *Eine Komponente ist eine physische Aggregation von zusammenhängenden mechanischen, elektrischen oder mechatronischen Teilen oder Baugruppen. Alle Komponenten werden physikalisch im Fahrzeug verbaut und besitzen somit zumindest mechanische Schnittstellen. Im Fokus dieser Arbeit stehen jedoch Komponenten, welche zusätzlich elektrische, elektronische beziehungsweise mechatronische Eigenschaften aufweisen und nachfolgend als E/E-Komponenten zusammengefasst werden. 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* [48], [79, S. 13].

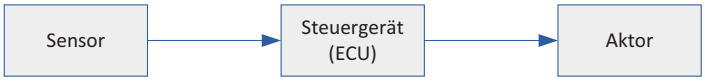

Abbildung 2.2: Vereinfachte Darstellung eines aus einem Sensor, einem Steuergerät und einem Aktor bestehenden E/E-Systems nach [7]

E/E-Komponenten sind entweder Sensoren, Aktoren oder Steuergeräte. Abbildung 2.2 zeigt eine vereinfachte Darstellung. Nach [7, S. 11], [114, S. 10], [120] werden die einzelnen Bestandteile definiert als:

Definition 2.6 (Steuergerät) *Eine ECU[8] ist die physische Umsetzung eines eingebetteten Systems. Es ist die Kontrolleinheit eines mechatronischen Systems, das Sensoren und Aktoren als Peripherie besitzen kann.*

Definition 2.7 (Sensor) *Ein Sensor[9] ist ein Fühler oder Aufnehmer, der eine physikalische oder chemische (meist nichtelektrische) Größe in eine elektrische Größe umsetzt.*

Definition 2.8 (Aktor) *Ein Aktor[10] setzt Signale geringer Leistung, die Stellinformationen beinhalten, in leistungsbehaftete Signale um, die in einer zur Prozessbeeinflussung notwendigen Energieform vorliegen. Sie bilden die Schnittstelle zwischen elektrischer Signal- bzw. Informationsverarbeitung und dem Prozess, also der Mechanik.*

Nachdem die vierte Ebene der Systemebenen beschrieben wurde, folgt nun die fünfte Ebene. Auf der fünften Ebene befinden sich die E/E-Bauteile:

Definition 2.9 (E/E-Bauteile) *Die E/E-Komponenten bestehen aus unterschiedlichen Bauteilen. Das zentrale Bauteil, auf dem die Funktion in Software umgesetzt ist, ist der Prozessor.*

Im Folgenden werden einige weitere Begriffe hinsichtlich der Prozessoren dargestellt. Die in eingebetteten Systemen derzeit am meisten eingesetzte Prozessorarchitektur ist der Einkernprozessor oder auch Singlecore-Prozessor. Dieser besteht aus nur einem Rechenkern (Prozessorkern). Die Bandbreite der Architektur reicht dabei von 8 Bit bis zu 64 Bit Prozessoren. Prinzipiell wird dabei zwischen der Von-Neumann-Architektur und der weiter verbreiteten Harvard-Architektur unterschieden. Die Von-Neumann-Architektur unterscheidet sich zur Harvard-Architektur in der Speicheranbindung. Bei der ersteren wird nur ein Speicher an den Prozessorkern angebunden, der sowohl Daten als auch Befehle beinhaltet. Bei der Harvard-Architektur werden zwei Speicher angebunden, einer für Befehle und einer für Adressen. Auf diese Weise können erheblich bessere Zugriffe auf die Speicher erreicht werden und somit ist diese Architektur die heutzutage am weitesten verbreitete [11, S. 30].

[8] *englisch*: *Electronic Control Unit*, Steuergerät
[9] *latein*: *sensus*, Empfindung, Gefühl
[10] oder Aktuator: *latein*: *actus*, Bewegung, Tätigkeit

Singlecore-Prozessoren haben daher eine Grenze hinsichtlich ihrer Leistungsfähigkeit. Um immer höhere Frequenzen bei der Taktung der Prozessoren zu erreichen, sind auch immer höhere Spannungen notwendig. Höhere Spannungen führen jedoch auch zu immer höherer Verlustleistung und somit zu einer thermischen Problematik. Um dennoch eine Steigerung der Rechenleistung zu erreichen, werden weitere Prozessorkerne parallelgeschaltet und in einen Chip integriert. In diesem Fall spricht man von Multicore-Prozessoren oder auch Mehrkernprozessoren. In Abbildung 2.3 ist eine beispielhafte Multicore-Prozessorarchitektur dargestellt.

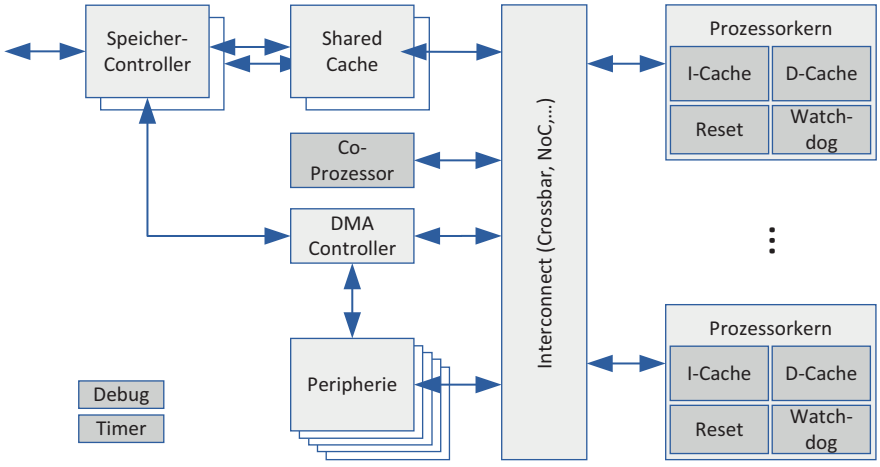

Abbildung 2.3: Abstrahierte und beispielhafte Architektur von Multicore-Prozessoren nach [11]

Ein weiterer Prozessortyp ist der Manycore-Prozessor. Gegenüber den Multicore-Prozessoren handelt es sich hierbei um Prozessoren, die deutlich mehr Prozessorkerne beinhalten. Bei den Multicore-Prozessoren kann aus Rückwärtskompatibilitätsgründen der Softwarecode für Singelcoreprozessoren ausgeführt werden. Bei Manycoreprozessoren ist das nicht mehr möglich. Dadurch kann die Energieeffizienz weiter gesteigert werden [100], [122]. Zusätzlich werden folgende Begriffe festgelegt:

Definition 2.10 (SoC - System on Chip) *Ein SoC integriert die notwendigen Komponenten eines Computers auf einem Die*[11]*. Zu den Komponenten zählen typischerweise der Hauptprozessor, der Speicher, die I/Os*[12] *und der Sekundärspeicher.*

[11] *englisch*: *Die*, Würfel oder Plättchen, ungehaustes Stück eines Halbleiter-Wafers
[12] *englisch*: *Input/Output*, Eingabe/Ausgabe

Definition 2.11 (MPSoC - Multiprozessor System on Chip) *Wenn ein SoC mit mehreren Prozessorkernen bestückt ist, wird dieser als MPSoC bezeichnet. Je nach integrierten Prozessoren wird noch in die folgenden Punkte unterschieden:*

Definition 2.12 (Homogener MPSoC) *Bei einem homogenen MPSoC sind die auf dem Die integrierten Prozessorkerne alle vom selben Typ. Wie beim SoC werden zusätzliche Peripherien auf dem Die mit integriert. Anders als die Prozessorkerne werden diese in der Regel jedoch nicht mehrfach verbaut. Bei der parallelen Verarbeitung auf den einzelnen Prozessorkernen kann es daher zu einem gleichzeitigen Zugriff zum Beispiel auf die Speichercontroller kommen, der bei Singlecore-Prozessoren aufgrund der sequenziellen Abarbeitung von Instruktionen so nicht möglich ist* [*11, S. 33*].

Definition 2.13 (Heterogener MPSoC) *Gegenüber einem homogenen MPSoC sind bei einem heterogenen MPSoC die verbauten Prozessorkerne von unterschiedlichem Typ. Hierbei kann es sich um Prozessorkerne mit unterschiedlicher Mikroarchitektur handeln, um spezielle Beschleuniger oder um rekonfigurierbare Logikarchitekturen. Bei der Peripherie verhält es sich genauso wie bei einem homogenen MPSoC, sodass es auch hier zu Engpässen bei der gemeinsam genutzten Ressource aufgrund der Parallelität in der Abarbeitung der Prozessorbefehlssätze kommen kann.*

Bei der rekonfigurierbaren Logik handelt es sich typischerweise um einen FPGA:

Definition 2.14 (FPGA - Field Programmable Gate Array) *Eines der zahlreichen Merkmale von FPGAs ist die Möglichkeit, bei der Programmierung auch die Schaltungsstruktur über eine Hardwarebeschreibungssprache in den Prozessor zu laden. Dieses unterscheidet sie von Singlecore- und Multicore-Prozessoren. Der Vorgang als solches wird als Konfigurieren bezeichnet. Im Feld können bei einem FPGA die Schaltungsstrukturen geändert werden. Dieses ist ein Alleinstellungsmerkmal gegenüber Singlecore- und Multicore-Prozessoren sowie ASICs*[13]*, die nach dem Verlassen des Herstellungsprozesses eine nicht mehr veränderbare Schaltungsstruktur haben.*

[13] *englisch*: *Application-Specific Integrated Circuit*, anwendungsspezifische integrierte Schaltung, eine als integrierter Schaltkreis realisierte, nicht mehr veränderbare elektronische Schaltung

Eine weitere notwendige Unterscheidung ist zwischen Mikrocontrollern und Mikroprozessoren festzulegen. Ein Mikrocontroller ist ein Ein-Chip-Mikrorechner mit speziell für Steuerungs- oder Kommunikationsaufgaben zugeschnittener Peripherie [79, S. 15], [99]. Ein erster Mehrkern-Mikrocontroller wurde 2006 auf den Markt gebracht. Zuvor gab es ausschließlich Einzelkern-Mikrocontroller. Ein Mikroprozessor ist im Gegensatz zum Mikrocontroller ein Universalrechner ohne Speicher, I/Os, Timer etc.

Die sechste und letzte Ebene ist die Softwareebene. Auf dieser Ebene ist die Software platziert, die auf den Prozessoren der darüberliegenden Ebene abläuft.

2.1.2 Abstraktionsebenen von E/E-Architekturen

In diesem Abschnitt soll definiert werden, was unter dem Begriff E/E-Architektur zu verstehen ist und aus welchen Teilen sich diese zusammensetzt.

Definition 2.15 (E/E-Architektur) *Eine E/E-Architektur beschreibt fahrzeugweit alle E/E-Systeme hinsichtlich der Struktur und Interaktion, d. h. der Schnittstellen, der funktionalen, logischen, elektrischen sowie physikalischen Vernetzung, der Kommunikation, der elektrischen Versorgung sowie der Topologie* [*79, S. 16*].

In Abbildung 2.4 ist das Modell bestehend aus vier Ebenen für eine detaillierte Beschreibung einer E/E-Architektur im Kraftfahrzeug dargestellt [127, S. 15 f.].

Diese vier Ebenen stellen jeweils eine spezifische Sicht auf die Architektur dar und sind miteinander verbunden. Je nach Verwender, ob Entwickler, Architekt, Produktstratege oder Kunde, wird die entsprechende Ebene oder auch Sicht angewendet. Die vier Ebenen werden wie folgt bezeichnet: Auf der obersten Ebene befindet sich der Funktionsumfang, darunterliegend ist die Funktions- bzw. Software-Architektur, auf der nächsten Ebene liegt die Vernetzungsarchitektur und schließlich folgt die Komponententopologie.

Funktionsumfang

Der Funktionsumfang befindet sich auf der obersten Ebene. Im Funktionsumfang sind sämtliche Funktionen mit ihren Merkmalen hierarchisch hinterlegt. Er besteht aus einem Basisanteil und dem Sonderausstattungsteil.

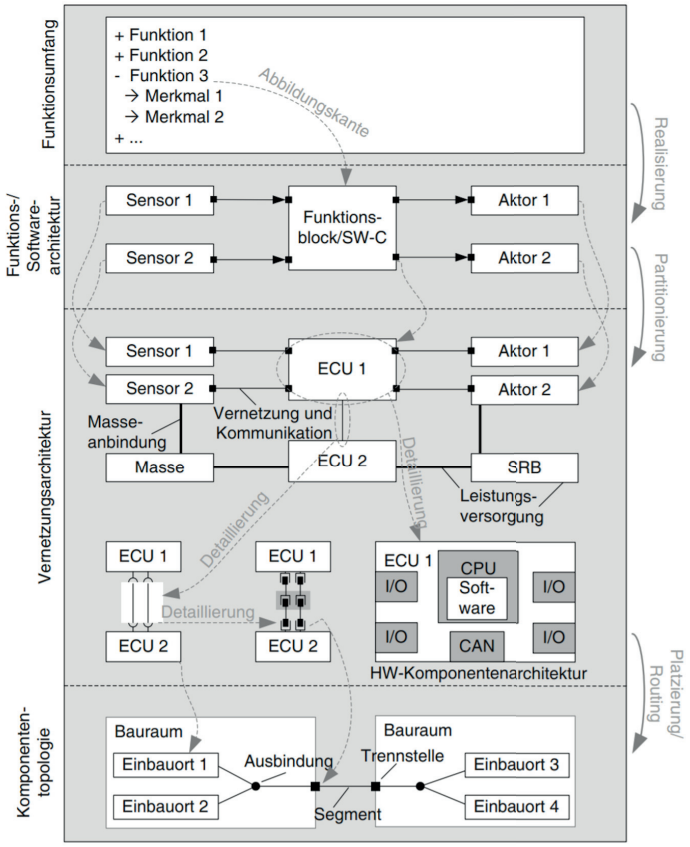

Abbildung 2.4: Hardware-Architektur: Vier Ebenen der E/E-Architektur, Quelle: [127, S. 16]

Funktions-/Software-Architektur

Die unter der Funktionsumfangsebene liegende Ebene wird als Funktions-/Software-Architektur oder auch funktionale E/E-Architektur bezeichnet. Hier sind die für die

Realisierung der Funktionen aus der darüber liegenden Ebene notwendigen Eingabe-, Funktions- und Ausgabeblöcke definiert.

Vernetzungsarchitektur

Die nächste Ebene wird Vernetzungsarchitektur oder auch logische E/E-Architektur genannt. Hier werden die einzelnen Softwarekomponenten den entsprechenden Hardware-Komponenten zugeordnet. Weitere Untergliederungen der Vernetzungsarchitektur sind die Kommunikationsstruktur, die Leistungsversorgung und die Komponentenarchitektur, sowie der Leitungssatz.

Komponententopologie

Die Komponententopologie oder auch geometrische E/E-Architektur ist auf der untersten Ebene angesiedelt. In dieser werden die Hardware-Komponenten der Vernetzungsarchitektur auf die Bauräume verteilt. Anhand der Verlegewege im Fahrzeug werden die Leitungen für Kommunikation und Leistungsversorgung dargestellt.

Eine E/E-Architektur beschreibt demzufolge durchgängig und fahrzeugweit die Zusammenhänge für Software, Hardware, Bordnetz, Leitungssatz und Topologie [79, S. 19].

Domänenarchitektur

Eine konkrete Ausprägung der E/E-Architektur ist die Domänenarchitektur. Hierbei werden durch Hochintegration (vgl. Abschnitt 2.1.4) verschiedene Funktionen auf Domänenleitrechner integriert. Die Einführung dieser Domänenleitrechner ist als evolutionärer Schritt anzusehen. Sie sind für die Datenverarbeitung innerhalb ihrer Domäne zuständig, sowie auch für die domänenübergreifenden Funktionen und die Kommunikation. Auf diese Weise können bisher getrennte Funktionen zusammengefasst werden und die bis jetzt vorhandene strikte Trennung auf Bauteilebene kann aufgeweicht werden. Werden die Leitrechner in geometrisch sinnvollen Bauräumen platziert, kann dieses auch zu einer Reduzierung des Kabelbaumumfangs führen. Mit Hilfe von standardisierten, frei programmierbaren Steuergeräteplattformen ist eine

freie Einteilung der Topologie möglich. Anstatt der heute üblichen Gatewaystruktur kommunizieren diese Hochleistungsrechner über einen Backbone mit sicherem und echtzeitfähigem Kommunikationsprotokoll mit hoher Datenrate zu geringen Kosten. Heutzutage sind insbesondere zeitkritische Anwendungen wie das Airbag-Crash-Signal aufgrund möglichst latenzfreier Übertragung über eine Stichleitung angebunden. Zukünftig wird dieses vielleicht auch über das Backbone übertragbar sein [31, S. 37].

Die zentralen Aufgaben in den Domänen Komfort, Sicherheit, Fahrerassistenz, Infotainment und Energieversorgung werden durch die Domänenleitrechner übernommen. An diese sind die im Fahrzeug verteilten Steuergeräte für die intelligenten Sensoren und Stellglieder angeschlossen. Auf den zentralen Domänenleitrechnern werden vorwiegend Funktionen in Software abgebildet, die einen hohen Vernetzungsgrad hinsichtlich Informationen und Stellbefehlen haben. Damit die Software wiederverwendbar bleibt ist eine Standard-Softwarearchitektur erforderlich, wie sie aktuell durch das AUTOSAR[14]-Konsortium [53] vorangetrieben wird [109, S. 211].

2.1.3 Entwicklungsprozess und Vorgehensmodelle

Für die Entwicklung von E/E-Systemen werden neben den erwähnten Umfängen auch Entwicklungsprozesse benötigt. Dabei hat sich in den letzten Jahren die modellbasierte Software-Entwicklung in der Automobilindustrie verbreitet. Die Gründe dafür liegen in dem Vorteil, dass eine Beschreibung von Algorithmen ohne Spielraum für Interpretationen durch die Verwendung von Software-Modellen ermöglicht wird. Dabei kommen ein einheitliches Vokabular und eine grafische Darstellung zum Einsatz. Auf diese Weise werden die Algorithmen verständlicher geschildert als durch sprachliche Beschreibungen oder Programmcodes.

Durch Software-Modelle wird ein Software-System aus verschiedenen Blickwinkeln beschrieben. Je nach Anforderung sind bestimmte Sichtweisen von besonderem Vorteil: Um zum Beispiel die Software-Architektur von Mikrocontrollern zu beschreiben, sind die Kontext- oder Schnittstellensicht, die Schichtensicht und die Sicht auf die möglichen Betriebszustände besonders vorteilhaft [118, S. 159].

[14] AUTomotive Open System ARchitecture

Im Folgenden werden einige Begriffe für den Entwicklungsbegriff dargelegt. Nach [118, S. 160] ergeben sich folgende Definitionen:

Definition 2.16 (Prozess) *Ein Prozess im Sinne eines Vorgehensmodells ist eine systematische, wiederkehrende Reihe logisch aufeinander folgender Schritte.*

Dabei werden die aufeinander folgenden Schritte definiert als:

Definition 2.17 (Prozessschritt) *Ein Prozessschritt ist eine in sich abgeschlossene Folge von Tätigkeiten, die als Ergebnis ein Artefakt liefert.*

Wobei ein Artefakt beschrieben wird als:

Definition 2.18 (Artefakt) *Ein Artefakt ist ein Zwischenergebnis, das von anderen Prozessschritten weiterverwendet wird. Artefakte elektronischer Systeme des Fahrzeugs sind beispielsweise die Spezifikation oder die Implementierung einer Software-Komponente, aber auch Hardware-Komponenten, Sollwertgeber, Sensoren oder Aktoren.*

Für jeden Prozessschritt muss eine Methode festgelegt werden. Dabei ist eine Methode nach [10] definiert als:

Definition 2.19 (Methode) *Eine Methode ist eine planmäßig angewandte, begründete Vorgehensweise zur Erreichung von festgelegten Zielen im Allgemeinen im Rahmen festgelegter Prinzipien.*

Ein Prozess oder auch Vorgehensmodell ist also ein allgemeiner Vorgehensrahmen, der die Art und Anordnung von Aktivitäten im Rahmen der SW-Entwicklung benennt und beschreibt. Dabei wird grundsätzlich zwischen zwei Varianten unterschieden, dem sequenziellen, streng phasenorientierten Modell und dem nicht linearen, evolutionären Modell. In der Praxis wird das Vorgehensmodell für konkrete Projekte und Unternehmen näher spezifiziert und es wird ein konkretes Prozessmodell festgelegt.

Typische Vorgehensmodelle sind unter anderen CMMI[15] [35], SPICE[16] [75], das Spiralmodell [27], das Wasserfallmodell [111] sowie das V-Modell 97 [66, S. 35 f.], [72], [118, S. 131].

Ein Prozessmodell ist eine konkrete Instanz eines Vorgehensmodells und definiert die konkreten Aktivitäten, ihre Reihenfolge und ihre Produkte für die Durchführung eines Projektes. Die exakte Definition ist abhängig von Projekttyp und Unternehmenspolitik sowie -kompetenz. Die einzelnen Aktivitäten müssen durch Methoden und Werkzeuge unterstützt werden [69].

V-Modell

Im Automobilbereich kommt hauptsächlich das V-Modell zum Einsatz. Dieses Modell zur Entwicklung von Systemen findet auch bei der Entwicklung von E/E-Umfängen seine Anwendung. In Abbildung 2.5 ist das vereinfachte V-Modell dargestellt.

Von dem V-Modell gibt es verschiedene Ausprägungen, somit gibt es nicht das eine gültige V-Modell. Es ist ein Leitfaden zum Planen und Durchführen von Entwicklungsprojekten unter Berücksichtigung des gesamten Systemlebenszyklus. Es wurde für eingebettete Systeme entwickelt und behandelt die Software als Komponente eines informationstechnischen Systems. Für die SW-Entwicklung von eingebetteten Systemen ist die Kopplung mit der System-Entwicklung und der HW-Entwicklung charakteristisch [69]. Zusätzlich integriert das V-Modell die Qualitätsprüfung in die Systemerstellung.

Bevorzugt wird das V-Modell für Systeme mit hohen Zuverlässigkeits- und Sicherheitsanforderungen eingesetzt, bei denen entsprechende Prüfschritte vorgeschrieben sind. Weiterhin kommt es für Systeme zum Einsatz, deren Komponenten verteilt entwickelt werden [118, S. 158]. Das V-Modell ist ein international anerkannter Entwicklungsstandard für IT-Systeme, der einheitlich und verbindlich festlegt, was zu tun ist, wie die Aufgaben durchzuführen sind und womit dies zu geschehen hat. Es umfasst das Vorgehensmodell, die Methodenzuordnung und die funktionalen Werkzeuganforderungen. Damit ist klar umrissen, in welchen Schritten und mit welchen Methoden die

[15] Capability Maturity Model Integration
[16] Software Process Improvement and Capability Determination

Entwicklungsarbeiten auszuführen sind und welche funktionalen Eigenschaften die zum Einsatz kommenden Werkzeuge aufweisen müssen.

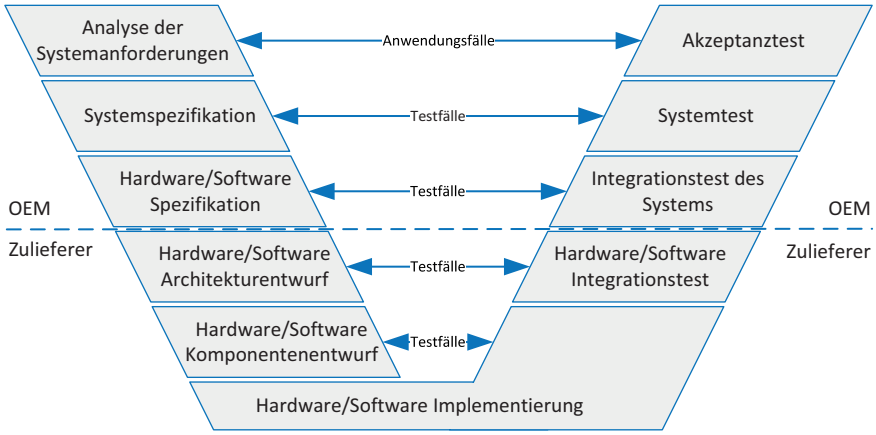

Abbildung 2.5: Entwicklungsprozess: Vereinfachtes V-Modell nach [69]

In Abbildung 2.5 ist der Tätigkeitsbereich der Systemerstellung dargestellt. Dieser ist definiert als der Kernprozess. Zusätzlich zu diesem Kernprozess sind folgende fünf Unterstützungsprozesse festgelegt [118]:

- das Konfigurationsmanagement
- das Projektmanagement
- das Lieferantenmanagement
- das Anforderungsmanagement
- die Qualitätssicherung

Das V-Modell 97 wurde durch das V-Modell XT [2] im Jahr 2005 abgelöst. Während der Fokus beim V-Modell 97 auf den Tätigkeiten liegt, ist das V-Modell XT auf die Produkte fokussiert. Beim V-Modell XT wurde eine Reihe zusätzlicher Konzepte eingeführt, wesentliche Neuerungen wurden eingebracht und bereits vorhandene Regelungen unter Berücksichtigung des Stands der Technik erweitert. Mit Hilfe der Konventionsabbildung aus dem V-Modell XT können die Begriffe aus dem V-Modell XT auf das V-Modell 97 abgebildet werden. Je nach Projektkontext lassen sich agile Vorgehensweisen wie Scrum gut mit dem V-Modell XT kombinieren und es so konkret ausgestalten.

Agile Software-Entwicklung

Agile Software-Entwicklung ist ein Weg, um den Entwicklungsprozess zu führen und zu organisieren. Er fokussiert auf direkte und regelmäßige Kommunikation und auf regelmäßige Lieferung von funktionierenden Softwareergebnissen. Kurze Iterationen und aktive Kundeneinbindung während des gesamten Entwicklungslebenszyklus stehen dabei genauso im Vordergrund wie auch die Reaktionsfreudigkeit hinsichtlich Veränderung anstatt auf Vermeidung von Veränderung. Diese Herangehensweise steht im Gegensatz zu den Wasserfallmodellen, welche auf genaue und detaillierte Planung und Entwurf im Vorfeld sowie Konformität zu den aufeinanderfolgenden Stufen des Plans Wert legen. Es gibt eine Vielzahl von agilen Methoden, die in der Softwareindustrie zum Einsatz kommen. Unter diesen sind Scrum [121], Kanban [50] und Extreme Programming (XP) [16] die am meisten verwendeten, häufig auch in einer Kombination [60, S. 12]. Continuous Integration [131] ist dabei die am häufigsten verwendete Tätigkeit.

Agile Methoden kommen dann zum Einsatz, wenn der gegenwärtige Entwicklungsprozess für die aktuellen Herausforderungen nicht mehr angemessen ist. Häufig genannte Herausforderungen dabei sind der Zuwachs von hoher Komplexität, das Tempo, in dem Veränderungen gefordert sind und die Verkürzung der Produkteinführungszeit. In einer Umfrage hat sich ergeben, dass die ISO 26262 als kompatibel zu agilen Methoden angesehen wird [137].

2.1.4 Weitere Begriffe im Zusammenhang mit dieser Arbeit

Neuromorphe Systeme

Neuromorphe Systeme können in zwei Gruppen eingeteilt werden: Zum einen in die biologisch inspirierten Systeme und zum anderen in die neuromimetischen Systeme. Bei den biologisch inspirierten Systemen verwenden Entwickler Prinzipien aus der Biologie, um effizientere und neue Lösungen auf Fragestellungen aus dem Ingenieurbereich zu finden: z.B. für Vision [25], [37], Aufmerksamkeit [71] und Learning [40].

Bei der zweiten Gruppe, den neuromimetischen Systemen, werden Ingenieurmittel und -werkzeuge benutzt, um neurowissenschaftliche Fragestellungen anzugehen, wie

z.B. zentrale Muster zu verstehen [123], die rhythmische Motorsteuerung [34], Learning [143] oder Bildverarbeitung [39], [112]. Im Rahmen dieser Arbeit wird der Begriff neuromorph im Sinne der zweiten Gruppe, den neuromimetischen Systemen, verwendet.

Virtualisierung

In der Industrie sind Virtualisierungslösungen seit längerem Stand der Technik. In der Automobilindustrie kommen diese Lösungen noch kaum zum Einsatz. Mit Hilfe dieser Technologie, können Hard- oder Software-Objekte durch ein ähnliches Objekt vom selben Typ mit Hilfe eines Abstraktions-Layers nachgebildet werden. Dadurch können virtuelle Geräte oder Dienste wie emulierte Hardware, Betriebssysteme, Datenspeicher oder Netzwerkressourcen erzeugt werden.

Bei der Virtualisierung wird durch ein Programm ein Modell eines Prozessors oder einer Systemarchitektur bereitgestellt. Dieses Modell ermöglicht das Ausführen von Programmen in der gleichen Form, wie der Prozessor oder die Systemarchitektur, die von dem Modell nachgebildet wird. Die Virtualisierung ist ein sehr breites Aktionsfeld und es existieren viele verschiedene Formen, die sich in ihrer Zielsetzung, der Art der modellierten Maschine sowie der zur Modellierung eingesetzten Technik unterscheiden.

Von IBM wurde eine virtuelle Maschine definiert, die eine vollständig isolierte Kopie der unterlagerten physischen Maschine darstellt. Auf diese Weise sollen sich Anwendungen und Betriebssysteme bei der Ausführung in der virtuellen Maschine exakt so verhalten, als werden diese auf einer realen Maschine ausgeführt.

Als Virtual Machine Monitor (VMM) werden die Softwarekomponenten bezeichnet, die zusammen eine virtuelle Maschine bereitstellen. Häufig wird die VMM auch als Hypervisor bezeichnet. Die VMM hat die Möglichkeit, eine Abstraktion der Maschine vorzunehmen. Diese Abstraktion ist mehrfach instanziierbar und es können somit auf einer einzelnen physischen Maschine mehrere virtuelle Maschinen zur Verfügung gestellt werden, in denen unterschiedliche Betriebssysteme und Programme gleichzeitig ausgeführt werden. Die Betriebssysteme und Programme in den virtuellen Maschinen werden als Gäste (*englisch: guests*) bezeichnet.

Die Verteilung von Ressourcen wie die Ein-/Ausgabegeräte, den Arbeitsspeicher, aber auch die zur Verfügung stehende Rechenzeit auf dem Prozessor oder den Prozessoren wird durch die VMM gesteuert. Wenn die VMM so gestaltet ist, dass die einzelnen Gäste exklusiv auf eine Untermenge der Ressourcen zugreifen können und es sichergestellt ist, dass kein anderer Gast auf die Ressourcen des anderen zugreifen kann, lassen sich die einzelnen Gäste von einander derart isolieren, als würden sie auf jeweils eigenen physikalischen Maschinen ausgeführt [81, S. 43]. Mit Hilfe dieser Isolierung kann die Rückwirkungsfreiheit oder auch Freedom From Interference sichergestellt werden.

Grundsätzlich können die Ansätze der Virtualisierung in die Voll- und Para-Virtualisierung unterteilt werden. Diese Konzepte können jeweils in Software implementiert werden, auch wenn keine Hardwareunterstützung zur Verfügung steht. Durch eine Unterstützung in der Hardware besteht jedoch die Möglichkeit, den benötigten Overhead für die VMM zu reduzieren [11, S. 36].

Im Automotive-Bereich können mit Hilfe der Virtualisierung unterschiedliche Funktionen auf einem Steuergerät zusammengefasst werden. Die einzelnen Funktionen können dabei teilweise sehr unterschiedliche Anforderungen an das Betriebssystem oder auch an den Entwicklungsprozess haben. Zusammen mit den heterogenen oder auch homogenen Multicore-Prozessoren können auf diese Weise einzelne Steuergeräte zusammengefasst werden. Man spricht dann von Hochintegration.

Hochintegration

Hochintegration ist ein Trend, der sich seit Jahren in der Automobilindustrie vollzieht. Anders als bei der Hardware, bei der man von Very Large Scale Integration (VLSI) spricht und darunter die Integration von immer mehr Transistoren und Funktionen auf einem *Die* versteht, geht es bei der Architekturarbeit darum, Funktionen auf eine Hochintegrationsplattform zu bringen. Ein Beispiel dafür sind die Domänenleitrechner, in denen viele Funktionen einer Fahrzeugdomäne zusammengefasst werden, nicht zuletzt, um Steuergeräte einzusparen [31].

2.2 Sicherheitskritische Systeme

Sicherheitskritische Systeme sind Systeme, bei denen der Verlust des Lebens oder eine Umweltkatastrophe vermieden werden muss [44, S. 3]. Beispiele dafür sind Steuerungssysteme von Kernkraftwerken, computergesteuerte Strahlungstherapiegeräte, Herzschrittmacher, Flugsteuerungssysteme, sowie einige Steuer- und Regelsysteme im Automobil. Bei den sicherheitskritischen Systemen sind deswegen Fehlerfälle zu vermeiden, also Situationen, in denen sich das System nicht gemäß seinen definierten Spezifikationen verhält. Um die Fehlerfälle zu vermeiden, gibt es grundsätzlich zwei Vorgehensweisen. Eine Vorgehensweise ist der Versuch, sämtliche Ursachen für mögliche Fehler während der Entwicklungszeit (*englisch: design-phase*) zu identifizieren und zu eliminieren. Dabei kann dieses Ziel durch einen sorgfältigeren Entwurf, eine erhöhte Anzahl von Tests und Verbesserungen zur Inbetriebnahme, Verwendung von geeigneteren Materialien und verbesserten Herstellungsverfahren angestrebt werden. Diese Vorgehensweise wird als Fehlervermeidung (*englisch: fault-avoidance*) bezeichnet. Bei komplexeren Systemen ist dieses sehr aufwendig und kostspielig und wird durch physikalische Gesetze sowie begrenzt vorhandene Zeit und Mittel limitiert. Hardwaredefekte aufgrund von Alterungsprozessen sind beispielsweise oft nicht vermeidbar. Eine zweite Vorgehensweise ist die Fehlertoleranz (*englisch: fault-tolerance*) bei der das System so ausgelegt wird, dass es Fehler toleriert und auch im Falle eines auftretenden Fehlers seine spezifizierte Funktionalität oder zumindest eine reduzierte Funktionalität aufrechterhält. Häufig findet sich eine Kombination aus beiden Vorgehensweisen, insbesondere dann, wenn die geforderten quantitativ messbaren Kenngrößen, wie zum Beispiel die Auftrittshäufigkeit von Fehlern für das Gesamtsystem, nicht durch die einzelnen Komponenten erreichbar ist [46, S. 9].

Im Automobilbereich, aber auch in anderen Industriebereichen, werden weitere Begriffe zur Untergliederung des Begriffs der Fehlertoleranz verwendet. Die einzelnen Begriffe geben an, inwieweit das System im Falle eines auftretenden Fehlers seine Funktionalität aufrechterhält. Die Aufrechterhaltung der vollen Funktionalität wird als Fail-Operational bezeichnet. In der nächsten Stufe wird eine Verringerung der Funktionalität als fail-soft bezeichnet. Wenn das System in einen sicheren Zustand überführt wird, von dem keine Gefahr ausgeht, dann wird diese Stufe der Fehlertoleranz als Fail-Safe bezeichnet. Bezogen auf den Anwendungsfall, werden die Anforderungen der jeweiligen Funktion an ein fehlertolerantes Verhalten während der Entwicklungszeit

spezifiziert. Bei einem fliegenden Flugzeug ist der Übergang in einen sicheren Zustand für das Gesamtsystem nicht möglich, da es unter allen Umständen manövrierbar sein muss. In der Tabelle 2.1 sind die Stufen nach [63] dargestellt.

Tabelle 2.1: Stufen der Fehlertoleranz nach [63]

Fehlertoleranzstufe	Bezeichnung
volle Fehlertoleranz	Fail-Operational
verringerte Leistungsfähigkeit	Fail-Soft, Graceful-Degredation
Übergang in einen sicheren Zustand	Fail-Safe

Um die Praxistauglichkeit eines Systems für bestimmte Aufgaben beurteilen zu können, werden häufig die Begriffe Zuverlässigkeit, Verfügbarkeit und Sicherheit verwendet, die im Folgenden Unterabschnitt erläutert werden.

2.2.1 Kenngrößen von Sicherheitskritischen Systemen

Die Hardware-Komponenten, aus denen technische Systeme bestehen, sind physikalischen Gesetzen unterworfen und Umwelteinflüssen ausgesetzt. Beide haben wiederum Einflüsse auf die Hardware-Komponenten in Form von Materialermüdung, Korrosion, Alterung, sowie auf Materialreinheit und die Fertigungsprozesse. Somit ist davon auszugehen, dass absolute Sicherheit bei Nichtvorhandensein jeglicher Gefahr auszuschließen ist. Es müssen also Regelungen getroffen werden, um technische Systeme hinsichtlich ihrer Sicherheit bewerten zu können. Nach [44, S. 6] ergibt sich folgende Definition:

Definition 2.20 (Zuverlässigkeit) *Die Zuverlässigkeit $R(t)$ (englisch: reliability) eines Systems zum Zeitpunkt t ist die Wahrscheinlichkeit, dass das System ohne Ausfälle in dem Intervall $[0, t]$ arbeitet, unter der Annahme, dass das System zum Zeitpunkt $t = 0$ korrekt funktioniert hat.*

Die Zuverlässigkeit oder Ausfallsicherheit ist eine Messgröße für die kontinuierliche Erbringung eines korrekten Services. Hohe Zuverlässigkeit wird in Situationen benötigt, in denen ein System beispielsweise ohne Unterbrechung funktionieren muss. Ein typisches Beispiel dafür ist der Herzschrittmacher. Eine andere Situation tritt auf,

wenn das System für Wartungszwecke nicht erreichbar ist, wie es bei Anwendungen für den fernen Weltraum der Fall ist. Der Begriff Zuverlässigkeit kann im Englischen auch mit dependability übersetzt werden. Der Begriff dependable bedeutet im deutschen wiederum auch verlässlich oder betriebssicher und wird somit für die Beschreibung technischer Systeme verwendet [92].

Die Fehlerwahrscheinlichkeit ist beschrieben mit $F(t) = 1 - R(t)$ und ist komplementär zu der Zuverlässigkeit. Sie gibt die Wahrscheinlichkeit an, mit der ein System in dem Intervall $[0, t]$ ausfällt. Dabei ist $f(t)$ die momentane Änderung von $F(t)$ und $\lambda(t)$ die Ausfallrate. Diese gibt die bedingte Wahrscheinlichkeit an, mit der ein System ausfällt, welches bis zum Zeitpunkt t funktioniert hat. Die Ausfallrate ergibt sich somit als:

$$\lambda(t) = \frac{f(t)}{R(t)} = \frac{f(t)}{1 - F(t)} \tag{2.1}$$

Wird nun angenommen, dass die Ausfallrate konstant ist mit $(\lambda(t) = \lambda)$, so gilt:

$$F(t) = 1 - e^{-\lambda t} \tag{2.2}$$

Daraus ergibt sich, dass die Zuverlässigkeit gegeben ist durch:

$$R(t) = e^{-\lambda t} \tag{2.3}$$

Neben dem Begriff der Zuverlässigkeit wird der Begriff der Verfügbarkeit benötigt. Dieser ist nach [44, S. 7] definiert als:

Definition 2.21 (Verfügbarkeit) *Die Verfügbarkeit $A(t)$ kann als das Verhältnis der Zeit, in der das System in dem Intervall $[0, t]$ funktioniert, zum gesamten Intervall interpretiert werden.*

Während die Zuverlässigkeit die Wahrscheinlichkeit für einen Ausfall angibt, bezieht sich Verfügbarkeit auf den zeitlichen Anteil, in welchem die Einheit zur Benutzung bereitsteht. Dabei werden auch die Reparaturzeiten berücksichtigt. Für eine Zuordnung von Verfügbarkeit und Ausfallzeit sind die entsprechenden Werte in Tabelle 2.2 dargestellt.

Auf lange Zeit ist A definiert als das Verhältnis von der durchschnittlichen Zeit bis zu einem Ausfall, der Mean Time To Failure (MTTF), zu der durchschnittlichen Zeit zwischen zwei aufeinanderfolgenden Ausfällen, der Mean Time Between Failures (MTBF).

Tabelle 2.2: Verfügbarkeit und die zugehörigen Ausfallzeiten nach [44]

Verfügbarkeit [%]	Ausfallzeit
90	36.5 Tage/Jahr
99	3.65 Tage/Jahr
99.9	8.76 h/Jahr
99.99	52 min/Jahr
99.999	5 min/Jahr
99.9999	31 s/Jahr

Die MTBF wiederum ist die Summe aus der MTTF und der durchschnittlichen Zeit zur Reparatur eines Fehlers, der MTTR[17] [87].

$$A = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR} \tag{2.4}$$

Die MTTF kann aus der Wahrscheinlichkeitsdichte $f(t)$ ermittelt werden. Es gilt [87]:

$$MTTF = \int_0^\infty t \cdot f(t)dt \tag{2.5}$$

Durch Umformungen und der Annahme von einem konstanten $\lambda(t)$, ergibt sich

$$MTTF = \int_0^\infty R(t)dt = \int_0^\infty e^{-\lambda t}dt = \frac{1}{\lambda}. \tag{2.6}$$

Wenn Abwesenheit von Gefahr für Menschen und Wertobjekte besteht, dann wird dieses als Sicherheit bezeichnet. Sicherheit ist in der deutschen Sprache doppelt belegt und kann mehrfach interpretiert werden. Zum einen gibt es den Begriff der Angriffssicherheit (*englisch: security*) und zum anderen den Begriff der Betriebssicherheit oder auch funktionalen Sicherheit (*englisch: safety*). Die Betriebssicherheit bezieht sich auf den Schutz seiner Umgebung vor einer Sache, also ein System, das sich entsprechend seiner

[17] Mean Time To Repair

Spezifikationen verhält und von dem keine nicht tolerierbare Gefährdung für Mensch und Natur ausgeht. Angriffssicherheit bezieht sich auf die Fähigkeit eines Systems, sich gegen böswillige Eingriffe von außen zu schützen. In dieser Arbeit wird der Begriff Sicherheit unter dem Aspekt der Betriebssicherheit verwendet. Üblicherweise wird zusammen mit dem Begriff der Sicherheit ebenfalls eine Aussage darüber geliefert, worauf sich die Sicherheit bezieht. Alternativ wird angegeben, wovor die betrachtete Sache oder der betrachtete Sachverhalt sicher ist [66, S. 42].

2.2.2 Einflussfaktoren für Sicherheitskritische Systeme

In Abschnitt 2.2.1 wurde bereits über Ausfallraten und die Zuverlässigkeit als Messgröße der Wahrscheinlichkeit für einen Ausfall eines Systems geschrieben. In diesem Abschnitt werden die Begriffe dafür genauer beschrieben. Ein Ausfall eines Systems basiert auf Fehlern. Der Begriff Fehler hat wie der zuvor eingeführte Begriff der Sicherheit eine Mehrfachbedeutung und muss detaillierter beschrieben werden. Dafür finden sich im Englischen die Begriffe fault - error - failure. Nach [66, S. 48] und [93] werden folgende Definitionen verwendet:

Definition 2.22 (Fehlerursache) *Eine Fehlerursache (englisch: fault) ist der Auslöser eines Fehlerzustandes.*

Definition 2.23 (Fehlerzustand) *Ein Fehlerzustand (englisch: error) ist ein Systemzustand, der dafür verantwortlich ist, dass eine Fehlerauswirkung bzw. ein Ausfall auftritt. Die Fehlerursache wird im System offenbar.*

Definition 2.24 (Fehlerauswirkung oder Ausfall) *Ein Ausfall oder eine Fehlerauswirkung (englisch: failure) ist die Abweichung der erbrachten Leistung von der in der Systemspezifikation geforderten Leistung. Ein Ausfall einer Komponente kann die Fehlerursache eines übergeordneten Systems sein.*

Die Art und Weise, in der ein System ausfallen kann, wird als failure-mode bezeichnet. Es gibt verschiedene Arten, wie ein System ausfallen kann und diese müssen während der Entwicklungsphase identifiziert werden. So kann z.B. ein Schloss in der Art ausfallen, dass damit nicht mehr abgeschlossen werden kann oder aber, dass damit nicht

mehr geöffnet werden kann. Diese failure-modes werden klassifiziert. Dabei werden sie eingeteilt basierend auf ihrer Domäne (Wert- oder Timing-Ausfälle), basierend auf ihrer Wahrnehmung durch den Benutzer des Systems (konsistente oder inkonsistente Ausfälle) und basierend auf Ihrer Auswirkung auf die Umwelt (unbedeutend, bedeutend und katastrophal). Die tatsächliche oder erwartete Auswirkung eines Ausfalls wird als failure-effect bezeichnet. Im Entwicklungsprozess von sicherheitskritischen Systemen werden die failure-effects analysiert und Gegenmaßnahmen werden dafür identifiziert [44, S. 10]

Eine Fehlerursache kann unterschiedlichen Ursprungs sein. Nach [44, S. 10] werden diese in die vier folgenden Kategorien eingeteilt: inkorrekte Spezifikation, inkorrekte Implementierung, Herstellungsfehler und externe Faktoren.

Zusätzlich ist der Begriff der Common-Cause-Failures einzuführen. Diese beschreibt die Ursache für das gleichzeitige Ausfallen von redundanten Komponenten. Ein typisches Beispiel dafür ist eine gemeinsame Stromversorgung zweier zueinander redundanter Komponenten. Fällt die Stromversorgung aus, so fallen beide Komponenten aus. Bei der Software kann es ebenfalls zu dieser Fehlerursache kommen. Gegenmaßnahmen sind unter dem Begriff der diversitären Entwicklung zusammengefasst. Wirkliche diversitäre Entwicklung benutzt dabei nicht nur unterschiedliche Software-Entwurfspattern auf höherer Abstraktionsebene, sondern basiert auf unterschiedlichen Entwicklungsteams, unterschiedlichen Entwurfsvorgaben und unterschiedlichen Software-Tools, um die Abhängigkeiten zwischen den einzelnen redundanten Software-Komponenten zu beseitigen.

Weiterhin müssen die Fehlerursachen in transient und permanent unterschieden werden. Permanente Fehlerursachen bleiben aktiv, bis eine Gegenmaßnahme eingeleitet wurde. Beispiele dafür sind Kurzschlüsse oder gebrochene Verbindungen aufgrund von physikalischen Hardwaredefekten. Transiente Fehlerursachen (*englisch: soft-error*) bleiben nur für ein kurzes Zeitintervall aktiv. Wenn die Fehler nur wiederholt auftreten, werden diese auch als intermittierend bezeichnet. Aufgrund der kurzen Auftretensdauer werden transiente Fehlerursachen häufig durch den Fehlerzustand entdeckt, der durch sie propagiert wird. Dabei ist diese Fehlerursache die dominierende bei heutigen ICs[18]: 98 % aller Fehlerursache sind bei Random Access Memory (RAM)

[18] *englisch*: *Integrated Circuit*, integrierter Schaltkreis

transiente Fehler. Die Gründe hierfür sind in der Regel umweltbedingte Einflüsse wie Alpha-Strahlung, atmosphärische Neutronen, elektrostatische Entladung, Spannungsschwankungen und Überhitzung.

Neben den hardwarebasierten gibt es auch softwarebasierte Fehlerursachen: Unter der Annahme, dass Software nicht altert, sind Entwicklungsfehler die Hauptursache für softwarebasierte Fehlerursachen und somit auf den Menschen zurückzuführen. Eine mögliche Gegenmaßnahme, um diese Fehlerursachen einzugrenzen ist die modellbasierte Softwareentwicklung.

Eine sehr detaillierte Auseinandersetzung zu diesem Thema ist in [5] zu finden.

2.2.3 Verfahren für Sicherheitskritische Systeme

Es gibt verschiedene Methoden und Techniken, die bei der Entwicklung von sicherheitskritischen Systemen zum Einsatz kommen. Fehlertoleranz ist eine davon, die in Kombination mit einer oder mehreren anderen Methoden und Techniken wie zum Beispiel der Fehlerprävention, der Fehlerentfernung und der Fehlervorhersage Verwendung findet.

Bezugnehmend auf die Fehlertoleranz ist deren oberstes Ziel die Entwicklung von betriebssicheren Systemen, die korrekt nach ihrer Spezifikation arbeiten, obwohl Fehlerursachen in dem System vorhanden sind. Nach [80] ist Fehlertoleranz definiert als:

Definition 2.25 (**Fehlertoleranz**) *Fehlertoleranz ist die Fähigkeit eines Systems seine bestimmungsgemäßen Funktionen in Gegenwart von Fehlern weiterhin auszuführen.*

Mit dem Begriff der Fehlertoleranz ist im weitesten Sinne die Zuverlässigkeit, der erfolgreiche Betrieb und das Fernbleiben von technischen Defekten verknüpft. Ein fehlertolerantes System sollte in der Lage sein, Fehler in einzelnen Hardware- oder Softwarekomponenten, Leistungsausfall oder andere unerwartete Fehler zu beherrschen und dabei immer noch gemäß seiner Spezifikation zu funktionieren.

Fehlertoleranz ist deswegen so wichtig, weil es im Prinzip unmöglich ist, ein perfektes System zu bauen. Das Hauptproblem dabei ist folgende Systematik: je komplexer das

System wird, desto geringer wird seine Zuverlässigkeit, sollten keine Kompensationsmaßnahmen getroffen werden. Besteht ein System beispielsweise aus 100 nicht-redundanten Komponenten und jede einzelne Komponente hat eine Zuverlässigkeit von 99.99 %, dann hat das Gesamtsystem eine Zuverlässigkeit von 99.01 %. Besteht das System hingegen aus 10.000 nicht-redundanten Komponenten mit einer jeweiligen Zuverlässigkeit von 99.99 %, dann liegt die Gesamtzuverlässigkeit bei lediglich 36.79 %.

In den meisten Anwendungen ist eine derartige niedrige Zuverlässigkeit inakzeptabel. Liegt die Anforderung an die Gesamtzuverlässigkeit an ein System mit 10.000 Komponenten bei 99 %, dann muss jede einzelne Komponente eine Zuverlässigkeit von 99.999 % erreichen. Diese Anforderung führt jedoch zu einem enormen Kostenanstieg.

Ein weiteres auftretendes Problem ist die Tatsache, dass Hardware- und Softwarefehler nicht vollständig beseitigt werden können, bevor das System auf den Markt kommt. Es ist unumgänglich, dass in der Entwicklungsphase bestimmte Umgebungseinflüsse nicht berücksichtigt wurden oder bestimmte Kundenfehlbedienungen nicht vorhergesehen wurden. Auch wenn ein System perfekt entworfen und implementiert wurde, ist es wahrscheinlich, dass ein Fehler aufgrund von äußeren Einflüssen hervorgerufen wird, den der Entwickler nicht unter seiner Kontrolle hat.

Fehlertoleranz kann auf verschiedene Weisen erreicht werden. Eine davon ist der Einsatz von Redundanz. Redundanz kann dann wiederum eingeteilt werden in strukturelle Redundanz (*englisch: space-redundancy*) und Zeitredundanz (*englisch: time-redundancy*). Erstere wird erreicht durch das Hinzufügen von Komponenten, während letztere durch wiederholtes Ausführen von Anweisungen erreicht wird. Strukturelle Redundanz wird wiederum weiter unterteilt in die Hardware-Redundanz, die Software-Redundanz und die Informations-Redundanz [44, S. 1 f.].

Neben der Redundanz sind noch weitere Anteile bei der Entwicklung eines fehlertoleranten Systems notwendig. Dazu gehören die Fehlerdiagnose, die Fehlerspezifikation und die Fehlerbehandlung. Bei der Fehlerbehandlung kann dann wiederum unterschieden werden in die Fehlermaskierung, die Fehlerbehebung und die Fehlerkompensation [46, S. 15].

2.2.4 Redundanz bei Sicherheitskritischen Systemen

Nach [46, S. 49] ist Redundanz wie folgt definiert:

Definition 2.26 (Redundanz) *Redundanz bezeichnet das funktionsbereite Vorhandensein von mehr technischen Mitteln, als für die spezifizierten Nutzfunktionen eines Systems benötigt werden (die Fehlertoleranz-Fähigkeit selbst wird in diesem Zusammenhang nicht als eigentliche Nutzfunktion eines Systems angesehen).*

Eine weitere Unterteilung der Redundanz erfolgt dann in die technischen Mittel, die für die Redundanz genutzt werden und die Aktivierung der Redundanz. Wie bereits in Abschnitt 2.2.3 erwähnt, gibt es bei den Mitteln die strukturelle Redundanz, die zeitliche Redundanz, die Informationsredundanz und die funktionelle Redundanz. Bei der Aktivierung der Redundanz wird unterschieden in die statische Redundanz und die dynamische Redundanz.

Definition 2.27 (statische Redundanz) *Statische Redundanz bezeichnet das Vorhandensein von redundanten technischen Mitteln, die während des gesamten Einsatzzeitraums aktiv zu den zu unterstützenden Funktionen beitragen (soweit diese redundanten Mittel nicht selbst fehlerhaft sind).*

Die statische Redundanz ist im Allgemeinen durch eine erhöhte Anzahl von Komponenten zur Erbringung einer Funktion, einem n-von-m-System dargestellt: Dieses stellt durch Mehrheitsentscheid die korrekte Funktionsausführung sicher. Alle Komponenten dieses Systems berechnen ein Ergebnis. Stimmt dieses Ergebnis mehrheitlich überein, wird von der Korrektheit des Ergebnisses ausgegangen. Dieser Vorgang wird auch als Fehlermaskierung bezeichnet und führt somit ein fehlerfreies Ergebnis herbei. Zugleich wird durch dieses System die Fehlererkennung mit durchgeführt [46].

Anders verhält es sich bei der dynamischen Redundanz, die wie folgt definiert ist:

Definition 2.28 (dynamische Redundanz) *Das Vorhandensein von redundanten technischen Mitteln, die erst im Ausnahmebetrieb (d. h. nach Auftreten eines Fehlers) aktiviert werden, um zu den zu unterstützenden Funktionen beizutragen, kennzeichnet die dynamische Redundanz.*

Bei der dynamischen Redundanz wird ebenfalls auf redundante Strukturen gesetzt. Diese sind jedoch nicht gleich zu Betriebsbeginn aktiv an dem Ergebnis beteiligt. Deswegen wird zwischen Primär- und Sekundärkomponenten unterschieden. Wird ein Fehler diagnostiziert, wird von der Primärkomponente auf die Sekundärkomponenten umgeschaltet. Die Dauer der Umschaltung hängt von den vorbereitenden Maßnahmen ab. Aus diesem Grund wird zwischen heißer und kalter Reserve (*englisch: hot-standby* und *cold-standby*) unterschieden. Bei der kalten Reserve sind die Komponenten ausgeschaltet und müssen erst aktiviert werden, wohingegen die Komponenten bei der heißen Reserve immer aktiv sind. Diese führt in den meisten Situationen zu einer verkürzten Umschaltzeit, kann aber ggf. die Lebensdauer der Komponenten verkürzen [46, S. 63].

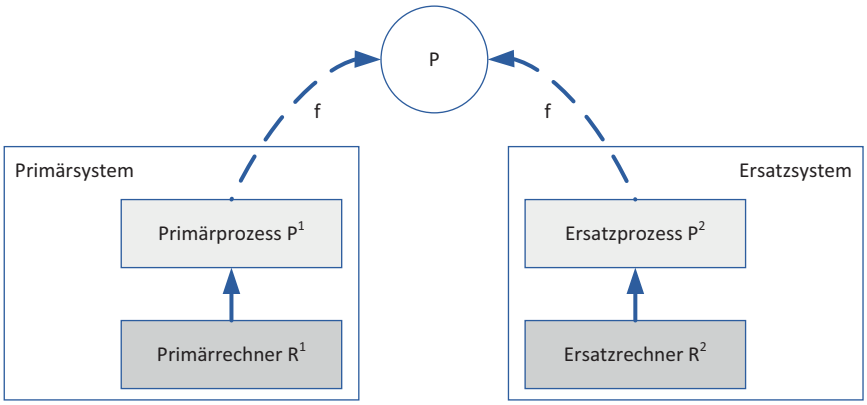

Abbildung 2.6: Dynamische, strukturelle Redundanz nach [46]

Neben der statischen und der dynamischen Redundanz gibt es auch die hybride Redundanz, die als Rekonfiguration bezeichnet wird. Diese ist eine Kombination aus der dynamischen Redundanz und der statischen Redundanz [46, S. 65]. Abbildung 2.6 zeigt die dynamische, strukturelle Redundanz bei der Hardware-Komponenten zum Beispiel durch Schalter ein- und ausgegliedert werden können. Definiert ist sie als:

Definition 2.29 (Hybridredundanz) *Eine Hybridredundanz bezeichnet die dynamische Veränderung (Rekonfigurierung) eines statisch redundanten Mittels.*

Aufbauend auf den bisherigen Definitionen ist die Rekonfiguration eine Kombination aus dynamischer und statischer Redundanz. Die Rekonfiguration benötigt stets die

dynamische strukturelle Redundanz. Ein Rekonfigurator in Form einer in Hardware und/oder Software implementierten Fehlertoleranz-Instanz verfügt über Möglichkeiten, defekte Komponenten auszugliedern, neue Komponenten einzugliedern und Komponenten zu verlagern. Kommunikation kann durch Ein- oder Ausschalten von Kommunikationsperipherie-Bausteinen vom restlichen Kommunikationsnetz getrennt oder hinzugefügt werden.

2.2.5 Normen für Sicherheitskritische Systeme

Für die funktionale Sicherheit (siehe auch Abschnitt 2.2.1) von Straßenfahrzeugen existiert die internationale Norm ISO 26262 [77]. Sie geht auf die speziellen Anforderungen für Serienentwicklung und Produktion von Straßenfahrzeugen ein und beschreibt einen Sicherheitslebenszyklus für Management, Entwicklung, Produktion, Betrieb, Service und Außerbetriebnahme. Des Weiteren erfolgt eine Einteilung in Phasen, Aktivitäten und Arbeitsergebnisse, die parallel zu den bestehenden Entwicklungs- und Produktlebenszyklen wie dem Automotive-V-Modell betrachtet werden sollen [66, S. 91].

Eine vollständige Darstellung der Kern- und Unterstützungsprozesse findet sich in [77]. Die ISO 26262 stellt eine Spezialisierung der IEC 61508 [73] für das Kraftfahrzeug dar, da sich im Automobil einige Besonderheiten ergeben. So liegt der Fokus der ISO 26262 auf Maßnahmen, um im Produktentstehungsprozess eine Fehlervermeidung zu erreichen. Eine weitere Besonderheit ist die Einführung der ASIL[19], um eine Einstufung von sicherheitskritischen Systemen vornehmen zu können. Das ASIL ist dabei eine Adaption des SIL[20] aus der IEC 61508 für den Automobilbereich. [66, S. 91]

Eine Übersicht der einzelnen Teile der Norm bietet die folgende Liste:

- Teil 1: Glossar (Vocabulary)
- Teil 2: Management der funktionalen Sicherheit
- Teil 3: Konzeptphase
- Teil 4: Produktentwicklung Systemebene

[19] Automotive Safety Integrity Level
[20] Safety Integrity Level

- Teil 5: Produktentwicklung Hardwareebene
- Teil 6: Produktentwicklung Softwareebene
- Teil 7: Produktion und Betrieb
- Teil 8: Unterstützende Prozesse
- Teil 9: ASIL- und sicherheitsorientierte Analysen
- Teil 10: Orientierungshilfen (informativ)

In der Abbildung 2.7 sind die einzelnen Bestandteile dargestellt. Besonders erwähnenswert sind dabei die Kern- (Prozessnummer 4 bis 6) und die Unterstützungsprozesse (Prozessnummer 8) (siehe hierzu auch Abschnitt 2.1.3).

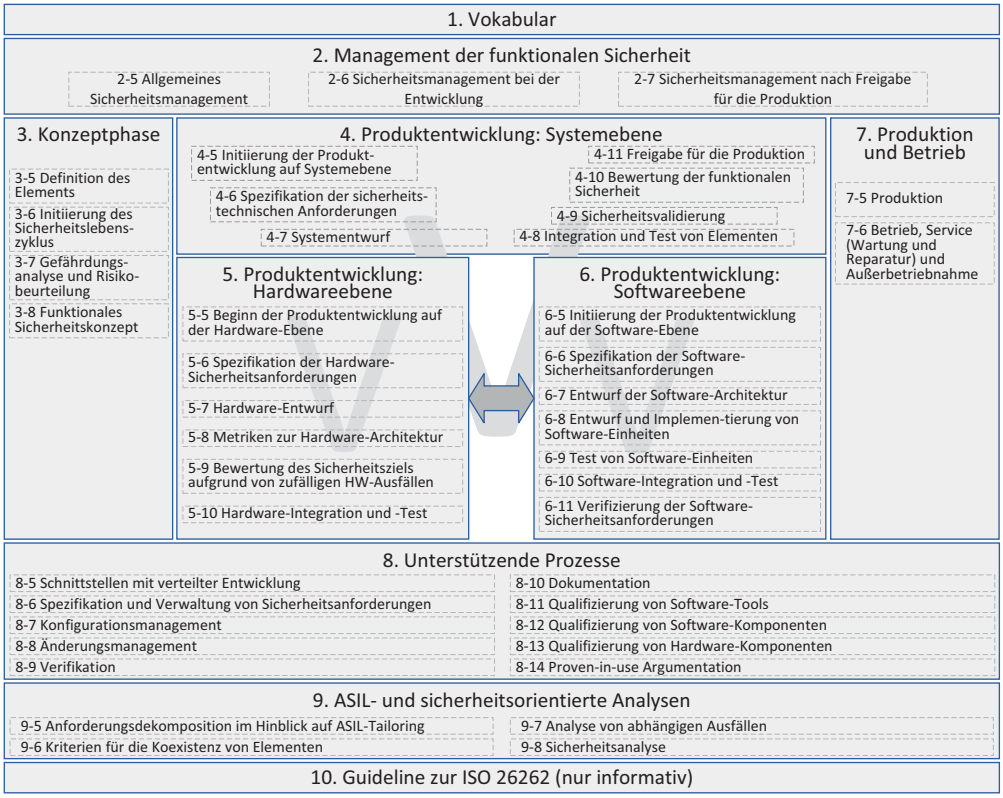

Abbildung 2.7: Übersicht der ISO 26262 nach [69]

Teil 3 der ISO-Norm beschreibt, wie eine Gefahren- und Risiko(GuR)-Analyse durchzuführen ist. Abgeleitet von der Klassifizierung von Gefährdungen in sogenannten ASIL, werden Sicherheitsziele sowie funktionale Sicherheitsanforderungen an das betrachtete System oder seine Teilsysteme definiert [66, S. 92]. Nach der Definition der ISO 26262 Klausel 7.12 wird eine ausreichende Freedom From Interference erreicht, in dem nur die Abwesenheit von kaskadierenden Fehlern nachgewiesen werden kann.

2.3 Fazit

Autonomes Fahren ist ein klar definiertes Ziel im Automobilbereich. Der Weg dorthin wird anhand der SAE-Level beschrieben. Um das Ziel zu erreichen und gleichzeitig wettbewerbsfähig zu bleiben, müssen weitere technische Möglichkeiten hinsichtlich ihrer Umsetzbarkeit im Automobil untersucht werden. Die Möglichkeiten setzen sich dabei aus den Konzepten und den zugehörigen Methoden zusammen. Die Zielsetzung dieser Arbeit ist es, die Hochintegration auf zentralen Rechnerplattformen für sicherheitskritische Funktionen mit Fail-Operational-Verhalten zu ermöglichen. Gleichzeitig soll dabei die strukturelle Redundanz im E/E-Gesamtsystem unter Zuhilfenahme von dynamischer Rekonfiguration minimiert werden. Dazu wird nach Abschluss der Arbeit das Arbeitsergebnis in Form von Lösungsbausteinen vorliegen. Neben den technischen Lösungsbausteinen sind die Entwicklungsprozesse für die Automobilindustrie stets von entscheidender Rolle. Um dynamische Rekonfiguration zu etablieren, muss also der Entwicklungsprozess von sicherheitskritischen Systemen, die auf dynamischer Rekonfiguration basieren, in den E/E-Entwicklungsprozess im Automobilsektor integriert werden.

Während der Konzeptentwicklungsphase in der Serienentwicklung von sicherheitskritischen Systemen sind Entwicklungs- und Konzeptingenieure mit der Aufgabe betreut, für eine Problemstellung das richtige Konzept auszuwählen. Dabei stellt sich die Frage, welches das geeignete System und das richtige Konzept für eine Anwendung ist. Um zu einer Entscheidung zu kommen, müssen Kategorien, Bewertungskriterien und Bewertungseinheiten von dynamisch rekonfigurierbaren Systemen aufgestellt werden, um unterschiedliche Konzepte evaluieren zu können. Verschiedene Kriterien fließen in diese Bewertungsmöglichkeit mit ein. Diese bestehen zum Beispiel aus den Garantien,

die die Systeme zur Verfügung stellen und bis zu welchem Umfang eine Qualifizierung für sie möglich ist. Die aus dem autonomen Fahren resultierende Anforderung von Funktionen an ein Fail-Operational-Verhalten ist ein weiteres Kriterium, das für dynamisch rekonfigurierbare Systeme überprüft werden muss. Es ist zu bewerten, ob dynamisch rekonfigurierbare Systeme diese Anforderung erfüllen können und bis zu welchem Grad.

Neben den Entwicklungsprozessen dienen Normen und Standards zur Vereinfachung und Vereinheitlichung von Entwicklungen. Für die Entwicklung von sicherheitskritischen Systemen ist die ISO 26262 derzeit der Standard. In dieser Norm ist jedoch die Technik der dynamischen Rekonfiguration für ASIL D-Systeme auf „nicht empfohlen“ eingestuft. Die ISO 26262 rät somit von dem Einsatz dynamischer Rekonfiguration für sicherheitskritische Konzepte ab. Es muss demnach ein Weg gefunden werden, wie dynamische Rekonfiguration im Automobil für sicherheitskritische Systeme prozessseitig zum Einsatz kommen kann. Für nachfolgende Versionen der ISO 26262 sind dynamisch rekonfigurierbare Systeme in den Standard zu integrieren.

Der Einsatz von AUTOSAR Adaptiv oberhalb der Virtualisierungsebene ist ein zentraler Punkt im Automobilbereich: Aufgrund von Hochintegration und dem Einsatz von Domänenleitrechnern kann zunehmend eine Zentralisierung durchgeführt werden, die auf Gesamtsystemebene zu Komplexitäts- und Kostenreduktion führen kann. AUTOSAR Adaptiv kommt dann als Gastbetriebssystem auf den Hochintegrationsplattformen zum Einsatz.

Dynamische Rekonfiguration für harte Echtzeitsysteme wird derzeit im Automobil weder für sicherheitskritische Systeme noch für nicht-sicherheitskritische Systeme eingesetzt. Aus diesem Grund wird im Rahmen dieser Arbeit eine Methode und eine Systematisierung zum Vorgehen in der Entwicklung von dynamisch rekonfigurierbaren Systemen für sicherheitskritische Systeme im Automobilbereich erarbeitet und der Bezug zu der ISO 26262 hergestellt. Dabei wird die Möglichkeit zur Modellierung verwendet. Weiterhin wird eine Bewertungsmöglichkeit erarbeitet, bei der die einzelnen Kriterien in eine Metrik einfließen und so einen Teil der Methode bilden. Es wird ein Konzept erarbeitet, um dynamische Rekonfiguration oberhalb der Virtualisierungsschicht auf High-Performance-Computern umzusetzen. Abschließend wird eine Bewertung für das implementierte Konzept anhand der erarbeiteten Kriterien durchgeführt.

Die Grundlagen sind hergestellt, um eine systematische, ingenieurmäßige Bearbeitung der Thematik durchzuführen. Ziel ist es aufzuzeigen, inwieweit dynamische Rekonfiguration für harte Echtzeitsystem im Fail-Operational-Kontext einsetzbar ist. Mithilfe der Systemabstraktionsebenen kann eine Einordnung der Techniken erfolgen. Weiterhin unterstützt die Kategorisierung bei der Vereinfachung, um dieses komplexe Thema zu bearbeiten. Im nächsten Kapitel wird zunächst der Stand der Technik exploriert und in Bezug zu diesem Vorhaben gesetzt.

3 Stand der Technik und Forschung

Nachdem zunächst die Motivation für diese Arbeit dargelegt wurde und die Grundlagen für den weiteren Verlauf geschaffen wurden, wird in diesem Kapitel auf den Stand der Technik eingegangen. Es erfolgt ein Überblick über relevante Themen und Vorarbeiten, die in den bereits aufgezeigten Zusammenhang gebracht werden. Abschließend werden die identifizierten offenen Bereiche aufgezeigt und gegen den dargelegten Stand der Technik abgegrenzt. Zunächst wird darauf eingegangen, in welchem Kontext die Arbeit entstanden ist, um ein Verständnis für die weiteren Ausarbeitungen zu schaffen.

3.1 Einbettung in das Forschungsvorhaben

Die vorliegende Arbeit wurde im Rahmen eines Forschungsförderprojekts erstellt. Das Projekt und dessen Ziele werden im Folgenden aufgezeigt.

3.1.1 Anforderungen und Systemkontext

Aufgrund der Entwicklung im Automobilbereich in Richtung des hoch- und vollautomatisierten Fahrens, steht bei der Entwicklung der gesamten E/E-Architektur im Kraftfahrzeug ein Paradigmenwechsel von störungssicheren (*englisch: fail-safe*) hin zu fehlertoleranten (*englisch: fault-tolerant*) Komponenten an. In diesen zukünftigen Betriebsmodi ist der Fahrer nicht immer sofort in der Lage die Kontrolle über das Fahrzeug im Fehlerfall zu übernehmen. Vielmehr wird es eine Übergabezeit geben bis der Fahrer die Verantwortung wieder übernehmen kann. Aus diesem Grund sind die für den Fahrbetrieb notwendigen Komponenten ausfallsicher auszulegen. In diesem Ansatz werden die Komponenten betrachtet, die zum Lenken und Bremsen benötigt werden. Dabei ist die Verdopplung dieser Komponenten wirtschaftlich und technisch

nicht immer effizient und die notwendige Redundanz soll daher mit rekonfigurierbaren Steuergeräten erreicht werden. Es wird insbesondere die Gestaltung zukünftiger Steuergeräteplattformen und systemübergreifender Lösungen unter Berücksichtigung des Einsatzes von serviceorientierten Architekturen als Basis für hochverfügbare Funktionen verfolgt. Diese Basis beinhaltet das Management von Rückfallebenen und das Umschalten aktiver Pfade, so dass Funktionen auf einen standardisierten Satz von Fail-Operational-Patterns und -Mechanismen zurückgreifen können, bzw. davon befreit werden, funktionsspezifische Mechanismen redundant aufzubauen. In der Software- und in der Hardwarearchitektur ist hierfür eine modulare, parametrierbare Bibliothek für Rekonfigurations- und Rückfallebenen-Management die einen Cross-Layer-Ansatz verfolgt zu entwickeln. Von einem ganzheitlichen Konzept der funktionalen Sicherheit für autonome Fahrfunktionen ausgehend, sind technische Realisierungen abzuleiten. Das Prinzip der Rekonfiguration ist in der Informationstechnologie keine Neuheit. Im Fahrzeug konzentriert sich die Rekonfiguration auf den Einsatz bei Drive-by-Wire-Komponenten. Hierbei werden Fehler erkannt und durch die Deaktivierung eines von zwei redundanten Pfaden isoliert. Eine Untersuchung zur kompletten Funktions-Rekonfiguration von komplexen Komponenten gibt es derzeit nicht (vergleiche [55]).

3.1.2 Forschungs- und Anwendungskontext

Diese Arbeit entstand im Rahmen des vom BMBF[1] geförderten Projekts AutoKonf[2] [57]. Das Buchstabenkurzwort AutoKonf steht für Automatische rekonfigurierbare Aktorikansteuerungen für ausfallsichere automatisierte Fahrfunktionen. Die Motivation, aus der dieses Projekt heraus entstanden ist, beruht auf der Situation, dass die E/E-Komponenten für zukünftige automatisierte Fahrfunktionen anstatt der bisherigen Anforderung an ein störungssicheres (*englisch: fail-safe*) Verhalten eine Anforderung an ein fehlertolerantes (*englisch: fault-tolerant*) Verhalten haben sollen. Dieser Wechsel erfordert eine Veränderung der gesamten E/E-Architektur. Für hoch- und vollautomatisiertes Fahren müssen die beiden sicherheitsrelevanten Systeme von Bremse und Lenkung jeweils redundant ausgelegt werden. Das Projekt AutoKonf [57] zielt darauf ab, den heutigen aus der Avionik übernommen Ansatz zu optimieren. Dieser heutige Ansatz ist in Bezug auf den Einsatz im Automobil hinsichtlich Bauraum, Kosten und

[1] Bundesministerium für Bildung und Forschung
[2] Automatische rekonfigurierbare Aktorikansteuerungen für ausfallsichere automatisierte Fahrfunktionen

Gewicht nicht ideal. Somit wird in der Projektlaufzeit von 2016 bis 2019 ein neues Konzept entwickelt, das diesen Ansatz für den Einsatz im Auto intelligent umgestaltet: Es wird ein Redundanzsteuergerät entwickelt, das im Fehlerfall entweder die Brems- oder die Lenkungsfunktion übernehmen kann. Das Prinzip ist in der Abbildung 3.1 dargestellt.

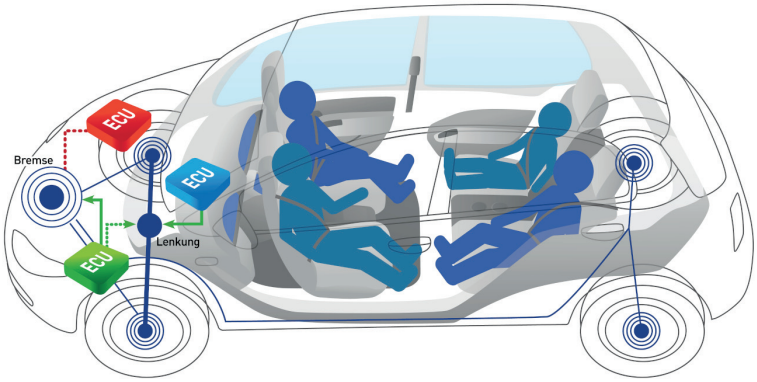

Abbildung 3.1: Schema einer E/E-Architektur zum Bremsen und Lenken, Quelle: [59]

Auf diese Weise wird die Anzahl an redundanten Komponenten reduziert und das Redundanzsteuergerät kann zu Laufzeit (*englisch: run-time*) dynamisch rekonfiguriert werden. Der Anspruch an ein fehlertolerantes System ist somit erfüllt und ein Fail-Operational-Zustand wird erreicht. Dafür wird ein Elektroniksystem entwickelt, das Signale und Stromversorgung dynamisch verteilt. Bei gleichbleibender Sicherheit wird dadurch die Komplexität der elektrischen und elektronischen Systeme für automatisiertes Fahren reduziert. Die entsprechenden Systeme werden günstiger, indem eine einzelne Redundanzkomponente für die Doppelfunktion Bremsen oder Lenken, die beiden Redundanzkomponenten für die jeweiligen einzelnen Funktionen ersetzt. Der Abschlussbericht zu diesem Projekt für das Teilvorhaben seitens BMW[3] ist unter [148] zu finden.

[3] Bayerische Motoren Werke

3.2 Ebenen für die dynamische Rekonfiguration

Zunächst werden die einzelnen Konzepte für dynamische Rekonfiguration analysiert und kategorisiert. Für die Kategorisierung bieten sich die Systemabstraktionsebenen (siehe 2.1.1) an.

3.2.1 Dynamische Rekonfiguration auf Bauteilebene

Fehlertolerante Chip[4]-Architekturen für Anwendungen im Automobilbereich werden in der Arbeit mit dem Titel „Fault-tolerant platforms for automotive safety-critical applications“ [9] beschrieben. Dabei erstellen die Autoren einen Überblick über verschiedene Architekturen, die in anderen Industrien seit mehreren Jahren zum Einsatz kommen und untersuchen, wie diese auf einem einzigen Halbleiterelement integriert werden können. Als Bewertungsmetrik werden die Größen Kosten, Performanz und Fehlerabdeckung sowie die Flexibilität hinsichtlich Breite der Einsetzbarkeit verwendet und für den Einsatz in X-by-Wire-Systemen untersucht. Eine dynamische Rekonfiguration geht aus der Arbeit nicht hervor. Sie bietet jedoch eine geeignete Grundlage, um verschiedene Prozessorarchitekturen zum Einsatz für fehlertolerante Systeme zu untersuchen.

Die Autoren der Arbeit „Fail-safe and fail-operational systems safeguarded with coded processing“ [30] verfolgen den Ansatz durch Software Coded Processing, die Redundanz für Fail-Safe und Fail-Operational-Systeme zu verringern. Dabei werden sowohl Singlecore- als auch Multicore-Prozessoren betrachtet. Das Prinzip des Software Coded Processing ermöglicht dabei die Hardwareredundanz durch diverse Redundanz in Software zu verringern. Es werden bezogen auf die ISO 26262 für die unterschiedlichen ASIL die entsprechenden MTTF angegeben. Sie kommen zu dem Schluss, dass das verwendete Vorgehen für Fail-Safe-Anwendungen geeignet ist, jedoch nicht für Fail-Operational-Anwendungen.

Einen Überblick über rekonfigurierbare Prozessor-Architekturen geben die Autoren in ihrer Arbeit mit dem Titel „Reconfigurable computing architecture survey and introduction“ [6]. Sie gehen dabei auch auf das Thema der dynamischen Rekonfiguration

[4] gemeint ist hier das Halbleiterplättchen

ein. Die Autoren führen den Unterschied zwischen statischer oder auch compile-time-Rekonfiguration und dynamischer oder auch run-time-Rekonfiguration an. Während die statische Rekonfiguration eine Konfiguration des FPGA-Prozessors über die gesamte Dauer der Anwendung vorsieht, verändert die dynamische Rekonfiguration die Hardwarestruktur des FPGAs während der Laufzeit und kann so zur Effizienzsteigerung durch hochoptimierte, anwendungsspezifische Strukturen genutzt werden.

Weiterhin werden die Hauptherausforderungen bei dynamischer Rekonfiguration aufgezeigt. Diese bestehen zum einen darin, dass Anteile abgetrennt werden, die nicht gleichzeitig abgearbeitet werden müssen, und zum anderen darin, die Koordination des Übergangs von einer Konfiguration zur nächsten sowie deren Zustandsübergaben. Der Aspekt der Echtzeitfähigkeit, sowie der Fehlertoleranz, werden in dieser Arbeit nicht betrachtet.

Um der Problematik zu begegnen, die Anforderungen an höhere Rechenleistung und niedrigeren Stromverbrauch zusammen mit geringeren Kosten und kürzeren Markteinführungszeiten zu erfüllen, wird in dem Paper „Dynamic self-reconfiguration of a MIPS[5]-based soft-core processor architecture“ [104] ein neuartiger Ansatz vorgestellt, dynamisch rekonfigurierbare ASPs[6] dafür zu verwenden. In dieser Arbeit wird ebenfalls nicht auf den Aspekt der Fehlertoleranz eingegangen.

Mit der Zielsetzung, die Kosten für Fail-Safe und Fail-Operational-Systeme im Automobilbereich zu minimieren, beschreiben die Autoren in ihrer Arbeit mit dem Titel „A flexible microcontroller architecture for fail-safe and fail-operational systems“ [98] eine Mikroprozessorarchitektur. Dabei vermeiden sie unnötige Redundanzen und beeinträchtigen die Performanz so wenig wie möglich. Die Autoren zeigen weiterhin, wie der Prozessor unter Berücksichtigung der ISO 61508 [73] und ISO 26262 [77] spezifiziert, entworfen und getestet wurde. Die Arbeit berücksichtigt das Thema von Fail-Operational, geht jedoch nicht auf die dynamische Rekonfiguration ein.

Das ARAMiS[7]-Projekt ist in der Zeit von 2011 bis 2015 erfolgreich durchgeführt worden. Das Ziel ist die Untersuchung des Einsatzes der Multicore-Technologie im Automo-

[5] *englisch*: *Microprocessor without Interlocked Pipeline Stages*, Mikroprozessor ohne verschränkte Pipeline-Stufen ist eine Befehlssatzarchitektur im RISC-Stil, die ab 1981 von John L. Hennessy und seinen Mitarbeitern an der Stanford-Universität entwickelt wurde.
[6] *englisch*: *Application Specific Processor*, anwendungsspezifischer Prozessor
[7] Automotive, Railway and Avionics Multicore System

bil. Die in diesem Projekt gewonnenen Erkenntnisse dienen als Grundlage, um eine erfolgreiche Vernetzung von Embedded Systems zu Cyber Physical Systems (CPS) zu ermöglichen.

Eine herausragende Fragestellung ist dabei die optimale Nutzung der Leistungsfähigkeit und Effizienz von Multicore-Prozessoren. Als Randbedingungen sind die im Automobil typischen Anforderungen der verschiedenen Fahrzeugdomänen (Infotainment, Fahrdynamik, Chassis, Powertrain, etc.) zu berücksichtigen. Neben der Anwendungsorientierung steht ebenfalls die Entwicklung einer Sicherheitsarchitektur im Vordergrund. Diese muss auf der ISO 26262, den Konzepten zur effizienten Parallelisierbarkeit auf Task- und Datenebene, als auch den Security-Aspekten basieren. Letztendlich muss der gesamte Entwicklungsprozess durch eine geeignete Toolchain bzw. einer AUTOSAR konformen Toolchain unterstützt werden.

Als weiteres Ziel wird durch Virtualisierung die Funktionsabschottung verbessert und die Grundlage für eine neue energieeffiziente Bordnetzarchitektur durch zentralisierte Verarbeitungseinheiten geschaffen. Gleichzeitig wird die Realisierung neuer, hoch vernetzter Funktionen in den bereits erwähnten Fahrzeugdomänen ermöglicht [18].

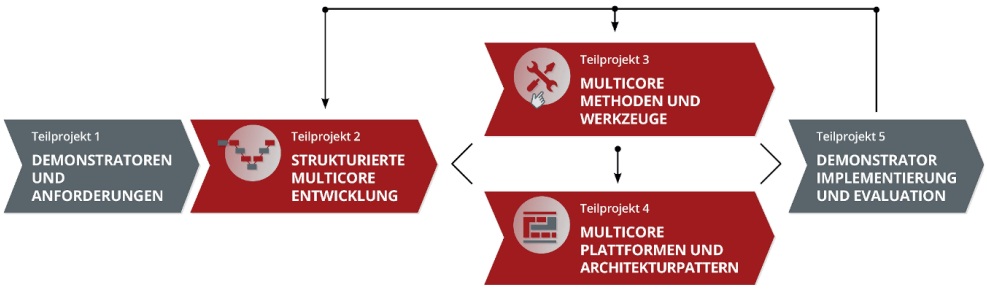

Abbildung 3.2: Projektstruktur des ARAMiS II-Projekts, Quelle: [17]

Die Projektlaufzeit von dem Projekt ARAMiS II umfasste den Zeitraum von 2016 bis 2019. In dem Nachfolgeprojekt zu ARAMiS werden die Entwicklungsprozesse und Plattformen, für die im ARAMiS-Projekt untersuchten und erarbeiteten Architekturen für den Einsatz von Multicore-SoCs optimiert und weiterentwickelt. Die Projektstruktur ist in Abbildung 3.2 dargestellt. Um den effizienten Einsatz von Multicore-Technologien zu ermöglichen, werden in diesem Projekt zum einen ein systematischer und strukturierter Ansatz zur Entwicklung von Multicore-Software und -Plattformen bereitge-

stellt. Zum anderen werden die notwendigen Methoden und Werkzeuge entwickelt, die diesen durchgängigen, strukturierten Multicore-Entwicklungsprozess realisieren. Letztendlich werden etablierte industrielle Plattformen unter Berücksichtigung von Multicore-spezifischen Anforderungen entwickelt und erweitert. Anhand von über zehn repräsentativen Use Cases aus den unterschiedlichen Anwendungsdomänen werden die Methoden, Werkzeuge und Plattformadaptionen abschließend evaluiert [17].

Die vorgestellten Arbeiten zeigen zum einen, in welcher Form dynamische Rekonfiguration auf Bauteilebene zum Einsatz kommt, und zum anderen, wie auf Bauteilebene die Grundlage für fehlertolerante Systeme gelegt wird. Im nächsten Abschnitt wird auf den Stand der Technik auf der Komponentenebene eingegangen.

3.2.2 Dynamische Rekonfiguration auf Komponentenebene

In der Arbeit „The System-Level Simplex Architecture for Improved Real-Time Embedded System Safety“ [8] führen die Autoren eine Simplex-Architektur auf Komponentenebene ein. In Abbildung 3.3 ist die logische Ansicht der Simplex-Architektur dargestellt.

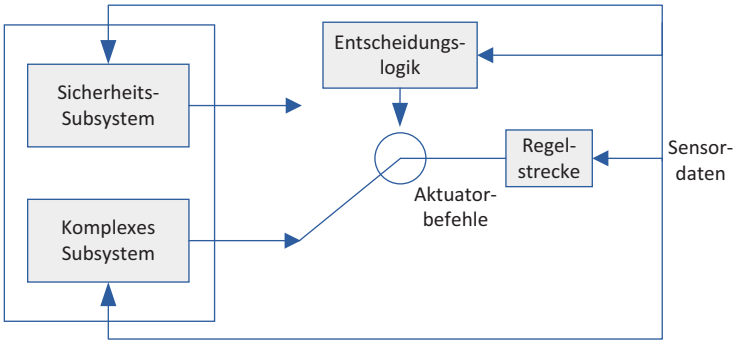

Abbildung 3.3: Logische Ansicht der Simplex-Architektur nach [8]

Der Titel spricht zwar von Systemebene, bezogen auf die eingeführten Systemabstraktionsebenen wird in der Arbeit jedoch ein Ansatz für die Komponentenebene vorgestellt. Unter Verwendung von Hardware-/Software-Co-Design, wird eine Lösung

vorgeschlagen, die sowohl für das Application Layer als auch für die darunterliegenden Schichten des Real Time Operating System (RTOS) und des Mikroprozessors Fail-Operational-Garantien zur Verfügung stellt. Weiterhin wird ein Ende-zu-Ende-Entwicklungsprozess vorgestellt, der automatisiert zusammengefügten und geprüften VHDL[8]-Code generiert. Eine Beispielimplementierung wird für einen Herzschrittmacher und die klassische invertierte Pendel-Anwendung bereitgestellt.

„Adaptives Monitoring für Mehrkernprozessoren in eingebetteten sicherheitskritischen Systemen“ [11] ist der Titel der Dissertation, in der der Autor Methoden vorstellt, um die Zuverlässigkeit von Multicore-Prozessoren zu steigern. Damit wird zu jenem Ziel beigetragen, Multicore-Prozessoren für eingebettete sicherheitskritische Anwendungen auch im Automobil einsetzbar zu machen. Um dieses Ziel zu erreichen, werden neuartige Methoden zur Überwachung und Fehlererkennung in Mehrkernprozessoren vorgestellt. Dabei werden auf Komponentenebene und Bauteilebene verschiedene Konzepte für Anwendungs- und Hardwareplattformen bereitgestellt. Neben der Überwachung und der Fehlererkennung werden Konzepte für die dynamische Migration von Funktionen für künftige Fail-Operational-Systeme zur Integration in einen Multicore-Prozessor vorgestellt.

Die Autoren der Arbeit „Towards Fail-Operational Systems on Controller Level Using Heterogeneous Multicore SoC Architectures and Hardware Support“ [12] zeigen in ihrer Arbeit einen Ansatz, um dynamische Rekonfiguration für sicherheitskritische Funktionen von einem Nominalsystem zu einem Fallback-System zu Laufzeit durchzuführen. Die Arbeit beinhaltet einen Fail-Operational-Ansatz auf Komponentenebene unter Einsatz eines heterogenen Multicore-Systems, das einen FPGA und Lockstep-Cores verwendet. Wird ein Fehler-Event ausgelöst, schaltet das System auf die Rückfallebene und übergibt dabei den Zustand. Dabei wird eine Beispielanwendung gezeigt, anhand derer die Umschaltzeiten gemessen werden.

Eine weitere Arbeit in dem Bereich der dynamischen Rekonfiguration auf Komponentenebene trägt den Titel „Markov Chain-based Reliability Analysis for Automotive Fail-Operational Systems“ [86]. In dieser Arbeit stellen die Autoren einen „2-out-of-2 Diagnostic Fail Safe (2oo2DFS)“-Ansatz vor, um ein Fail-Operational-System für den Automobilbereich einzusetzen. Hierfür werden Zuverlässigkeitsberechnungen für eine

[8] Very High Speed Integrated Circuit Hardware Description Language

symmetrische und eine asymmetrische Ausprägung durchgeführt und anhand eines Beispiels einer elektrischen Lenkkraftverstärkung evaluiert. Um die Zuverlässigkeit und die MTTF zu berechnen, wird mithilfe von Markov-Ketten ein Modell dafür erstellt.

Abbildung 3.4 zeigt einen Ansatz für Fail-Operational-Systeme, basierend auf dynamischer Rekonfiguration, der von den Autoren in der Arbeit „Fail-operational in safety-related automotive multi-core systems“ [85] beschrieben wird.

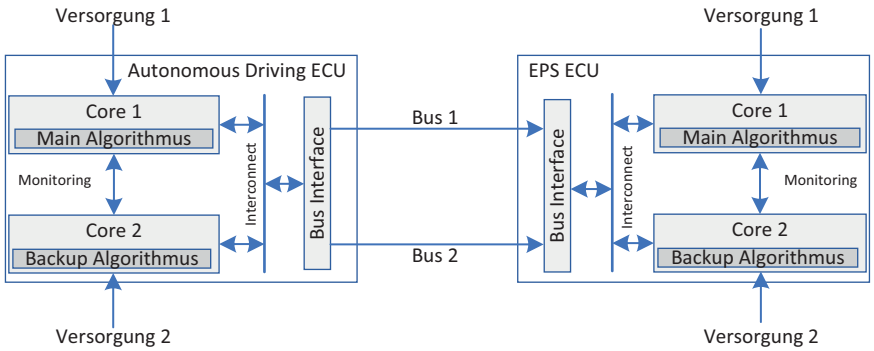

Abbildung 3.4: 2oo2DFS innerhalb von hochintegrierten ECUs nach [85]

Die Autoren erwähnen die Möglichkeit, das Konzept auf zwei ECUs aufzuteilen. Gezeigt und untersucht wird dieser Ansatz jedoch nicht. Es wird in der Arbeit ein 2oo2DFS-Ansatz beschrieben, der sowohl in einer EPS-ECU als auch in einem darüber liegenden Fahrrechner zum Einsatz kommt. Eine konkrete Umsetzung wird in der Arbeit nicht weiter beschrieben.

„A Controller Safety Concept Based on Software-Implemented Fault Tolerance for Fail-Operational Automotive Applications“ [58] ist eine Arbeit, in der eine kosteneffiziente Rechenplattform für Fail-Operational-Systeme auf Komponentenebene vorgestellt wird. In dem dargelegten Konzept wird auf Komponentenebene zwischen dem Nominalsystem und der Rückfallebene dynamisch rekonfiguriert, sobald ein Fehler erkannt wurde. Das Alleinstellungsmerkmal dieser Arbeit ist eine Fehlerdiagnose über Software Coded Processing anstatt über Lockstep-Cores. Des Weiteren wird ein asymmetrischer Ansatz vorgeschlagen, der eine Limp-Home[9]-Funktionalität zur Verfügung stellt. Das Software Coded Processing-Verfahren wird auch in der Arbeit [30] verwendet. Dort

[9] Limp-Home - nach Hause humpeln, reduzierte Funktionalität

kommt für das Fail-Operational-System ein 1oo2D-System zum Einsatz während bei dieser Arbeit ein 2oo2-System bzw. ein 1oo1D und ein Limp-Home-Controller zum Einsatz kommt. In dieser Arbeit wird das Software Coded Processing-Verfahren für Fail-Operational-Systeme empfohlen.

Drei verschiedene Architekturen werden in der Arbeit mit dem Titel „Fehlertolerante Systeme im Fahrzeug – von *fail-safe* zu *fail-operational*“ [128] vorgestellt. Zwei davon verwenden eine dynamische Rekonfiguration. Die erste Architektur ist eine 2oo3-Architektur, die zweite ist eine Duo-Duplex-Architektur und die dritte eine hybride Architektur. Alle drei sind auf Komponentenebene dargestellt. Die Arbeit beschränkt sich auf die prinzipielle Darstellung der Architekturen, ohne konkrete Umsetzungen anzugeben.

„Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives“ [124] ist eine Arbeit in der, aufbauend auf den grundlegenden Konzepten von fail-silent und Fail-Operational-Systemen, eine Fail-Operational-Architektur für Brake-by-Wire-Systeme vorgestellt wird. Die Entwurfsentscheidungen werden durch angemessene Begründungen und Entwurfskompromisse unterstützt. Des Weiteren werden eine Sicherheits- und eine Zuverlässigkeitsanalyse gemäß der ISO 26262 durchgeführt. Das vorgestellte Konzept berücksichtigt das komplette Bremssystem und führt dort die unterschiedlichen Methoden zur Zuverlässigkeitsanalyse durch. Die dynamische Rekonfiguration findet auf Komponentenebene statt. Dabei arbeiten zwei fail-silent-Einheiten parallel zueinander. Sobald eine ausfällt, übernimmt die andere Einheit. Kommuniziert wird über redundante Busse. Die beiden Einheiten besitzen jeweils eine getrennte Stromversorgung. Das vorgestellte System kann Kommunikationsfehler und Fehler aufgrund eines Stromausfalls tolerieren.

Der Autor der Arbeit mit dem Titel „Fault tolerant digital systems“ [107] stellt einige Grundprinzipien von fehlertoleranten Systemen dar. Dabei geht er auf die Attribute von Fehlertoleranz, den Entwicklungsansatz, die Fehlervermeidung, das redundante System und die Fehlererkennung ein. Beim letzteren stellt er insbesondere die dazu möglichen Ansätze wie der Verdopplung, der Codierung, der Prüfsumme[10], des Watchdogs, der Konsistenz- und der Fähigkeitsüberprüfung dar. Weiterhin zeigt er für die Fehlermaskierung ein Triple Modular Redundancy(TMR)-System und den Einsatz

[10] *englisch: checksum*

von Error Correcting Codes (ECC). Darüber hinaus stellt er prinzipiell in seiner Arbeit dynamische Rekonfiguration dar. Die Arbeit gibt einen Überblick über fehlertolerante Systeme und welche Techniken dazu notwendig sind.

Weitere Arbeiten in diesem Bereich sind „Fehlertoleranzverfahren“ [46], „Basic concepts and taxonomy of dependable and secure computing“ [5], „Fault-tolerant computing“ [103] und „Fault-Tolerant Design“ [44]. Mit Fehlertoleranz in der Avionik befasst sich der Autor in der Arbeit mit dem Titel „Fault-Tolerant Avionics“ [67]. Im Automobilbereich thematisiert die Arbeit „Diagnosis in Automotive Systems: A Survey“ [91] das Thema Fehlertoleranz. Die Autoren zeigen dort unter anderem modellbasierte Diagnose von Fehlern. Dynamische Rekonfiguration wird in dieser Arbeit, anders als in der Arbeit über Fehlertoleranz, in der Avionik nicht erwähnt. „Survey on fault-tolerant vehicle design“ [135] stellt Fehlertoleranz im Fahrzeug umfassender dar, geht jedoch dabei ebenfalls nicht auf die Möglichkeiten der dynamischen Rekonfiguration ein.

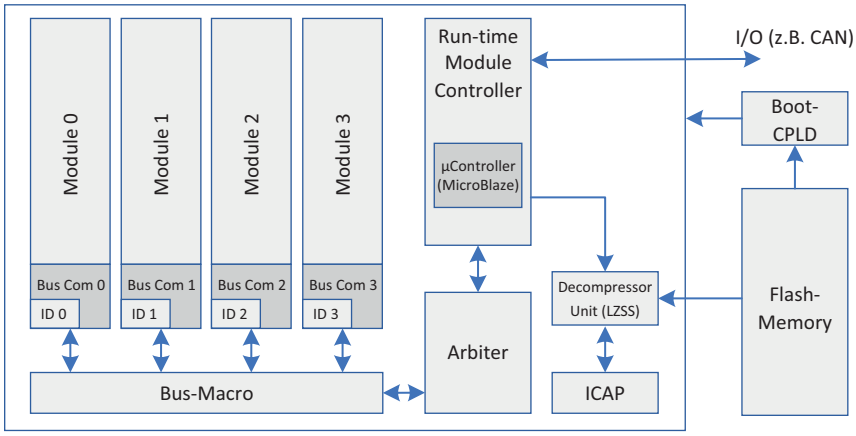

Abbildung 3.5: Rekonfigurierbares System für Automotive-Anwendungen nach [19]

Mit Hilfe von dynamischer und partieller Rekonfiguration haben die Autoren von „Dynamic and Partial FPGA-Exploitation“ [19] verschiedene Chassisfunktionen im Automobilbereich auf ein Steuergerät platziert. Zum Einsatz kommt dabei ein Steuergerät, das über einen Controller Area Network(CAN)-Bus mit dem Rest des Fahrzeugnetzwerkes verbunden ist. Weiterhin ist auf dem Steuergerät ein FPGA verbaut. In dem FPGA ist ein MicroBlaze (MB) als Softcore untergebracht. Die Architektur dazu ist in Abbildung 3.5 dargestellt.

Auf dem FPGA befinden sich vier Slots, auf die die unterschiedlichen Funktionen partitioniert werden können. Sind alle vier Slots belegt, wird geprüft, ob eine Funktion sich im Leerlauf befindet. Die älteste Funktion wird dann gelöscht. Auf einem externen Flashspeicher sind alle notwendigen Interieurfunktionen abgelegt, die bei Bedarf auf den FPGA rekonfiguriert werden können. Auf diese Weise können der Energieverbrauch und die Anzahl von Steuergeräten im Fahrzeug reduziert werden. Die Funktionen erfüllen Echtzeitanforderungen bis zu einer Antwortzeit von $10\,\mathrm{ms}$. Mit Hilfe von Messungen haben die Autoren gezeigt, dass auf diese Weise Energie eingespart werden kann. In der Arbeit gehen die Autoren auf dynamische Rekonfiguration innerhalb einer Komponente ein, der Aspekt eines Fail-Operational-Systems ist in dieser Arbeit nicht berücksichtigt.

Eine umfassende Arbeit zu Fail-Operational-Systemen im Automobilbereich hat der Autor der Dissertation mit dem Titel „Fail-operational automotive systems“ [119] erstellt. Er zeigt darin den Stand der Technik zu technischen Lösungen von Systemen im Automobilbereich, die sich auf funktionale Sicherheit beziehen und geht dabei auch auf verschiedene verkehrswesenbezogene Industrien ein. Des Weiteren wird die ISO 26262 [77] unter dem Aspekt von Fail-Operational-Systemen untersucht und auf ihre Anwendbarkeit auf solche Systeme hin geprüft. Als Ergebnis werden notwendige Verbesserungsvorschläge aufgezeigt. Zusätzlich sind in der Arbeit typische Software- und Hardware-Architekturen für Fail-Operational-Systeme unter dem Gesichtspunkt der ISO 26262 und einer Anwendung im Automobilbereich dargestellt.

Der Autor kommt zu dem Ergebnis, dass für Fail-Operational-Systeme im Automobil nur selten eine volle Redundanz notwendig ist und daher Entwickler in die Lage versetzt werden müssen, die notwendige Redundanz an der richtigen Stelle in der Architektur platzieren zu können. Mit diesem Ziel liefert der Autor in seiner Arbeit eine generische, strukturierte Fehleranalyse und zeigt eine Lösung für eine Anwendung im Automobilbereich für das aus anderen Industrien bekannte Problem der Redundanzallokation auf. Hierfür wird ein komplexes mathematisches Modell abgeleitet, die Zuverlässigkeitsmetrik für die Hardware erstellt und an realistischen Beispielen validiert. Der Aspekt der dynamischen Rekonfiguration wird an einigen Stellen aufgegriffen. So schlägt der Autor vor, die ISO 26262 um die Möglichkeit der Rekonfiguration zu erweitern (vgl. [119, S. 93]. Die dynamische Rekonfiguration wird in der Arbeit nicht weiter an einem Beispiel gezeigt.

In der Arbeit „Architecture of By-Wire Systems Design Elements and Comparative Methodology“ [38] vergleichen die Autoren auf Komponentenebene eine geteilte redundante Architektur (*englisch: shared redundancy architecture*) mit einer vollständig redundanten Architektur (*englisch: fully replicated architecture*). Bei der geteilten redundanten Architektur wird softwarebasierte Redundanz und dynamische Rekonfiguration für ein System, bestehend aus den Funktionen Bremsen, Lenken und Motordrosselung, vorgeschlagen. Es werden Grundelemente für die Entwicklung identifiziert und Hauptkriterien für einen Vergleich der unterschiedlichen Architekturen bereitgestellt. Darüber hinaus wird eine Methode aufgezeigt, um verschiedene Konzepte zu evaluieren. Der Ablauf der dynamischen Rekonfiguration wird in der Arbeit nicht weiter dargelegt.

Dynamische Rekonfiguration mit Hilfe von Software wird in der Arbeit „Software Deployment Analysis for Mixed Reliability Automotive Systems“ [20] ausgearbeitet. Der Autor führt einen Ansatz ein, um eine formalisierte Analyse durchzuführen, ob ein Systementwurf sämtliche Fail-Operational-Anforderungen der Funktionen erfüllt. Dabei berücksichtigt er verschiedene Szenarien von ausfallenden Systemelementen, wie zum Beispiel das Ausfallen von Ausführungseinheiten oder ausfallende Softwarekomponenten. Er zielt auf mixed-criticality und mixed-reliability im Automobilbereich ab.

Neben den Fail-Operational-Anforderungen werden auch Degradationskonzepte untersucht. Für die Systemanalyse wird zunächst ein formales Modell aufgesetzt, das die Funktionseigenschaften, mögliche Funktionsdegradationen, die Beziehungen zwischen den Funktionen und den Softwarekomponenten, die Kommunikationszusammenhänge, die Ausführungseinheiten und das Deployment der Funktionen auf den Ausführungseinheiten repräsentiert.

Weiterhin wird eine strukturierte Analyse der notwendigen Degradationslevel durchgeführt. In der Analyse werden zudem valide Deployments der Softwarekomponenten auf die Ausführungseinheiten automatisch synthetisiert, die die Fail-Operational-Anforderungen erfüllen und das geforderte Level der Degradation minimieren. Darüber hinaus werden angemessene Level von Redundanzen in das Deployment eingebaut, um ein Fail-over zu ermöglichen. Das formale Modell, die Randbedingungen und die Optimierungskriterien werden durch eine lineare Arithmetik und logische Operatoren

beschrieben. Abschließend wird der Ansatz an drei Beispielen aus dem Automobilbereich gezeigt.

3.2.3 Dynamische Rekonfiguration auf Systemebene

Bezogen auf die Systemabstraktionsebenen wurden Arbeiten auf der Bauteil- und Komponentenebene untersucht. Als nächstes werden einige Projekte auf der Systemebene vorgestellt. Zunächst wird dabei auf das Projekt IT_Motive 2020 eingegangen, das im Folgenden kurz umrissen wird.

IT_Motive 2020 – Real-Time Capable Virtualized Information and Communication Technology Infrastructure for Automotive Systems

Die IT_Motive 2020-Projektlaufzeit belief sich auf 3 Jahre. Das Projekt war im Rahmen der Car@TUM-Kooperation angesiedelt. Motivation für dieses Projekt war die Tatsache, dass die Innovationen in einem aktuellen Fahrzeug zu 90 % durch Elektronik und Software getrieben werden. Die Entwicklungszyklen von Elektronik und Software sind jedoch wesentlich schneller als der typische Lebenszyklus eines Fahrzeugs. Aktuelle Fahrzeug-IT-Architekturen leiden jedoch unter der begrenzten Erweiterbarkeit, weil sie auf vielen verschiedenen, spezifischen Produkten basieren, wie zum Beispiel bis zu sieben unterschiedlichen Bus Typen und ungefähr 50 bis 70 unterschiedlichen ECUs. Im Projekt IT_Motive 2020 wurde aus diesem Grund eine neuartige Fahrzeug-IT-Architektur untersucht, die auf einem Universalrechner mit Ethernet-Verbindung basiert.

Zukünftige Automotive-Control-Units werden dem generellen Trend für eingebettete Systeme folgen und mit Multicore-Prozessoren bestückt sein. Die höhere Rechenleistung wird benötigt, weil aufgrund zunehmender Interaktionen von Funktionen verschiedene Domänen zusammengefasst werden müssen. Die verschiedenen Funktionen mit unterschiedlichen Anforderungen hinsichtlich Echtzeitfähigkeit, Datendurchsatz oder Performanz werden auf einer geteilten Plattform zusammengeführt und benötigen daher eine effektive Organisation mit niedrigem Overhead. Dabei sind Lösungen aus dem Konzept der Multiple Independent Levels of Security and Safety (MILS) verwendet worden, um Anwendungen mit unterschiedlichen Anforderungen an Echtzeit

und Performanz auf einer geteilten Multicore-Plattform darzustellen. Das entwickelte Konzept wurde an einem Embedded Board mit einer Intel Core 2 Duo CPU inklusive einer on-board Gigabit-Ethernet-Netzwerkverbindung validiert [65]. Die erarbeitete Architektur ist in Abbildung 3.6 dargestellt.

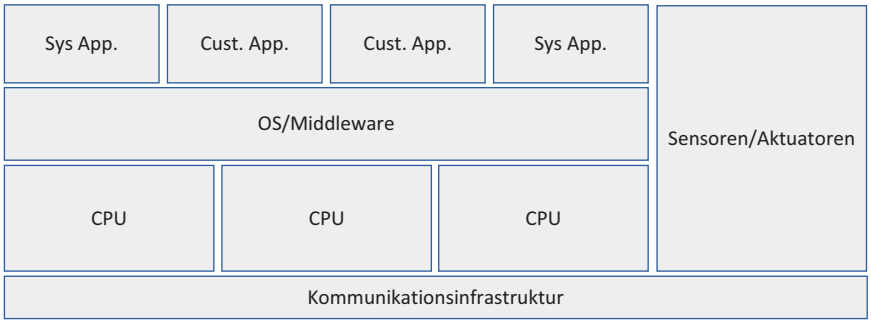

Abbildung 3.6: Fahrzeug-IT-Architektur, Quelle: [43]

In „A Real-Time Capable Virtualized Information and Communication Technology Infrastructure for Automotive Systems“ [43] wurden die im IT_Motive 2020-Projekt erfolgreich entwickelten Ansätze für fehlerresiliente Systeme dargestellt. Rerouting von Kommunikationswegen, dynamische Rekonfiguration und Task-Migration im Hot-Standby kommen hierbei zum Einsatz, um eine Fehlertoleranz zu erreichen. Weiterhin wurden die Sicherheitsaspekte zum Schutz vor Hackerangriffen durch Einsatz eines Hypervisors dargestellt. Strukturen und Technologien, die in standardisierten Geschäfts-IT-Umgebungen zum Einsatz kommen, wurden an die Automobilanforderungen adaptiert. Angegangen wurden die besonderen Herausforderungen im Hinblick auf eine zukünftige Fahrzeug-IT-Architektur bezüglich gleichzeitiger Übertragung und Berechnung auf Multicore-Prozessoren von Soft Real-Time-Tasks, wie zum Beispiel $25\,\mathrm{fps}$ Videodekodierung und Hard Real-Time-Tasks mit $1\,\mathrm{ms}$ Closed-Loop-Regelschleifen. Darüber hinaus wurden effiziente Lösungen für den Speicher- und I/O-Zugriff auf geteilte Ressourcen erarbeitet. In Summe wurde aufgrund der hohen Komplexität heutiger Fahrzeug-IT-Architekturen und der damit verbundenen schwierigen Wartbarkeit und der hohen Kosten eine neuartige IT-Architektur für zukünftige Fahrzeuge vorgestellt. Diese basiert auf den folgenden fünf grundlegenden Prinzipien Zentralisierung, Homogenisierung, Virtualisierung, Relokation und Fault-Tolerance.

Der Aspekt der Echtzeitfähigkeit wurde im IT_Motive 2020-Projekt mit Hilfe eines Ethernet-Switches umgesetzt, der um Real-Time-Fähigkeiten erweitertet wurde. Dabei kam der IEEE 802.3-Standard zum Einsatz. Zusätzlich kamen ein echtzeitfähiges Scheduling und eine Partitionierung zum Einsatz, mit der virtuelle Kanäle umgeschaltet werden konnten und der logische Pfad vom physischen Pfad getrennt werden konnte. Es handelt sich hierbei um ein ereignisgesteuertes System[11], das gegenüber einem zeitgesteuerten System[12] eine sehr komplexe Berechnung der Echtzeitfähigkeit erfordert. Aus diesem Grund wurde hier ein analytischer Ermittlungsansatz gewählt, der für die Echtzeitfähigkeit eine obere Schranke liefert. Es ist bekannt, dass dieser Ansatz bei einem ereignisgesteuerten System zu einer deutlichen Überschätzung führt und die tatsächliche Ausführungszeit wesentlich niedriger ist. Trotzdem muss sichergestellt werden, dass diese obere Schranke nicht verletzt wird. Hierfür kam eine UPC[13]-Instanz zum Einsatz.

Dynamische Rekonfiguration wurde im IT_Motive 2020-Projekt sowohl bei dem Entwurf der Software-Architektur berücksichtigt als auch bei der Sensorauslegung für verschiedene Öffnungswinkel von Lidar[14] und Radar[15]. In der Software Architektur wurde eine API[16] zum Austausch von internen Zustandsinformationen implementiert, die für eine zustandsbehaftete dynamische Rekonfiguration benötigt werden.

Einige Ideen aus dem IT_Motive 2020-Projekt werden in dieser Arbeit aufgegriffen und mit in das AutoKonf-Projekt integriert. Diese sind insbesondere der Einsatz von Universalrechnern auf Systemebene. Auch die Abschätzung der Echtzeitfähigkeit wird in dieser Arbeit analytisch durchgeführt, allerdings wird diese Berechnung noch um einen simulativen Ansatz ergänzt (vgl. 6.3.2).

[11] *englisch*: *event-triggered System*, ist ein System, dem die Erzeugung, Erkennung, die Verwendung und die Reaktion auf Ereignisse zugrunde liegt
[12] *englisch*: *time-triggered System*, ist ein Computersystem, das eine oder mehrere Gruppen von Aufgaben nach einem vorher festgelegten und eingestellten Aufgabenplan ausführt
[13] *englisch*: *usage parameter control*, ist definiert als eine Reihe von Aktionen, die das Netzwerk zur Überwachung und Kontrolle des Datenverkehrs durchführt
[14] ist ein Verfahren zur Entfernungsmessung, bei dem das Ziel mit Laserlicht beleuchtet und die Reflexion mit einem Sensor gemessen wird
[15] ist ein Erkennungssystem, das mit Hilfe von Funkwellen die Entfernung, den Winkel oder die Geschwindigkeit von Objekten bestimmt
[16] Application Programming Interface

RACE – Robust and reliable Automotive Computing Environment for future eCars

Im Rahmen des RACE-Projekts wird der stetig wachsenden Softwarekomplexität Rechnung getragen. Die ständig zunehmende Anzahl von Softwarefunktionalitäten führt direkt zu höherer Komplexität bei der Entwicklung und Konfiguration. Aktuelle Studien zeigen, dass dieser Trend sich in Zukunft so weiter entwickeln wird. Zusätzlich kommen neue Felder hinzu, wie das der Fahrerassistenzsysteme, im Besonderen das hoch- und vollautomatisierte Fahren sowie Parken.

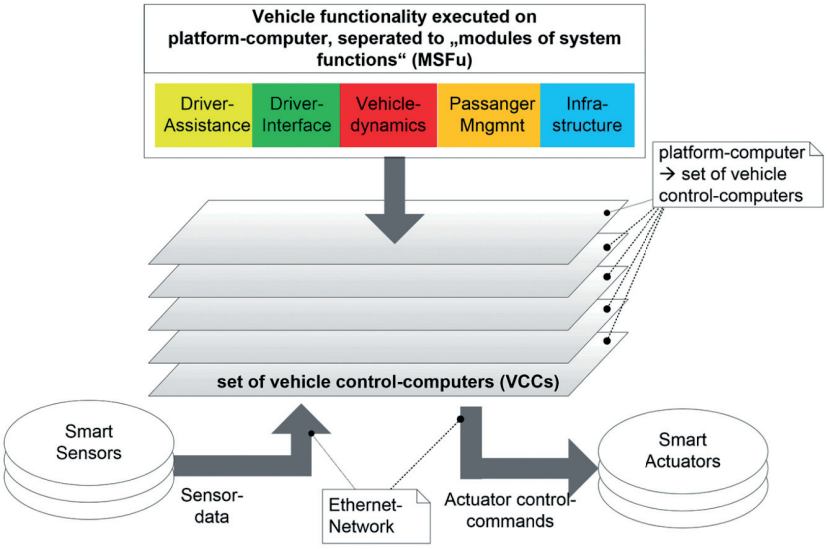

Abbildung 3.7: Eine zentralisierte Recheneinheit als Kern der Plattform, Quelle: [3]

Daraus resultieren Anforderungen an Erweiterbarkeit und Zuverlässigkeit für zukünftige E/E-Architekturen. Viele dieser Funktionen benötigen Zugriff auf Informationen von Systemen – und damit verbunden auch auf Kommunikationssysteme – die über das gesamte Fahrzeug verteilt sind. Das wiederum erhöht die Komplexität. AUTOSAR [53] bietet dafür jedoch keine Methoden, um diese immer weiter steigende Komplexität zu reduzieren. Somit wird im RACE-Projekt eine zentrale Rechnerarchitektur entwickelt, die an die erprobten Architekturen aus der Avionik und der Prozessautomatisierung angelehnt ist. Der zentralisierte Rechner als Kern der Plattform ist in Abbildung 3.7 dargestellt.

Die Limitierungen, die AUTOSAR diesbezüglich hat, werden dabei berücksichtigt, so dass eine Umgebung für sicherheitskritische Anwendungen geschaffen wird. Schnittstellen für Verifikation und Test werden dabei ebenso mit in das Konzept eingearbeitet wie eine verlässliche Kommunikation zu Aktuatoren und Sensoren. Ziel des Projekts ist eine vereinfachte standardisierte Informations- und Kommunikationsstruktur, um Funktionen per Software über den gesamten Lebenszyklus des Fahrzeugs zu aktualisieren und nachrüsten zu können.

Die entwickelte ausfallsichere, zentralisierte Architektur besteht aus einer redundanten Kommunikationsinfrastruktur, die auf einer Switched-Ethernet-Topologie basiert, einer redundanten Stromversorgung und redundanten Hochleistungs-Multicore-Rechnern. Weiterhin wird die Test- und Entwicklungsphase verkürzt und die Grundlage für autonomes Fahren, Drive-by-Wire, Konnektivität sowie Car2X-Kommunikation geschaffen. Die zentrale Komponente beim RACE-Projekt ist eine System- und Softwarearchitektur, die auf einem Middleware-basierten Ansatz aufsetzt, der eine rekonfigurierbares, Quality of Service(QoS)-bewusstes Echtzeitsystem bereitstellt. Dieses stellt Safety-, Security- und Reliability-Eigenschaften bereit und setzt auf einer Duo-Duplex-Architektur mit einem Ethernet-Kommunikationsring auf [125].

In der Arbeit „Ethernet-Based and Function-Independent Vehicle Control-Platform: Motivation, Idea and Technical Concept Fulfilling Quantitative Safety-Requirements from ISO 26262“ [3] gehen die Autoren insbesondere auf das im RACE-Projekt verwendete Ethernet-basierte Kommunikationssystem ein. Die vorgeschlagene Architektur beinhaltet einen n-Duplex-Ansatz für Redundanz und zeigt Skalierbarkeit mit funktionaler Performanz und Verfügbarkeit. Das Plattformkonzept basiert auf einer klaren Trennung von Verantwortlichkeiten.

SafeAdapt – Safe Adaptation for Reliable and Energy-Efficient E/E Architectures

Von 2013 bis 2016 ist im Projekt SafeAdapt ein neuartiges E/E-Architekturkonzept für vollektrische Fahrzeuge entwickelt worden, um die Robustheit, die Verfügbarkeit sowie die Kosteneffizienz von Safety-relevanten Systemen bei gleichzeitigem Erhalten der funktionalen Sicherheit zu steigern. Die zugrundeliegende E/E-Architektur wurde vom RACE-Projekt übernommen. Der innere Ring wurde durch ein Time-Triggered Ethernet (TTE) ersetzt. Der verfolgte Ansatz reduziert die Systemkomplexität und die

Komplexität bei den Interaktionen, indem eine generische, systemweite Definition im Umgang mit Fehlern und Anpassungen vorgenommen wird. Dies ist insbesondere für die zunehmende Zuverlässigkeit und Effizienz beim Energieverbrauch, bei den Kosten und bei der Einfachheit im Systementwurf wichtig. In Abbildung 3.8 ist diese Architektur dargestellt.

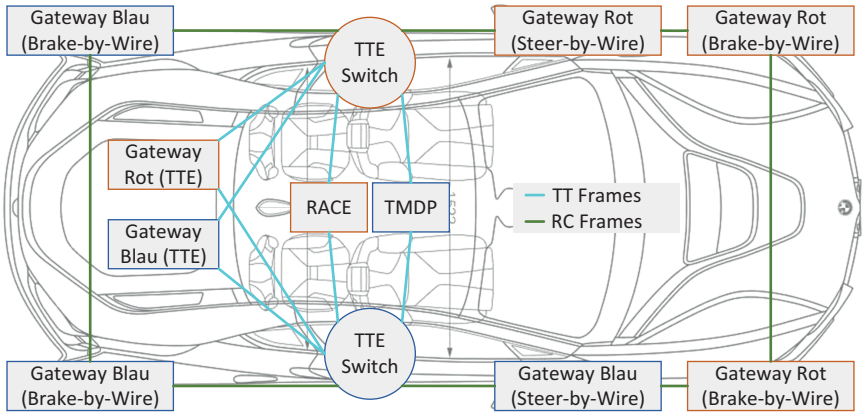

Abbildung 3.8: Überblick über die Hardware-Architektur nach [49]

Ein weiteres Projektziel ist die erhöhte Safety und Verfügbarkeit durch die Möglichkeit, komplexe Fehler zu handhaben. Im Besonderen wurden Fehler berücksichtigt, bei denen gegenwärtige Systeme keine geeigneten Lösungen anbieten. Darüber hinaus wird die Materialliste reduziert, indem die Anzahl der Steuergeräte verkleinert wird. Dies wird erreicht, indem ein generisches Fehlermanagement basierend auf dem SafeAdapt-Platform-Core umgesetzt wird. Der Ansatz für funktionale Sicherheit, der im SafeAdapt-Projekt verfolgt wird, entspricht den Vorgaben aus der ISO 26262. Des Weiteren sind Softwareupdate-Mechanismen implementiert, um über den gesamten Lebenslauf Funktionen hinzufügen zu können. Letztendlich wird auch die Energieeffizienz der E/E-Architektur verbessert.

Das Projekt SafeAdapt stellt einen integrierten Ansatz zur Verfügung, um adaptive Systeme zu entwickeln. Hierbei reicht die Bandbreite von der Unterstützung durch eine Werkzeugkette, einer Referenzarchitektur, dem Modellieren des Systementwurfs bis hin zu einer frühen Verifikation und Validierung. Hierdurch werden ebenfalls die Entwicklungs- und Absicherungskosten wie auch die Time-to-Market reduziert. Die

Anwendbarkeit wird anhand von zwei Beispielen demonstriert: Einem Brake-by-Wire-Beispiel, das Redundanz mit Hot-Standby-Slaves beinhaltet und einem Beispiel für die Geschwindigkeitsregelung, das ein Szenario mit einer angemessenen Degradation zeigt [49].

SAFER – System-level Architecture for Failure Evasion in Real-time Applications

Von 2012 bis 2013 wurde das Projekt SAFER [83] durchgeführt. Darin wurde eine Systemarchitektur für zuverlässige autonome Fahrzeuge eingeführt, die auch eine Degradation in Fehlerszenarien unterstützt. Die SAFER-Architektur unterstützt die Fehlertoleranz durch die Verwendung von Redundanzmechanismen, die auf Cold-Standby-Slaves, Hot-Standby-Slaves und Task-Neuausführungen basieren. Die Autoren berücksichtigen auch eine ordnungsgemäße Degradation des Systems, beispielsweise im Falle eines Ausfalls von Prozessorkarten.

In ihrer Arbeit sind sie der Ansicht, dass die angemessene Degradation von Fahrzeugfunktionen in Abhängigkeit von verschiedenen Situationen durchgeführt werden sollte. Wenn beispielsweise ein Bildverarbeitungsalgorithmus für die Fußgängererkennung aufgrund eines Ausfalls eines Mikrocontrollers ausfällt, sollte die Reaktion beim Fahren auf einer Autobahn anders sein als beim Fahren in einem Stadtgebiet, da im letzteren Fall Fußgänger mit größerer Wahrscheinlichkeit anwesend sind. Prozessorfehler und Taskfehler werden in der SAFER-Architektur dadurch erkannt, dass der aktuelle Gesundheitszustand und der Zustand jedes einzelnen Tasks überwacht und diese Information verteilt wird. Wird ein zeitbasierter Fehler oder ein eventbasierter Fehler erkannt, wird das System rekonfiguriert, um die Funktion aufrecht zu erhalten.

Im SAFER-Projekt wird eine formale Analyse für das Worst-Case-Zeitverhalten der SAFER-Eigenschaften durchgeführt. Weiterhin wird eine Systemmodellierung mit dem Werkzeug SysWeaver durchgeführt, um das Zeitverhalten zu beschreiben. Implementiert wurde SAFER auf Ubuntu und einem Prototypenfahrzeug, das an der 2007 DARPA Urban Challenge teilgenommen hat [84].

HiBoard – Hochzuverlässige und intelligente Bordnetztopologien für automatisierte Fahrzeuge

Dieses Projekt wurde im Zeitraum von 2016 bis 2019 bearbeitet und erforschte den Einsatz von zukünftigen Bordnetztopologien in hoch- und vollautomatisierten Fahrzeugen. Die zugrundeliegende Motivation war dabei die Befähigung von Elektronik und Sensorik, um eine Umsetzung für automatisierte Fahrfunktionen zu ermöglichen. Dabei müssen nicht nur die Einzelkomponenten befähigt werden, sondern insbesondere auch das Gesamtsystem, um eine Ausfallsicherheit und somit stets eine sichere Fahrt zu gewährleisten.

Beim hoch- und vollautomatisierten Fahren wird im Fehlerfall der Mensch nicht mehr eingreifen. Aus diesem Grund muss das System jederzeit eine stabile Rückfallebene einnehmen können. Mit diesem Ziel wurden im HiBoard-Projekt zukünftige Bordnetztopologien für den Einsatz in hoch- und vollautomatisierten Fahrzeugen erforscht. Vollständig redundante Systeme wurden dabei durch den Einsatz von einem intelligenten Verbund aller elektronischer Komponenten vermieden und damit eine sehr hohe Zuverlässigkeit und Fehlertoleranz des Bordnetzes ermöglicht.

Um dieses zu erreichen, wurden dynamische Umleitungen über intelligente Schnittstellen, aktive dezentrale Energiespeicher für temporäre Spannungsversorgung sicherheitskritischer Komponenten, Fehlerdetektion und Zustandsüberwachung von Steck- und Kabelverbindungen sowie Software- und Entwicklungswerkzeuge zum Entwurf intelligenter Bordnetzsysteme entwickelt [68].

3.2.4 Beispiele von Alternativen X-by-Wire-Systemen

Es gibt verschiedene Umsetzungen eines Fail-Operational-Systems in der Automobilindustrie, die im Folgenden beispielhaft aufgezeigt werden. Dabei werden diese Ansätze gegliedert nach Steer-by-Wire und Brake-by-Wire-Ansätzen. Brake-by-Wire-Ansätze ermöglichen neue Kundenfunktionen wie zum Beispiel die Individualisierung der Bremspedalrückmeldung zum Kunden beim Betätigen oder der Einsatz bei elektrischen Antrieben, bei denen das Gaspedal gleichzeitig die Bremsfunktion teilweise übernimmt.

Aber auch geometrische Vorteile können durch den Einsatz dieser Systeme gehoben werden. Da sich neue Möglichkeiten für die Verlegung von Leitungen und die Positionierung von Steuergeräten ergeben. Nicht zuletzt sei der verkürzte Bremsweg in Notbremssituation zu erwähnen.

Auch der Einsatz beim autonomen Fahren muss hervorgehoben werden, sollten zum Beispiel keine Bremspedale mehr im Fahrgastraum verbaut sein. Bei den Steer-by-Wire-Systemen verhält es sich ähnlich. Auch hier sind der Einsatz beim autonomen Fahren, die Individualisierung der Kundenwahrnehmung, Notfallmanöver und die geometrischen Vorteile zu erwähnen. Hier kann der geometrische Vorteil im Entfall oder Verkürzung der Lenksäule liegen, das wiederum zu einer Verlagerung der Stirnwandposition und somit zu einem vergrößerten Fahrgastraum führen kann.

Eine der großen Herausforderungen für die Automobilindustrie ist die Realisierung von Fail-Operational-Funktionen für hoch- oder vollautomatisiert fahrende Fahrzeuge. Innerhalb von E/E-Architekturen gibt es wiederum mehrere Themen, die dabei berücksichtigt werden müssen. Diese sind unter anderem der benötigte Bauraum von zusätzlichen Steuergeräten aufgrund der benötigen Redundanz, die elektrische Anbindung, aber auch in dem Entwicklungsprozess als solchen. Im Folgenden werden technische Lösungen betrachtet, die eine Redundanz oder eine Rekonfiguration zum Einsatz bringen.

Steer-by-Wire-Ansätze

Im Jahr 2013 führte Nissan in seinem Kraftfahrzeug (Kfz) Infiniti Q50 ein dreifach redundantes Steer-by-Wire-System ein. In diesem Auto werden drei Steuergeräte verwendet, die ein statisch redundantes *n*-*m*-System darstellen, für das die zugrunde liegende Theorie in [46] erläutert wird. Andere Forschungsaktivitäten zeigen statisch redundante Ansätze wie in [23] dargestellt. Weitere Lösungen, die von einer Standardlenkung abweichen, finden sich zum Beispiel in [47]. Hier wird eine Einzelradlenkung verwendet. In diesem Fall wird durch die Einzelradlenkung ein einzelner Ausfallpunkt verhindert. Wenn eines von vier Lenksystemen ausfällt, kann die Lenkung trotzdem stattfinden, indem die verbleibenden drei Systeme auf intelligente Weise genutzt werden.

Brake-by-Wire-Ansätze

In Bezug auf die Bremse wurde 2014 ein Brake-by-Wire-System in die Formel 1 aufgenommen; frühe Versionen wurden 2005 eingeführt und sind in Fahrzeugen von Toyota Estima, Toyota Prius und Mercedes E und SL zu finden. Trotzdem wurden die Systeme eingestellt. In [115] wird ein Überblick über die Vorteile gegeben und in [14] ein konventionelles Bremssystem mit einem Fallback beschrieben. In [61] ist ein Beispiel für ein konventionelles Bremssystem gegeben. Es zeigt den Einbau in ein Prototypenfahrzeug. Aus den Arbeiten stellt sich heraus, dass die dynamische Rekonfiguration eines Echtzeit-Embedded-Systems in der Automobilindustrie für Bremsen und Lenken bisher noch nicht gezeigt wurde. Sie eröffnet jedoch neue Möglichkeiten für Forschung und Industrie. Für die Avionikindustrie wird in [51] ein Überblick über einige Redundanzkonzepte gegeben. Diese können im Weiteren zur Orientierung für entsprechende Architekturen herangezogen werden.

3.3 Entwicklungsmethoden für dynamisch rekonfigurierbare Systeme

3.3.1 Modellierung und Einsatz von serviceorientierten Architekturen (SOA)

Es gibt keine etablierten und bekannten Ansätze, um den Entwurf und die Überprüfung einer zuverlässigen dynamischen Rekonfiguration in ausfallsicheren Systemen für Kraftfahrzeuge zu handhaben. Trotzdem existieren einige Forschungsarbeiten, die im Folgenden kurz vorgestellt werden. In „Elastic Service Provision for Intelligent Vehicle Functions“ [89] wird ein konzeptioneller Rahmen bereitgestellt. In dieser Arbeit beschreiben die Autoren Artificial Intelligence(AI)-gesteuerte und automatisch ausgelöste Maßnahmen sowie regelbasierte Strategien zusammen mit einer Publish-Subscribe-Middleware, um auf inkonsistente Fahrzeugumgebungsbedingungen zu reagieren.

In der Arbeit mit dem Titel „On Service-Orientation for Automotive Software“ [90] untersuchen die Autoren die Anwendbarkeit von SOA für sicherheitskritische Embedded-

Automotive-Softwaresysteme. Das Fazit daraus ist, dass die ersten Ergebnisse in der Tat für das Automotive Software Engineering geeignet sind und den Komplexitäts- und Sicherheitsanforderungen gerecht werden. Die in dieser vorherigen Studie durchgeführten Arbeiten werden in dieser Arbeit verwendet und um den modellbasierten Ansatz erweitert. Anschließend werden sie zu dynamisch rekonfigurierbaren Systemen erweitert. Darüber hinaus wird die Lücke zwischen dem modellbasierten Entwurf von SOAs und der dynamischen Rekonfiguration geschlossen.

Es gibt bereits mehrere Studien zum modellbasierten SOA-Design in der IT[17]-Branche. In [32] werden die Theorie und ein formales Servicemodell für SOA vorgestellt. Ein Framework mit einer SOA-Schichtenarchitektur unter Verwendung modellbasierter Techniken wird in [36] vorgestellt. Es sind auch einige Forschungen im Bereich der Automobilindustrie verfügbar. Ein umfassender Überblick über verschiedene Aspekte einschließlich SOA und modellbasiertem Design ist in [33] zusammengefasst. In [78] wird eine domänenspezifische Modellierung für Automotive-Services vorgeschlagen, um die spezifischen Anforderungen im Automotive-Bereich zu erfüllen. Eine dynamische Rekonfiguration wird in dieser Untersuchung jedoch noch nicht berücksichtigt.

Middleware-Konzepte

Die Technologien und Middleware-Konzepte sind beispielsweise Scalable Service-Oriented Middleware over IP (SOME/IP) [133], AUTOSAR Adaptive [53], Data Distribution Service (DDS) [129], GENIVI [56] und Rich Services [88]. SOME/IP ist eine Automotive-Middleware-Lösung, die für die Integration in AUTOSAR konzipiert wurde. Es unterstützt eine breite Palette von Middleware-Funktionen wie Serialisierung, Remote Procedure Call (RPC), Service Discovery, Publish/Subscribe und Segmentierung von User Datagram Protocol(UDP)-Nachrichten. Die AUTOSAR Adaptive-Plattform soll die dynamische Bereitstellung von Kundenanwendungen unterstützen. Es bietet eine Umgebung für Anwendungen, die High-End-Rechenleistung erfordern, während Funktionen, die aus eingebetteten Systemen stammen, wie Sicherheitsdeterminismus und Echtzeitfunktionen, beibehalten werden.

DDS ist ein Data-Centric Publish-Subscribe(DCPS)-Modell für die verteilte Anwendungskommunikation und -integration. Die Spezifikation ermöglicht die effiziente

[17] Informationstechnik

Übermittlung von Informationen von Datenbereitstellern an korrespondierende Nutzer und zielt auf Echtzeitanwendungen ab. Die GENIVI-Entwicklungsplattform ist ein Open Source-Projekt für die Automobilindustrie. Rich Services ist ein Architekturmuster, mit dem sich cyber-physische Systeme systematisch entwerfen und implementieren lassen. Es legt besonderen Wert auf dynamischen Wandel.

3.3.2 Zuverlässigkeitsanalyse von E/E-Architekturen

Zuverlässigkeit ist kein neues Forschungsthema, auch nicht im Automotive-Bereich. Verschiedene Untersuchungen wurden bereits vollzogen. Im Folgenden werden einige ausgewählte Arbeiten vorgestellt. In der Forschungsarbeit „Dynamic Fault Tree Analysis: State-of-the-Art in Modelling, Analysis and Tools“ [4] konzentrieren sich die Autoren auf die Modellierung des Antriebssystems von Drohnen mit mehreren Rotoren mithilfe von Markov-Ketten. Unter Verwendung des vorgeschlagenen Modells werden sowohl die Zuverlässigkeit als auch die mittlere Ausfallzeit (MTTF) des Antriebssystems bewertet.

Die Arbeit „Reliability Analysis of Multi-rotor UAV Based on Fault Tree and Monte Carlo Simulation“ [134] zeigt eine Zuverlässigkeitsanalyse von einem Unmanned Aerial Vehicle (UAV) mit mehreren Rotoren durch eine Fehlerbaum-Modellierung, gefolgt von einem Monte-Carlo-Simulationsexperiment zur quantitativen Analyse der Systemzuverlässigkeit. Um die Zuverlässigkeit des Systems zu messen, werden die Wichtigkeit der Basiseinheit und die Schwäche der Basiseinheit durch die Berechnung des vorgeschlagenen Zuverlässigkeitsbewertungsindex definiert.

In der Arbeit „Fuzzy dynamic fault tree analysis for electro-mechanical actuator based on algebraic model with common-cause failures“ [142] stellen die Autoren einen Ansatz vor, der verschiedene Ansätze miteinander kombiniert. Die Fehlerbaumanalyse, die Common-Cause-Fehleranalyse und die Analyse mithilfe von Markov-Ketten werden für elektromechanische Antriebe angewendet. Die Fehlerbaumanalyse erfasst Fehlerereignisse sowohl für elektrische als auch für mechanische Teile. Die Zuverlässigkeit des mechanischen Teils wird jedoch separat über Markov-Ketten analysiert und in den Fehlerbaum integriert. Die Zuverlässigkeit eines 2oo2DFS für eine symmetrische und eine asymmetrische elektronische Servolenkung wird in der Arbeit mit dem Titel „Markov Chain-based Reliability Analysis for Automotive Fail-Operational

Systems“ [86] anhand eines Markov-Kettenmodells bewertet. Eine generische Methode für eine Bottom-up-ASIL-Zerlegung, die bei der Entwicklung eines neuen Produkts verwendet werden kann, ist in „A Generic Method for a Bottom-Up ASIL Decomposition“ [52] dargestellt. Darüber hinaus wird eine Common-Cause-Fehleranalyse durchgeführt. Auf diese Weise wird sichergestellt, dass keine Common-Cause-Fehler vorhanden sind.

3.4 Bewertung Stand der Technik und Fazit

In der Arbeit „Fail-operational in safety-related automotive multi-core systems“ [85] wurde eine Fail-Operational-Architektur auf Systemebene entwickelt, jedoch ohne dynamische Rekonfiguration. Die Arbeit „Adaptives Monitoring für Mehrkernprozessoren in eingebetteten sicherheitskritischen Systemen“ [11] entwickelte eine Fail-Operational-Architektur, jedoch nicht auf Systemebene. Als Kern der Arbeit und als Ansatz für die eigene Idee wird eine Kombination aus diesen beiden Arbeiten konzipiert und in das AutoKonf-Projekt mit integriert. Es wurde demnach noch nicht untersucht, wie dynamische Rekonfiguration auf Systemebene im Automobil dargestellt wird.

Die Motivation hierfür leitet sich aus der funktionalen Hochintegration auf Systemebene und dem Übergang von Fail-Safe- zu Fail-Operational-Systemen (vgl. Abschnitt 1.1 und Tabelle 2.1) ab. Die hierfür erforderlichen unterschiedlichen Prozessortypen (vgl. Abschnitt 2.1.1) spielen dabei eine entscheidende Rolle für welchen Umfang sie dabei zum Einsatz kommen. Es wird gezeigt, welche Applikationsanteile auf einem High-Performance oder einem echtzeitfähigen Core aber auch auf einem FPGA verwendet werden können. Aus diesem Grund wird in dieser Dissertation eine MPSoC-Plattform verwendet. Des Weiteren findet die Verwendung von serviceorientierten Architekturen in zunehmendem Maße auch in der Automobilindustrie statt (vgl. Abschnitt 3.3.1). Dieser Aspekt wird ebenfalls in dieser Arbeit untersucht. Aufgrund der verteilten Systeme im Fahrzeug (vgl. Abschnitt 2.1.1) kommen verschiedene Middleware-Konzepte dabei zum Einsatz. Dazu wird in dieser Dissertation auf SOME/IP gesetzt, allerdings werden auch andere Middleware-Konzepte untersucht (vgl. Abschnitt 3.3.1 und Abschnitt 5.5.1). In Abbildung 3.9 ist der in dieser Arbeit untersuchte neue Ansatz schematisch skizziert.

Der erarbeitete Ansatz bietet unterschiedliche Perspektiven, die derzeit in der Automobilindustrie untersucht werden. Die langfristige Kostensenkung durch den Einsatz von serviceorientierten Systemen, die höhere Flexibilität, die Möglichkeit, verteilte Anwendungen auch über die Fahrzeuggrenzen hinaus einzusetzen, die Hochintegration auf wenigen zentralen Komponenten aufgrund der hohen Funktionsvernetzung und die damit verbundene Update-Fähigkeit, sowie die Trennung von Aktuator-nahen Regelschleifen und Regelschleifen für Kundenfunktionen sind nur einige Beispiele, die dabei zu nennen sind.

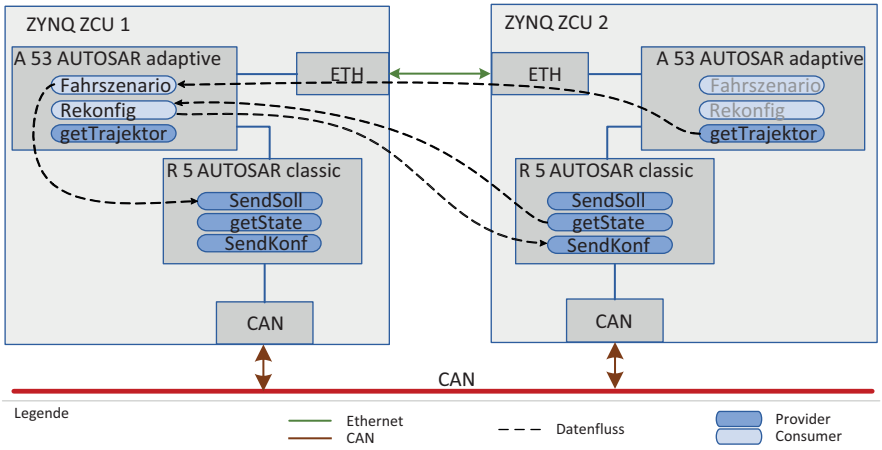

Abbildung 3.9: Eigener Ansatz zur Projektrealisierung

Die Arbeit wird sich aus einer hardware- und einer softwarebasierten Redundanz zusammensetzen. Dabei kommt bei der softwarebasierten Redundanz eine serviceorientierte Architektur zum Einsatz. Bei der hardwarebasierten Redundanz kommt zum einen die Simplex-Architektur [8] zum Einsatz, aber auch ein Konzept, wie es aus dem SAFER-Projekt [83] bekannt ist. In der Diplomarbeit „Entwicklung einer heterogenen Hardware/Software-Architektur für Fail-Operational-Systeme“ [41] wird die Umsetzung der Simplex-Architektur auf einem MPSoC exakt beschrieben.

Für die Realisierung wird ein zweischichtiges Konzept umgesetzt, das zunächst eine Rekonfiguration innerhalb der Komponente vorsieht. Abschließend wird eine Rekonfiguration von einer Komponente auf eine zweite Komponente implementiert. Mit dieser Basis wird bezüglich der Systemabstraktionsebenen auf allen Ebenen die Methodik der dynamischen Rekonfiguration durchgehend systematisch eingesetzt.

Das zugrundeliegende technische Konzept basiert auf einem Xilinx Zynq Ultrascale+ Board. Das Board ist in zwei Hauptkomponenten unterteilt: Das Processing System (PS) und die Programmable Logic (PL). Die zentralen Komponenten des PS sind: Eine Application Processing Unit (APU) und eine Real-Time Processing Unit (RPU). Weiterhin steht eine Graphics Processing Unit (GPU), sowie externe Interfaces (Ethernet und CAN) zur Verfügung. Die PL kann spezifisch konfiguriert werden und ist an das PS angebunden. Beide Teile werden in dieser Dissertation verwendet. Die APU stellt hohe Rechenleistung bereit und es kommt auf ihr ein Linux OS zum Einsatz, unter dem die Applikation in Form des Trajektorien-Providers läuft. Die RPU dient als Schnittstelle zwischen der APU und den CAN-Schnittstellen.

Das gesamte System besteht aus zwei Boards mit identischer Architektur und wird in dieser Dissertation als DRS bezeichnet. Die Aufgabe jedes einzelnen Boards (Nominal oder Rückfallebene) wird durch die SW-Variante bestimmt. Die Ethernet-Schnittstelle wird zur Kommunikation zwischen den beiden Boards verwendet. Die beiden CAN-Schnittstellen werden zur Anbindung des DRS an den Simulator (vgl. Abschnitt 5.1.4) eingesetzt.

Aufbauend auf der Arbeit „Entwicklung einer heterogenen Hardware/Software-Architektur für Fail-Operational-Systeme“ [41] wird die Dynamic Simplex Architecture (DSA) in dieser Dissertation für den Einsatz als redundante und Real-Time-fähige Komponente verwendet, die die zuverlässige Kommunikation zu den CAN-Schnittstellen sicherstellt (vgl. Abschnitt 6.1.2, Abschnitt 6.1.3 und [96]). Aus diesem Teil ist auch ein Patent entstanden [P4].

Die Kommunikation zwischen den Boards, also der Applikation in Form der Provider und der Consumer wird über SOME/IP realisiert (vgl. Abschnitt 6.1.5 und Abbildung 6.15). Die Kommunikation zwischen den Prozessoren (APU, RPU, MB und Platform Management Unit (PMU)) verwenden ein proprietäres Protokoll.

Zusätzlich zu den technischen Umfängen wurden einige Methoden in dieser Dissertation angewendet. Im Rahmen der Entwicklung von Systemen in der Automobilbranche werden immer wieder Simulationsmöglichkeiten benötigt, um die Ergebnisse, Ideen und Konzepte zu evaluieren. Auf dem Weg bis zum fertigen Produkt werden Evaluierungsschleifen durchgeführt. Mit Hilfe des Active Replay (vgl. Abschnitt 6.2) können in dieser Arbeit Möglichkeiten aufgezeigt werden, das System zu evaluieren, zu testen und zu qualifizieren. Dabei wird die ISO 26262 angewendet (vgl. Abschnitt 5.2.1).

Nach Abschluss der Arbeit liegt ein Konzept vor, wie dynamische Rekonfiguration im Kraftfahrzeug im Fail-Operational-Kontext umgesetzt werden kann. Anhand dieses Aufbaus für eine Anwendung von dynamischer Rekonfiguration im Fail-Operational-Kontext wird das Konzept durch ein Experiment nachgewiesen und die Messung der Rekonfigurationszeiten durchgeführt.

Neben der Umsetzung des Minimum Viable Products (MVPs) werden im nächsten Abschnitt zwei weitere Aspekte beschrieben. Zum einen wird eine Modellierung des Systems dargestellt (vgl. Abschnitt 5.3). Auf diese Weise kann der beschriebene Ansatz auch auf andere Problemstellungen für dynamische Rekonfiguration angewendet werden. Mit Hilfe des erstellen Modells kann die maximale Ausführungszeit abgeschätzt werden (vgl. Abschnitt 6.3). Zum anderen wird eine Zuverlässigkeitsanalyse mit Hilfe der Markov-Ketten Monte Carlo-Methode (MCMC) (vgl. Abschnitt 5.4, Abschnitt 6.4 und [22]) durchgeführt. Der vorgestellte Ansatz kann wiederum auch auf andere Architekturen angewendet werden, um die Schwachstellen in einer frühen Entwicklungsphase zu identifizieren oder auch verschiedene Architekturen miteinander zu vergleichen.

In diesem Kapitel ist der Stand der Technik beschrieben. Einzelne Arbeiten und Projekte wurden auf die zuvor eingeführten Systemabstraktionsebenen eingeordnet und beschrieben. Auf diese Weise wird ein Verständnis für das Themenfeld geschaffen, in dem diese Arbeit platziert ist. Aus dieser Einordnung wurden die bestehenden Lücken abgeleitet, die im nächsten Kapitel bei der Detaillierung des eigenen Ansatzes aufgegriffen werden. Es wird dort zunächst die Motivation herausgestellt, des Weiteren wird die verwendete Methodik präsentiert und abschließend eine ausführliche Anforderungsanalyse durchgeführt. Später wird darauf aufbauend die konkrete Umsetzung anhand von Beispielen gezeigt und deren Evaluierung vollzogen.

4 Ganzheitliche Methodik und Anforderungen an dynamische Rekonfiguration

In den folgenden Kapiteln 5 und 6 werden Methoden und Ansätze zum Einsatz von dynamischer Rekonfiguration für Fail-Operational-Systeme im Automobil beschrieben. In diesem Kapitel werden diese Methoden und Ansätze zueinander in Bezug gesetzt und gesamtheitlich betrachtet. Die Motivation dazu wurde in Abschnitt 3.4 dargelegt. Dieses Kapitel zeigt den Überblick der in dieser Arbeit vorgestellten Methoden und Ansätze für den Einsatz der dynamischen Rekonfiguration auf den in Abbildung 2.1 eingeführten Ebenen und zeigt auf, wie diese miteinander kombiniert werden können. Es wird dabei herausgearbeitet, wie eine Kombination der Ansätze auf den einzelnen Ebenen möglich ist.

In dieser Dissertation liegt der Fokus auf der Umsetzung einer dynamischen Rekonfiguration basierend auf serviceorientierten Architekturen. Aus diesem Grund sind die Ausarbeitungen im Folgenden stärker auf die Software ausgerichtet. Die eingesetzten Hardware-Plattformen werden dabei allerdings ebenfalls mit betrachtet. Eine dynamische Rekonfiguration zu Laufzeit der PL eines FPGAs ist nicht Bestandteil dieser Arbeit. Dennoch wird die PL eines MPSoC bei der Betrachtung der Lösungsansätze mit berücksichtigt. Deren dynamische Rekonfiguration im Sinne einer dynamisch partiellen Rekonfiguration wird hierbei nicht durchgeführt.

Die Anforderungen an die Ansätze fallen je nach Anwendungsfall sehr unterschiedlich aus und sind aus diesem Grund zu berücksichtigen. Die notwendigen Anpassungen fließen in Form von Parametrierungen oder Konfigurationen in der Umsetzung der Ansätze und Methoden mit ein. Die Komponenten hierfür bieten die Voraussetzung dafür bereits. Weiterhin wird dargelegt, in welchem Ausmaß die erarbeitete Metho-

dik auf andere Architekturen übertragbar ist und anwendungsbezogen skaliert bzw. parametriert werden kann. Dazu werden etwaige Einschränkungen aufgezeigt.

4.1 Motivation und Abgrenzung

In dem vorangegangenen Kapitel 3 wurde ein Überblick über verschiedene Ansätze zur Realisierung von dynamisch rekonfigurierbaren Systemen auf den unterschiedlichen Ebenen dargelegt. Eine spezifische, ganzheitliche Methode ist jedoch nicht vorhanden. Um die dynamische Rekonfiguration in den Automobilbereich zu bringen, wird im Folgenden ein entsprechender Entwurf vorgestellt.

- Modellbasierter Entwurf (vgl. Abschnitt 5.3):
  - Erfassung eines modellbasierten Entwurfs für die dynamische Rekonfiguration.
  - Verknüpfung des modellbasierten Entwurfs mit einer Simulation der dynamischen Rekonfiguration.
  - Modellhafte Berücksichtigung der Komplexität vom Bauteil bis zum System-of-Systems.
- Simulation in frühen Entwicklungsphasen (vgl. Abschnitt 6.2):
  - Ermittlung von KPI[1]s für die Systemauslegung.
  - Analyse des Echtzeitverhaltens.
  - Simulative Betrachtung und Einbeziehung eines HILs[2] in den Entwicklungsprozess von dynamisch rekonfigurierbaren Systemen.
  - Einsatz von Simulationstechniken sowohl zur Ermittlung als auch zur Verifikation.

[1] Key-Performance-Indicator
[2] Hardware in the Loop

- Evaluation von dynamisch rekonfigurierbaren E/E-Systemen (vgl. Abschnitt 5.4):
  - Berücksichtigung der Zertifizierung, um dynamisch rekonfigurierbare Systeme im Automobil für sicherheitskritische Anwendungen zu etablieren.
  - Einbindung einer iterativen Modellanpassung in frühen Phasen der Entwicklung.
  - Durchführung der Bestimmung des Sicherheitslevels des Gesamtsystems.

Im folgenden Abschnitt 4.2 wird ein Lösungskonzept für die oben angegebenen Punkte präsentiert.

Die Methodik integriert:

- eine Software- und Hardware-Bibliothek für die dynamische Rekonfiguration,
- die Anforderungsanalyse zur Ermittlung der KPIs für die Systemauslegung,
- die Ableitung notwendiger Anforderungen,
- den modellbasierten Entwurf,
- eine Evaluierungsmöglichkeit in der frühen Phase,
- die Ermittlung der Entwurfsszenarien,
- sowie den Test und die Verifikation.

Diese Umfänge werden in den etablierten Entwicklungsprozess im Automobilbereich eingeordnet. Dieses Vorgehen wird ebenfalls im folgenden Abschnitt 4.2 gezeigt. Konkrete Konzepte und die Realisierung des Prototyps werden in Kapitel 5 und Kapitel 6 dargestellt.

4.2 Kontextliche Methodik und Einbindung in den Entwicklungsprozess

Im Kern der Arbeit steht eine Systematik, um dynamische Rekonfiguration für sicherheitskritische Anwendungsfälle, sowohl für Systeme als auch über Fahrzeuggrenzen

hinaus, zu entwickeln. Mithilfe der erarbeiteten Methodik zur Simulation und Bewertung wird insbesondere eine notwendige Zertifizierung unterstützt. In den Kapiteln 5 und 6 werden Ansätze und Konzepte vorgestellt, die sowohl unabhängig voneinander als auch auf einem MPSoC oder mehreren MPSoCs jeweils in Kombination zum Einsatz kommen können. Insgesamt kann durch die Kombination das Abfangen von Fehlern und die Aufrechterhaltung des Fail-Operational-Verhaltens der entsprechenden Funktion gewährleistet werden. In Abschnitt 6.4 werden die genauen Auswirkungen dazu dargelegt. Durch die konsistente Berücksichtigung der Ebenen wird hierdurch eine gesamthafte Darstellung erreicht.

Die drei adressierten Ebenen sind hierbei die Ebene System-of-Systems, die Systemebene und die Komponentenebene. Weitere Ebenen können der Methodik jedoch hinzugefügt werden. Zur Erreichung des aus der GuR resultierenden Safety-Ziels wird die in Abschnitt 6.4 beschriebene Methodik herangezogen und durch Kombination kann das entsprechende Ziel iterativ erreicht werden. Mögliche Kombinationen sind in Tabelle 4.1 dargestellt.

Tabelle 4.1: Kombinationsmöglichkeiten der einzelnen Ebenen der dynamischen Rekonfiguration

Ebene	System-of-Systems	System	Komponente
System-of-Systems		x	x
System	x		x
Komponente	x	x	

Die Auswahl der Konstellation wird auf Basis der gewünschten Tiefe der Rekonfiguration durchgeführt, bzw. der Ergebnisse aus Abschnitt 6.4. Betrachtet man die verschiedenen Ansätze, wird deutlich, dass diese sowohl in HW als auch SW abgebildet werden. Das übergreifende Monitoring und Steuern wird durch die Cross-Layer-Rekonfigurationslogik übernommen (vgl. Abschnitt 5.3.2). Dabei kann diese Logik als global operierende Einheit dienen, von der aus die Rekonfiguration startet. Basierend auf den vorgestellten Konzepten kann die Rekonfiguration je nach Anwendungsfall auf den verschiedenen Ebenen ausgeführt werden. Für den Einsatz der vorgestellten Konzepte im Kontext von sicherheitskritischen Anwendungen mit ASIL-Einstufung und Echtzeitverhalten, werden in dieser Dissertation die notwendigen Bewertungsmethoden zur Verfügung gestellt (vgl. Abschnitt 5.4, Abschnitt 6.3 und Abschnitt 6.4).

Die Übertragbarkeit und die Skalierbarkeit der vorgestellten Ansätze sind grundsätzlich gegeben. Je nach zugrunde liegender HW- und SW-Plattform (vgl. Abschnitt 5.5.1

und Abschnitt 5.5.2) sind verschiedene Einsatzmöglichkeiten vorhanden. In der vorliegenden Arbeit wurde auf eine heterogene Multicore-Architektur mit PL aufgebaut. Aber auch heterogene Commercial Off The Shelf(COTS)-Plattformen mit und ohne PL oder dedizierten HW-Beschleunigern für KI[3]-Ansätze können Verwendung finden. In Tabelle 4.2 ist die Verwendung der Ansätze in einer COTS-Architektur dargestellt.

Tabelle 4.2: Einsatzmöglichkeiten der einzelnen Ebenen der dynamischen Rekonfiguration in COTS

Ebene	Mikro-prozessor	Mikro-controller	HW-Beschleuniger	Bemerkung
System-of-Systems	x	x	x	Die Rekonfiguration kann rein in SW umgesetzt werden.
System	x	x	x	Die Umsetzung ohne PL ist grundsätzlich möglich.
Komponente		(x)		Erfordert einen Safety-Prozessor oder eine PL.

Somit kann ein reales Hochintegrationsszenario unter Berücksichtigung von echtzeitfähigen Regelkreisen mit niedrigen Latenzen, rechenintensiven Berechnungen auf den High-Performance-Kernen sowie dedizierten HW-Beschleunigern für KI-Umfänge zum Beispiel in der Bildverarbeitung dargestellt werden. Eine PL ist in der frühen Phase der Entwicklung von großem Vorteil und bietet entsprechende Flexibilität, die in späteren Phasen zum Beispiel durch den Einsatz von einem ASIC zur kostenoptimalen Auslegung ersetzt wird. Die Berücksichtigung der verschiedenen Architekturen stellt die Übertragbarkeit der Ansätze weitgehend sicher. Deren Skalierbarkeit wird durch die Anzahl der zur Verfügung stehenden Cores und den Einsatz der Virtualisierungslösung nicht zuletzt aber durch die verwendete Middleware umgesetzt. Auf diese Weise wird ein breites Anwendungsspektrum abgedeckt. Entscheidend ist bei der Verwendung der Ansätze die Untersuchung auf gemeinsame Fehlerquellen (CCF[4]), die gesondert durchzuführen ist.

Ein weiteres Ziel dieser Arbeit, ist eine erweiterbare Bibliothek, die wiederverwendbare Pattern zur Verfügung stellt, um Ingenieuren einen Startpunkt für den Einsatz von dynamischer Rekonfiguration mit Fail-Operational-Kontext im Automobil zu bieten. Auf diese Weise wird die Beherrschbarkeit der Komplexität gewährleistet und letzt-

[3] Künstliche Intelligenz
[4] Common Cause Failure

endlich die Homologation dieser Umfänge ermöglicht. Für die Wiederverwendbarkeit der Ansätze wurde auf die Generik geachtet, um die Methoden auf unterschiedlichste Anwendungen anzupassen, sie zu bewerten und somit den entsprechenden Nachweis für die Homologation zu erbringen. Die gezeigten SW- und HW-Pattern zum Beispiel in Form der Architekturen (vgl. Abschnitt 5.1.2) und der verwendeten Protokolle (vgl. Abschnitt 5.5.1) sind aufgrund der eingeführten Ebenen modular zu verwenden und den entsprechenden Anforderungen anzupassen. In Abbildung 4.1 ist dieser ganzheitliche Ansatz dargestellt:

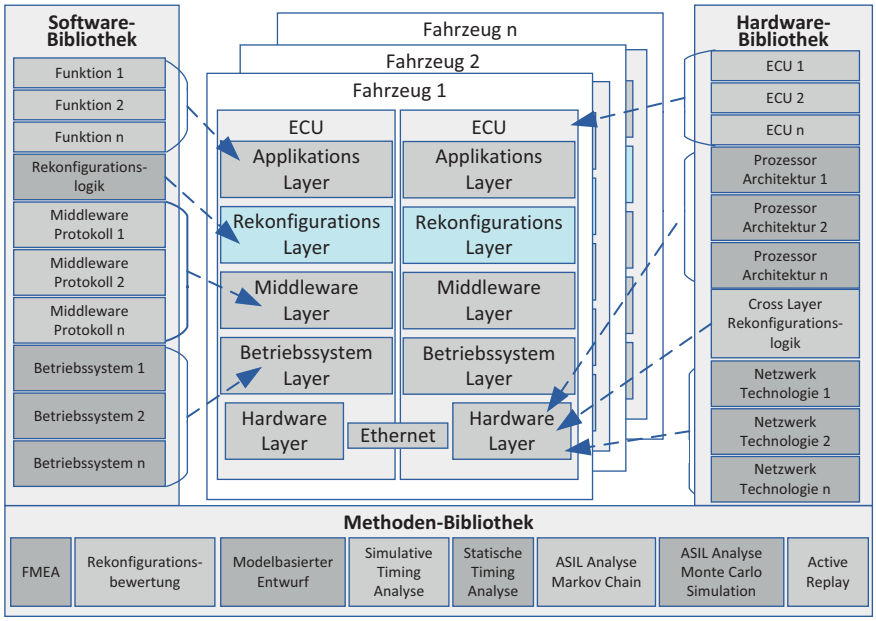

Abbildung 4.1: Ganzheitlicher Ansatz zur Entwicklung von dynamisch rekonfigurierbaren E/E-Systemen

Dieser Lösungsbaukasten wird in den etablierten Entwicklungsprozess der Automobilindustrie integriert. Auf diese Weise wird es möglich, sicherheitskritische Echtzeitsysteme basierend auf der dynamischen Rekonfiguration einzelner Ebenen zu entwickeln. Beispielsweise können serviceorientierte Architekturen (vgl. Abschnitt 5.3) mit verschiedenen Protokollen (vgl. Abschnitt 5.3.1 und Abschnitt 5.5.1) kombiniert werden. In der HW-Bibliothek finden sich die unterschiedlichen HW-Architekturen wieder (vgl. Tabelle 4.2), die mit den E/E-Architekturen (vgl. Abschnitt 5.1.2) verbunden werden. Weiterhin sind hier auch die Konzepte aus Abschnitt 5.3.2 und Abschnitt 6.1.6 einge-

bunden. Je nach Gesamtkonfiguration werden die entsprechenden Netzwerktopologien (z.B.: CAN oder Ethernet) eingesetzt. Der Einsatz in Fail-Operational-Systemen wird dann in der Kombination mit den SW-Anteilen ermöglicht.

Der zugrunde liegende modellbasierte Ansatz ermöglicht die nachhaltige Wiederverwendbarkeit von Teilumfängen, da die Funktion zunächst auf der logischen Architektur spezifiziert wird. Dabei kann diese Ebene der E/E-Architektur innerhalb einer Fahrzeuggeneration und darüber hinaus stabil bleiben [117]. Die Umsetzung aus dem Hardware-/ Software-Baukasten auf der technischen Ebene der Architektur kann sich hingegen in einem anderen Zyklus ändern und unterschiedliche Ausprägungen annehmen. Basierend auf der Evaluierung der Ergebnisse können somit iterative Verbesserungen beziehungsweise Anpassungen der Architektur vorgenommen werden (vgl. Abschnitt 6.4). Nach der Modellierung stehen domänenübergreifende Simulationsergebnisse zur Verfügung, die zur Zertifizierung herangezogen werden. Der Lösungsbaukasten ebnet den Weg für die Entwicklung von dynamisch rekonfigurierbaren Systemen für sicherheitskritische Anwendungen auf allen Ebenen.

Die Bibliothek kann mit weiteren Konzepten erweitertet werden. Deren Auswahl ist stark von der Architektur und dem Einsatz abhängig. Die Einbindung in den Entwicklungsprozess stellt deswegen einen weiteren Punkt dieser Dissertation dar.

Im Automobilbereich und insbesondere bei der Entwicklung von sicherheitskritischen Systemen kommen das V-Modell (vgl. Abbildung 2.5) und die ISO 26262 (vgl. Abbildung 2.7) zum Einsatz. Diese sieht vor, zunächst die Zielfunktion zu bestimmen und deren Spezifikation durchzuführen. Im Folgenden kann dann das Sicherheitslevel und das Sicherheitsziel ermittelt werden (vgl. Abschnitt 5.2.1 und Abschnitt 6.2.3). Daraus ergeben sich die Anforderungen an die zu verwendenden Safety-Mechanismen, um den Systementwurf durchzuführen.

Aus den Anforderungen an die Rekonfigurationszeit können die Mechanismen aus der Bibliothek ausgewählt werden und im weiteren Verlauf des Systementwurfs verwendet werden. Die Auswahl kann anhand der Tabelle 4.1 durchgeführt werden. Zusätzlich zu diesen Anforderungen gibt es im Entwurfsraum einschränkende Größen, die aus der verwendeten HW-Architektur resultieren. Tabelle 4.2 stellt diesen Zusammenhang dar, der wiederum bei der Auswahl berücksichtigt wird. Insbesondere die Berücksichtigung der non-funktionalen Anforderungen und die Iteration bei der Auswahl der

Mechanismen ausgehend von der MCMC-Evaluierung führen bei der Entwicklung letztendlich zur Umsetzung. In Abbildung 4.2 ist der Ablauf dazu dargestellt.

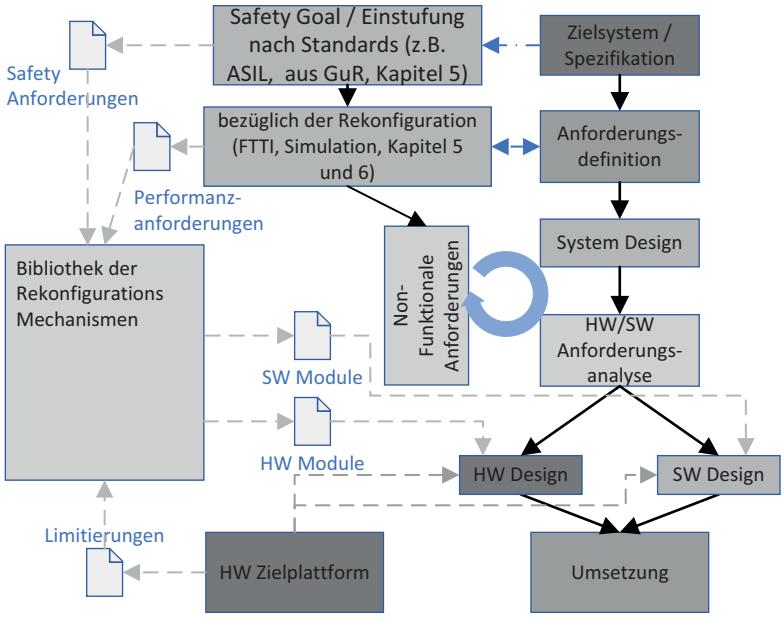

Abbildung 4.2: Der Entwicklungsprozess mit Einbezug des vorgestellten Bibliothekskonzepts und der iterativen Bewertung der non-funktionalen Anforderungen

Somit werden die Eingangsartefakte der Bibliothek zur Auswahl der Mechanismen bestimmt. Aus der Bibliothek werden die bereits existierenden, wiederverwendbaren HW- und SW-Module in den Entwurfsprozess zurückgeführt, um durch die MCMC-Evaluierung das Sicherheitsziel zu erreichen.

Ergibt die Evaluierung eine Erreichung des Sicherheitsziel, kann eine Optimierung des Gesamtsystems zum Beispiel unter Kostenaspekten durchgeführt werden. Eine automatisierte Optimierung wurde in dieser Dissertation jedoch nicht weiterverfolgt. In Abbildung 4.3 ist der Auswahlprozess der Mechanismen innerhalb der Bibliothek dargestellt. Die zu durchlaufenden Schritte sind dabei detailliert dargestellt.

Durch dieses systematische und methodische Vorgehen und die definierten Ein- und Ausgangsartefakte können reproduzierbare und nachvollziehbare Ergebnisse in der Entwicklung von sicherheitskritischen Systemen basierend auf der dynamischen Re-

konfiguration erreicht werden. Zusammen stellt das den zentralen Baustein für die Zertifizierung bzw. der Homologation dar.

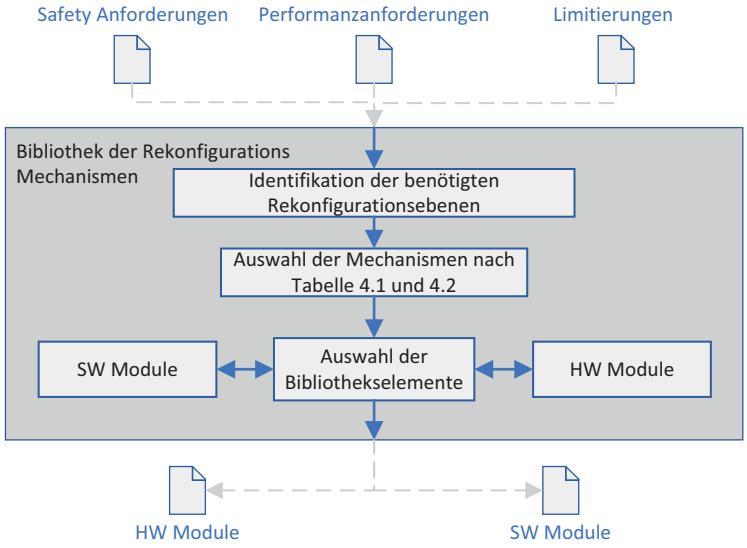

Abbildung 4.3: Auswahlprozess der Mechanismen innerhalb der Bibliothek

Im folgenden Abschnitt werden Anforderungen abgeleitet, die später durch die Methodiken erfüllt werden müssen. Dabei spielt die Identifikation notwendiger Parameter zur Berücksichtigung während der Simulation eine entscheidende Rolle. Anhand von relevanten Anwendungsfällen zum Aufbau der E/E-Architektur, die in der Konzeptphase zur Evaluierung verwendet werden, kann später der Einsatz dieser Methodiken aufgezeigt werden.

4.3 Anforderungen

4.3.1 Modellbasierter Entwurf

Einige Verbesserungspotenziale wurden in Abschnitt 4.1 bereits erwähnt. Aufbauend auf diesen Potenzialen soll eine architekturweite Systemevaluierung durchgeführt werden. Der modellbasierte Entwurf ermöglicht dafür eine konsistente und formale Beschreibung. Es entsteht dadurch die notwendige Nachvollziehbarkeit und Transparenz.

Als Folge davon können ebenenübergreifende Artefakte abgeleitet werden, wie zum Beispiel die Anforderungen an die ausführenden Hardware-Komponenten. Für eine nachfolgende Simulation sind diese abgeleiteten Anforderungen essenziell, um das Verhalten der Hardware-Komponenten zu berücksichtigen. Dieser Umfang findet auf der logischen Ebene statt, die realisierungsunabhängig ist und eine Stabilität fahrzeuggenerationenübergreifend ermöglicht. Eine Schnittstellenspezifikation der Funktions- und Informationseinheiten sorgt dafür, dass bestehende logische Architekturen nicht geändert werden müssen.

Durch den modellbasierten Entwurf kann eine Erhöhung der Wiederverwendbarkeit der zu entwickelnden Funktionen gewährleistet werden. Mithilfe der Funktionsbibliothek kann somit zu einer Steigerung der Produktivität beigetragen werden. Die getrennte, unabhängige Modellierung der Applikation und der ausführenden Plattform ist eine Methode zur Entwurfsraumexploration[5] der Gesamtarchitektur [113]. Auf diese Weise können Entwurfsentscheidungen getroffen werden, wie die Wahl der Topologie der Hardware-Architektur (zum Beispiel zentrale oder verteilte Steuergeräte-Architektur), die Verwendung von Single- oder Multicore-Prozessoren, die Verteilung von Funktionen auf die ausführende Hardware und der Einfluss auf das Verhalten der Anwendung [1]. Mithilfe der Entwurfsraumexploration kann somit in frühen Entwicklungsphasen das System rechtzeitig evaluiert werden.

4.3.2 Key-Performance-Indicator für die Systemauslegung

In der Entwicklung von E/E-Systemen im automobilen Umfeld spielt die Anforderungsanalyse und die Spezifikation eine entscheidende Rolle. Die Ermittlung von KPIs zur Systemauslegung ist eine nicht zu unterschätzende Aufgabe und von großer Bedeutung. Die KPIs beeinflussen explizit das Systemverhalten. In der Kundenwahrnehmung entscheiden diese Parameter häufig über die Qualität der Funktionsausprägung bzw. darüber, ob bei Abnahmeversuchen zum Beispiel durch die BASt die Zulassung erfolgen kann. Auf der anderen Seite bedeuten zu enge Vorgaben bei den KPIs hohe Ausgaben auf der Fahrzeuginfrastrukturseite. Die Systemkosten steigen und machen es dadurch unattraktiv. Im schlimmsten Fall wird eine Innovation verhindert. Dieser Prozess ist bereits in Abbildung 4.2 dargestellt.

[5] *englisch: Design Space Exploration*

Für dynamisch rekonfigurierbare Systeme gibt es für den Automobilbereich diese KPIs nicht, es kann also auf keine Erfahrungswerte zurückgegriffen werden. Um die Arbeit zur Entwicklung von dynamisch rekonfigurierbaren Systemen zu erleichtern und die richtigen KPIs zu finden, zu identifizieren und zu bestimmen, soll eine Methodik etabliert werden. Diese Methodik soll Bestandteil von dem vorhandenen Entwicklungsprozess sein. Vorhandene Mittel und Möglichkeiten sollen dafür verwendet werden. Diese können zum Beispiel virtuelle Simulationsumgebungen zur Ermittlung von Rekonfigurationszeiten oder auch bestehende Softwarewerkzeuge sein, um zum Beispiel die Kriterien der Echtzeitfähigkeit zu ermitteln. Hierfür gilt es unter anderem die maximale Ausführungszeit bezogen auf den Anwendungsfall abzuschätzen.

In dieser Arbeit wurde das beispielhaft für die Lenkung und die Bremse durchgeführt. In Abschnitt 5.2.1, Abschnitt 6.1.6 und Abschnitt 6.2 sind die entsprechenden Ansätze zur Ermittlung der KPIs dargestellt. Insbesondere die Rekonfigurationszeit und das Sicherheitsziel sind dabei die entscheidenden KPIs. Die jeweilige Erreichung sind in Abschnitt 6.3.3 und Abschnitt 6.4.3 dargelegt.

4.3.3 Evaluierungsmöglichkeit in früher Entwurfsphase

Der Evaluierungsmöglichkeit von Systemen in einer frühen Entwurfsphase kommt eine besondere Bedeutung zu. Zu diesem Zeitpunkt können Stellhebel identifiziert und verändert werden, um eine Systemauslegung gemäß den Vorgaben zu erreichen. Bevor die Umsetzungsphase beginnt, soll also bereits in der frühen Entwurfsphase die Möglichkeit gegeben sein, die Systemparameter zu verifizieren. Je später dieses im Entwicklungsprozess passiert, umso kostenintensiver werden die Systemkosten bei der Entwicklung dieser Komponenten. Möglichkeiten dazu bestehen in simulativen Ansätzen, die entsprechend angewendet, notwendige Erkenntnisse hervorbringen.

Auch für ein dynamisch rekonfigurierbares System muss diese Möglichkeit gegeben sein. Für sicherheitskritische Systeme ist dabei ein wesentlicher Aspekt die ASIL-Fähigkeit des Systems. Zunächst werden mit Hilfe der GuR-Analyse potenzielle Gefahren der Betrachtungseinheit identifiziert, eingestuft, Ziele definiert und einem ASIL zugeordnet. Es soll nun in der frühen Entwurfsphase eine Möglichkeit gefunden werden, durch iterative Veränderung das System den Anforderungen anzupassen.

Neben dieser zu erarbeiteten Methodik soll es auch möglich sein, unterschiedliche E/E-Architekturen zu bewerten. Im Rahmen einer Konzeptausschreibung für ein neu zu entwickelndes System soll das richtige Konzept für die nächste Serienentwicklung ausgewählt werden. Dazu ist es notwendig dieses evaluieren zu können. Hierfür müssen die entsprechenden Parameter, Metriken und das Vorgehen identifiziert werden, um später die Bewertung durchführen, Potenziale zu ermitteln und die entsprechende Auswahl zu treffen.

4.3.4 Entwurfsszenarien und Validierung

Im Rahmen der Konzeptarbeit werden in der Entwicklung von automobilen E/E-Systemen zunächst verschiedene Konzeptentwürfe erstellt. Im nächsten Schritt werden diese Entwürfe bewertet, um schließlich das richtige Konzept weiter voranzutreiben, ein MVP zu erstellen, dieses zu evaluieren und schließlich das finale Konzept zur Serienreife zu bringen. Für diese erste Konzeptentwurfsphase wird ein weiteres Vorgehen benötigt. Hierfür werden zunächst unterschiedliche Entwurfsszenarien aufgestellt und aus den vorgeschlagenen Entwürfen kann dann das geeignete Szenario für den entsprechenden Anwendungsfall ausgewählt werden. Beispielhaft wurde in dieser Arbeit dieses Vorgehen zum Entwurf von dynamisch rekonfigurierbaren Systemen in Abschnitt 5.5.2 durchgeführt. Diese Szenarien sind eines der zu definierenden Artefakte in dem Entwicklungsprozess und fließen als Limitierungen in den Gesamtprozess mit ein. In Abbildung 4.2 und Abbildung 4.3 ist jeweils dieses Artefakt im Gesamtkontext bzw. bezogen auf die Bibliothek dargestellt.

4.3.5 Test und Verifikation

Das V-Modell ist das etablierte Vorgehensmodell zur Entwicklung von E/E-Systemen im Fahrzeug. Wichtige Prozessschritte innerhalb des Modells geben dabei zu unterschiedlichen Phasen in der Entwicklung wiederkehrende Tests und Verifikationen des entwickelten Systems vor (vgl. Abbildung 2.5). Dabei wird das System überprüft, ob es fehlerfrei funktioniert und ebenso, ob die Anforderungen an das System in Form der KPIs auch erfüllt werden.

Um zu diesen Punkten eine Aussage zu liefern, soll für den Test und die Verifikation wiederum ein vorhandener Simulator verwendet werden, um die erarbeiteten KPIs zu testen.

4.3.6 Zusammenfassung der Anforderungen

In diesem Kapitel wurde die ganzheitliche Methodik für den Einsatz von dynamischer Rekonfiguration für sicherheitskritische Echtzeitsysteme im Automobil vorgestellt. Beginnend mit der Motivation und der Abgrenzung (Abschnitt 4.1), weiterführend mit der kontextlichen Methodik und Einbindung in den Entwicklungsprozess (Abschnitt 4.2) wurden abschließend die Anforderungen einzeln herausgearbeitet, die im Folgenden zusammengefasst werden als einzelne Anforderungen (*englisch: requirements* (*REQ*)) versehen mit einer fortlaufenden Nummerierung. Diese einzelnen Anforderungen werden in den Kapiteln 5 und 6 dieser Dissertation beantwortet.

Modellbasierter Entwurf

REQ01 Für die dynamische Rekonfiguration soll eine SW-Bibliothek vorhanden sein, die einzelne Lösungselemente für die Rekonfiguration bereitstellt. Diese besteht aus vorhandenen Lösungselementen und kann entsprechend erweitert werden.

REQ02 Zusätzlich soll eine HW-Bibliothek vorhanden sein, die die dynamische Rekonfiguration unterstützt. Bereits vorhandene Lösungen auf HW-Ebene, die eine dynamische Rekonfiguration zum Beispiel in einem MPSoC oder FPGA zeigen, finden sich hier wieder.

REQ03 Der Lösungsansatz soll eine Rekonfiguration auf den Systemebenen, die im Kapitel 2 und Kapitel 3 aufgezeigt wurden, ermöglichen.

REQ04 Die dynamische Rekonfiguration soll nicht nur auf SW- oder HW-Lösungen beschränkt sein, sondern vielmehr den Nutzen aus beiden ziehen können. Die Rekonfigurationslogik soll dabei sowohl für die SW- als auch HW-Ebene berücksichtigt sein.

REQ05 Um von vornherein die zunehmende Komplexität von E/E-Systemen zu berücksichtigen und eine hohe Qualität und eine Wiederverwendbarkeit zu gewährleisten, soll ein modellbasierter Entwurf zum Einsatz kommen.

REQ06 Aufgrund des geplanten Einsatzes für sicherheitskritische Anwendungen muss es möglich sein, durch Simulation notwendige Parameter zur Bestimmung des Echtzeitverhaltens zu ermitteln.

REQ07 Um den Lösungsbaukasten im Automobilbereich zu etablieren, soll der Lösungsansatz in den Standardentwicklungsprozess der Automobilindustrie eingeordnet sein.

Key-Performance-Indicator für die Systemauslegung

REQ08 Für die Systemauslegung in der Anforderungsanalysephase sollen relevante KPIs ermittelt werden. Diese sollen dann im weiteren Verlauf des Entwicklungsprozesses in den folgenden Phasen Wiederverwendung finden, wie zum Beispiel bei der Konzeptentwurfs- und -validierungsphase.

REQ09 Zur Ermittlung der relevanten KPIs soll geprüft werden, welche Softwarewerkzeuge und Methodiken dazu eingesetzt werden können. Insbesondere der Einsatz von Simulationssoftware ist dabei zu prüfen.

Evaluierungsmöglichkeit in früher Entwurfsphase

REQ10 Um das Konzept in einer frühen Phase auf das Erreichen seiner Anforderungen überprüfen und anpassen zu können, soll die Möglichkeit vorhanden sein, eine iterative Anpassung des Entwicklungskonzepts durchzuführen.

REQ11 Bei der Evaluierung soll insbesondere die Überprüfung des für sicherheitskritische Anwendungen wesentliche Merkmal des erreichten ASIL in einer frühen Phase möglich sein. Daraus ableitend soll wiederum der iterative Anpassungsprozess gestartet werden.

REQ12 Mit Hilfe der Evaluierung soll es möglich sein, verschiedene Konzepte untereinander zu vergleichen und zu bewerten. Daraus resultierend sollen dann die Konzeptentscheidungen gefällt werden.

Entwurfsszenarien

REQ13 Die Entwurfsentscheidung für dynamische rekonfigurierbare Systeme soll auf Basis verschiedener aufgestellter Szenarien getroffen werden. Hierfür ist es erforderlich verschiedene Entwurfsszenarien zu ermitteln und der Entscheidungsfindung herbeizuführen.

REQ14 Die unterschiedlichen Entwurfsszenarien sollen basierend auf Metriken evaluiert werden und im Anschluss in den Entscheidungsprozess eingesteuert werden. Hierfür ist eine Methodik zu erarbeiten.

Test und Verifikation

REQ15 Gemäß dem V-Modell, dem etablierten Entwicklungsprozess in der Automobilindustrie, muss das entwickelte System verschiedenen Tests und Verifikationen unterzogen werden. Hierbei sind insbesondere die ermittelten KPIs zu evaluieren.

Randbedingungen und allgemeine Anforderungen

REQ16 Aufgrund der Komplexität von E/E-Systemen kann das Konzept im Gesamtverbund nicht durch den OEM allein entwickelt werden, sondern soll vielmehr im Rahmen des AutoKonf-Projekts umgesetzt und evaluiert werden.

REQ17 Zusätzlich zu der Anforderung der HW-, SW-Bibliothek soll der Lösungsbaukasten verschiedene Middleware-Konzepte berücksichtigen, um die technische Realisierungsebene frei zu gestalten.

REQ18 Neben den zu betrachtenden Middleware-Konzepten sollen auch verschiedene Betriebssysteme, die typischerweise im automobilen Umfeld zum Einsatz kommen, im Lösungsbaukasten enthalten sein.

REQ19 Da sicherheitskritische Anwendungen eine zentrale Rolle spielen, sind die erforderlichen Normen und Standards zur Zertifizierung entsprechend in der Betrachtung zu berücksichtigen.

In dieser Zusammenfassung wurden in sechs Paragrafen die Anforderungen zusammengefasst, die durch diese Dissertation beantwortet werden. Die Anforderungen zum Paragraf Modellbasierter Entwurf werden in Abschnitt 5.3 dargestellt. Die KPIs für die Systemauslegung werden in Abschnitt 5.2.1 und Abschnitt 6.2 detailliert beschrieben. In Abschnitt 6.3 und Abschnitt 6.4 ist die Evaluierungsmöglichkeit in frühen Entwurfsphasen dargelegt. Die Entwurfsszenarien und Validierung ist beschrieben in Abschnitt 5.5.2, Abschnitt 6.1.7, Abschnitt 6.2.4, Abschnitt 6.3.3 und Abschnitt 6.4.3. Der Test und die Verifikation ist in Abschnitt 5.5.3 erklärt. Alle Schritte sind Teil des V-Modells und der ISO 26262 und wurden methodisch und systematisch vollzogen, um dynamische Rekonfiguration für automobile Echtzeitsystem im Fail-Operational-Kontext Zertifizierungs- und Homologationsfähig umzusetzen.

5 Dynamische Rekonfiguration für sicherheitskritische Systeme im automobilen Umfeld

In diesem Kapitel wird ein Überblick über die in dieser Arbeit vorgestellten Methoden zur Entwicklung von dynamisch rekonfigurierbaren System für sicherheitskritische Echtzeitsysteme im automobilen Umfeld gegeben. Dabei wird auf die einzelnen Arbeitsweisen eingegangen und verdeutlicht, wie diese in den Standard-Entwicklungsprozess der Automobilindustrie eingeordnet werden können.

Die Anforderungen, die im vorherigen Kapitel herausgearbeitet wurden, werden im Folgenden wieder aufgegriffen und es wird veranschaulicht, wie diese mit den vorgestellten Vorgehensweisen und Lösungsansätzen erfüllt werden können. Des Weiteren wird gezeigt, wie die erarbeiteten Methoden und Ansätze für verschiedene Architekturkonzepte zum Einsatz gebracht werden können.

Die Methoden und Konzepte in diesem Kapitel sowie deren prototypische Umsetzung bauen auf den eigenen Veröffentlichungen [144, 145, 151, 150, 154, 155] auf.

5.1 Konzept

Das Gesamtkonzept in Form einer Bibliothek ist in Abbildung 4.1 dargestellt. Es folgt ein Überblick der einzelnen Bestandteile dieser Bibliothek. Weiterhin wird die übergeordnete Architektur erklärt (Abschnitt 5.1.2) sowie die Erweiterung dieser Architektur mit dem verteilten Fahrrechneransatz (Abschnitt 5.1.3) und die verwendete Simulationsumgebung (Abschnitt 5.1.4).

5.1.1 Überblick

In dieser Arbeit wurde ein neues, systematisches und parametrierbares Bibliothekskonzept erarbeitet, das aus Methoden und Software- sowie Hardwareansätzen für den Einsatz von dynamischer Rekonfiguration bei sicherheitskritischen Echtzeitsystemen im automobilen Umfeld besteht. Es kann herangezogen werden, um unterschiedliche Architekturen zu entwerfen und dynamische Rekonfiguration im Automobil umzusetzen.

Durch die Kombination der verschiedenen Lösungsansätze kann ein E/E-Architekturentwurf auf den Anwendungsfall zugeschnitten werden. Folgende Methoden wurden dafür erarbeitet:

1. Modellbasierter Entwurf von dynamisch rekonfigurierbaren Systemen
2. Methodik zur Ermittlung der Rekonfigurationszeit
3. Analyse zur Abschätzung der Ausführungszeiten
4. Ende-zu-Ende-Bewertung der Latenz
5. Einordnung von Lösungsszenarien
6. Integration des dynamischen Systemverhaltens in Markov-Ketten
7. Monte-Carlo-Simulation zur Berechnung der komplexen Modellierung

Neben den Methoden sind sowohl Software- als auch Hardwareansätze zum Einsatz gekommen. Bei den Softwareansätzen wurden insbesondere folgende Verfahren untersucht:

1. Geeigneter Einsatz serviceorientierter Architekturen.
2. Untersuchung und Bewertung verschiedener Middleware-Konzepte.
3. Verwendung eines MPSoCs zur Darstellung der Verwendung von High-Performance-Kernen für die Berechnung von Trajektorien[1] und echtzeitfähigen Kernen für die Behandlung von sicherheitskritischen Umfängen

[1] *latein*: *traiectus*, Bahn oder Bewegungspfad eines Objektes

4. Verwendung von zwei High-Performance-Rechnern, verbunden über Ethernet zur Darstellung der dynamischen Rekonfiguration außerhalb von Steuergerätegrenzen.

Neben den Softwareansätzen wurden auch Hardwareansätze in das Gesamtkonzept integriert und unterstützen methodisch das Bibliothekskonzept. Bei den Hardwarekonzepten wurden insbesondere folgende untersucht:

1. Erweiterung der VHDL-Programmierung des FPGAs, um eine gesicherte Zustandsübertragung über CAN im Moment des Kontextwechsels zu ermöglichen (vgl. Abschnitt 6.1.3).

2. Verwendung der Simplex-Methode, um die Echtzeitkommunikation im Fail-Operational-Kontext zwischen dem Real-Time Core des MPSoCs, dem FPGA und dem CAN-Controller zu gewährleisten (vgl. Abschnitt 6.1.1).

In dem Kapitel „Stand der Technik“ wurden verschiedene Entwürfe untersucht und in den erarbeiteten Systemebenen angeordnet. Diese Konzepte können prinzipiell individuell in das Bibliothekskonzept mit aufgenommen werden. In dieser Arbeit wurde das jedoch nicht weiter untersucht. Im Weiteren werden die einzelnen neu erarbeiteten bzw. erweiterten Methoden in diesem Kapitel tiefergehend beschrieben. Auf die exakte Anwendung der Software- und Hardwareumfänge aus dem Bibliothekskonzept wird im Kapitel 6 detailliert eingegangen. Dabei werden die hier beschriebenen Methoden auf die konkrete Umsetzung am Beispiel eines Anwendungsfalls gezeigt.

5.1.2 Architekturbeschreibung

Im AutoKonf-Projekt wurde versucht, die Anzahl der einzuführenden redundanten Komponenten für das voll- und hochautomatisierte Fahren zu begrenzen. Auf diese Weise kann eine redundanzarme technische Architektur für ausfallsichere Systeme entwickelt werden. Um dieses Ziel zu erreichen, wird die Gesamtzahl der redundanten Geräte reduziert, die die Möglichkeit bieten, während des Betriebs neu konfiguriert zu werden.

In „Automatically reconfigurable actuator control for reliable autonomous driving functions (AutoKonf)“ [147] wurden die Ergebnisse aus dem Projekt AutoKonf darge-

stellt. Die dabei entstandene Architektur ist in Abbildung 5.1 dargestellt und wird im Folgenden beschrieben.

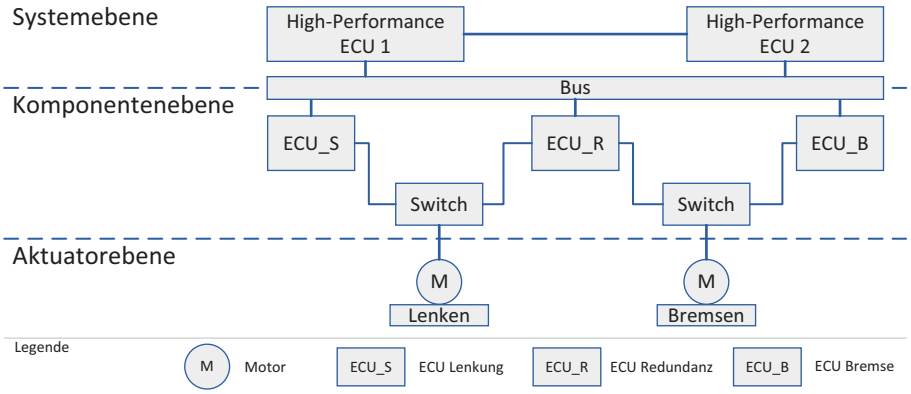

Abbildung 5.1: Technische Architektur im AutoKonf-Projekt nach [147]

Die Grundidee der AutoKonf-Architektur ist es, eine Komponente für zwei verschiedene Zwecke zu verwenden. Im Fall des AutoKonf-Projekts sind es das Bremsen oder das Lenken. Auf diese Weise könnte diese einzelne Zweifunktionskomponente zwei Einzelfunktionskomponenten ersetzen.

Somit übernimmt jene Komponente, die zusätzlich zu den regulären Steuergeräten entwickelt werden muss, die Funktion der fehlerhaften Komponente. Um dieser Herausforderung zu begegnen, wird im AutoKonf-Projekt untersucht, wie die ECUs von den Aktuatoren getrennt werden, welche fehlertolerante Infrastruktur erforderlich ist und wie diese mit rekonfigurierbaren ECUs verwendet wird. Die AutoKonf-Architektur wird in dieser Arbeit noch um die Fahrrechnerebene erweitert. Auf dieser ergänzten Ebene kommen zwei MPSoC-Boards zum Einsatz. Des Weiteren wird eine serviceorientierte Architektur verwendet.

Zielsetzung im AutoKonf-Projekt

Ziel des AutoKonf-Projekts ist es, ein redundantes Steuergerät zu entwickeln, das im Fehlerfall ohne Funktionsbeeinträchtigung entweder ein primäres Brems- oder Lenksteuergerät ersetzen kann. Die automatische Rekonfiguration ist ein zentrales Element für hochautomatisierte Fahrzeuge (vgl. Abschnitt 1.1: SAE Automation Level 4): In diesem Stadium darf ein Ausfall der Brems- und Lenk-ECU den Betrieb des autonomen

Autos nicht verhindern. Im AutoKonf-Projekt wird ein Einzelfehler betrachtet, eine gleichzeitige Fehlfunktion der Brems- und der Lenksteuereinheit ist hingegen nicht abgedeckt.

Beschreibung der Ebenen im AutoKonf-Projekt

Die E/E-Architektur ist hauptsächlich in drei verschiedene Schichten unterteilt:

1. Auf der Systemebene befinden sich die Fahrzeug-Hochleistungssteuergeräte einschließlich der Trajektorienberechnung, der Kommunikation mit anderen Fahrzeugsystemen und der Rekonfigurationslogik.
2. Auf der Komponentenebene sind die primären Aktuator-ECUs und die redundante Aktuator-ECU einschließlich der Schalter untergebracht. Ihre Aufgabe ist es, Steuersignale zu generieren und an die entsprechenden Aktuatoren zu senden.
3. Die Stellantriebsebene ist eine Ebene, in der sich Elektromotoren einschließlich mechanischer Komponenten befinden. Hier ist keine Mikrocontroller-Logik integriert.

Systemebene Für die Hochleistungssteuergeräte auf dieser Ebene wurden Xilinx Zynq ZCU102-Boards ausgewählt, auf denen ein Linux-Betriebssystem implementiert wurde. Die Datenverbindungen zwischen den Steuergeräten auf der Systemebene basieren auf Ethernet, wobei das Ziel darin besteht, eine serviceorientierte Middleware in Form von SOME/IP [133] zu implementieren. Zwischen den ECUs auf der System- und der Komponentenschicht wurde ein CAN-Bus implementiert. Diese Kommunikationsverbindung ist für den Aufbau die einfachste Option. Die erforderlichen Anforderungen hinsichtlich des Timings können hiermit erfüllt werden.

Komponentenebene Für dieses Projekt wurden sowohl Brems- als auch Lenksysteme ausgewählt, um das Prinzip eines Fail-Operational-Systems ohne vollständige Komponentenvervielfältigung aufzuzeigen. Es gibt mehrere Gründe, sich insbesondere für diese beiden sicherheitsrelevanten Teilsysteme zu entscheiden:

- Beide Systeme haben die höchsten Sicherheitsanforderungen (ASIL D).
- Die ausfallbedingte Funktionalität beider Systeme ist für die Zukunft des autonomen Fahrens obligatorisch.

- Beide Systeme haben eine ähnliche Architektur-Topologie: Die Leistungs-ECU muss einen bürstenlosen Gleichstrommotor steuern.
- Beide Motoren haben die gleiche Leistungsklasse (im Bereich von $1\,\mathrm{kW}$ oder weniger).
- In der Vergangenheit haben beide Systeme dieselben Entwicklungsschritte durchlaufen. Angefangen bei einer vollständig mechanischen Einheit über eine elektrohydraulische Unterstützung bis hin zum Fahren mit komplett elektromechanischen X-by-Wire-Lösungen.

Im AutoKonf-Projekt wurde die Komponentenebene folgendermaßen definiert und begrenzt: Für jedes Lenk- und Bremssystem kommt eine Haupt-ECU zur Steuerung des Stellmotors zum Einsatz. Während für das Lenksystem diese Definition uneingeschränkt verwendet werden kann und derzeit die am häufigsten verwendete Lösung ist, gibt es einige Anmerkungen zur Bremseinheit. Hier wird ausschließlich die Bremskraftverstärker-ECU berücksichtigt. Die Fahrzeugstabilitätsfunktion ist außerhalb des Projektumfangs. Diese Funktion ist die meiste Zeit nicht aktiv und wird lediglich in einigen Sonderfällen aktiviert, in denen das fahrdynamische Verhalten instabil werden kann. Für das Lenksystem wird eine EPS-ECU eingesetzt, um den Lenkmotor auf der Aktuatorebene zu steuern. Für das Bremssystem kann dieselbe ECU auf der Grundlage der oben genannten Annahmen verwendet werden.

Sowohl ECU_S als auch ECU_B sind für die Steuerung ihrer angeschlossenen Aktuatoren verantwortlich, wie in Abbildung 5.1 zu sehen ist. Im Falle eines Fehlers auf der Grundlage ihrer eigenen Sicherheitsstrategien werden diese deaktiviert. Dazwischen befindet sich eine redundante ECU_R, die während des normalen Betriebs in den Hot-Standby-Modus versetzt wird. Diese ECU_R ist sowohl mit Lenk- als auch mit Bremssoftware ausgestattet und startet einen entsprechenden Software-Block, falls eines der primären Steuergeräte eine Fehlfunktion aufweist. Alle drei Steuergeräte haben die gleiche Topologie bezüglich Schnittstellen und HW-Komponenten. Auf diese Weise sind Kosteneinsparungspotenziale möglich.

Alle ECUs sind mit einer mehrfach vorhandenen Stromversorgung ausgestattet und verfügen über einen redundanten Kommunikationsbus zu den Hochleistungs-ECUs des Fahrzeugs. Auf der anderen Seite sind alle Steuergeräte über sogenannte Schalter mit den Aktoren verbunden. Diese Schalter haben die Aufgabe, die entsprechende

Leistungselektronik an die Ausgänge der Antriebsmotoren anzuschließen. Basierend auf der Sicherheitsanalyse haben die Schalter die höchste ASIL D-Sicherheitsstufe und sind die Schlüsselkomponenten in der vorgeschlagenen Architektur.

Aktuatorebene Als Stellglied kann ein typischer EPS-Lenkmotor oder ein Bremskraftverstärkermotor verwendet werden. Beide Motoren sind mit einem Motorpositionssensor ausgestattet, der erforderlich ist, um eine feldorientierte Steuerung durchzuführen und ihre Position auszulesen. Systemspezifische Drehmoment- oder Drucksensoren werden hier nicht berücksichtigt und fallen daher nicht in den Geltungsbereich.

5.1.3 High-Performance Computing-Plattformen

Die AutoKonf-Architektur wurde auf der obersten Ebene um zwei MPSoC-Boards erweitert. Auf diese Weise kann eine dynamisch rekonfigurierbare Architektur erreicht werden. Eine Detaillierung dieser Ebene ist in der Abbildung 5.2 zu sehen.

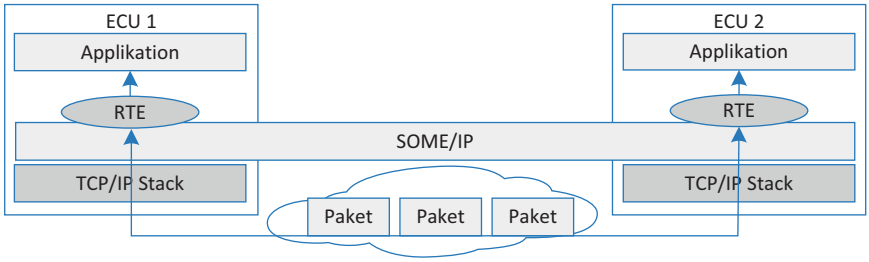

Abbildung 5.2: Ebenen innerhalb der High-Performance Computing-Plattformen

Der betrachtete Anwendungsfall in dieser Arbeit ist das voll- und hochautomatisierte Fahren. Gemäß der SAE-Level sind es die Level 4 und 5, die hier adressiert werden. Für beide Level sind das Bremsen und Lenken die zentralen Funktionen. Aus diesem Grund wird in dieser Arbeit anhand dieser beiden Funktionen die dynamische Rekonfiguration dargestellt. Das Ziel der statischen Redundanz (vgl. Abschnitt 2.2.4) ist die Auswirkungen von Fehlern in einer Komponente zu maskieren oder zu verbergen. Sie wird auch als maskierende Redundanz oder passive Redundanz bezeichnet. Bei der statischen Redundanz sind keine aktiven Maßnahmen durch das System vorhanden,

wie z. B. die Aktivierung von Backup-Komponenten. Beispiele für statische Redundanz sind die aktive Replikation mit Voting oder TMR.

Dynamisch rekonfigurierende Redundanz bedeutet, aktive Wiederherstellungstechniken wie Rekonfiguration oder Selbstheilung zu verwenden. Sie wird auch aktive Redundanz genannt. Beispiele dafür sind Systeme mit Standby-Ersatzteilen, die ausgefallene Einheiten ersetzen (vgl. [20]). Sowohl durch statische Redundanz als auch durch dynamische Rekonfiguration kann ein Fail-Operational-Verhalten erzielt werden. Beide Anteile werden in dieser Arbeit dargestellt und für den Automotive-Einsatz untersucht.

Auf diese Weise werden folgende Ziele verfolgt: eine Hochintegration zu ermöglichen, ein Fail-Operational-Verhalten zu erreichen und die zugehörigen Methoden zur Verfügung zu stellen. Aufgrund der hohen Anforderung bezüglich des Sicherheitsziels bei den Funktionen Bremsen und Lenken, sind diese beiden Funktionen geeignete Forschungsobjekte. Zukünftig werden insbesondere die CPS-Systeme die Fail-Operational-Anforderungen aber auch die Security-Anforderungen bestimmen [18].

Die ausgewählte Architektur und das in dieser Dissertation erstellte Bibliothekskonzept bieten sowohl für die Brems- und Lenkfunktionen als auch für zukünftige CPS-Systeme die Möglichkeit dynamische Rekonfiguration anzuwenden und dadurch die hohen Anforderungen an Echtzeitverhalten und Sicherheitsziele beispielhaft nachzuweisen. Die dynamische Rekonfiguration ist neben der statischen Redundanz das technische Konzept, um ein Fail-Operational-Verhalten zu erreichen. Aus diesem Grund sind beide Varianten in der Kombination Bestandteil dieser Arbeit.

5.1.4 Simulationsumgebung

Um die Auswirkung der Rekonfigurationszeit auf das Fahrverhalten sowie die Wahrnehmung des Fahrers anhand vorgegebener Fahrmanöver und Fehlerszenarien bestimmen zu können, wurde ein Simulator aufgebaut, bei dem die Daten des Fahrrechners manipuliert werden können. Anhand von Probandentests und verschiedener Simulationen konnte ein Richtwert für die maximale Rekonfigurationszeit abgeleitet werden. Die Bitrate zwischen dem CAN-Computer und dem Lenkrad ist mit $1\,\mathrm{Mbps}$ angegeben.

Die Bitrate für die Pedalerie und die Gangwahl beträgt 500 kbps. In der Abbildung 5.3 ist der Aufbau des Gesamtsystems dargestellt.

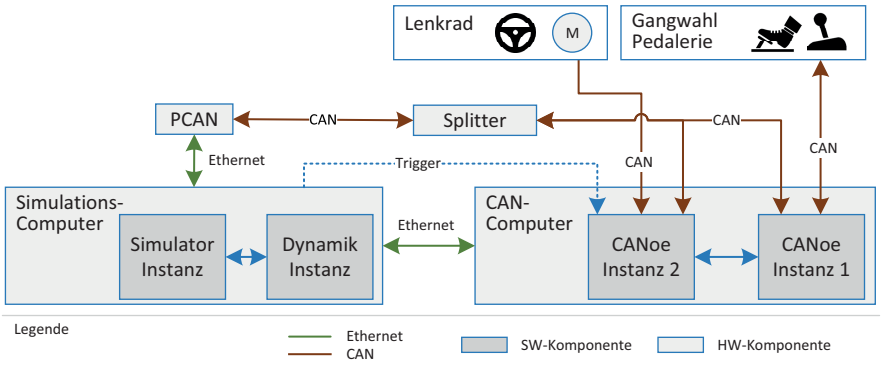

Abbildung 5.3: Simulationsumgebung zur Ermittlung der Rekonfigurationszeiten nach [155]

Zur Datenmanipulation unterbricht ein CANcaseXL[2] die CAN-Verbindung zwischen dem Lenkrad und dem Simulator und leitet die CAN-Frames über den Fahrrechner (CANoe[3]) um. Der Fahrrechner nimmt dann die Manipulation der Signale vor. Dieses wird für drei Szenarien genutzt:

1. Fehler werden während der Fahrt eines realen Fahrers eingestreut, um die Wahrnehmung zu prüfen.
   a) VDA[4]-Ausweichtest
   b) Schnelle Geradeausfahrt
2. Es wird eine Fahrt aufgezeichnet. Mit diesen Daten wird eine Referenzfahrt simuliert. Dabei wird ein Fehler in Form eines Lenkausfalls produziert, um anschließend die Abweichung von der Referenzfahrt aufzunehmen und auswerten zu können.

[2] Das CANcaseXL ist ein USB-Interface mit einem leistungsfähigen 32Bit 64MHz Mikrocontroller und zwei CAN-Controllern zur Verarbeitung von CAN-Botschaften
[3] CANoe ist eine Entwicklungs- und Test-Software der Vector Informatik GmbH
[4] Verband der Automobilindustrie

Datenmanipulation während der Fahrt

Das CANcaseXL stellt zusammen mit der zugehörigen CANoe-Software den Aufbau zur Datenmanipulation dar. Dabei wird das CANcaseXL zwischen das SENSO-Wheel und einem CAN auf Ethernet Gateway (PCAN) eingesetzt, um die Daten abzufangen und diese mittels CAPL[5]-Skripts in der CANoe-Konfiguration zu manipulieren. Das CANcaseXL ist per Universal Serial Bus (USB) an einen Rechner angeschlossen, auf dem die zugehörige SW CANoe installiert ist. Die Signale werden mittels CAPL-Skripts in CANoe manipuliert. Die vom SENSO-Wheel gesendeten und empfangenen Signale sind in SW spezifiziert.

Tabelle 5.1: Betriebsart Normalmodus Soll-Wert - CAN-Nachricht 201h

Identifier: 201h			DLC: 8 Byte	
Richtung: Host –> SENSO-Wheel			Zykluszeit: 1 ms	
	Einheit	**Wertebereich**	**Byte**	**Bits**
Soll-Moment	[mNm]	−32 768–32 767	1, 2	0..15
Reibung	[mNm]	0–5000	3, 4	0..15
Dämpfung	[mNm/min]	0–500	5, 6	0..15
Federsteifigkeit	[mNm/°]	0–2500	7, 8	0..15

Zur Vollständigkeit sind hier die beiden wichtigsten CAN-Nachrichten in Tabelle 5.1 und Tabelle 5.2 aufgelistet.

Tabelle 5.2: Betriebsart Normalmodus Ist-Wert - CAN-Nachricht 211h

Identifier: 211h			DLC: 8 Byte	
Richtung: SENSO-Wheel –> Host			Zykluszeit: 1 ms	
	Einheit	**Wertebereich**	**Byte**	**Bits**
Position	[Ink]	$-1 \cdot 2^{31}$–$1 \cdot 2^{31}-1$	1, 2, 3, 4	0..31
Geschwindigkeit	[1/min]	−32 768–32 767	5, 6	0..15
Ist-Moment	[mNm]	−32 768–32 767	7, 8	0..15

In den Tabellen sind jeweils der Identifier, die Zykluszeit, die Einheit, sowie der Wertebereich und die Senderichtung angegeben. Die Nachricht, deren Eigenschaften in

[5] Communication Access Programming Language ist eine von Vector Informatik entwickelte Programmiersprache, die in den weit verbreiteten Software-Werkzeugen CANoe und CANalyzer zur Verfügung steht

Tabelle 5.1 aufgezeigt ist, wird vom Host zum SENSO-Wheel geschickt. Tabelle 5.2 beinhaltet die Nachricht, die vom SENSO-Wheel wieder zurück an den Host geschickt wird.

Zur Nachrichtenmanipulation am Rechner mit CANoe werden folgende Schritte durchgeführt:

1. Initialisierung und Start
    a) Initialisierung der Simulationsumgebung. Dabei werden auch die Variablen Maximalmoment, Maximalgradient, Inkremente pro Umdrehung, Dämpfung und Totzeit auf Default-Werte gesetzt.
2. Empfang und Manipulation von CAN-Botschaften
    a) Unterscheidung und Überprüfung von CAN-Botschaften mit ID[6] 0x201h und 0x211h. Übrige CAN-Botschaften werden durchgeroutet.
    b) Je nach Eingabe im Panel am CANoe-Rechner werden entsprechende Daten geändert.
    c) Durch Betätigen der Taste „Fehler1“ im Panel des CANoe-Rechners werden die entsprechenden CAN-Botschaften für eine bestimmte Dauer (Totzeit) manipuliert.
    d) Anschließend werden neu eingegebene Werte auf Gültigkeit überprüft.
3. Verschicken von CAN-Botschaften an den Simulator.

Während einer Probefahrt im Szenario VDA-Ausweichtest ist es möglich mit CANoe den Lenkausfall zu simulieren (vgl. Abbildung 6.9). Dabei kann nicht nur die Ausfalldauer eingestellt werden, sondern auch das Gegenmoment und Dämpfung während des Ausfalls, somit lässt sich der Lenkausfall wahrnehmen und die Rekonfigurationszeit reproduzierbar ermitteln (vgl. Abschnitt 6.2.4). Um eine Probefahrt aufzunehmen, werden die CAN-Nachrichten vom SENSO-Wheel sowie Pedalerie und Getriebe am CAN-Bus anhand CANoe aufgenommen. Neben der Ermittlung der Rekonfigurationszeit wird der Simulator ebenfalls zur KPI-Messung des Simplex-Moduls und des Fail-Operational-Moduls verwendet (vgl. Abschnitt 6.1.7). Der Aufbau und die

[6] Identifier

Einbindung des Simulators zusammen mit der E/E-Architektur sind somit wichtige Bestandteile dieser Arbeit, um die entsprechenden Evaluierungen durchführen zu können.

5.2 Methodik für einen rekonfigurierbaren, fehlertoleranten E/E-Architekturentwurf für Brems- und Lenksteuersysteme

Nachdem im vorangegangenen Abschnitt das Ziel vom AutoKonf-Projekt und die AutoKonf-Architektur beschrieben wurden, wird in diesem Abschnitt auf das Vorgehen eingegangen. Es wird dargelegt, wie die Architektur entstanden ist und welche Spezifikation sie zu erfüllen hat. Zunächst werden die Fahrszenarien dargestellt, die von Brems- und Lenksystemen des voll- und hochautomatisierten Fahrens unterstützt werden. Danach werden Beispiele aus der entsprechenden GuR-Analyse gezeigt. Das Vorgehen ist in der eigenen Veröffentlichung „Dynamic Reconfiguration for Real-Time Automotive Embedded Systems in Fail-Operational Context“ [154] beschrieben und wird zur Klarheit hier noch einmal geschildert. Folgende Schritte werden für die Architekturauslegung und die Ableitung der Anforderungen durchgeführt:

1. Durchführen der GuR-Analyse zur Identifizierung und Bewertung von Störungen nach ISO 26262 [77] (Abschnitt 5.2.1).
2. Entwerfen einer funktionalen E/E-Architektur zur logischen Implementierung von Rekonfigurationsmechanismen (Abschnitt 5.2.2).
3. Entwerfen einer technischen E/E-Architektur, die die funktionale E/E-Architektur implementiert (Abschnitt 5.2.3).

Abschließend werden alternative Systemkonzepte vorgestellt und deren Bewertung durchgeführt (Abschnitt 5.2.4).

5.2.1 GuR-Analyse zur Identifizierung und Bewertung von Fehlern nach ISO 26262

Ein grundlegender KPI für dynamisch rekonfigurierbare Systeme ist stets die Zeit, die für den Rekonfigurationsvorgang benötigt wird. Dabei ist es essenziell zu wissen, was die obere Schranke für diese Zeitspanne ist. Um eine spätere Zertifizierung und Homologation des Gesamtsystems zu erhalten, ist ein Bezug zu der ISO 26262 unabdingbar. Diese sieht eine Fehlerreaktionszeit vor. Im Rahmen dieser Arbeit wird die Fehlerreaktionszeit als Rekonfigurationszeit umdefiniert. Die Definition des Fault Tolerant Time Intervals (FTTIs) wird um diesen neuen Begriff ergänzt und in Abbildung 5.4 in den Zusammenhang zu dem FTTI gestellt.

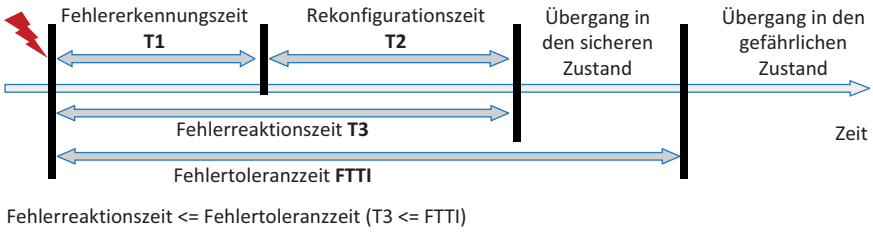

Abbildung 5.4: Definition der Rekonfigurationszeit nach [154]

Für die Brems- und Lenksteuerfunktionen werden Fehler simuliert, um die Zeit zwischen dem Auftreten eines Fehlers und eines resultierenden gefährlichen Ereignisses, das einen Schaden (Unfall) verursachen kann, zu berechnen. Diese Zeitspanne ist in Abbildung 5.4 als Fehlertoleranz-Zeitintervall (FTTI) dargestellt.

Tabelle 5.3: Beispielausgabe der GuR-Analyse

Failure Event	Safe State	Safety Goal	ASIL	FTTI
Ausfall der primären Lenkfunktion	Aufrechterhaltung der Funktion durch Rekonfiguration	Vermeidung eines signifikanten Ausfalls der angegebenen Funktion für länger als $t = x$ (FTTI)	D	130 ms
Ausfall der primären Bremsfunktion	Aufrechterhaltung der Funktion durch Rekonfiguration	Vermeidung eines signifikanten Ausfalls der angegebenen Funktion für länger als $t = x$ (FTTI)	D	160 ms

Sowohl für die Brems- als auch für die Lenksteuerungsfunktion berücksichtigt der sichere Zustand die Aufrechterhaltung der Brems- und Lenksteuerungsfunktionen. In der Tabelle 5.3 sind zwei Beispiele als Ergebnis der GuR-Analyse gemäß der Norm ISO 26262 [77] aufgezeigt. Darin sind die Fehler, die Klassifizierungen der Fehler nach den ASIL, den entsprechenden Sicherheitszuständen und Sicherheitszielen sowie der FTTIs aufgeführt [154]. Die in Tabelle 5.3 aufgestellten Spalten ASIL und FTTI sind zwei zentrale KPIs und werden als Eingangsartefakte für die Rekonfigurationsbibliothek verwendet (vgl. Abbildung 4.2, Abbildung 4.3 und Abschnitt 4.3.6).

5.2.2 Entwurf einer funktionalen E/E-Architektur zur logischen Implementierung von Rekonfigurationsmechanismen

Im Folgenden wird die Entwicklung einer funktionalen E/E-Architektur beschrieben. Dazu werden zunächst die Module der funktionalen E/E-Architektur abgeleitet. Anschließend wird der Rekonfigurationsmechanismus für die Brems- und Lenksteuerungsfunktionen entwickelt und anhand von ausgetauschten Nachrichten zwischen einzelnen Modulen, die an den entsprechenden Rekonfigurationsszenarien beteiligt sind, modelliert. Die funktionale Architektur besteht aus den einzelnen Modulen und deren Kommunikationsabhängigkeiten. Dabei beinhaltet sie wiederum drei Ebenen:

1. Auf der obersten Ebene befinden sich Module zur Fehlerbehandlung der Bremse und der Lenkung.
2. Auf der mittleren Ebene befinden sich drei Steuermodule, eines zum Bremsen, eines zum Lenken sowie das redundante Rekonfigurationssteuermodul, welches das Bremsen oder Lenken ausführen kann.
3. Auf der untersten Ebene befinden sich die beiden Aktuatormodule, die eine Bremse und eine Lenkung darstellen.

In Abbildung 5.5 ist die funktionale E/E-Architektur für die Funktionen Bremse und Lenkung dargestellt. Neben den Modulen der funktionalen E/E-Architektur werden auch die Rekonfigurationsmechanismen dargestellt, die für die Brems- und Lenksteuerfunktion in ihrem Aufbau und ihrer Funktionsweise ähnlich sind.

Diese Mechanismen umfassen drei Nachrichten:

1. Die erste Nachricht modelliert die Übermittlung der Fehler als Benachrichtigung von dem Brems- oder Lenkungssteuerungsmodul an das entsprechende Fehlerbehandlungsmodul.

2. Als Nächstes befiehlt das Modul zur Fehlerbehandlung dem redundanten Modul zum Bremsen und Lenken, den Hot-Standby-Modus zu verlassen, um die Bremse oder Lenkung zu steuern, je nachdem, welches Modul ausgefallen ist. Es wird davon ausgegangen, dass entweder das primäre Modul zum Bremsen oder das primäre Modul zum Lenken ausfällt, aber nicht beide.

3. Zuletzt werden Verhaltensmodelle für die beiden Rekonfigurationsmechanismen spezifiziert, in denen die Zeiteinschränkungen aus Tabelle 5.3 und Tabelle 6.5 verwendet werden.

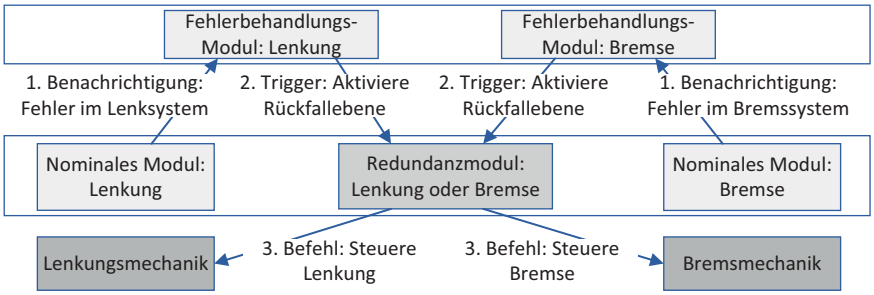

Abbildung 5.5: Funktionale E/E-Architektur nach [154]

Durch den konsequenten Einsatz der Ebenen und der Modularisierung, gelingt die Wiederverwendbarkeit der Einzelmodule. Dies gestattet langfristig eine Kostensenkung und eine Komplexitätsreduktion in der Entwicklung. Die Einzelmodule sind aufgrund der Unterstützung durch definierte Schnittstellen einfacher zu warten. Auch eine Skalierbarkeit ist durch dieses Vorgehen begünstigt, da ein Fahrzeugausstattungsvariante eventuell eine andere Variante oder Version eines Moduls benötigt und somit ausgetauscht werden kann. Dieses ist wiederum vorteilhaft für das Variantenmanagement bei der E/E-Architekturentwicklung. Des Weiteren können die Entwicklungsstrukturen im Unternehmen besser skalieren, weil Teilumfänge einfacher an entsprechend spezialisierte Firmen vergeben werden können oder innerhalb der Firma optimaler verteilt werden können [154].

5.2.3 Entwurf einer technischen E/E-Architektur, zur Implementierung der funktionalen E/E-Architektur

In Abschnitt 5.1.2 wurde die technische E/E-Architektur bereits beschrieben. Im Folgenden werden die Module aus der funktionalen E/E-Architektur auf die technische E/E-Architektur abgebildet.

Die Module zur Fehlerbehandlung von Bremse und Lenkung werden auf den zwei Hochleistungssteuergeräten eingesetzt, auf denen Linux implementiert ist.

Als Nächstes werden die Module der mittleren Ebene der funktionalen E/E-Architektur auf Standard-Kfz-Steuergeräten umgesetzt, die für die Steuerung von Brems- und Lenkaktuatoren optimiert sind. Zusätzlich werden Trenn- und Schließschalter entwickelt. Die Schalter senden bei einem Steuergeräteausfall entweder pulsweitenmodulierte Steuersignale von den primären ECUs oder von der Rekonfigurations-ECU zum Brems- oder Lenkungsaktor.

Auf der untersten Ebene der funktionalen E/E-Architektur aus der Abbildung 5.5 werden die Module für die Bremse und die Lenkungsaktorik durch bürstenlose Gleichstrommotoren ersetzt.

In der technischen E/E-Architektur werden neben den elektrischen Signalen, die über gewöhnliche Leitungen gesendet werden, Datenverbindungen über vorhandene Kfz-Netzwerktechnologien dargestellt, um die Kommunikationsabhängigkeiten der funktionalen E/E-Architektur zu implementieren. In dieser Hinsicht sind die Datenverbindungen zwischen den Steuergeräten auf der obersten Ebene Ethernet-basiert, da eine serviceorientierte Middleware in Form von SOME/IP implementiert wird [132]. Zwischen den Steuergeräten auf der obersten und mittleren Ebene kommt ein CAN-Bus zum Einsatz. Dieser stellt die einfachste Möglichkeit in der Umsetzung dar und erfüllt die Anforderungen in Bezug auf das Timing.

5.2.4 Alternative Systemkonzepte und moderne Architekturen

Die in Abbildung 5.1 gezeigte Systemarchitektur ist eine realisierbare, jedoch nicht die einzige Lösung. Im AutoKonf-Projekt [59] wurden während der Definitionsphase der

Systemanforderungen mehrere Kriterien festgelegt, um den dargestellten Ansatz mit anderen möglichen Architekturen zu validieren.

Tabelle 5.4: Vergleich alternativer Architekturen nach [147]

Kriterium	Einbauraum	EMV[7]	Sicherheit	Gewicht	Kosten	Gesamt
Gewichtungsfaktor	0.5	0.2	1.0	0.3	0.6	
Konzept						
4 Motoren mit eigener Leistungselektronik und Schaltern	1.00	0.33	0.14	1.00	1.00	1.61
6-phasige Motoren, die direkt an die Steuergeräte angeschlossen werden, ohne die Schalter zu öffnen	0.57	1.00	0.57	0.95	0.78	1.81
6-phasige Motoren mit Öffnungsschaltern	0.71	1.00	0.29	0.97	0.85	1.64
Redundante ECU ohne redundante Leistungselektronik	0.14	0.67	1.00	0.53	0.28	1.53
Redundante ECU einschließlich redundanter Leistungselektronik und Schalter	0.57	0.33	0.43	0.68	0.66	1.38
Redundantes Steuergerät und Leistungselektronik ohne Öffnungsschalter	0.43	1.00	0.86	0.63	0.44	1.72
AutoKonf Vorschlag	0.43	1.00	0.43	0.65	0.51	1.34

Die Tabelle 5.4 gibt in der ersten Spalte die unterschiedlichen Konzepte an, die untersucht wurden. Die Konzepte selbst sind aus einer Kombination der zur Lösung beitragenden Komponenten entstanden. Die einzelnen Komponenten bestehen aus einem 3-Phasen-Motor mit eigener Leistungselektronik, einem 6-Phasen-Motor mit direkter Anbindung an das Steuergerät, Trennschalter und redundanter ECU. Jede Kombination wurden empirisch, in diesem Fall basierend auf Erfahrungswerten, analysiert und anhand der vordefinierten Merkmale Einbauraum, EMV[8], Sicherheit, Gewicht und Kosten validiert. Die Zusammenfassung dazu ist in Tabelle 5.4 aufgelistet:

Die Wichtigkeit aller Kriterien wurde durch einen Gewichtungsfaktor bewertet. Danach wurde jeder einzelne Konzeptwert mit dem Gewichtungsfaktor multipliziert und alle Werte addiert, um den endgültigen Gesamtwert zu erhalten. Je niedriger dieser ist, desto geeigneter ist der aktuelle Entwurf in Bezug auf die vordefinierten Merkmale. Basierend

[8] Elektromagnetische Verträglichkeit

auf diesem Vergleich hat sich die AutoKonf-Architektur mit der besten Bewertung als vielversprechende Lösung erwiesen.

Das hier verwendete Vorgehen zeigt eine Möglichkeit auf, wie empirisch ein erster E/E-Architektur- Grobentwurf skizziert werden kann. Dieser fließt wiederum zu einem späteren Zeitpunkt des Entwicklungsprozesses in die Bewertung ein (vgl. Abschnitt 6.4).

5.3 Modellbasierte Entwurfsmethodik für die dynamische Rekonfiguration

Im vorherigen Abschnitt wurde gezeigt, wie die Spezifikation für ein dynamisch rekonfigurierbares System ermittelt wird. In diesem Abschnitt wird nun gezeigt, wie ein dynamisch rekonfigurierbares System modelliert wird. Dazu werden zunächst die benötigten Elemente eingeführt (Abschnitt 5.3.1), eine Rekonfigurationsebene für eine Cross-Layer-Zusammenspiel aufgezeigt (Abschnitt 5.3.2), das Vorgehen zur Modellierung des serviceorientierten Systems beschrieben (Abschnitt 5.3.3) und die Analyse zur Abschätzung der Ausführungszeiten vorgestellt (Abschnitt 5.3.4).

5.3.1 Modularer Aufbau dynamisch rekonfigurierbarer, serviceorientierter Systeme

Es wird im Folgenden ein modularer Werkzeugkasten zur Verfügung gestellt, der aus drei Hauptelementen für die Entwicklung eines dynamisch rekonfigurierbaren Systems im ausfallsicheren Kontext besteht. Als erstes modulares Element werden die Grundkomponenten vorgestellt und erläutert. Das zweite modulare Element besteht aus verschiedenen Entwurfsmustern für ein dynamisch rekonfigurierbares System. Das dritte modulare Element enthält Bewertungskriterien, anhand derer der Systementwickler das richtige Muster auswählen kann.

Erstes modulares Element: Die Grundkomponenten

Zunächst wird das erste modulare Element aufgeschlüsselt. Weiterhin werden drei verschiedene Ebenen eingeführt. Zu Beginn werden zunächst vier Grundkomponenten vorgestellt, die in Abbildung 5.6 zu sehen sind.

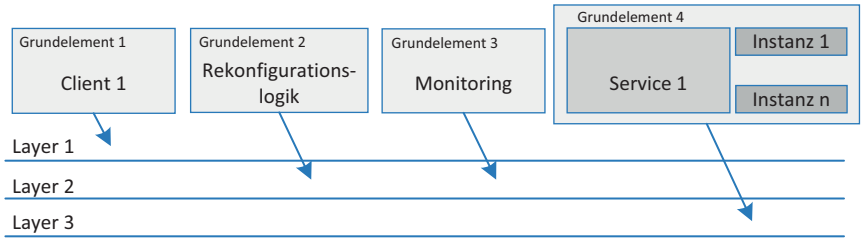

Abbildung 5.6: Erstes modulares Element - Die Grundkomponenten nach [145]

Die vier Komponenten sind demnach:

1. Client,
2. Rekonfigurationslogik,
3. Überwachung,
4. und Service mit n-Instanzen.

Die erste Komponente ist der Client. Der Client ist die Anwendung selbst, die verschiedene Dienste wie Sensoren, Aktuatoren oder eine Trajektorienplanung nutzt. Eine Beispielanwendung ist die Lenkfunktion oder die Bremsfunktion. Da die Funktion dynamisch neu konfiguriert werden soll, wird eine zweite Komponente benötigt, die Rekonfigurationslogik. Die Rekonfigurationslogik führt die Rekonfiguration durch. Die dritte Komponente ist die Überwachungskomponente, die Fehlerereignisse erkennt und als Auslöser für den dynamischen Rekonfigurationsprozess benötigt wird. Die vierte Komponente ist der Anbieter eines Dienstes: Diese Komponente kann mehrere Instanzen haben, mindestens aber zwei.

Neben den vier Basiskomponenten werden drei verschiedene Ebenen gezeigt. Auf diese werden die Basiskomponenten zugewiesen, wie in Abbildung 5.6 dargestellt. Auf Ebene

1 befindet sich der Client, während auf Ebene 2 die Rekonfigurationslogik und die Überwachung liegen. Der Service wird der Ebene 3 zugewiesen. Mit diesem Konzept wird eine Trennung einzelner Softwareteile unterstützt und daher die Wiederverwendung von Softwareteilen gefördert.

Zweites modulares Element: Die Entwurfsmuster

Als nächstes wird ein Überblick über Entwurfsmuster für dynamisch rekonfigurierbare Systeme gegeben. Diese Muster werden im Folgenden erläutert und verschiedene Alternativen aufgezeigt. Für jedes werden die Vor- und Nachteile dargestellt. Die notwendigen Entwurfsmuster sind wie folgt:

1. Zentrale Architektur: zentrale Middleware (siehe dazu Absatz *Entwurfsmuster einer zentralen Architektur*)
2. dezentrale Architektur: jeweils mit Überwachung (siehe dazu Absatz *Entwurfsmuster einer dezentralen Architektur*)
3. Domänenorientierte Architektur: eine Middleware pro Domäne (siehe dazu Absatz *Entwurfsmuster einer Domänenarchitektur*)

Erstellt werden diese Muster, indem die vier Grundkomponenten aus dem vorherigen Abschnitt übernommen werden. Abhängig davon, wie sie implementiert werden, können diese drei Entwurfsmuster extrahiert werden. Dabei bildet der zentrale Ansatz das eine Ende der Entwurfsmöglichkeiten, während der dezentrale Ansatz das andere darstellt. Das dritte Entwurfsmuster ist somit eine Mischung aus dem zentralen und dem dezentralen Ansatz.

Entwurfsmuster einer zentralen Architektur Übersichten zu zentralen Architekturen im Automobilbereich finden sich in „RACE“ [125], „Future Automotive Architecture and the Impact of IT Trends“ [130] und „Architectures of High Performance Computing“ [105].

Das Entwurfsmuster einer zentralen Architektur ist in Abbildung 5.7 gezeigt. Es sind drei Dienste mit jeweils *n*-Instanzen dargestellt. Die Logik zur Überwachung und Rekonfiguration ist in der zentralen Middleware implementiert. Die zentrale Middleware

entscheidet über die Serviceabonnements zwischen den Instanzen der Serviceanbieter und der Clients.

Beispielsweise gibt es mehrere Instanzen von Service Provider 1, in diesem Fall für die Trajektorienplanung. Es werden zwei Fälle angenommen: Eine Instanz läuft auf einer ECU und eine andere auf einer anderen ECU. Der Client stellt eine Verbindung zur zentralen Middleware her und fragt die Trajektorienplanung an. Die zentrale Middleware überträgt diese Anforderung wie ein Gateway und fordert die verschiedenen Instanzen des Dienstanbieters auf, die Trajektorienplanung bereitzustellen. Dann stellt die zentrale Middleware die Verbindung zu einer der Instanzen her. Die Verbindung wird über ein Dienstprotokoll hergestellt, das mit SOME/IP implementiert werden kann.

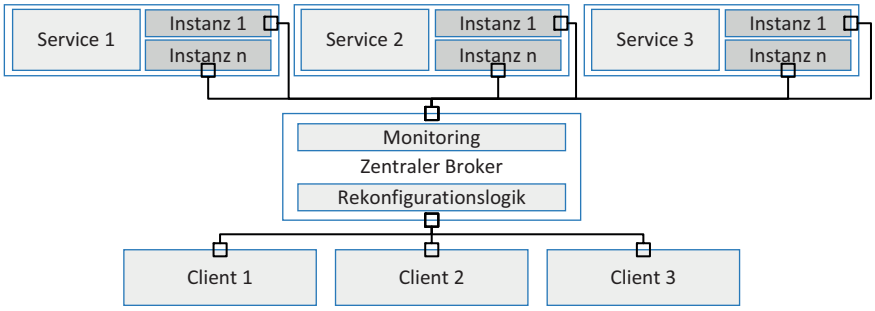

Abbildung 5.7: Entwurfsmuster einer zentralen Architektur nach [145]

Bei einem Ausfall einer Instanz wird die zentrale Middleware von der Überwachungslogik informiert. Nun fordert die zentrale Middleware die zweite Instanz des Dienstanbieters auf, für die Trajektorienplanung den Dienst für den Client bereitzustellen. Der Client bemerkt nicht, dass jetzt eine andere Instanz die Trajektorienplanung bereitstellt.

Entwurfsmuster einer dezentralen Architektur Ein Beispiel für eine dezentrale Architektur ist in „Architectures of High Performance Computing“ [105] beschrieben. Es gibt wieder zwei Instanzen von Service Provider 1. Die Verbindung wird ebenfalls über SOME/IP hergestellt. In diesem Fall wird dies jedoch vom Client selbst durchgeführt. Außerdem werden die Überwachung und die Neukonfigurationslogik im Client ausgeführt, da keine zentrale Middleware vorhanden ist. Der Client prüft nun, ob die

verbleibenden Instanzen des Dienstanbieters verfügbar sind und abonniert eine davon. In jedem Client muss die Überwachungs- und die Rekonfigurationslogik implementiert werden. Daher verursacht dieses Entwurfsmuster einen höheren Entwicklungsaufwand beim Client. Dieses Muster hat keine zentrale Middleware und ist in Abbildung 5.8 dargestellt.

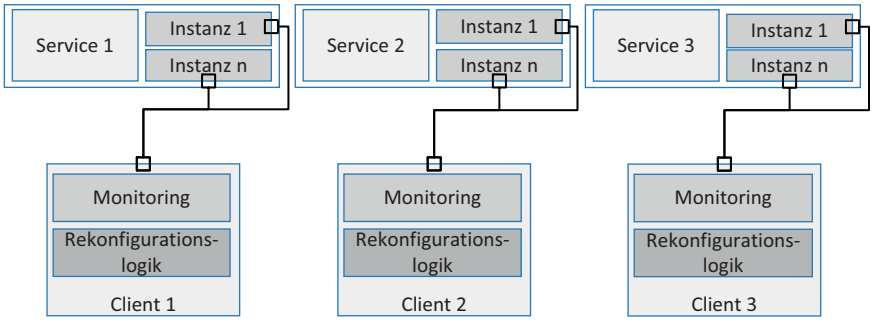

Abbildung 5.8: Entwurfsmuster einer dezentralen Architektur nach [145]

Entwurfsmuster einer Domänenarchitektur In den beiden Veröffentlichungen „Architectures of High Performance Computing“ [105] und „Elektrik/Elektronik-Architekturen im Kraftfahrzeug“ [127] werden Domänenarchitekturen für die Automobilindustrie diskutiert. In diesen Architekturen sind die folgenden Automobildomänen vorhanden (vgl. 2.1.1):

1. Fahrwerk
2. Antrieb
3. Chassis
4. Infotainment
5. Innenraum
6. Fahrerassistenz

Eine Domänenarchitektur verwendet Aspekte der zentralen und der dezentralen Architektur. In einer Domänenarchitektur existieren Domain Control Units (DCUs), auf denen u.a. domänenübergreifende Funktionen ausgeführt werden. Dieses verhält sich

genau wie in der zentralen Architektur, jedoch in diesem Fall nur für eine einzelne Domäne. Andere Funktionen werden auf Steuergeräten ausgeführt, die sich unterhalb der DCU befinden. Um die richtigen Entwurfsmuster auszuwählen, ist eine Bewertung erforderlich.

Drittes modulares Element - Entwurfsbewertung

Im Entwurfsprozess der E/E-Architektur ist bereits zu Beginn eine bedeutende Entscheidung über den grundlegenden Aufbau zu treffen. Für diesen Aufbau stehen prinzipiell die im Abschnitt 5.3.1 gezeigten Entwurfsmuster und deren Kombination zur Verfügung. Um die Entscheidungsfindung zu unterstützen, wurde im Rahmen dieser Dissertation eine Entwurfsbewertung durchgeführt (vgl. [145]).

Bezüglich der Bewertungskriterien wurde die Anzahl der Rekonfigurationslogiken, die Möglichkeit Fail-Operational-Anforderungen zu erfüllen, der benötigte Testaufwand sowie die Wartbarkeit des Gesamtsystems betrachtet. In einem zentralen Entwurfsmuster gibt es nur eine Rekonfigurationslogik, in einem dezentralen Entwurf entspricht die Menge dieser Logiken der Summe der Clients und in einer Domänenarchitektur entspricht die Anzahl der Rekonfigurationslogiken der Anzahl der Domänen. Dies ergibt sich aus den zuvor eingeführten Grundkomponenten und Entwurfsmustern.

Eine Fail-Operational-Anforderung kann mit allen drei Entwurfsmustern erfüllt werden. Somit ist die entscheidende Voraussetzung bei allen drei Entwürfen gegeben. Die zugehörige Entwurfsbewertung ist in Tabelle 5.5 dargestellt:

Tabelle 5.5: Bewertungskriterien für Entwurfsmuster

Bewertungskriterien	**Zentrale Architektur**	**Dezentrale Architektur**	**Domänen-architektur**
Rekonfigurationslogik	1	Anzahl der Clients	Anzahl der Domänen
Fail-Operational	+	+	+
Entwicklungsaufwand	+	-	-
Testaufwand	+	-	-
Wartbarkeit	+	-	-

Der Entwicklungs- und Testaufwand für die zentralen Entwurfsmuster muss nur einmal durchgeführt werden, bzw. müssen hier verschiedene Entwicklerteams aufgrund

der hohen ASIL-Anforderung an diese zentrale Komponente eingesetzt werden. Für Entwürfe einer dezentralen und einer Domänenarchitektur hingegen stimmt diese Metrik mit der Anzahl der Clients oder Domänen überein und zeigt einen Mehraufwand für das Gesamtunternehmen auf, weil die Entwicklung und der Test immer plattformspezifisch sind, auch wenn es sich um COTS-Plattformen handelt. In Bezug auf die Wartbarkeit besteht eine Korrelation zu der Summe der Rekonfigurationslogiken innerhalb der verwendeten Architektur. Auch hier bedingt die Anzahl der notwendigen Logiken den Mehrwert, weil eine Änderung für jedes einzelne System geplant, durchgeführt und getestet werden muss.

5.3.2 Cross-Layer-Rekonfigurationslogik

Zwischen der Anwendungsschicht und der Middleware-Schicht wird eine zusätzliche Rekonfigurationsschicht eingeführt. Diese Schicht beherbergt die Logik für die Rekonfiguration und ermöglicht einen ebenenübergreifenden Rekonfigurationsansatz. Dieser kann angewendet werden, um die dynamischen Rekonfigurationsansätze auf der Hardwareebene mit den Ansätzen auf der Softwareebene zu verbinden.

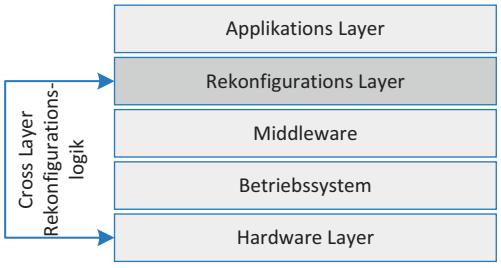

Abbildung 5.9: Rekonfigurationsebene und ebenenübergreifende Rekonfigurationslogik

Die Cross-Layer-Rekonfigurationslogik verbindet die dynamische Rekonfiguration umgesetzt durch die DSA, die auf der LPD[9] unter Einbeziehung der PMU, der RPU und des MB auf der PL zum Einsatz kommt, mit der dynamischen Rekonfiguration realisiert durch den serviceorientierten Ansatz, der über die Middleware auf der APU implementiert wurde. Abbildung 5.13 zeigt diese einzelnen SW- und HW-Komponenten. In Ab-

[9] Low-Power Domain

bildung 5.9 ist der in dieser Dissertation erarbeitete Cross-Layer-Rekonfigurationslogik eingebettet in den Kontext der Ebenen dargestellt.

Neben der DSA ist für die Cross-Layer-Rekonfigurationslogik die Verwendung des Service-Konzepts von entscheidender Bedeutung. Die CAN-Kommunikation wird für die Middleware als Service implementiert. Auf diese Weise kann die Abstraktion der HW-Ebene mit der Implementierung der DSA von der Middleware getrennt werden. Die Cross-Layer-Rekonfigurationslogik kann nun bei einer Degradation auf das zweite Board umschalten und den CAN-Provider des anderen Boards verwenden.

Für den Einsatz dieser Logik ist daher ein QoS-Konzept notwendig, so dass der CAN-Provider über die einzelnen Degradationsstufen der DSA informiert ist. Liegt ein Totalausfall des CAN-Providers vor, wird eine Stop-Offering-Nachricht an die Middleware geschickt, und die Rekonfigurationslogik kann nun einen weiteren Service abonnieren, der dieselbe Funktionalität zur Verfügung stellt. In diesem Fall ist es der CAN-Provider auf der Rückfallebene. Das zugehörige QoS-Konzept ist in Tabelle 6.1 und in Abschnitt 6.1.6 beschrieben.

Die Kommunikation zwischen den Prozessoren basiert auf Message Buffern und Inter-Prozessor Interrupts. Diese Methode wurde gemäß den Vorgaben seitens Xilinx für das Ultrascale+ MPSoC implementiert und an das Projekt angepasst [141]. Die Zuordnung der Kanäle im Projekt ist in Tabelle 5.6 dargestellt.

Tabelle 5.6: Verwendete IPC-Kanäle.

Kanal	Prozess
0	APU (Betriebssystem)
1	RPU
3-6	PMU
9	APU (Applikation)
10	MicroBlaze

Zum Austausch der Daten wird der 32 Byte große Message-Buffer verwendet. Der Interrupt wird zur Ablaufkontrolle verwendet. Jedem Prozessor werden ein oder mehrere Sender-Kanäle zugewiesen. Ausgehend von diesem Kanal, kann mit allen anderen vorhandenen Kanälen kommuniziert werden.

Beim Senden wird zunächst das korrekte Offset der Message-Buffer berechnet. Anschließend wird das Interrupt-Flag überprüft. Ist dieses nicht gesetzt, werden alle vorherigen

Nachrichten vom Empfänger verarbeitet und die neue Nachricht wird in den Message-Buffer geschrieben und der Interrupt ausgelöst. Falls das Interrupt-Flag gesetzt ist, wurde die Verarbeitung beim Empfänger nicht abgeschlossen und die Nachricht wird verworfen. In der Interrupt-Service-Routine des Empfängers wird ebenfalls das korrekte Offset berechnet. Anschließend wird die Nachricht gelesen und das Interrupt-Flag gelöscht.

5.3.3 Modellierung serviceorientierter Systeme

Für die eingeführten Entwurfsmuster, die auf der Idee der Serviceorientierung basieren, muss eine Methodik erstellt werden. Diese soll auf dem entsprechenden Systemdesign der Laufzeitarchitektur und der Bewertung der durchgängigen Latenz ausgerichtet sein. Zunächst soll das System unter Verwendung des modellbasierten Ansatzes entworfen werden. Relevante Systemelemente sollen dabei für eine spätere Überprüfung präzise modelliert werden. Das modellierte System kann dann für eine erste Evaluierung des Rekonfigurationsprozesses benutzt werden. Danach könnte eine weitere Überprüfung unter Verwendung des simulativen Ansatzes realisiert werden, um die Zuverlässigkeit des dynamischen Rekonfigurationsprozesses sicherzustellen. Die Implementierung sollte nur durchgeführt werden, wenn die Verifizierungsergebnisse sowohl der statischen Analyse als auch der simulativen Bewertung die entsprechenden Anforderungen, wie zum Beispiel die maximale Rekonfigurationszeit, erfüllen.

Abbildung 5.10: Methodik – Prozessschritte nach [145]

In Abbildung 5.10 wird ein Entwicklungsprozess zum Entwerfen und Entwickeln eines zuverlässigen, dynamisch rekonfigurierbaren Systems, das eine Anforderung an ein Fail-Operational-Verhalten aufweist, vorgeschlagen.

Die Modellierung eines SOA-basierten Systems, das dynamische Komponenten kombiniert, ist eine herausfordernde Aufgabe. Der hier vorgestellte Ansatz konzentriert sich auf die Modellierung des Systems in verschiedenen Abstraktionsebenen, die für

die dynamische Rekonfiguration am relevantesten sind: Service, Software und Hardware. Wie in Abbildung 5.11 dargestellt, müssen beim Entwurf eines Systemmodells unterschiedliche Ebenen berücksichtigt werden.

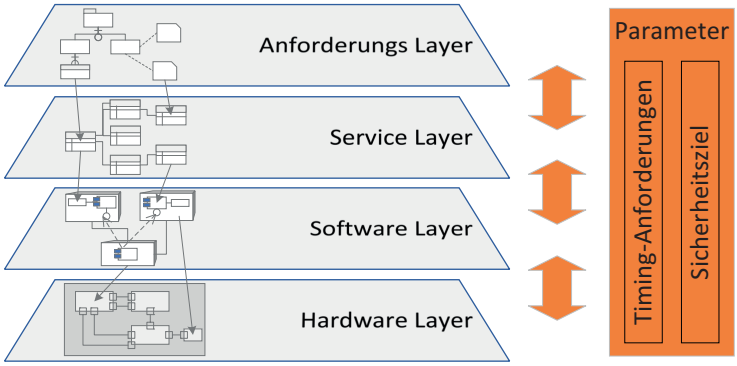

Abbildung 5.11: Ebenenarchitektur für die Systemmodellierung nach [145]

Darüber hinaus gibt es weitere Ebenen, zum Beispiel die Anforderungsebene und die Funktionsebene, die in dieser Arbeit nicht weiter berücksichtigt werden. Dieser Ansatz ermöglicht die genaue Beschreibung der verschiedenen Ebenen, sodass der Rekonfigurationsprozess anhand der modellierten Informationen überprüft werden kann.

Service-Modellierung

Die Service-Modellierung auf der entsprechenden Ebene aus Abbildung 5.11 zeigt die abstrakte Ebene innerhalb des dargestellten Systemdesign-Ansatzes. Die Hauptaufgabe in diesem Zusammenhang ist die Definition von Diensteschnittstellen. In diesen wird beschrieben, wie auf bereitgestellte Dienste von Clients zugegriffen werden kann. Weiterhin definiert diese, wie die Dienste auf die Software- und Hardwareebene abgebildet werden. Dazu wurden in dieser Arbeit die folgenden vier Ebenen festgelegt: die Hardwareebene, die Softwareebene, die Serviceebene und die Anforderungsebene [145].

Service-Interface-Design Service-Interfaces beschreiben den Datenaustausch zwischen Services und Clients mit unterschiedlichen Interaktionsmustern. In Tabelle 5.7

werden Interaktionsmuster nach erforderlichen Datenkanälen in Bezug auf Dienste und Clients berücksichtigt. Neben der Beschreibung von Datenkanälen können Service-Interfaces auch Echtzeiteigenschaften wie Timing oder die Erfüllung von Sicherheitsanforderungen anzeigen, die ihnen zugewiesen werden können.

Tabelle 5.7: Bewertung von Entwurfsmustern

Interaktionsmuster		**Ereignisbasierte Ausführungszeit**	**Ereignisbasiert (regelmäßige Aktualisierung)**	**Request-Response**
Client	Eingabedatenkanal	Ja	Ja	Ja
	Ausgabedatenkanal	Nein	Nein	Ja
Service	Eingabedatenkanal	Nein	Nein	Ja
	Ausgabedatenkanal	Ja	Ja	Ja

SOME/IP unterscheidet hierbei drei verschiedene Nachrichtentypen. Eine **Notification** wird vom Provider an den Consumer verschickt und informiert über ein eingetretenes Event. Eine **Method** stellt einen normalen Funktionsaufruf dar. Der Consumer schickt einen Request an den Provider und der Provider antwortet mit dem Rückgabewert der Funktion. **Fire-and-Forget** ist identisch mit einer Method, es wird vom Provider jedoch keine Antwort versendet. Tabelle 5.8 zeigt beispielhaft einige verwendete Methoden:

Tabelle 5.8: SOME/IP-Methoden IDs

Name	**ID**	**Provider**	**Typ**
can_receive	0x8100	CAN_PR_x	Notification
can_send	0x100	CAN_PR_x	Fire-and-Forget
traj_get	0x110	TRAJ_PR_x	Method

Der Trajektorien-Provider stellt auf Anfrage die nächste Lenkradposition sowie die Befehle der Pedale bereit. Hierfür wird die Methode traj_get verwendet. In der Request-Botschaft übermittelt die Applikation den gewünschten Index innerhalb des Traces. Der Provider liest die entsprechenden Werte aus der Datei und schickt die Antwort an die Applikation. Neben den Trajektorien wird auch ein Return-Code übermittelt. Dieser enthält Informationen über den Status des Providers. Tritt ein Fehler beim Lesen der Datei auf, wird der Rückgabewert entsprechend gesetzt und die Applikation muss zum Backup-Provider wechseln, um die korrekte Funktion des Systems zu gewährleisten.

Der IPC[10]-CAN-Provider dient als Schnittstelle zwischen SOME/IP und der proprietären Inter-Prozessor-Kommunikation innerhalb eines Boards. Über SOME/IP wird die Fire-and-Forget-Methode can_send angeboten. Zusätzlich erhält der Consumer des Services (Applikation) die empfangenen CAN-Frames über eine Notification. Der IPC-CAN-Provider empfängt neben den CAN-Frames diverse Statusmeldungen der DSA. Beim Wechsel vom R5-Kern auf den MicroBlaze verschickt die PMU eine entsprechende Nachricht, sodass der IPC-CAN-Provider die nachfolgende CAN-Kommunikation über den MicroBlaze abwickelt.

Das Umschalten zwischen dem R5 und dem MicroBlaze geschieht transparent für die Applikation. Der Ausfall eines CAN-Interfaces wird vom R5 bzw. dem MicroBlaze erkannt und an den IPC-CAN-Provider weitergeleitet. In diesem Fall meldet dieser Service sich beim SOME/IP-Router ab.

Software-Modellierung

Auf der Softwareebene ist das Instanziieren von Diensten und Clients implementiert. Software-Komponenten stellen dabei Instanzen von Diensten und Clients zur Verfügung. Die Task einer Softwareeinheit spiegelt die Anforderungen an Ressourcen und Echtzeitanforderungen wider. Sie kann zum Planen der benötigten Rechenressourcen verwendet werden.

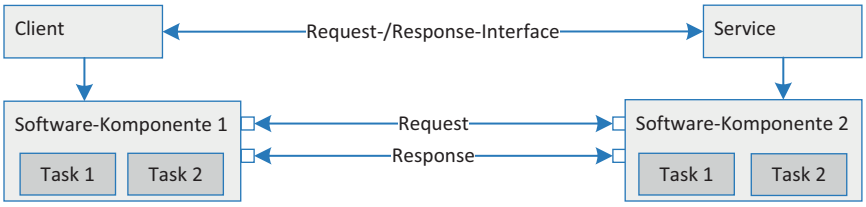

Abbildung 5.12: Beziehung zwischen Service, Clients und Software-Komponenten nach [145]

Die Beschreibung der einzelnen Tasks ist in Abschnitt 6.3.2 dargestellt. Notwendige Parameter wie die maximale Ausführungszeit können als Teil einer Software-Komponente beschrieben werden, die dann für den Verifizierungsprozess verwendet werden. Neben den Software-Komponenten sind in Abbildung 5.12 auch die Funktionen innerhalb

[10] Inter Processor Communication

dieser Komponenten und eine Client-/Service-Abhängigkeit in Form eines Request-/Response-Kommunikationsmusters dargestellt.

Abbildung 6.14 in Abschnitt 6.3.2 zeigt die einzelnen Services und deren Interfaces. Für das AutoKonf-Projekt wurden zwei verschiedene Services definiert. Der Trajektorien-Provider (TRAJ_PR_x) stellt die Daten des Fahrszenarios zur Verfügung und der IPC-CAN-Provider (CAN_PR_x) übergibt diese an die DSA.

Tabelle 5.9: SOME/IP-Service IDs

Applikation	Applikations-ID	Service-ID	Instanz-ID	Board
CAN_PR_0	0x10	0x1000	0x100	1
CAN_PR_1	0x11	0x1000	0x101	2
AutoKonf	0x20	-	-	1
TRAJ_PR_0	0x30	0x1001	0x110	1
TRAJ_PR_1	0x31	0x1001	0x111	2

Auf jedem Fahrrechner ist eine Instanz von jedem Typ aktiv, um im Fehlerfall den Provider wechseln zu können. Zur Identifizierung der Software-Komponenten in SOME/IP sind die Applikations-, Service- und Instanz-IDs in Tabelle 5.9 definiert und vergeben.

Hardware-Modellierung

Die verwendeten Hardware-Elemente für den Einsatz von Software-Komponenten und die Beziehung zwischen diesen Elementen werden auf der Hardwareebene beschrieben. Die Bereitstellung von Softwareeinheiten in physischen Netzwerken, also der Netzwerktopologie, wird von den Tasks aus dem Modell der Softwarearchitektur auf den modellierten Hardware-Elementen zur Verfügung gestellt. In dieser Hinsicht wird die Auswahl geeigneter Computerressourcen durch Fokussierung auf die Ressourcen- und Echtzeitanforderungen von Tasks abgeleitet.

Basierend auf dem abgeleiteten statischen Modell der Systemarchitektur werden die Verhaltensaspekte der resultierenden Run-Time-Architektur beschrieben. Da die Verwendung von serviceorientierten Architekturen im Fokus steht, werden Laufzeitaspekte hauptsächlich durch dynamische Kommunikationsbindung erfasst.

Abbildung 5.13 zeigt die Verwendung der Software-Komponenten auf der entsprechenden Hardware.

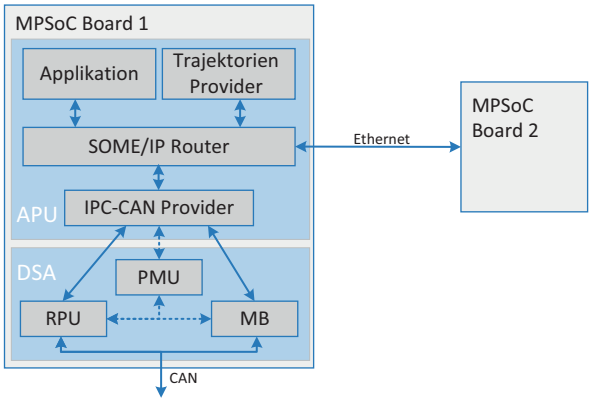

Abbildung 5.13: Modellierung der verwendeten HW- und SW-Komponenten

Service-Discovery

Die Service-Discovery implementiert die dynamische Bindung der Kommunikation mithilfe eines Protokolls, das Client-/Service-Abhängigkeiten zur Laufzeit auflöst. Zu diesem Zweck erhalten Client-Instanzen für bestimmte Dienste Offer-Nachrichten. Anschließend erfolgt das Abonnement in Form einer Antwortnachricht vom Client zum Service.

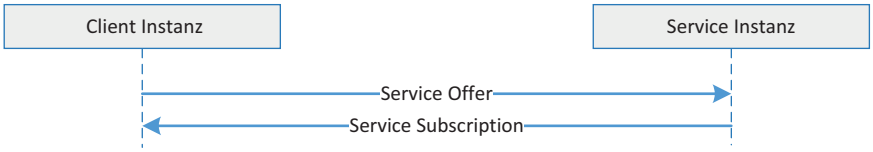

Abbildung 5.14: Service-Discovery nach [145]

Abbildung 5.14 zeigt eine einfache Darstellung des Service-Discovery-Mechanismus mit einem Client und einem Service. Im Zusammenhang dieser Arbeit wurden unter anderem ein Trajektorien-Provider und ein CAN-Kommunikations-Provider implementiert,

die den Dienst des jeweiligen anderen abonniert haben. Mit dieser Architekturauslegung ist es möglich, die Provider auch außerhalb der HW-Plattform zu integrieren und dynamisch zu- und abzuschalten.

Der CAN-Kommunikations-Service ist in Abschnitt 6.1.6 beschrieben. Die dynamische Rekonfiguration basierend auf dem Service-Discovery-Konzept wird als Überblick in Abbildung 6.6 gezeigt. Der Gesamtablauf der Rekonfiguration im Zusammenspiel der einzelnen Komponenten ist in Abbildung 6.15 dargestellt.

Service-Nutzung im Gesamtablauf

Nach Abschluss der Service-Discovery können Clients die angeforderten Services verwenden. In dieser Hinsicht ist die Nutzung eines Dienstes häufig in eine Kette von Client-/Service-Abhängigkeiten eingebettet. In Abbildung 5.15 ist ein Beispiel für ein Netzwerk von einem Client und zwei Diensten dargestellt. Die Abbildung zeigt ein Ethernet-Netzwerk, das drei Steuergeräte über einen Switch miteinander verbindet. Eine typische Folge von Client-/Service-Abhängigkeiten beginnt mit dem Einsatz eines Sensordienstes durch eine Client-Anwendung, z. B. eine Fahrerassistenzanwendung.

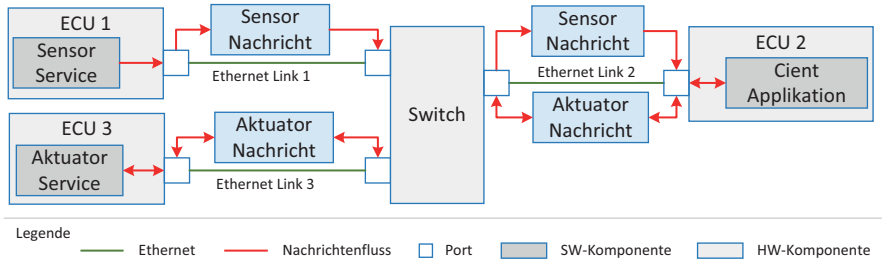

Abbildung 5.15: Analyse des Systemdesigns nach [145]

Nach einem Verarbeitungsschritt in der Client-Anwendung wird die nachfolgende Verwendung eines Aktuatordienstes z. B. eines Motors zum Steuern der Bremse oder der Lenkung ausgeführt. Dieser ist mit Ports ausgestattet, die eine Duplexübertragung von Daten ermöglicht. Die auf den ECUs implementierte Software veranschaulicht einen Sensordienst, eine Clientanwendung und einen Aktuatordienst. Ein ereignisbasiertes

Interaktionsmuster stellt die Interaktion zwischen dem Sensorservice und der Client-Anwendung her. Im Gegensatz dazu wird für den Aktuatordienst und die Client-Anwendung ein Request-/Response-Interaktionsmuster angewendet.

5.3.4 Analyse zur Abschätzung der Ausführungszeiten

Insbesondere bei sicherheitskritischen Funktionen müssen bestimmte Timing-Anforderungen erfüllt werden. Ein Beispiel dafür ist die Strecke, um Daten von einem Sensor-Service zu empfangen, in einer Client-Anwendung zu verarbeiten und schließlich die Service-Schnittstelle eines Aktuator-Service zu steuern.

Diese Timing-Anforderung soll unterhalb einer bestimmten Grenze bleiben. Um diese Eigenschaft zu bewerten, wird im Rahmen des Entwurfsansatzes eine durchgängige Bewertung der Latenz erstellt. Diese Beurteilung erfolgt in zwei Schritten und ist im Folgenden detailliert beschrieben. Die zugehörige Anwendung dieser Methodik und die entsprechenden Ergebnisse sind in Abschnitt 6.3.2 dargestellt.

1. Statische Analyse: das Aufstellen einer Ende-zu-Ende-Latenzzeitanforderung in einem Netzwerk interagierender Dienste und Clients. Die Rekonfigurationszeit wird mit Hilfe von statischen Systemmodellen berechnet.

2. Simulative Bewertung: die Überprüfung der Anforderung durch simulationsbasierte Evaluierungstechniken.

Die ISO 26262 erwähnt explizit die Erfüllung von drei grundlegenden nicht-funktionalen Anforderungen. Abwesenheit von Laufzeitfehlern, Ausführungszeit und Speicherbedarf. In dieser Arbeit wurde ausschließlich die Ausführungszeit tiefergehend betrachtet, so dass die hier beschriebene Analyse der Ausführungszeiten zum Einsatz kommt (vgl. [82]).

Statische Analyse des Systems

Bei der statischen Analyse soll das zu entwickelnde System mit einem modellbasierten Entwicklungswerkzeug modelliert werden. Dieses Systemmodell wird dann als Eingabe für die statische Analyse verwendet. Die Ausführungszeiten werden auf der Softwareebene in Kombination mit der ausgewählten Hardware ermittelt. Die übrigen

Kommunikationskomponenten wie Bussysteme und andere Netzwerklatenzen werden ebenfalls in der Hardwareebene modelliert.

Dies allein liefert nur die Basiszeitinformation in dem System. Um die Ende-zu-Ende-Latenz der Rekonfiguration eines Services zu berechnen, müssen auch die Verhaltensaspekte modelliert werden. Deswegen soll der Rekonfigurationsprozess explizit anhand eines Sequenzdiagramms beschrieben werden. Mit diesem Ansatz ergibt sich eine explizite Beschreibung des Rekonfigurationsprozesses. Der Mehrwert ist nicht nur die Möglichkeit, die Rekonfigurationszeiten abzuschätzen und zu berechnen, sondern auch die Beschreibung und Rückverfolgbarkeit des Prozesses. Alle diese ermittelten Daten können für die spätere Überprüfung und Zertifizierung eines solchen Systems relevant sein.

Simulative Bewertung des Timing-Verhaltens

Im simulativen Bewertungsprozess wird das modellierte System mithilfe eines Simulationstools nachgestellt. Auf diese Weise können Werte wie die Ausführungszeit für die spätere Verifizierung abgeleitet werden. Mögliche Tools zur Systemauslegung und -bewertung werden bereits in der Praxis angewendet. Eine Möglichkeit hierzu bietet die Tool-Suite chronSIM / chronVAL [70], die sich auf Echtzeiteigenschaften eingebetteter Systemnetzwerke konzentriert. Das Tool besitzt zwei wesentliche Stärken: Zum einen ist eine simulationsbasierte Bewertung der Timing-Eigenschaften möglich, zum anderen kann eine Einschätzung basierend auf einem analytischen Ansatz erzielt werden. Für die simulative Beurteilung sind folgende Prozessschritte erforderlich:

1. Zunächst muss das System modelliert werden.
2. Danach muss der Software-Code geschrieben werden.
3. Als Nächstes wird der Code in das Simulationswerkzeug aufgenommen.
4. Festlegung der Timing-Eigenschaften.
5. Abschließende Ausführung der Simulation.

Das hier beschriebene Verfahren wird für die Abschätzung der Echtzeiteigenschaften in dieser Dissertation gemäß diesen Schritten eingesetzt. Die Anwendung dieser Methodik auf den konkreten Einsatzfall und die daraus resultierenden Ergebnisse sind in Abschnitt 6.3 aufgeführt.

Dieses Verfahren ergibt eine obere Schranke für die Ausführungszeiten, die großzügig bemessen ist. Eine exakte Bestimmung der WCET[11] ist aufgrund der Systemkomplexität nur mit bedeutend höherem Aufwand verbunden und nicht Bestandteil dieser Arbeit (vgl. Abschnitt 3.2.3).

5.4 Verfahren zur Evaluierung in frühen Entwurfsphasen

Um in einer frühen Entwurfsphase eine Evaluierung über die Zuverlässigkeit des Systems geben zu können, wird in diesem Abschnitt zunächst eine Unterscheidung von deduktiven und induktiven Analysemethoden gegeben (Abschnitt 5.4.1), danach wird auf die Markov-Ketten eingegangen (Abschnitt 5.4.2) und abschließend eine Kombination mit der Monte-Carlo-Simulation zur Berücksichtigung von Ungenauigkeiten in der Ermittlung von Failure In Time(FIT)-Raten vorgestellt (Abschnitt 5.4.3). Der folgende Abschnitt basiert unter anderem auf Ausarbeitungen einer im Rahmen dieser Dissertation betreuten Masterarbeit [22] und einer daraus resultierenden eigenen Veröffentlichung [144].

5.4.1 Zuverlässigkeitsmodellierung

Modellierungs- und Analysemethoden werden zur Vorhersage der Ausfallhäufigkeit verwendet, und um die Wahrscheinlichkeit der Verletzung eines Sicherheitsziels aufgrund von Ausfällen abzuschätzen. Die Methoden zur Analyse dafür können in zwei allgemeine Kategorien eingeteilt werden: induktive sowie deduktive Verfahrensweisen. Bei einer induktiven Systemanalyse wird ein bestimmter Fehler oder auslösendes

[11] Worst-Case Execution Time

Ereignis angenommen. Es wird versucht, die Auswirkungen dieses Fehlers auf den gesamten Systemausfall zu ermitteln.

Die induktiven Methoden werden somit dafür verwendet, mögliche Systemzustände zu identifizieren, in diesem Fall handelt es sich um die Fail-States. Beispiele für induktive Systemanalysen sind die Failure Modes, Effects and Diagnostic Analysis (FMEDA) und Failure Mode Effect and Criticality Analysis (FMECA). Diese werden in Form von Tabellen dokumentiert, in denen die Funktions-, System- oder Subsystemfehler beschrieben sind.

In einer deduktiven Systemanalyse wird ein Systemfehler angenommen und versucht herauszufinden, welche Modi des System- oder Komponentenverhaltens zu einem Systemausfall beitragen. Ein Beispiel dafür ist die Fault Tree Analysis (FTA), bei der ein Fehlerszenario in Betracht gezogen und in seine etwaigen Auslöser zerlegt wird. Die Darstellung erfolgt in einer Baumstruktur. Jede mögliche Ursache wird untersucht und weiter verfeinert, bis die grundlegenden Ursachen des Versagens ermittelt sind.

Ähnlich wie die FTA können auch Zuverlässigkeitsdiagramme und Zuverlässigkeitsblockdiagramme (RDB[12]) verwendet werden, um verschiedene Kombinationen von Komponentenfehlern zu spezifizieren, die zu einem bestimmten Zustand oder Leistungsniveau eines Systems führen.

5.4.2 Markov-Ketten-Modell zur Integration des dynamischen Systemverhaltens

Markov-Ketten repräsentieren diskrete dynamische Systeme. Dynamische Fehlerbäume erweitern eine traditionelle FTA um dynamisches Systemverhalten wie die Sequenzabhängigkeit und einen gemeinsamen Ressourcenpool (stochastische Modelle). Diese Besonderheit erlaubt es, das dynamische Verhalten in Markov-Ketten zu integrieren, die für die Zuverlässigkeitsanalyse verwendet werden.

Die zwei Hauptkonzepte von Markov-Ketten sind Systemzustände und Zustandsübergänge. Zur Beschreibung der Systemzuverlässigkeit stellt jeder Zustand des Markov-

[12] Reliability Block Diagram

Modells im Allgemeinen eine eindeutige Kombination von fehlerhaften und fehlerfreien Komponenten dar.

Im Laufe der Zeit und während Ausfälle auftreten, geht das System von einem Zustand in einen anderen über, bis der Ausfallzustand normalerweise der Systemausfall erreicht ist.

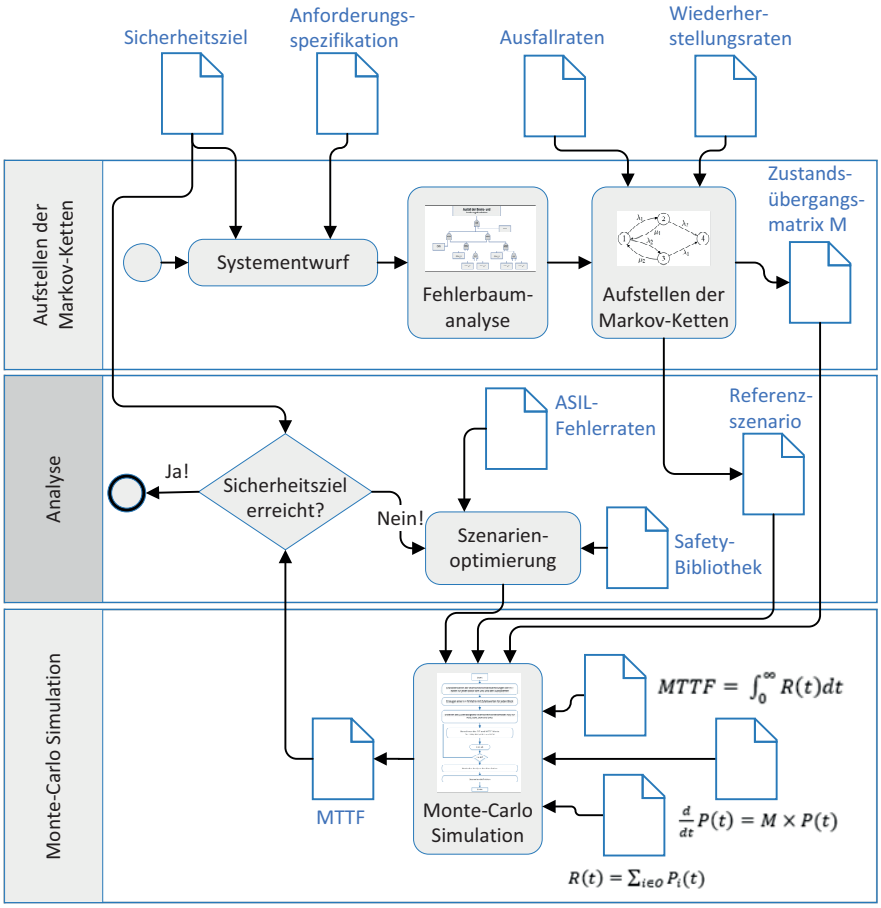

Abbildung 5.16: Gesamthaftes Vorgehen zur Systemanalyse von E/E-Architekturen mit der Markov-Ketten Monte-Carlo-Methode

Es handelt sich bei der hier verwendeten Markov-Kette also um eine endliche, diskrete Variante. Die Zustandsübergänge sind gekennzeichnet durch folgende Parameter:

1. Ausfallraten
2. Fehlerabdeckungsfaktoren
3. Reparaturraten/Wiederherstellungsraten

Eine MPSoC-Plattform stellt bereits ein komplexes System dar für das die Ausfallratenermittlung nicht trivial ist. In dieser Arbeit besteht das Gesamtsystem, die E/E-Architektur, aus wesentlich mehr Komponenten. Für dieses Gesamtsystem soll nun eine Zuverlässigkeitsanalyse durchgeführt werden, um eine Aussage über das Erreichen des geforderten Sicherheitsziels aus der Anforderungsanalyse treffen zu können (vgl. Abschnitt 5.2.1).

Abbildung 5.16 zeigt das gesamthafte Vorgehen zur Systemevaluierung unter Einsatz der Markov-Ketten Monte-Carlo-Methode. Von den drei Parametern der Zustandsübergänge wird im Folgenden nur die Ausfallrate herangezogen. Da davon ausgegangen wird, dass keine Selbstheilung bei Ausfall einer Komponente durchgeführt wird, liegt bei der hier aufgestellten Markov-Kette nur eine Richtung der Zustandsübergänge vor. Die Gesamtausfallrate erschließt sich aus den Einzelausfallraten der Komponenten, die in den entsprechenden Dokumenten der Hersteller zu finden sind [140]. Die Ausfallraten beruhen auf möglichen Fehlern, die zu einem Ausfall führen können. Dabei wird zwischen Soft-Errors und Hard-Errors unterschieden [139], die jeweils zu einem Single Event Effect (SEE) führen.

5.4.3 Monte-Carlo-Simulation zur Berechnung der komplexen Modellierung

FIT-Raten sind im Allgemeinen mit Ungenauigkeiten behaftet, die sich aus den zu ihrer Bestimmung verwendeten Methoden ergeben. Zu diesen Ungenauigkeiten gehören Messfehler, statistische Ungenauigkeiten und Transformationsungenauigkeit. Verständlicherweise ist es schwierig, diese Ungenauigkeiten zu charakterisieren und in das Markov-Kettenmodell einzubeziehen. Der Zweck der MCS[13] ist es, einen Ansatz

[13] Monte-Carlo-Simulation

zu verwenden, der FIT-Ratenschwankungen berücksichtigt und somit das Konfidenzniveau der Zuverlässigkeit für das Gesamtsystem erhöht. MCS ist eine wichtige Berechnungstechnik zur Verbesserung der statistischen Tests. Die für die Modellierung und Zuverlässigkeitsanalyse verwendete stochastische Methode wird unter Berücksichtigung der folgenden Annahmen angewendet:

1. Die Fehlererkennungseinheit ist perfekt, d.h. jeder mögliche Fehler wird erkannt.
2. Die Fehlerabdeckung durch entsprechende Maßnahmen liegt bei $100\,\%$.
3. Reparaturraten werden nicht berücksichtigt, daher werden nur irreparable Fehler[14] angenommen.

Die Annahme in Punkt 1 wird getroffen, weil die Werte der FIT-Raten bereits die FIT für die Erkennungsmechanismen des Modells enthalten. Auf diese Weise wird das Modell vereinfacht. Der gleiche Ansatz wird für jeden Block in jedem Subsystem verwendet, wobei jeder Block verschiedene Hardware-Elemente umfasst, einschließlich redundanter Elemente und Fehlererkennungselemente. Dieser Ansatz ermöglicht die intelligente Aufteilung und Partitionierung eines komplexen Systemmodells in kleinere Teilsystemmodelle, wodurch die Wiederverwendbarkeit jedes Teilsystemmodells ermöglicht wird [13].

Zur Vereinfachung der Systemmodellierung kann die Behauptung in Punkt 2 aufgestellt werden, dass die Fehlerabdeckung zu $100\,\%$ berücksichtigt wurde, da die zur Fehlererkennung und FIT-Bestimmung verwendeten Methoden und Tests auf Näherungen, unterschiedlichen Konfidenzniveaus und gesammelten Daten beruhen [140]. Punkt 3 kommt zum Einsatz, da Reparaturraten zunächst zur Vereinfachung des Systems ebenfalls ausgeschlossen werden. Auf diese Weise wird der Betrachtungszeitraum nur bis zu einem Ausfall gewählt. Diese Aspekte führen zu Ungenauigkeiten bezüglich des Modells und seiner Ergebnisse. Um das Modell in einen Zertifizierungsrahmen zu integrieren, sind folgende Ansätze möglich:

1. Kombination aus zufälliger Simulation und statistischer Modellprüfung
2. Modellbasierte Musteridentifikation zum Eingrenzen von Ergebnissen und Zustandsräumen [13]

[14] FIT-Raten aus Soft-Errors fließen in die Betrachtung nicht mit ein

Aus diesen Ungenauigkeiten ergeben sich potenzielle Auswirkungen und Abweichungen. Deshalb wird die Monte-Carlo-Simulation in die Bewertungsmethode mit einbezogen, um diese Implikationen zu ermitteln.

Wie bereits erwähnt, besteht die E/E-Architektur aus drei Steuergeräten und dem CAN-Bus. Jede FIT-Rate der einzelnen Steuergeräte-Funktionsblöcke umfasst nicht nur das Steuergerät selbst und die zugrunde liegende Software- und Hardware-Redundanz, sondern auch die Schnittstellen zu Brems- und Lenksystemen.

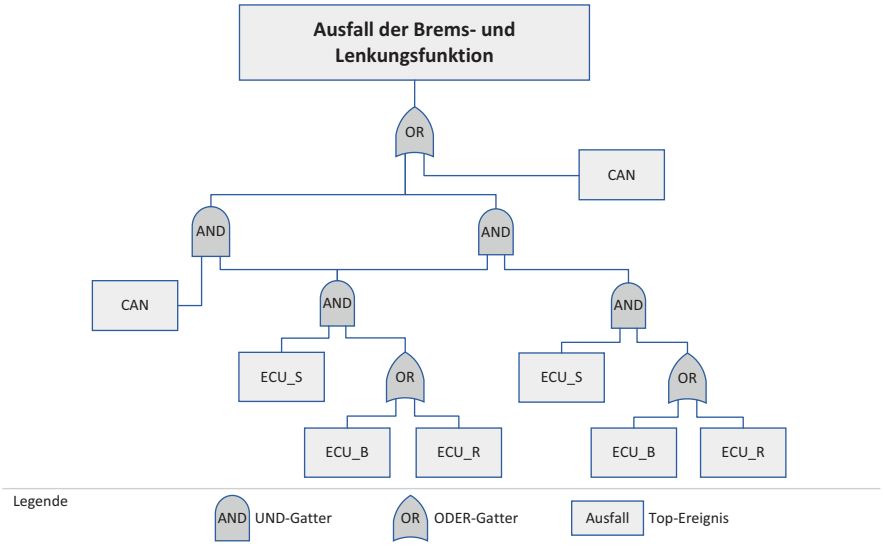

Abbildung 5.17: Fehlerbaumanalyse der E/E-Architektur nach [144]

Die E/E-Architektur kann mithilfe der deduktiven Methode der FTA von oben nach unten analysiert werden. Das TOP-Ereignis repräsentiert den Totalausfall der Lenk- und Bremsfunktion. Die Ereignisse, die zu dem TOP-Ereignis führen resultieren aus der Kombination von Ausfällen, entweder der Steuergeräte oder des CAN-Bus.

Die einzelnen Kombinationen sind in der Fehlerbaumanalyse entsprechend dargestellt. Diese Aggregation ist das Ergebnis früherer FMEDA- und FTA-Analysen, die für jedes Steuergerät als Subsystem im Rahmen des AutoKonf-Projekts durchgeführt wurden [59]. Die entsprechende FTA für die E/E-Architektur ist in Abbildung 5.17 dargestellt.

Wie in Abschnitt 5.4.2 bereits erwähnt, kann die FTA auch als Markov-Kette repräsentiert werden. Die aus der Fehlerbaumanalyse der E/E-Architektur resultierende Markov-Kette ist in Abbildung 5.18 dargestellt.

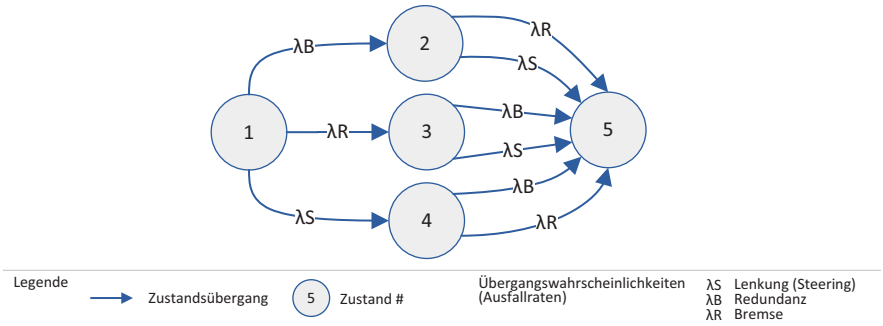

Abbildung 5.18: Übergangsgraph für die beschriebene Markov-Kette der E/E-Architektur nach [144]

Die möglichen Zustände sind in Tabelle 5.10 beschrieben. Die Zustände 1 bis 4 sind Betriebszustände, Zustand 5 ist ein Fehlerzustand.

Tabelle 5.10: Zustände der Markov-Kette für das E/E-System

Zustand	Beschreibung
1	Die drei ECUs sind betriebsbereit
2	Zwei Steuergeräte sind betriebsbereit (Bremssteuergerät ausgefallen)
3	Zwei ECUs sind betriebsbereit (redundante ECU ausgefallen)
4	Zwei Steuergeräte sind betriebsbereit (Steuergerät ausgefallen)
5	Das System ist ausgefallen (zwei Steuergeräte sind ausgefallen)

In einem Hot-Standby-Modus mit einer Ersatzkonfiguration sind mindestens zwei Module erforderlich, damit das System betriebsbereit ist. Wenn also die dritte Komponente ausfällt, fällt das System aus.

Bei dieser Zuverlässigkeitsbewertung kann das System nach einem Ausfall nicht repariert werden. Die Ausfallraten λ_B, λ_S, λ_R repräsentieren jeweils die Ausfallraten für den CAN-Bus, die Brems-, Lenk- und die redundanten ECUs.

Die folgenden Differentialgleichungen stellen die Zustandsübergangsmatrix für die in Abbildung 5.18 und Tabelle 5.10 gezeigte Markov-Kette dar:

$$\begin{aligned}
\dot{P}_1 &= -(\lambda_B + \lambda_S + \lambda_R)P_1; \\
\dot{P}_2 &= \lambda_B P_1 - (\lambda_S + \lambda_R)P_2; \\
\dot{P}_3 &= \lambda_R P_1 - (\lambda_S + \lambda_B)P_3; \\
\dot{P}_4 &= \lambda_S P_1 - (\lambda_B + \lambda_R)P_4; \\
\dot{P}_5 &= (\lambda_S + \lambda_R)P_2 + (\lambda_S + \lambda_B)P_3 + (\lambda_B + \lambda_R)P_4;
\end{aligned} \tag{5.1}$$

Unter der Annahme, dass der Ausgangszustand fehlerfrei ist, gelten folgende Bedingungen:

$$\begin{aligned}
&P_1(t=0) = 1; \\
&P_2(t=0) = P_3(t=0) = P_4(t=0) = P_5(t=0) = 0;
\end{aligned} \tag{5.2}$$

Mit diesen Anfangsbedingungen ist es nun möglich, die Differentialgleichungen zu lösen und die resultierenden Zustandswahrscheinlichkeiten für jeden Zustand zu bestimmen. Die Systemzuverlässigkeit $R(t)$, Verfügbarkeit oder Sicherheit kann als Summe der Wahrscheinlichkeiten berechnet werden, die für alle Betriebszustände übernommen wurden.

$$\begin{aligned}
&P_1 bis P_4; \\
R(t) &= \sum_{i=1}^{4} P_i(t);
\end{aligned} \tag{5.3}$$

Die DSA ist auf den LPD- und PL-Domänen der Xilinx Zynq ZCU 102-Boards implementiert und vom SOA-System isoliert, obwohl es auf demselben Board umgesetzt ist. Die möglichen Zustände sind in Tabelle 5.11 beschrieben.

λ_{RPU} und λ_{MB} sind die Ausfallraten für die RPU($m_{complex}$) bzw. MB ($m_{fallback}$). λ_K umfasst die Ausfallraten der auf der LPD (m_{atom}, S_{atom}, S_{state} und der jeweiligen Verbindung) implementierten Module (K-Module), XPPU[15] ($S_{protection}$), PMU ($m_{control}$), NOC[16]-Modul (stellvertretend für die Verbindungen, I/O zwischen LPD und PL) und

[15] Xilinx Peripheral Protection Unit
[16] Network-On-Chip

sofort einsatzbereite Module, die bestimmte Funktionen wie die CSU[17] für sicheres Booten haben.

Der gleiche Ansatz, der für das Hardware Dependant E/E System (HDS) verwendet wird, wird auf die DSA und ihre FTA angewendet. Die Markov-Kette für die DSA ist in Abbildung 5.19 dargestellt.

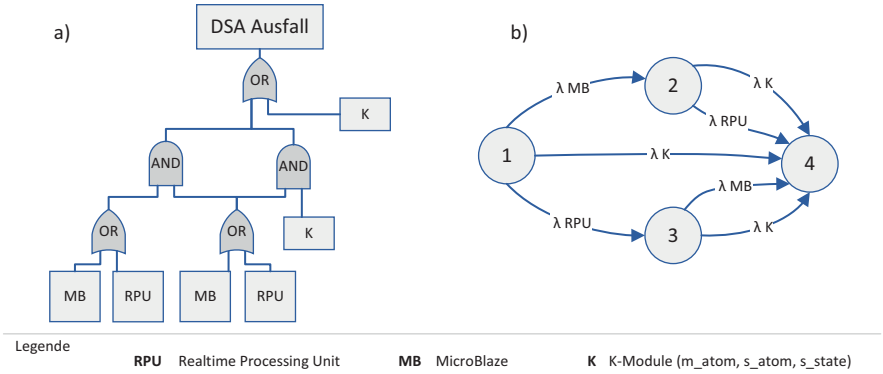

Abbildung 5.19: DSA: a) Fehlerbaumanalyse b) Markov-Kette nach [144]

Die zugehörigen Fehlerzustände sind in Tabelle 5.11 zusammengefasst.

Tabelle 5.11: Zustände der Markov-Kette für DSA

Zustand	Beschreibung
1	Alle Module sind betriebsbereit
2	Fehler auf dem MB. RPU- und K-Module sind betriebsbereit.
3	Fehler an der RPU. MB- und K-Module sind betriebsbereit.
4	Systemfehler. K-Module sind ausgefallen oder RPU und MB sind ausgefallen.

Für das DSA-Modell werden die gleichen Annahmen und Bedingungen verwendet, wie sie auch beim HDS zum Einsatz gekommen sind. Die Berechnung der DSA-Zuverlässigkeit folgt der identischen Regel der Summe der Wahrscheinlichkeiten von Zustand P_1 bis P_3.

[17] Configuration Security Unit

Die Zustandsübergangsmatrix für die in Abbildung 5.19 und Tabelle 5.11 gezeigte HDS-Markov-Kette kann durch die folgenden Differenzialgleichungen ausgedrückt werden:

$$\begin{aligned}
\dot{P}_1 &= -(\lambda_{MB} + \lambda_K + \lambda_{RPU})P_1; \\
\dot{P}_2 &= \lambda_{MB}P_1 - (\lambda_K + \lambda_{RPU})P_2; \\
\dot{P}_3 &= \lambda_{RPU}P_1 - (\lambda_K + \lambda_{MB})P_3; \\
\dot{P}_4 &= \lambda_K P_1 + (\lambda_K + \lambda_{RPU})P_2 + (\lambda_K + \lambda_{MB})P_3;
\end{aligned} \tag{5.4}$$

Wie bereits erwähnt, umfasst das DRS zwei Subsysteme. Jede APU muss den Status ihrer jeweiligen DSA-Module verfolgen und mit der anderen APU kommunizieren, um die Durchführbarkeit eines Kontextwechsels zwischen einer fehlerhaften DSA und dem redundanten DSA zu kontrollieren.

Bei einem Ausfall einer APU kann die andere APU diesen Ausfall quittieren und den Kontextwechsel zwischen den DSAs durchführen. Dieser Ansatz wird unter Berücksichtigung des gesamten DRS (HDS, DSA und SOA) analysiert. Die möglichen Fehlerzustände sind in Tabelle 5.12 beschrieben.

Tabelle 5.12: Zustände der Markov-Kette für DRS

Zustand	Beschreibung
1	Alle Systeme sind betriebsbereit.
2	Fehler am DSA A. Verbleibende Systeme sind betriebsbereit.
3	Fehler in der SOA A. Die verbleibenden Systeme sind betriebsbereit.
4	Fehler in der SOA B. Die verbleibenden Systeme sind betriebsbereit.
5	Fehler am DSA B. Die verbleibenden Systeme sind betriebsbereit.
6	Ausfall des Subsystems A (DSA A + SOA A). Subsystem B ist betriebsbereit.
7	Ausfall des Subsystems B (DSA B + SOA B). Subsystem A ist betriebsbereit.
8	Das System ist ausgefallen.

Die Analyse umfasst die vorherigen Systeme und ist im Sinne einer Top-Down-Analyse durchgeführt. Der vollständige Funktionsverlust stellt das TOP-Ereignis dar und führt ebenso wie bei dem HDS zum Verlust des Lenkens und Bremsens. Anders als beim HDS umfasst das Hauptereignis des DRS jedoch auch den Ausfall des DSA und anderer weniger kritischer Funktionen, die möglicherweise enthalten sind, aber im Rahmen dieser Arbeit nicht bewertet werden. Die DRS-Fehlerbaumanalyse ist in Abbildung 5.20 dargestellt.

Für das DRS gilt der gleiche Ansatz wie für das DSA und das HDS. Die Differenzialgleichungen werden auf die gleiche Weise wie für die vorherigen Subsysteme abgeleitet. Für das DRS-Modell werden ebenfalls die im HDS verwendeten Annahmen und Bedingungen verwendet. Die Berechnung der DSA-Zuverlässigkeit folgt der gleichen Berechnungsvorschrift, bei der die Summe der Wahrscheinlichkeiten von Zustand P_1 bis P_7 gebildet wird.

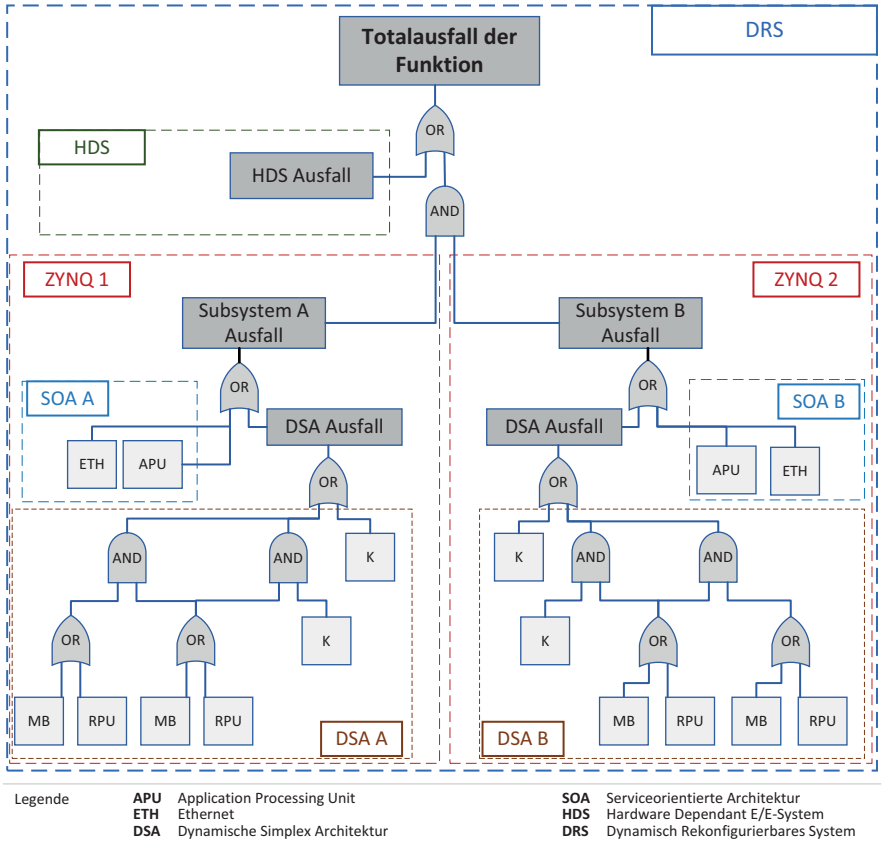

Abbildung 5.20: Fehlerbaumanalyse vom DRS nach [144]

Die FTA kann somit in eine Markov-Kette überführt werden, die in Abbildung 5.21 dargestellt ist. Die Erweiterung der FTA um das dynamische Verhalten durch die Verwendung von Markov-Ketten ermöglicht eine Vereinfachung der Betrachtung der einzelnen Zustandsübergänge des Gesamtsystems.

Die Zerlegung in die einzelnen Teilsysteme und die anschließende Komposition zum Gesamtsystem tragen ebenfalls dazu bei, eine Zuverlässigkeitsanalyse einer vorliegenden E/E-Architektur durchführen zu können.

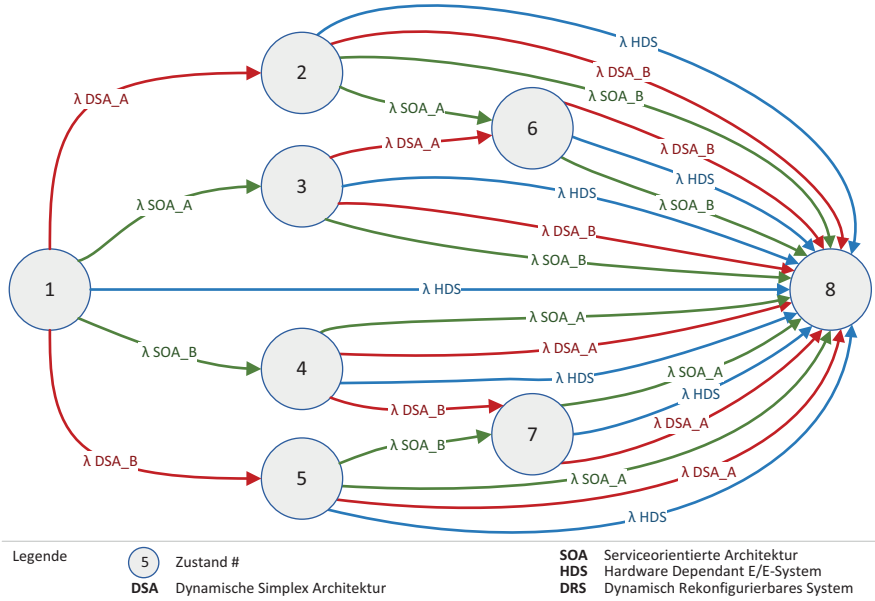

Abbildung 5.21: Markov-Kette von DRS nach [144]

Aufgrund der Komplexität bei der Ermittlung der Ausfallraten, ermöglicht die Monte-Carlo-Simulation nun, die komplexe Modellierung zu berechnen. In Abschnitt 5.4 wurde somit die Methode vorgestellt, die dazugehörigen Ergebnisse sind in Abschnitt 6.4 dargestellt. Bekannt ist diese Methode auch als MCMC.

5.5 Parametrierbare, modulare Bibliothek von Lösungsbausteinen

In diesem Abschnitt werden die einzelnen Middleware-Konzepte eingeführt (vgl. Abschnitt 5.5.1), die Erstellung von Lösungsszenarien aufgezeigt (vgl. Abschnitt 5.5.2), sowie die Einordnung in den Standardentwicklungsprozess vorgenommen (vgl. Abschnitt 5.5.3).

5.5.1 Middleware-Konzepte für den Einsatz im Automotive-Umfeld

Mit der Einführung von Automotive-Ethernet wurde die verfügbare Bandbreite im Fahrzeug deutlich erhöht und die Internet Protocol(IP)-basierte Kommunikation eingeführt. Die ersten Anwendungen waren die On-Board-Diagnose und insbesondere das Flash-Verfahren. Die klassischen Domänen wie Antriebsstrang, Karosserie oder Fahrwerk folgten wie bisher dem Ansatz der signalbasierten, statisch konfigurierten Kommunikation. Bus-Technologien wie CAN, Local Interconnect Network (LIN) oder FlexRay[18] wurden weiterhin eingesetzt.

SOME/IP war das erste Middleware-Protokoll, das die Vorteile der Ethernet-Technologie nutzte. Seit einiger Zeit werden zunehmend Protokolle aus dem Bereich der IT und des IoT[19] diskutiert. Das Hauptmerkmal ist der datenzentrierte Ansatz. Die diskutierten Protokolle sind das DDS-Protokoll und das MQTT[20]-Protokoll. Ein Vergleich der Protokolle, die den Automobilsektor betreffen, ist in Tabelle 5.13 zu finden.

Tabelle 5.13: Bewertung verschiedener Middleware-Protokolle

	Infotainment / Driver Assistance	**S**afety Critical	**H**ard Real-Time
SOME/IP	+	+	+
DDS	o	++	+
MQTT	+	+	-

Aus Protokollsicht heraus wird SOME/IP bezogen auf seine Leistungsfähigkeit nicht hinter DDS eingeordnet. Es sind keine grundlegenden Widersprüche zur funktionalen Sicherheit von SOME/IP vorhanden und es liegt kein Engpass bei erhöhtem Datenverkehr vor. Die Standardisierung ist gegeben und SOME/IP ist nahtlos in AUTOSAR integriert. Skalierbarkeit und Ressourcenverbrauch schneiden gut ab. Es wurden bereits zahlreiche Leistungsbewertungen der Protokolle durchgeführt [108].

[18] serielles, deterministisches und fehlertolerantes Feldbussystem für den Einsatz im Automobil
[19] Internet of Things
[20] Message Queuing Telemetry Transport

5.5.2 Lösungsszenarien

In Abschnitt 4.2 wurde eine kontextliche Methodik vorgestellt. In dem Ablaufplan (vgl. Abbildung 4.2) ist an erster Stelle und als Eingangsgröße für die nachfolgenden Schritte die Auswahl des Zielsystems und die Spezifikation zu finden. Für die erste Systemauslegung der E/E-Architektur kommt der morphologische[21] Kasten oder auch die morphologische Matrix zum Einsatz.

Der morphologische Kasten ist eine Methode zur systematischen Analyse komplexer Aufgabenstellungen, bei der in diesem Anwendungsbeispiel die E/E-Architektur in seine Elemente und Parameter zerlegt wird. In dieser Arbeit wurde bei der Belegung des morphologischen Kastens als Elemente das Betriebssystem, die Middleware, die Netzwerktechnologie, die Ebene der dynamischen Rekonfiguration sowie das verwendete Entwurfsmuster (vgl. Abschnitt 5.3.1) verwendet. Als Parameter kommen dann die einzelnen Lösungsbausteine zum Einsatz. Abbildung 5.22 zeigt den erarbeiteten morphologischen Kasten zur Darstellung verschiedener Lösungsszenarien. Die Parameter setzen sich aus der Bibliothek der Lösungsbausteine zusammen und können genauso wie die Bibliothek selbst, um zusätzliche Parameter erweitert werden.

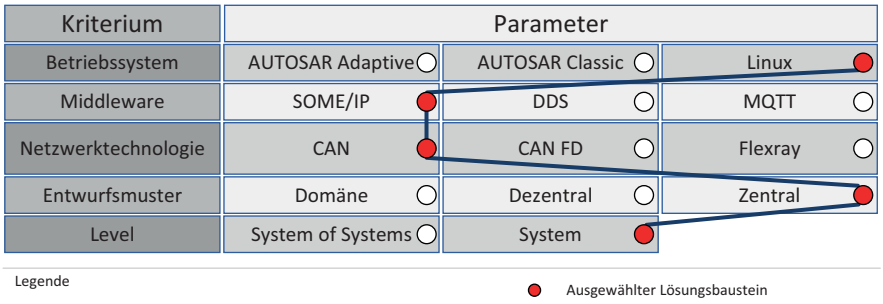

Abbildung 5.22: Morphologischer Kasten zur Entwicklung von Szenarien

In Abbildung 5.22 ist ein Szenario hervorgehoben, das folgende Parameter verwendet. Die Anwendung von Linux auf der Betriebssystemebene, SOME/IP als Middleware, CAN als Netzwerktechnologie, die Fahrzeugebene als Ebene, auf der die dynamische Rekonfiguration durchgeführt wird und der zentrale Entwurfsansatz. Dieses Szenario

[21] *altgriechisch: morphé*, Gestalt, Form, und *lógos*, Wort, Lehre, Vernunft, Sinn

wurde in dieser Arbeit prototypisch umgesetzt. Dessen Implementierung und Evaluierung folgt im Abschnitt 6.1.5.

5.5.3 Einordnung in den Entwicklungsprozess

Abschließend wird das gesamte Verfahren in den Standardentwicklungsprozess der Automobilindustrie integriert. Abbildung 5.23 zeigt die verwendete Bibliothek und die in den Entwicklungsprozess eingeordneten Methoden. Dabei gibt es eine enge Verbindung zwischen dem Vorgehen, das in Abschnitt 4.2 aufgezeigt wurde und dem V-Modell (vgl. Abschnitt 2.1.3).

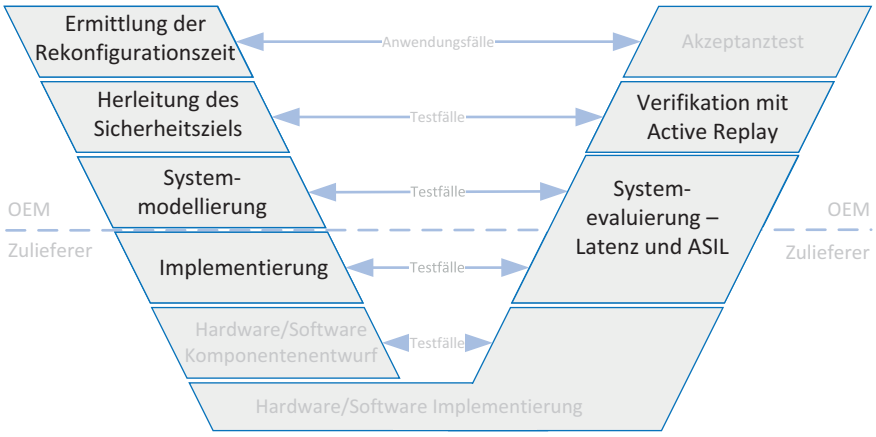

Abbildung 5.23: Einordnung in den Entwicklungsprozess

Bis jetzt wurden die Methoden in dieser Dissertation jeweils für sich betrachtet. Aufgrund der Situation, dass das V-Modell in weiten Teilen der Entwicklung von Systemen im Automobil nach wie vor den Standardprozess abbildet, werden im Folgenden die einzelnen verwendeten Methoden noch einmal zusammengefasst und in das V-Modell eingeordnet. Dabei wird auch herausgearbeitet, welche Neuheiten in dieser Dissertation elaboriert wurden und welche Vorteile sich daraus ergeben.

1. Als erstes ist die Erweiterung des **Ebenenkonzepts** um die Ebene System-of-Systems (vgl. Abschnitt 2.1.1) und die Verwendung im Kontext des Fail-Operational-Verhaltens zu nennen. Hieraus resultieren auch zwei Patente, die im

Rahmen dieser Arbeit entstanden sind [P3], [P6]. Diese Betrachtungsweise, eine dynamische Rekonfiguration über Fahrzeuggrenzen hinaus durchzuführen, wurde bisher nicht durchgeführt. Hieraus können Vorteile in der langfristigen Kostenreduktion der Systeme zum Einsatz in autonomen Fahrzeugen führen, die somit die Marktdurchdringung dieser gearteten Funktionen erhöhen.

2. Zu Beginn des Entwicklungsprozesses steht die Auswahl des **Entwurfsmusters** (vgl. Abschnitt 5.3.1) für die zu entwickelnde E/E-Architektur. Diese befindet sich im V-Modell ganz am Anfang. Diese Formalisierung des Prozesses ist so derzeit in der Automobilindustrie nicht vorhanden. Es liegen verschiedene Entwurfsmuster vor, jedoch sind diese historisch gewachsen. Eine bewusste, formalisierte Entscheidung unter Berücksichtigung der dynamischen Rekonfiguration wurde bis jetzt nicht vorgestellt und in den Entwicklungsprozess, bzw. das V-Modell mit integriert. Darüber hinaus hat diese Entscheidung Auswirkungen auf die weiteren Entwurfsprozesse insbesondere hinsichtlich der dynamischen Rekonfiguration für ein Fail-Operational-Verhalten. Die Berücksichtigung und die Integration in den Gesamtprozess wurden bisher nicht untersucht und sind eine Neuheit dieser Arbeit.

3. Für den nächsten Schritt im Entwicklungsprozess wurde eine Methode vorgestellt, um die **Rekonfigurationszeit** (vgl. Abschnitt 5.2.1 und Abschnitt 6.2) des Systems zu ermitteln. Hierfür wurde zunächst die Herleitung aus der ISO 26262 herbeigeführt, somit formal dargestellt und danach die experimentelle Ermittlung des Wertes aufgezeigt. Auch dieser Schritt ist eine Eingangsgröße für die in dieser Arbeit vorgestellte ganzheitliche Methodik. Für einen gesamthaften Prozess zur Entwicklung einer E/E-Architektur im Fahrzeug mit dynamischer Rekonfiguration im Fail-Operational-Kontext wurde dieser Schritt bis jetzt nicht in dieser Form beschrieben und stellt somit ebenfalls eine Neuheit dar. Vorteile ergeben sich hieraus in der erhöhten Zertifizierbarkeit und Homologationsfähigkeit von dynamischer Rekonfiguration im Automobil.

4. Als nächster Punkt wurde die **Modellierung** des Gesamtsystems vollzogen (vgl. Abschnitt 5.3 und Abschnitt 5.4). Hierbei wurde zum einen die serviceorientierte Architektur herangezogen und zum anderen die Fehlerbaumanalyse verwendet. Diese beiden Punkte für sich stellen keine Neuheit dar, sind allerdings wiederum im Bezug zum Gesamtsystem als Neuheit zu werten, weil die dynamische

Rekonfiguration und die Evaluierung im Fail-Operational-Kontext unter Zuhilfenahme der Modellierung einen Mehrwert bringen. Auf diese Weise kann die Anforderung aus der ISO 26262 bezüglich des Nachweises der Ausführungszeit durchgeführt werden. Insgesamt resultiert der Mehrwert der Modellierung in der langfristigen Kostenreduktion, der Unterstützung zur Zertifizierbarkeit und Homologationsfähigkeit, sowie der Beherrschbarkeit der Komplexität. Auch der Aufbau einer Bibliothek und der Einsatz wiederverwendbarer Teilumfänge, eine weitere Anforderung aus der ISO 26262, kann durch den Modellierungsansatz gewährleistet werden. Die Modellierung ist in diesem Fall ein Befähiger für die folgenden Punkte und bietet somit zusätzliche Vorteile.

5. Der Einsatz und die Bewertung von unterschiedlichen **Middleware-Konzepten** (vgl. Abschnitt 5.5.1) im Kontext des Fail-Operational-Verhaltens sind ebenfalls eine Neuheit. Die verschiedenen Middleware-Konzepte wurden zwar für sich jeweils im Kontext des Fail-Operational-Verhaltens betrachtet, eine Gegenüberstellung und somit der bewusste Einsatz als Lösungsbaustein für dynamisch rekonfigurierbare Systeme ist bis jetzt jedoch nicht erfolgt. Als Vorteil ergibt sich daraus ein modularer, parametrisierbarer und erweiterbarer Lösungsbaukasten für den Einsatz in automobilen Systemen.

6. Im Rahmen der Entwicklung stellt sich immer wieder die Herausforderung der **Evaluierung** der Systeme (vgl. Abschnitt 6.3 und Abschnitt 6.4) hinsichtlich unterschiedlicher Kriterien. Im Fail-Operational-Kontext sind diese Kriterien, die Latenz bezüglich des Echtzeitverhaltens und die Erreichung des Sicherheitsziels. Beide Umfängen müssen nach Möglichkeit bereits in einer frühen Entwurfsphase durchgeführt werden, um die Entwicklungskosten gering zu halten. Durch den Einsatz der simulativen und statischen Analyse des Echtzeitverhaltens wurde im Kontext der dynamischen Rekonfiguration ein Novum geschaffen. Alleinstehend sind diese Ansätze keine Neuheit, allerdings ist der Einsatz im Bereich der dynamischen Rekonfiguration und die Integration in die Bibliothek etwas Neues, das den Vorteil bietet, einen Standardprozess zur Verfügung zu stellen. Mit dem Einsatz der MCMC-Methode konnte für die Evaluierung zur Erreichung des Sicherheitsziels ein Verfahren zur Anwendung kommen, um die hochkomplexen Systeme basierend auf MPSoCs bewerten zu können.

7. Letztendlich ist die abschließende **Verifikation am Simulator** des dynamisch rekonfigurierbaren Systems eine Neuheit, die es ermöglicht ebenfalls durch einen simulativen Ansatz und die Verlagerung von Testmöglichkeiten in einer frühen Phase von der Straße in ein Labor, die Entwicklungskosten zu senken (vgl. Abschnitt 6.2.4). Gerade für das verwendete Beispiel des Steer-By-Wire-Ansatzes sind die hier gelegten Grundlagen von Vorteil.

In Kapitel 6 werden die in diesem Kapitel vorgestellten Methoden beispielhaft implementiert und die daraus resultierenden Ergebnisse diskutiert.

6 Anwendung und Evaluation der Methodik

In diesem Kapitel werden die in Kapitel 5 vorgestellten Vorgehensweisen und Entwürfe sowie die erzielten Musterbeispiele an ausgesuchten Anwendungsfällen gezeigt und bewertet. Die nachfolgend dargestellten Ergebnisse beruhen auf den eigenen Veröffentlichungen [144, 145, 151, 150, 154, 155].

6.1 Verwendeter Anwendungsfall für die dynamische Rekonfiguration

In diesem Abschnitt werden die einzelnen Umsetzungen zusammengesetzt. Zunächst wird dazu die Simplex-Architektur verwendet (Abschnitt 6.1.1), erweitert um eine ausfallsichere CAN-Kommunikation (Abschnitt 6.1.2) und der zugehörigen Kapselung (Abschnitt 6.1.3), die Integration des Algorithmus in einen bestehenden Simulator (Abschnitt 6.1.4), deren Implementierung mithilfe einer serviceorientierten Architektur (Abschnitt 6.1.5), sowie der Einsatz eines Fail-Operational-Moduls für die CAN-Kommunikation als Service (Abschnitt 6.1.6). Abschließend werden die Simulationsergebnisse diskutiert. Die folgenden Umfänge basieren auf einer Masterarbeit, die im Rahmen dieser Dissertation in Kooperation mit dem ITIV entstanden ist [96], des Weiteren ist aus dieser Arbeit auch ein Patent [P4] und eine Veröffentlichung extrahiert worden [151].

6.1.1 Vorbereitungen für die Cross-Layer-Rekonfiguration

In der Arbeit „Towards Fail-Operational Systems on Controller Level Using Heterogeneous Multicore SoC Architectures and Hardware Support“ [12] wurde eine erweiterte Simplex-Architektur vorgestellt. In dieser Arbeit wird die erweiterte Simplex-Architektur, die auf der LPD des MPSoCs eingesetzt wird als Lösungsbaustein für die

dynamische Rekonfiguration auf der Komponentenebene verwendet. Auf diese Weise kann die zusätzliche Redundanz für den R5-Kern, die in dem FPGA in Form eines MicroBlaze Soft-Core allokiert ist, als Redundanz verwendet werden. Hierbei wird in dieser Arbeit die CAN-Kommunikation für den Einsatz im Anwendungsfall herangezogen. Auf diese Weise kann ein Fail-Operational-Verhalten für die CAN-Kommunikation auf der Komponentenebene erreicht werden.

Dafür wird in Abschnitt 6.1.2 zunächst der Sequenzablauf geschildert, bei dem ein möglicher Statusübergabefehler identifiziert wurde und behoben wurde. In Abschnitt 6.1.3 wird die Implementierung dieser Funktionalität in die erweiterte Simplex-Architektur beschrieben. Abschnitt 6.1.6 beschreibt die Integration dieser Funktionalität als Service für eine CAN-Kommunikation mit Fail-Operational-Verhalten und der Einbindung in das SOME/IP-Konzept, das auch durch andere Middleware-Konzepte ersetzt werden kann.

Auf diese Weise wird ein Beispiel für die Anwendung der modularen Bibliothek (vgl. Abschnitt 4.2) auf der Ebene Komponente und System unter Einsatz der Cross-Layer-Rekonfiguration dargestellt. Dieses Zusammenspiel zwischen der HW-basierten erweiterten Simplex-Architektur auf der Komponentenebene und der Middleware-basierten Rekonfiguration auf der Systemebene ist eine Neuheit, die in dieser Dissertation zum ersten Mal vorgestellt wird.

6.1.2 Sequenzablauf der CAN-Kommunikation

Ein Teil eines Fail-Operational-Systems im Automobilbereich ist die CAN-Kommunikation. Der Ablauf dafür ergibt sich wie folgt: Das primäre System führt seine Funktionalität aus. Währenddessen ändert das System seinen Zustand. Zu diesem Zeitpunkt tritt ein Fehler auf, der zu einer Kontextumschaltung auf das Fallback-System führt. Das Fallback-System liest den zuletzt geschriebenen Status aus der Statusverwaltung. Dieser Status ist jedoch nicht der aktuellste, da der Kontextschalter das primäre System freigegeben hat, bevor der neueste Status in die Statusverwaltung geschrieben werden konnte.

Diese Situation führt zu einer inkohärenten Definition des Zustands. Ein Lockstep-Fehler tritt auf und es findet eine Kontextumschaltung statt, bevor das Primärsystem seinen Status in die Statusverwaltung schreiben kann. Daher ist es notwendig, sowohl

den Kontextwechsel als auch den Zustandsspeicher in einer atomaren Funktion zu kapseln. Abbildung 6.1 zeigt ein Szenario, bei dem während der Kontextumschaltung ein inkohärenter Zustand auftritt.

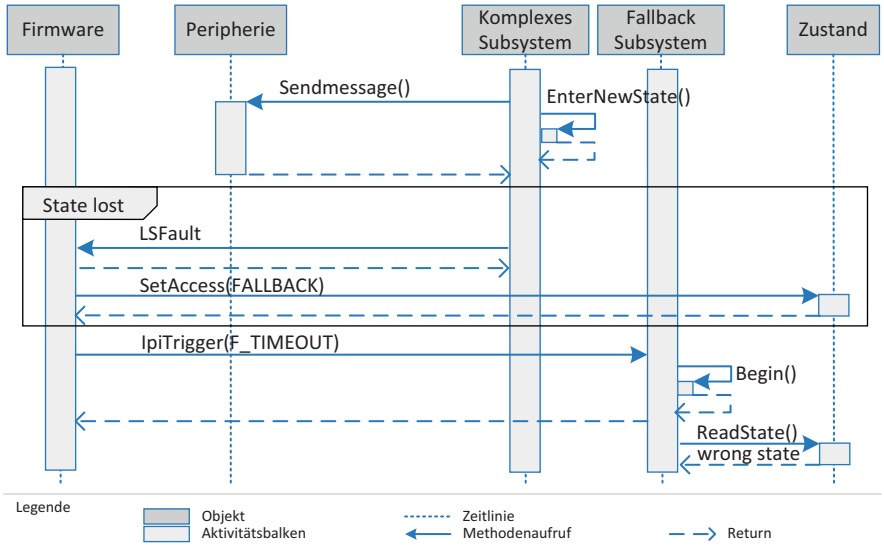

Abbildung 6.1: Sequenzdiagramm: Kontextwechsel ohne Sicherung des letzten Status nach [151]

Für diesen Fall konzentriert sich diese Arbeit auf die Kommunikation mit einer Peripheriekomponente, der CAN-Komponente. Das System, das nach einem Kontextwechsel aktiv ist, sollte die Information haben, ob und in welchem Zustand eine Kommunikation mit der Peripheriekomponente stattgefunden hat.

Das entwickelte Modul bietet daher die Möglichkeit der Kommunikation zwischen dem Peripheriegerät und dem Zustandsspeicher. Als Erstes müssen die Daten für die Kommunikation mit den beiden Befehlen *SendMessage*() und *SendState*() übertragen werden. Im zweiten Schritt wird die Ausführung der atomaren Funktion durch einen einzelnen Befehl *Ack*() ausgelöst. Beide werden dann unabhängig voneinander im Modul ausgeführt.

Der Ablauf des erweiterten Kontextwechsels ist in Abbildung 6.2 dargestellt. Die Abbildung zeigt, dass die Funktionalität von Kommunikation und Zustandsspeicherung auf die entwickelte atomare Funktion abgegeben wird.

Für den Fall, dass vor dem Auslösen des Acknowledge ein Lockstep-Fehler aufgetreten ist, wird keine Funktion ausgeführt. Wird die Bestätigung gesendet, wird sowohl die Kommunikation als auch die Zustandsspeicherung durchgeführt.

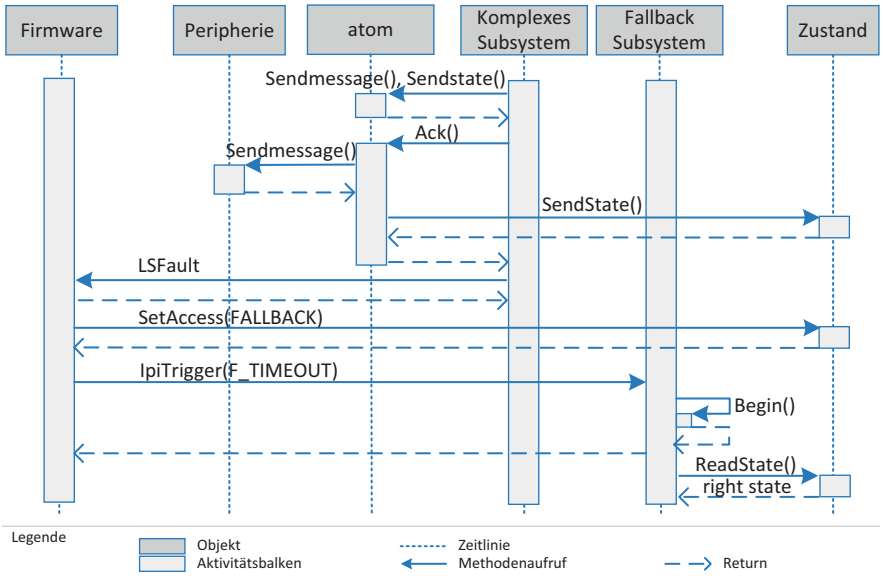

Abbildung 6.2: Sequenzdiagramm: Erweiterte Kontextumschaltung nach [151]

6.1.3 Hardware-Modul zur Kapselung der Kommunikation

In diesem Abschnitt wird die Integration des Moduls *atom* in die erweiterte Simplex-Architektur beschrieben. Dafür gibt es zwei Anforderungen:

1. der Zugriff sollte von anderen Master-Modulen auf das Modul möglich sein.
2. das Modul sollte auf andere Slave-Module zugreifen können, wie in Abbildung 6.3 dargestellt.

Um diese Anforderungen zu erfüllen, besteht das Modul *atom* selbst aus einer Master- (m_{atom}) und einer Slave-Einheit (s_{atom}). Das Primärsystem $m_{complex}$ sowie das Fallback-System $m_{fallback}$ müssen Schreibzugriff auf das Modul *atom* haben. Da $s_{protection}$ den Zugriff auf *atom* nicht regeln kann, muss der Zugriffsschutz von $s_{protection}$ selbst vorgenommen werden. Außerdem muss der Zugriffsschutz für die Statusverwaltung

s_{state} von s_{state} selbst erfolgen. Dafür wird *atom* vom aktiven System immer durch $m_{control}$ informiert. Außerdem muss *atom* Zugriff auf die Slave-Module haben, die das Peripheriegerät für die Kommunikation darstellen, um die entsprechenden Nachrichten zu übertragen. Für diesen Fall ist s_2 das Schnittstellenmodul zum Senden der Nachrichten. Das Modul m_{atom} kann darauf zugreifen.

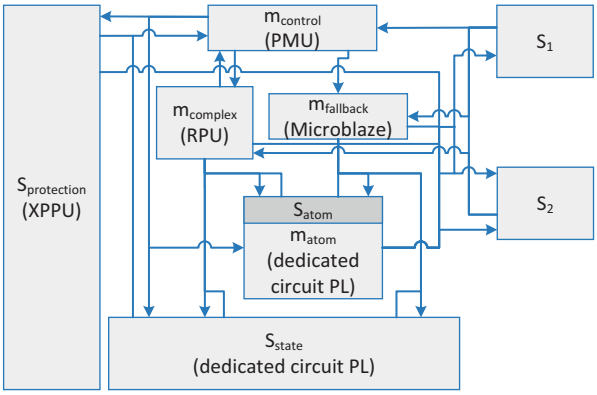

Abbildung 6.3: Logische Architektur der erweiterten Simplex-Architektur nach [151]

Bei der Entwicklung des Moduls *atom* werden Elemente für die Zustandsverwaltung wiederverwendet, da sie teilweise die gleiche Funktionalität erfüllen. Diese Prozedur ermöglicht die Implementierung des Zugriffs von $m_{complex}$ und $m_{fallback}$ auf *atom* über dieselbe Schnittstelle. Das Modul ist in der Hardware-Beschreibungssprache für VHDL geschrieben und läuft auf der PL.

6.1.4 Systemintegration der Gesamtsimulation

Der vorhandene Simulator und die Anbindung der Xilinx Plattformen über CAN wurde bereits in Abschnitt 5.1.4 beschrieben. Die Gründe für den Einsatz eines Simulators sind in Abschnitt 3.4 präsentiert. In Abschnitt 5.3.3 wurden die Services, über die der Simulator angesteuert wird, in Form des CAN- und des Trajektorien-Providers dargelegt. Diese kommen auf der APU der Xilinx Plattformen zum Einsatz. Abschnitt 5.3.2 beschreibt den IPC-CAN-Provider, der die Verbindung zwischen der HW und der SW umsetzt. Die Systemintegration der Gesamtsimulation ist in Abbildung 6.4 dargestellt.

Hier sind die einzelnen beschriebenen Komponenten zusammengesetzt, die Messungen eingezeichnet und die Ein- und Ausgangsartefakte skizziert.

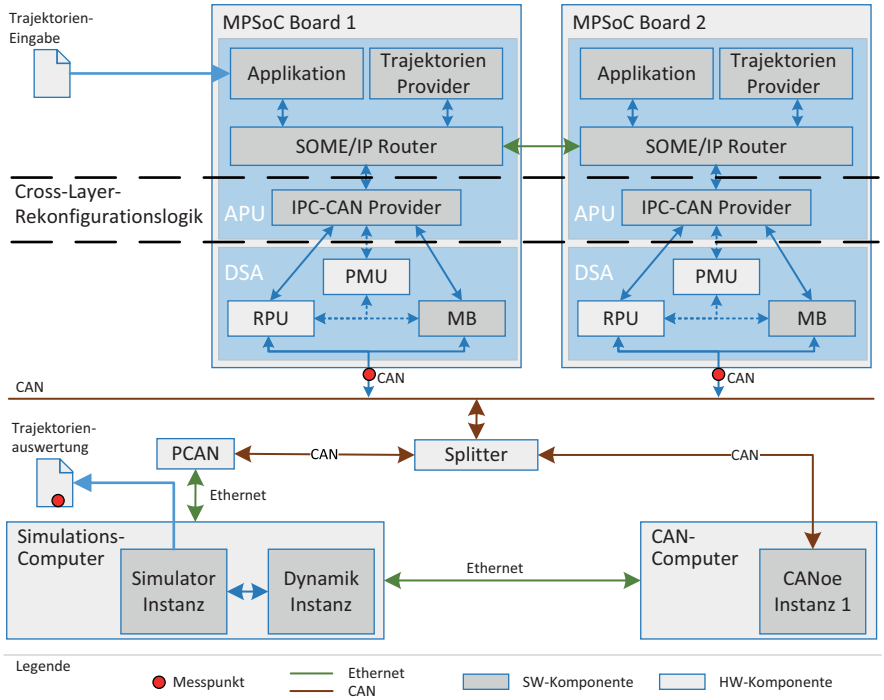

Abbildung 6.4: Systemintegration der Gesamtsimulation

Die erweiterte Simplex-Architektur wird zusammen mit dem entwickelten Hardware-Modul aus Abschnitt 6.1.2 über die in Abschnitt 5.3.2 beschriebenen Kanäle angebunden. Somit kann der praktische Einsatz für Funktionen mit Anforderungen an ein Fail-Operational-Verhalten demonstriert werden. Wie in „Dynamic Reconfiguration for Real-Time Automotive Embedded Systems in Fail-Operational Context“ [154] beschrieben, wird für dieses Experiment die Lenkfunktion verwendet. Das dabei zum Einsatz kommende Fahrszenario ist in Abschnitt 6.2 detailliert erläutert. Eine grundlegende Frage ist beispielsweise, wie sich ein Lockstep-Fehler auf die Lenkfunktionalität auswirkt. Durch Verwendung der ermittelten Rekonfigurationszeiten können die Messungen abschließend bewertet werden (vgl. Abschnitt 6.2.4).

6.1.5 Integration von serviceorientierten Architekturen

Um die aufgezeichneten CAN-Nachrichten des Simulators abzuspielen, wird die Systemarchitektur erweitert. Die Simplex-Architektur verwendete bisher nicht die APU auf dem MPSoC des Xilinx Zynq ZCU102-Boards. Die Integration der Simplex-Architektur und deren Verbindung als Service, der auf der APU implementiert ist, wurde bereits in Abschnitt 5.3.2 beschrieben. Darüber hinaus wird ein zweites MPSoC-Board hinzugefügt und damit eine weitere Fallback-Ebene generiert. Auf diese Weise kann anschaulich die Umsetzung der Rekonfiguration über zwei Plattformen hinweg gezeigt werden.

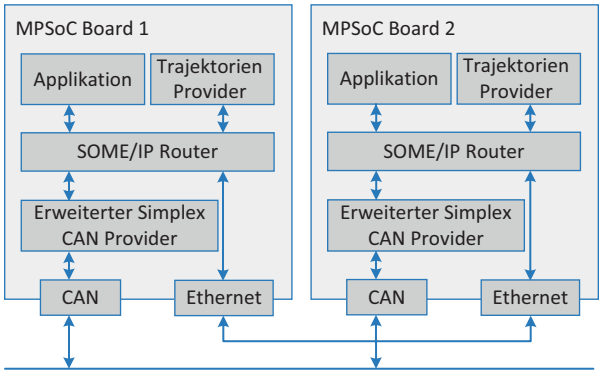

Abbildung 6.5: Serviceorientierte-Architektur auf Systemebene nach [151]

In Abbildung 6.5 ist die serviceorientierte Architektur mit dem Mapping auf die Hardware zu sehen. Bei einem Ausfall des CAN-Controllers auf dem Hauptpfad kann das Board auf den CAN-Controller des zweiten Boards zugreifen und dessen Peripheriegeräte verwenden. Auf diese Weise kann eine Fail-Operational-Funktionalität der CAN-Kommunikation bereitgestellt werden. Aus Sicht der APU wird dieser Umfang als Service bereitgestellt. Wenn auf einem Board ein Fehler auftritt, kann der von dem anderen Board bereitgestellte Service verwendet werden. Das Verfahren hierzu basiert auf dem Service-Discovery und dem Service-Subscription, das in Abschnitt 5.3.3 beschrieben ist. Erhält die Middleware das Signal Stop-Offering, bzw. beobachtet das Monitoring den Entfall des Service, schaltet die Rekonfigurationslogik auf die Rückfallebene für den CAN-Provider um, in diesem Fall CAN_PR_1, der auf dem zweiten MPSoC-Board implementiert ist. Die Service-ID ist in diesem Fall zwischen dem CAN_PR_0 und CAN_PR_1 identisch, die Applikations-ID und die Instanz-ID unterscheiden sich jedoch. Auf diese Weise ist der Service für den Consumer nicht zu unterscheiden, die

Rekonfigurationslogik kann jedoch genau bestimmen, auf welche Instanz des Providers umgeschaltet werden muss (vgl. Tabelle 5.9).

6.1.6 Fail-Operational-Modul als Service

Die Simplex-Architektur stellt zusammen mit dem Hardware-Modul einen Service zum Senden von CAN-Nachrichten und zum Identifizieren eines Systemzustands bereit. In Abbildung 6.6 ist der prinzipielle Ablauf der Service-Kommunikation dargestellt.

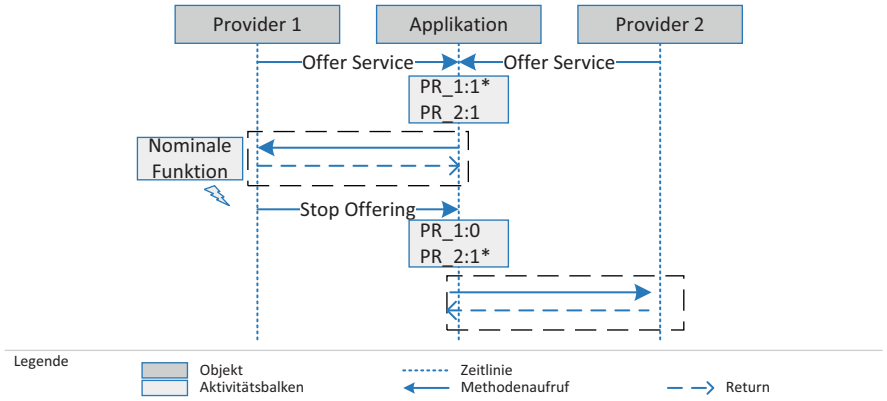

Abbildung 6.6: Sequenzdiagramm: dynamische Rekonfiguration, aufbauend auf einer serviceorientierten Architektur nach [151]

Wenn beim Senden einer CAN-Nachricht ein Fehler auftritt, wird die APU über den Ausfall des Controllers und seinen aktuellen Status informiert. Dieser Umfang ist so implementiert, dass auf einen Acknowledge-Fehler reagiert wird. Dieser Fehler kann erzwungen werden, indem lediglich der CAN-Bus getrennt wird. Während der Laufzeit oder beim Systemstart ist es möglich, den Kontextstatus durch Ausgeben eines Befehls abzurufen. Darüber hinaus bietet die Architektur die Möglichkeit, zu Testzwecken einen Lockstep-Fehler der RPU auszulösen.

Zunächst wird die Kommunikationssequenz zwischen der APU und der Simplex-Architektur definiert:

1. Der Absender einer Nachricht schreibt Daten, die gesendet werden müssen, in den Nachrichtenpuffer.

2. Der Sender löst über das Trigger-Register einen Interrupt des Empfängers aus.
3. Der Empfänger erhält den Interrupt und liest den entsprechenden Nachrichtenpuffer.
4. Der Empfänger führt den Service aus.
5. Der Empfänger bestätigt den Interrupt durch Zurücksetzen des Statusregisters.

Diese Schritte 1 bis 5 repräsentieren die Sequenz zum Senden von CAN-Nachrichten. Je nach Status der erweiterten Simplex-Architektur wird der Service von der RPU oder von dem MB angefordert. Beim Empfang wird die CAN-Nachricht an die APU gesendet.

In der Firmware der PMU wurden folgende Funktionen der Simplex-Firmware als Modul implementiert, zu Testzwecken wurde eine vereinfachte Version mit folgenden Funktionen verwendet:

- Ein Lockstep-Fehler der RPU kann erkannt werden und führt zu einer Kontextumschaltung auf das Fallback-System.
- Die Kontextumschaltung enthält das Zurücksetzen des primären Systems, das Zurückschalten wird nicht unterstützt.
- Abhängig vom aktiven Kontext wird das Zugriffsrecht über die XPPU erteilt.

Neben dem Service zum Versenden von CAN-Nachrichten, wurden weitere Services implementiert. Einer dieser zusätzlichen Services stellt ein QoS-Konzept bereit, das im Rahmen dieser Arbeit entworfen wurde.

Auf Basis des jeweiligen Status können die entsprechenden Fallback-Systeme aktiviert werden oder ggf. andere Maßnahmen ergriffen werden, wie zum Beispiel die Benachrichtigung des Fahrers.

Bei dem Service zur Feststellung des QoS sendet das Fallback-Subsystem dem PMU-Subsystem bei Änderungen seinen Status. Diese Information wird über einen Benachrichtigungs-Service zur Verfügung gestellt.

Das PMU-Subsystem verwendet die entsprechenden Änderungsmeldungen, um auf Veränderungen des QoS-Levels zu reagieren. Die einzelnen Stufen des QoS-Konzepts sind in Tabelle 6.1 beschrieben. Ein Beispiel für das zugehörige Klassendiagramm ist in Abbildung 6.7 dargestellt.

Das APU-Subsystem nutzt die Information über die Änderungen der Servicequalität, um eine Rekonfigurationsstrategie abzuleiten. Die PMU benachrichtigt das System über den Integritätsstatus des Fallback-Systems.

Tabelle 6.1: Quality of Service-Level für die dynamische Rekonfiguration im ausfallsicheren Kontext

Level	Beschreibung
6	volle Funktionalität
5	das Fallback-System ist aufgrund eines Fehlers im Primärsystem aktiv
4	das primäre System wird zurückgesetzt
3	das primäre System ist betriebsbereit, um nach dem Zurücksetzen vom Fallback-System zurückzuschalten
2	Fehler im Voter des Fallback-Systems, verringerte Zuverlässigkeit
1	Ausfall des Fallback-Systems

In diesem Fall wird die APU über die Servicequalität informiert. Konkret werden die Informationen über den QoS aus Tabelle 6.1 als Entscheidungskriterium verwendet, ob eine Funktionalität auf dem anderen Subsystem dynamisch rekonfiguriert werden kann.

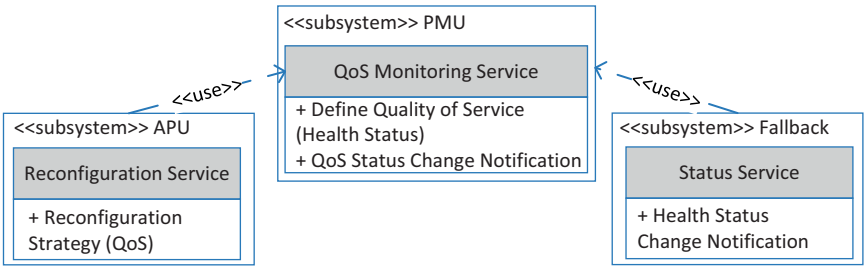

Abbildung 6.7: Klassendiagramm: QoS-Umfänge im Zusammenspiel mit der dynamischen Rekonfiguration nach [151]

In der aktuellen Implementierung wurde dies für Status 5 und 6 nur ansatzweise durchgeführt. Für zukünftige Systeme ist dieser Integritätsstatus ein wichtiger Teil, der ein fehlerfreies Betriebsverhalten garantiert.

6.1.7 Simulationsergebnisse und Diskussion

Leistungsmessung des CAN-Moduls

Das Modul *atom* ist so konzipiert, dass der Zustandsspeicher und die Kommunikation mit dem CAN-Controller in einer atomaren Funktion zusammengefasst sind. Die Abbildung 6.3 zeigt, dass der Pfad zum Senden einer CAN-Nachricht vom aktiven Modul über das Modul *atom* zum CAN-Controller führt. Dies ist ein Umweg gegenüber der direkten Kommunikation. Daher ist zu prüfen, ob dies den maximalen Durchsatz von CAN-Nachrichten einschränkt.

Aus diesem Grund wurde ein Test durchgeführt, der zunächst CAN-Nachrichten direkt von der RPU an den CAN-Controller und dann über eine Instanz von *atom* sendet. Dabei wird nur die Zeit für die Übertragung der Nachrichten an die Steuerung gemessen und nicht die Übertragungszeit auf dem CAN-Bus. Für die Messung wird einer der vier verfügbaren TTC[1] mit einer Taktfrequenz von $100\,\mathrm{MHz}$ verwendet.

Die Übertragung von 60 Nachrichten ohne das Modul benötigt 6.481 Ticks, dies entspricht $64.81\,\mu\mathrm{s}$. Eine Nachricht besteht dabei aus vier Wörtern mit einer Breite von $32\,\mathrm{Bit}$. Mit dem Hardware-Modul *atom* benötigt die Übertragung hingegen 11.994 Ticks oder $119.94\,\mu\mathrm{s}$. Zwischen diesen beiden Ergebnissen liegt in etwa der Faktor 2.

Tabelle 6.2: Messung des Durchsatzes für die CAN-Kommunikation

	Durchsatz 60 Nachrichten		**max-Durchsatz**
	[Ticks]	**[µs]**	**[Nachrichten/s]**
ohne Modul *atom*	6481	64.81	
mit Modul *atom*	1199	119.94	500 000
Theoretische Obergrenze CAN-Bus			21 276

Mit dem Modul *atom* kann ein Durchsatz von ca. 500.000 Nachrichten/s erreicht werden. Die gesendeten Nachrichten des CAN-Controllers bestehen aus mindestens $47\,\mathrm{Bit}$. Dies ist der Fall, wenn die Nachrichten nur den 11-Bit-Identifier und keine Daten enthalten. Für CAN wird die übliche Übertragungsrate von $1\,\mathrm{MBit\,s^{-1}}$ angenommen. Dies ergibt eine Übertragungsrate von 21.276 Nachrichten/s. Der Durchsatz mit dem Modul *atom*

[1] Triple Timer Counter

liegt damit über dem theoretisch möglichen Durchsatz eines CAN-Controllers. Durch die Verwendung des Moduls *atom* werden keine Einschränkungen der maximalen Datenübertragungsrate des CAN-Busses erwartet. In Tabelle 6.2 sind diese Ergebnisse zusammengefasst.

Neben dem Durchsatz ist die Latenz ein wichtiges Attribut des Moduls. Daher wird dieses Merkmal mit demselben Timer gemessen wie im vorhergehenden Experiment. Eine CAN-Nachricht wird für die Messung einmal direkt an den CAN-Controller und das zweite Mal über das Modul *atom* gesendet. Für das direkte Senden wird die vom Xilinx-Treiber zur Verfügung gestellte Funktion *XCanPs_Send* verwendet. Diese Funktion bietet neben dem Schreiben von vier $32\,\mathrm{Bit}$-Wortnachrichten auch weitere Funktionen an. Aufgrund dieser zusätzlichen Funktionen treten einige Ungenauigkeiten auf: Da ausschließlich ein Vergleich zwischen dem Senden einer Nachricht mit und ohne das Modul *atom* durchgeführt werden muss, können die zusätzlichen Umfänge vernachlässigt werden.

Tabelle 6.3: Messung der Latenz für die CAN-Kommunikation

	∅-Latenz	
	[Ticks]	**[s]**
ohne Modul *atom*	155	$1.55 \cdot 10^{-6}$
mit Modul *atom*	284	$2.84 \cdot 10^{-6}$
Anforderung Rekonfigurationszeit		$100 \cdot 10^{-3}$

Die Messung mit dem Modul *atom* erfolgt mit der Funktion *AtomManager_SendCan*. Ihre Aufgabe ist es, zunächst die Verfügbarkeit des Moduls zu prüfen. Anschließend werden die vier $32\,\mathrm{Bit}$-Wortnachrichten und optional ein Zustand in das Modul geschrieben. In diesem Experiment besteht der Status aus einer $32\,\mathrm{Bit}$-Wortnachricht. Die Messung endet, nachdem das Modul seine Verfügbarkeit erneut bestätigt hat. Das Ergebnis als Durchschnitt aus 50 Messungen zeigt, dass die benötigte Zeit zum Senden der angegebenen Nachricht ohne das Modul 155 Ticks beträgt, das entspricht $1.55\,\mu\mathrm{s}$. Wird das Modul *atom* verwendet, werden 284 Ticks benötigt, das wiederum $2.84\,\mu\mathrm{s}$ entspricht. Bezogen auf die Gesamtrekonfigurationszeit fällt diese Zeit jedoch nicht ins Gewicht. Diese Ergebnisse sind in Tabelle 6.3 zusammengefasst.

Messung der Umschaltzeiten auf Komponenten- und Systemebene

Auf Komponentenebene wurden Messungen bereits in einer anderen Arbeit durchgeführt, an die in dieser Arbeit angeknüpft wird [12]. Das entwickelte Hardware-Modul für die CAN-Kommunikation wurde jedoch nicht berücksichtigt. Weiterhin wurde die Messung mit einer speziellen Firmware ohne APU durchgeführt. Da die APU für die serviceorientierte Architektur eingesetzt und Linux auf der APU ausgeführt wird, kommt eine andere Firmware-Version zur Anwendung. Diese wird beispielsweise um die Energieverwaltungsfunktion ergänzt. Dies kann zu zusätzlichen Ausführungszeiten führen, die identifiziert werden sollen.

In diesem Testaufbau wird die Möglichkeit der PMU genutzt, einen Lockstep-Fehler in die RPU einzubringen. Aus diesem Grund wird ein Programm verwendet, das auf der APU ausgeführt wird und diesen Befehl an die PMU sendet. Die PMU startet die Zeitmessung, indem sie den aktuellen Wert eines Timers speichert. Anschließend wird der Wert zum Einfügen des Lockstep-Fehlers in das Injektionsregister der RPU geschrieben. Jetzt wird die PMU erneut über den ausgelösten Lockstep-Fehler in der RPU informiert. Dadurch startet die PMU das Fallback-System und wartet auf eine Antwort. Alle diese Informationen werden über IPI[2] übertragen. Nach erfolgreicher Kontextumschaltung auf das Fallback-System liest die PMU erneut den Wert des Timers ein und die Messung ist beendet. Die Differenz dieser beiden Zeitgeberwerte stellt die Kontextumschaltung vom Primär-System zum Fallback-System dar. Das Ergebnis dieser Messung ist die Dauer für den Kontextwechsel von $t_1 = 89\,\mu\mathrm{s}$.

Tabelle 6.4: Zusammenfassung der ermittelten Umschaltzeiten

Abstraktionsebene	**Aktion**	**gemessene Zeit** [s]
Komponente	context-switch t_1 :	$89 \cdot 10^{-6}$
System	fail-over t_2 :	$1.535 \cdot 10^{-3}$
Anforderung	Rekonfigurationszeit t_3 :	$100 \cdot 10^{-3}$

In der Tabelle 6.4 ist eine Übersicht der relevanten Messzeiten dargestellt. Nachdem die Messungen der Umschaltzeit auf Komponentenebene durchgeführt wurden, konnte die

[2] Inter Processor Interrupt

Umschaltzeit auf Systemebene zusammen mit der serviceorientierten Architektur ermittelt werden. Die beiden MPSoC-Boards sind über Ethernet miteinander verbunden. Auf beiden Boards wird die Simplex-Architektur verwendet, um Acknowledge-Fehler der CAN-Controller zu erkennen.

Die Acknowledge-Fehler werden an einen Hardware-Pin auf dem Board gesendet. Dieser Hardware-Pin wechselt nach dem Auftreten eines Acknowledge-Fehlers auf eine logische Eins. Für die Messungen werden CAN-Nachrichten zyklisch in einem Zeitintervall mit der Differenz $\Delta t = 1\,\mathrm{ms}$ gesendet. Der CAN-Stecker wird zu einem beliebigen Zeitpunkt von dem aktiven Board abgezogen und die Trennung führt zu einem Acknowldege-Fehler. Das hat zur Folge, dass sich der Status des Hardware-Pins auf diesem Board zu einer logischen Eins ändert. Zusätzlich werden die Informationen über den Fehler an die APU gesendet. Die APU leitet daraufhin die Umschaltung auf das zweite MPSoC-Board ein. Dies führt dazu, dass die CAN-Kommunikation an das zweite MPSoC-Board übergeben wird. Bei diesem serviceorientierten Ansatz werden die nächsten CAN-Nachrichten nach dem erkannten Acknowledge-Fehler direkt von dem zweiten Board gesendet. Die gemessene Zeit zwischen dem Erkennen des Fehlers und dem erfolgreichen Senden der ersten CAN-Nachricht von dem zweiten Board nach dem Umschalten dauert $t_2 = 1.535\,\mathrm{ms}$.

6.2 Methodik zur Ermittlung der Rekonfigurationszeiten

In diesem Abschnitt werden zwei neue Simulationstechniken vorgestellt, mit denen zwei wichtige Werte für die Rekonfiguration extrahiert werden können. Der Ansatz basiert auf einem Beispiel für den Verlust der Lenksteuerungsfunktionalität. Der erste Wert ist die gesamte Systemrekonfigurationszeit, die vom menschlichen Fahrer im Falle eines Verlusts der Lenksteuerungsfunktionalität wahrgenommen wird (Abschnitt 6.2.1). Der zweite Wert ist die maximal mögliche Rekonfigurationszeit, um ein bestimmtes Manöver auszuführen (Abschnitt 6.2.2). In Abbildung 6.8 sind die Timing-Evaluationen im Zusammenhang mit der Anforderungsanalyse dargestellt.

Um die relevanten Werte zu erhalten und die Simulationen mit der realen Welt zu verknüpfen, wird eine Simulation aufgebaut. Mit dieser Simulation wird ein Beispiel-

manöver gefahren. Im Folgenden wird dargelegt, wie die technische Ausgangsbasis erreicht wird, um den Simulationsaufbau so zu manipulieren, dass der Fahrer unterschiedliche Rekonfigurationszeiten wahrnimmt. Weiterhin wird beschrieben, wie die technischen Voraussetzungen ausgelegt sind, um die Rekonfigurationszeiten zu erfassen und eine Grenzfunktion zu ermitteln. Zusammenfassend werden in diesem Abschnitt zwei Simulationstechniken für die Lenkfunktion dargestellt, um auf das FTTI von Ende-zu-Ende zu schließen und somit eine Entwurfsvorgabe zu erhalten.

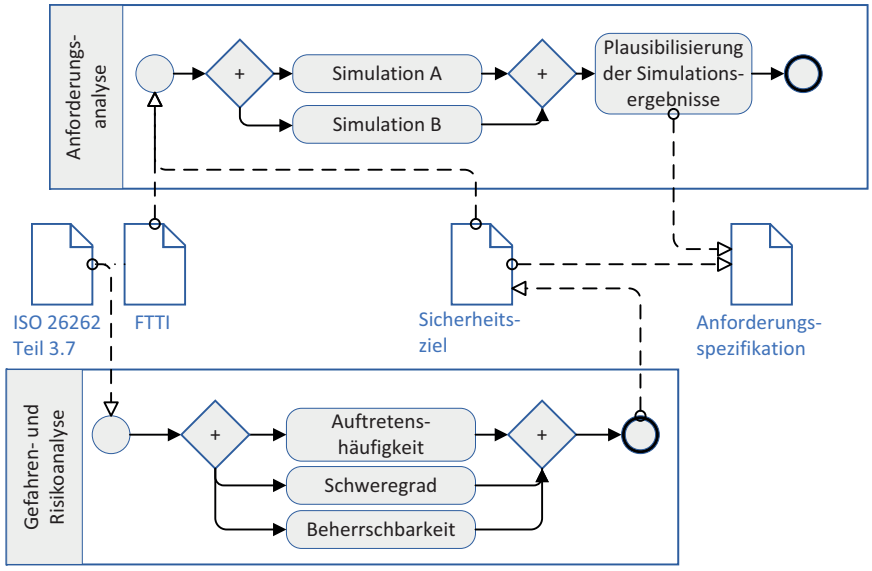

Abbildung 6.8: Gesamtbild der Timing-Evaluationen

Ende-zu-Ende bedeutet in diesem Fall, wie viel Zeit dem Gesamtsystem zur Verfügung steht, um den Fehler zu erkennen, auf diesen Fehler zu reagieren und das System neu zu konfigurieren. Auf diese Weise wird das System in einen sicheren Zustand versetzt. Die Simulationsmethoden werden auf Systemebene angewendet, d.h. es handelt sich um das gesamte Spektrum der Steuergeräteklassen. Die Integrationseinheit als Steuergerät der Klasse 1 sowie die Aktoren als Steuergerät der Klasse 3 werden angesprochen. Als Kundenfunktion wird die Lenksteuerfunktion ausgewählt, die simuliert werden soll. Der Simulator ist ein statischer Simulator mit einem Force-Feedback-Lenkrad. Das Lenkrad ist über CAN mit dem CAN-Computer verbunden. Auf diesem Computer werden die CAN-Signale manipuliert und dann an die Fahrdynamik des Simulationsrechners übertragen. Darüber hinaus ist es mit dieser Konfiguration möglich, eine Fahrt

in einem virtuellen Szenario aufzuzeichnen und wiederzugeben. Der beschriebene Aufbau ist in Abbildung 5.3 dargestellt.

Das Hauptziel dieser Simulationen ist es, die maximale Ausschaltzeit t_{off} zu ermitteln. In diesem Fall ist t_{off} eine Zeitspanne und wird mithilfe der FTTI identifiziert. Die Ausschaltzeit t_{off} ist damit ein Parameter, der in der grafischen CAN-Benutzeroberfläche auf Werte zwischen $t_{off} = 100\,\mathrm{ms}$ bis $t_{off} = 5000\,\mathrm{ms}$ mit einer Schrittgröße von $\Delta t_{off} = 1\,\mathrm{ms}$ eingestellt werden kann.

6.2.1 Simulationsmethode A – Fahrerlebnis

Die erste Simulationsmethode besteht darin, die gesamte Systemrekonfigurationszeit zu identifizieren, die von einem menschlichen Fahrer wahrgenommen werden kann. Fünf Fahrer sollen eine Schätzung durchführen, um einen Eindruck davon zu gewinnen, welche Zeitspanne ein Mensch bei einem Funktionsausfall wahrnimmt. Ein Versuchsleiter kann über eine Benutzeroberfläche den Ausfall auslösen. Die Simulation konzentriert sich auf einen Fehler, bei dem die maximale Lenkung während der Ausschaltzeit t_{off} stattfindet. Die Fahrerkandidaten sollen versuchen, das Manöver „doppelter Spurwechsel“ erfolgreich abzuschließen.

Hierbei wird der Fahrer auf das Fehlerereignis auf eine Weise aufmerksam gemacht, dass es dem realen Szenario so nahe wie möglich kommt. In einem realen Szenario wäre die Lenkradunterstützung also nicht mehr verfügbar. Dies führt zu einer Lenkung, die schwerer zu betätigen ist. Durch Einstellen der entsprechenden Parameter der verwendeten Lenkrad-Hardware kann dieses Ziel gut angenähert werden.

Der Aufbau der Simulation ist folgendermaßen gestaltet: Im virtuellen Szenario wird das in der Abbildung 6.9 dargestellte Manöver des doppelten Spurwechsels durchgeführt. Zunächst absolviert der Fahrer die Strecke ohne einen Ausfall. Vor dem zweiten Durchgang wird die Ausschaltzeit von dem Versuchsleiter auf $t_{off} = 100\,\mathrm{ms}$ gesetzt und der Fahrer muss innerhalb der Pylonengasse bleiben. Dabei ist eine Berührung mit den Pylonen zu vermeiden. Hat der Fahrer keine Störungen im Lenkverhalten des Fahrzeugs bemerkt, wird die Probefahrt wiederholt und die Ausschaltzeit t_{off} um $\Delta t_{off} = 50\,\mathrm{ms}$ vom Versuchsleiter erhöht. Dies wird fortgesetzt, bis der Fahrer eine Störung bemerkt.

6.2.2 Simulationsmethode B – Simulation in the Loop

Die zweite Methode wird verwendet, um das FTTI eines simulierten Fahrzeugs abzuleiten. Zu diesem Zweck wird ein Fehlerszenario während des Fahrens des in Abbildung 6.9 dargestellten Manövers betrachtet.

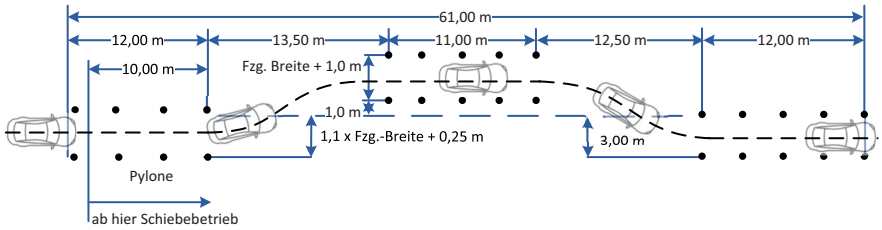

Abbildung 6.9: Doppelter Spurwechsel nach ISO 3888 Teil 2 nach [74]

Für das Fehlerszenario werden zwei verschiedene Fehler gewählt:

1. Der erste Fehler stellt eine Lenkunterstützung dar, die den Lenkwinkel feststellt. Eine Bewegung des Lenkrads hat dann keine Auswirkung auf das tatsächliche Lenkverhalten des Fahrzeugs.

2. Der zweite Fehler besteht darin, dass während der Ausschaltzeit der Lenkunterstützung ein beliebiger Lenkradwinkel an das System übergeben werden kann. Auf diese Weise kann der kritischste Lenkwinkel angewendet werden.

Um eine Einstellgröße für den zweiten Fehler zu ermitteln, wird angenommen, dass der Fahrer in der Lage ist, zweieinhalb Umdrehungen des Lenkrads innerhalb von

$$t_{turn} = 2000\,\mathrm{ms} \tag{6.1}$$

zu lenken. Der Gesamtlenkradwinkel ergibt sich aus:

$$\varphi_{total} = 360° \cdot 2.5 = 900° \tag{6.2}$$

Mit der Definition der Lenkradwinkelgeschwindigkeit

$$v_s = \frac{\delta\varphi}{\delta t} \tag{6.3}$$

und der Verwendung von φ_{total} und t_{turn} führt dies zu

$$v_s = \frac{\delta\varphi_{total}}{t_{turn}} = 0.45\,°/\mathrm{ms} \tag{6.4}$$

Dieser Wert wird als maximal möglicher Wert verwendet, da der Lenkwinkel in einem realen Szenario keine Sprungfunktion sein kann. Damit kann folgende lineare Gleichung definiert werden:

$$\Delta\varphi(t) = v_s \cdot t \tag{6.5}$$

Sobald das Fehlerereignis eintritt, kann berechnet werden, um welchen Betrag der Lenkradwinkel während der Ausschaltzeit abweichen wird. Durch Addition oder Subtraktion dieser Abweichung zur Lenkwinkelkurve, resultiert daraus das in Abbildung 6.11 gezeigte Ergebnis. Da der Lenkwinkel keine Sprungfunktion sein kann, muss auf die ursprüngliche Lenkwinkelkurve zurückgelenkt werden. Dies geschieht mit derselben maximalen v_s, jedoch mit einem entgegengesetzten Vorzeichen.

Im zweiten linearen Teil wird davon ausgegangen, dass das System wieder betriebsbereit ist. Hier ist eine Lenkung möglich. In der Praxis würde das System in wenigen Millisekunden einen Fehler feststellen. Gemäß der Tabelle 6.5 wird dieser Wert mit $30\,\mathrm{ms}$ angenommen. Der Lenkwinkel weicht demnach wesentlich weniger ab. Basierend auf diesen Fehlerszenarien verlässt das simulierte Fahrzeug, das mit einer Geschwindigkeit von $50\,\mathrm{km/h}$ fährt, die primäre Trajektorie. Auf dieser Grundlage wird die Zeit ab dem Aufrufen des Ausfalls und dem anschließenden Zusammenstoß des Fahrzeugs mit den Pylonen bestimmt, die das maximale FTTI ergibt.

6.2.3 Timing-Evaluationen für die dynamische Rekonfiguration

Als Nächstes werden die Anforderungen für die Rekonfiguration der Brems- und Lenksteuerfunktion abgeleitet. In dieser Hinsicht wird ein formales Modell für die zeitlichen Einschränkungen in Bezug auf die Rekonfiguration betrachtet. In diesem Modell bezieht sich das FTTI auf die FRT[3]. Die FRT besteht aus der Fehlererkennungszeit und der Rekonfigurationszeit. Für eine erfolgreiche Neukonfiguration darf die FRT nicht größer als das FTTI sein (vgl. Abschnitt 5.2.1). Beispielhaft sind zwei Fehlerereignisse

[3] Fault Reaction Time

aufgelistet. Zum einen der Verlust der primären Lenkfunktion und zum anderen der Verlust der primären Bremsfunktion.

Tabelle 6.5: Beispiel für Rekonfigurationsanforderungen: Fehlererkennung und Rekonfigurationszeit

Failure Event	**Failure Detection Time** [ms]	**Reconfiguration Time** [ms]
Verlust der primären Lenkfunktion	30	100
Verlust der primären Bremsfunktion	30	130

In der Tabelle 6.5 werden die Werte für die Fehlererkennungszeit und die Rekonfigurationszeit veranschaulicht. Die Fehlererkennungszeit wird jeweils mit $t = 30\,\mathrm{ms}$ angenommen, die maximale Rekonfigurationszeit ergibt sich bei der Lenkfunktion zu $t = 100\,\mathrm{ms}$ und bei der Bremsfunktion zu $t = 130\,\mathrm{ms}$.

6.2.4 Simulationsergebnisse und Diskussion

Ergebnisse der Simulationsmethode A

Die Simulationsmethode A wurde mit fünf Fahrern durchgeführt. Während der Ausschaltzeit wird eine Lenkgeschwindigkeit von $v_s = 1\,^\circ/\mathrm{ms}$ in der entgegengesetzten Lenkrichtung angegeben. Die Ergebnisse befinden sich in Tabelle 6.6.

Tabelle 6.6: Ergebnisse aus der Simulationsmethode A

Ausfallzeit [ms]	**Fahrer #1-5**
100	bemerkbar; Fahrversuch erfolgreich
150	bemerkbar; Fahrversuch erfolgreich
200	bemerkbar; Fahrversuch noch erfolgreich
225	bemerkbar; Fahrversuch gerade noch erfolgreich
250	bemerkbar; Fahrversuch nicht erfolgreich
300	bemerkbar; Fahrversuch nicht erfolgreich

Die Durchführung der Simulationsmethode A zeigt auch, dass es wichtig ist, an welcher Stelle des Kurses der Ausfall eingeleitet wird. Daher bemerken die Fahrer das Versagen nicht so intensiv an einer Stelle mit starken Lenkbewegungen wie einen Ausfall bei langsamen Lenkbewegungen. Bis $t_{off} = 200\,\mathrm{ms}$ können die Fahrer den Kurs noch

durchführen, aber mit $t_{off} = 250\,\text{ms}$ können sie den Kurs nicht mehr erfolgreich abschließen.

Ergebnisse der Simulationsmethode B

Für die bereits beschriebene Simulationsmethode B wurden Messungen aufgezeichnet und jeweils mit drei verschiedenen Ausschaltzeiten manipuliert, dargestellt in Tabelle 6.7:

Tabelle 6.7: Verwendete Ausfallzeiten für Simulationsmethode B

Versuch #	Ausfallzeit t_{off} [ms]
1	100
2	130
3	140

Diese drei unterschiedlichen Ausfallzeiten wurden in der Simulation für den Lenkausfall verwendet. Dabei wurden die verschiedenen Trajektorien aufgezeichnet. Anschließend wurde eine Visualisierung der Daten vorgenommen, um eine Auswertung durchführen zu können. Durch einen Vergleich der aufgezeichneten Fahrspuren, der Fahrzeugbreite und dem Fahrszenario kann die maximale Ausfallzeit ermittelt werden, bei der das Manöver ohne Kollision erfolgreich absolviert werden kann.

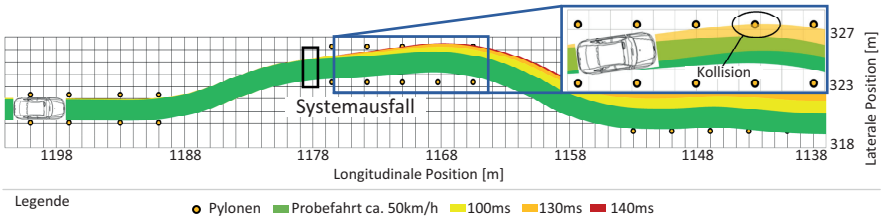

Abbildung 6.10: Ausweichmanöver mit einer Falschlenkung von $t_{off} = 100\,\text{ms}$, $t_{off} = 130\,\text{ms}$ und $t_{off} = 140\,\text{ms}$ nach [155]

In Abbildung 6.10 sind die Trajektorien für die jeweilige Ausfallzeit dargestellt. Zu erkennen ist, dass mit einer Ausfallzeit von $t_{off_3} = 140\,\text{ms}$ eine Kollision mit den Pylonen auftritt, während mit einer Ausfallzeit von $t_{off_1} = 100\,\text{ms}$ und $t_{off_2} = 130\,\text{ms}$

keine Kollisionen verzeichnet werden. Daraus ergibt sich eine obere Abschätzung der maximalen FTTI mit $t_{off_2} = 130\,\mathrm{ms}$.

Die Stärke der grünen Linie entspricht der Breite eines $1.9\,\mathrm{m}$ breiten Autos. Während das Lenkrad um $900°$ gedreht wird, bewegen sich die Räder nur um $75°$. Dies führt zu einer Lenkübersetzung von $R = 0.083$. Innerhalb von $140\,\mathrm{ms}$ dreht sich das Lenkrad um $63°$, während sich die Vorderräder um $5.23°$ drehen.

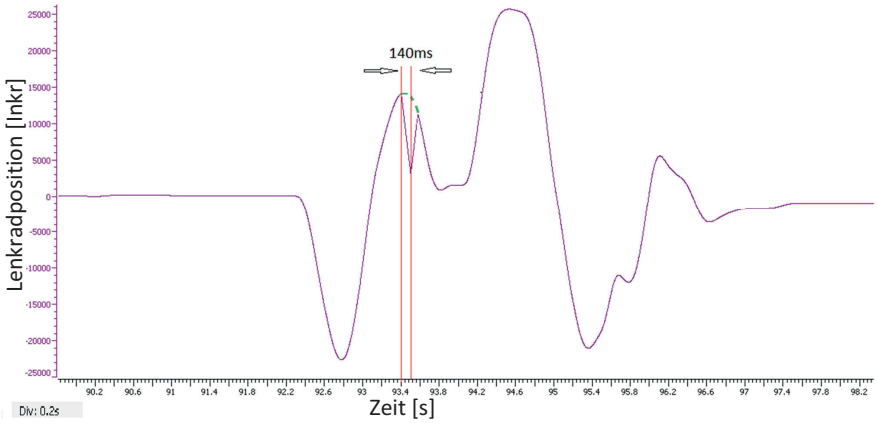

Abbildung 6.11: Lenkwinkelverlauf mit einer Falschlenkung von $t_{off} = 140\,\mathrm{ms}$ @ $50\,\mathrm{km/h}$ nach [155]

In der Abbildung 6.11 ist der Lenkradwinkel für eine normale Fahrt mit der Ausfallzeit von $t_{off_3} = 140\,\mathrm{ms}$ dargestellt. Die gestrichelte Linie zeigt den Verlauf ohne Störung, die durchgezogene Linie hingegen den Verlauf des Versagens der Funktion. Auf der x-Achse wird die Zeit in Sekunden und auf der y-Achse der Lenkradwinkel in Inkrementen angegeben. Es wird hier von einer linearen Übertragungsfunktion ausgegangen. Die Lenkung in diesem Versuch beträgt weniger als $8\,\%$ des maximal möglichen Lenkwinkels.

6.3 Modellbasierte Entwurfsmethodik für die dynamische Rekonfiguration

Im Folgenden wird der Entwurfsansatz auf eine beispielhafte Fahrzeugfunktion in Form einer automatisch gesteuerten Bremse angewendet. Die einzelnen erforderlichen Schritte werden dafür detailliert erläutert. Zunächst wird dazu der betrachtete Anwendungsfall beschrieben, es werden die jeweiligen Services und Clients identifiziert und es erfolgt ein Zuweisung des Anwendungsfalls auf die Services und Clients (Abschnitt 6.3.1). Danach werden die Service- und Client-seitigen Tasks sowie die entsprechenden Routinen aufgezeigt (Abschnitt 6.3.2). Abschließend wird die maximale Ausführungszeit der Rekonfiguration ermittelt (Abschnitt 6.3.3).

6.3.1 Anwendungsanforderungen und Fallbeispiele für die dynamische Rekonfiguration

Für die Beurteilung wird der Schwerpunkt auf die Bewertung der Ende-zu-Ende-Latenz im Falle einer Rekonfiguration gelegt. Zu diesem Zweck wird davon ausgegangen, dass ein in Betrieb befindliches Netzwerk von Services und Clients beeinträchtigt wird.

Unter einem laufenden Netzwerk ist zu verstehen, dass jeder Client mit seinen entsprechenden Services verbunden ist und diese benutzt. Der Begriff der Beeinträchtigung bedeutet in diesem Zusammenhang, dass Clients jene Services, an die sie ursprünglich gebunden waren, nicht mehr verwenden.

In diesem Fall muss die Service-Erkennung nochmals ausgeführt werden, um Fallback-Service-Instanzen zu identifizieren und folglich die Architektur zu rekonfigurieren. Jeder Client wird dabei erneut an die entsprechenden Services gebunden.

In diesem Anwendungsfall kommt die Trajektorienplanung in zwei Instanzen als Service zur Anwendung. Der Client stellt die Bremsfunktion dar, die Rekonfigurations- und Überwachungslogik sind ebenfalls auf dem Client implementiert.

6.3.2 Implementierung und Modellierung der Systemarchitektur

Implementierung für die Simulation

Die Implementierung basiert auf der Realisierung verschiedener Arten von Aufgaben, die von den Netzwerkteilnehmern in Abbildung 6.12 bereitgestellt werden. Hierbei wird zwischen Service-Tasks der Trajektorienplanung, den Client-Tasks der Bremsfunktion und der Aktuator-Task des Bremsaktuators unterschieden. Für den Service werden die folgenden Tasks eingeführt.

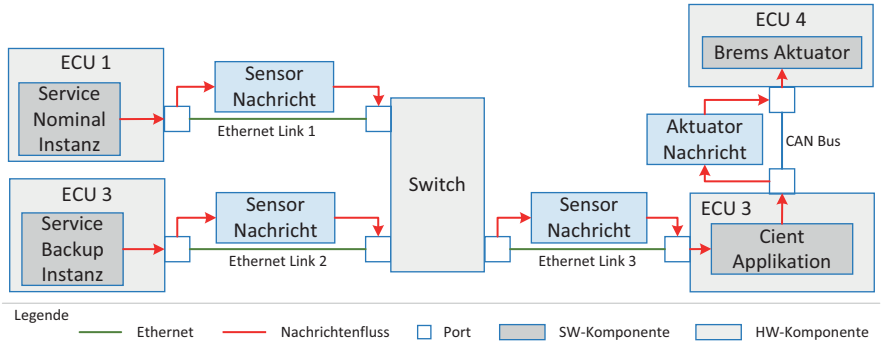

Abbildung 6.12: Beispielarchitektur zur Bewertung der maximalen Ausführungszeit nach [145]

Task 1: Offer-Service-Task: Der Offer-Service-Task sendet regelmäßig Ethernet-Nachrichten an die Client-Anwendung.

Task 2: Interrupt-Service-Routine: Die Interrupt-Service-Routine verarbeitet eingehende Ethernet-Nachrichten, die die Service-Subscriptions der Client-Anwendung darstellen. Die Subscription wird durch Extrahieren der MAC[4]-Adresse des Clients erreicht.

[4] Media Access Control

Task 3: Send-Event-Task: Nachdem ein Client einen Service abonniert hat, sendet der Send-Event-Task regelmäßig Ereignisse an die Client-Anwendung. Diese Ereignisse repräsentieren den aktuellen Sensorwert. Zum Senden von Ereignissen wird die MAC-Adresse des Clients benutzt. Diese wurde von der Interrupt-Service-Routine extrahiert. Alle $10\,\mathrm{ms}$ wird ein Ereignis gesendet.

Auf der Client-Seite werden folgende Tasks verwendet:

Task 1: Interrupt-Service-Routine: Ethernet-Nachrichten werden von der Interrupt-Service-Routine der Client-Anwendung verarbeitet. Diese stellen Service-Offers dar und lösen auch die Aufgabe von Subscriptions aus.

Task 2: Subscription-Task: Der Subscription-Task wertet die von der Interrupt-Service-Routine empfangenen Ethernet-Nachrichten aus, indem das Ziel der Service-Instanzen in Form ihrer MAC-Adresse gefiltert wird. Anschließend wird diese MAC-Adresse in eine Subscription-Nachricht eingefügt und an die entsprechende Service-Instanz gesendet.

Task 3: Monitoring-Task: Der Monitoring-Task wertet aus, ob eine bestimmte Sensor-Service-Instanz die Ereignisse innerhalb des Zeitraums bereitstellt. In dieser Hinsicht wird eine Service-Instanz als beeinträchtigt angesehen, wenn weniger als sieben Ereignisse innerhalb eines Intervalls von $100\,\mathrm{ms}$ empfangen werden. Infolgedessen muss der Client die Fallback-Instanz des Trajektorienservice abonnieren, indem er zuvor die Aufgabe der Subscription einleitet.

Task 4: Request-Action-Task: Der Request-Action-Task sendet regelmäßig Anforderungen an den Aktuator-Task. Diese Anforderungen weisen den Bremsaktuator unter anderem an zu bremsen.

Der Aktuator-Task ist wie folgt umgesetzt: Der Bremsaktuator ist für den Empfang und die Verarbeitung von Befehlen der Client-Anwendung verantwortlich.

Statische Modellierung

Im vorgeschlagenen Entwicklungsprozess muss das System zunächst modelliert werden, um die Rekonfigurationszeit zu ermitteln und diese in einem ersten Schritt zu überprüfen. Zu diesem Zweck wurde ein benutzerdefiniertes modellbasiertes Entwicklungstool aus dem Projekt SysKitHW [95] unter Verwendung des Eclipse-Modellierungs-Frameworks implementiert.

Mit den benutzerdefinierten Metamodellen können die verschiedene Ebenen in Übereinstimmung mit den spezifischen Anforderungen modelliert werden. In diesem Fall müssen die Hardware-Architektur, die Software-Architektur, die Service-Schicht und das Rekonfigurations-Szenario modelliert werden. Die Abbildung 6.13 zeigt das Hardware-Diagramm des modellierten Beispielsystems mit den in dieser Konfiguration verwendeten zwei MPSoCs.

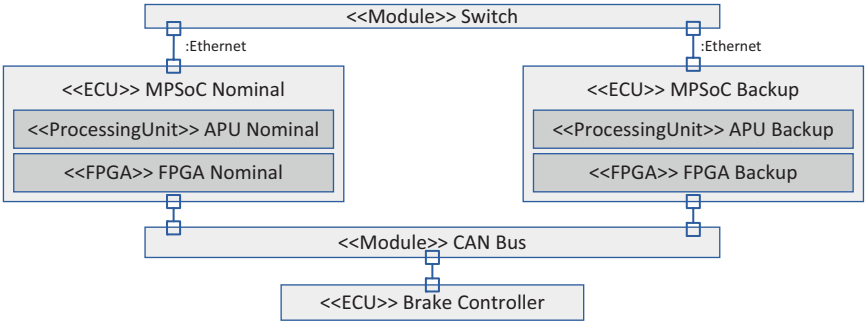

Abbildung 6.13: Hardware-Diagramm des modellierten Systems nach [145]

Die Trajektorienplanung ist auf beiden MPSoCs implementiert und wird jeweils in der APU ausgeführt. Der Service-Consumer wird ebenfalls in der APU ausgeführt. Ein MPSoC wird als Primär-Instanz und der andere als Fallback-Instanz betrachtet. Diese beiden MPSoCs kommunizieren miteinander über einen Switch und eine Ethernet-Verbindung. Der FPGA-Teil des MPSoCs wird verwendet, um Informationen auf den CAN-Bus umzuleiten. Beide Plattformen bieten den gleichen Trajektorienplanungs-Service an. Die Client-Anwendung wird ebenfalls von beiden MPSoCs als Service-Consumer implementiert, die wiederum die beiden Trajektorienplanungs-Services verwenden können. Die SOME/IP-Anwendung realisiert die Service-Erkennung und die Service-Offer-Funktionalität.

Um die Rekonfigurationszeit so genau wie möglich zu berechnen, sollten die SOME/IP-Middleware-Funktion und die Routing-Funktion des Switches berücksichtigt und auch in der Software-Schicht modelliert werden. Als Nächstes wird die maximale Ausführungszeit dieser Anwendungen bestimmt. Diese wird von dem Entwickler als Eingabegröße für die Berechnung der Rekonfigurationszeit benötigt. Hierzu können entweder ein separates Tool zum Schätzen des Timings verwendet oder empirisch abgeleitete Werte benutzt werden. In Abbildung 6.14 ist die modellierte Softwareschicht des Beispielsystems gezeigt.

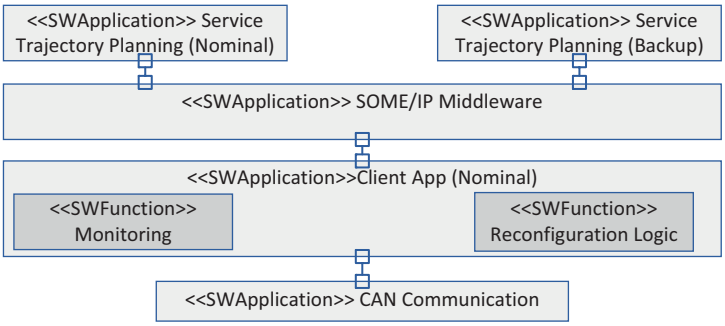

Abbildung 6.14: Software-Diagramm des modellierten Systems nach [145]

Der Ablauf des Rekonfigurationsvorgangs ist wie folgt: Zunächst fordert die Client-Anwendung den Service von der primären Instanz an, wobei auch der Monitoring-Task ausgeführt wird. Wenn der primäre Service nicht mehr verfügbar ist, erhält die Client-Anwendung keine Antwort mehr. Nach dem Ablauf des voreingestellten Timeouts benachrichtigt die SOME/IP-Middleware die Client-Applikation darüber, dass der aktuelle Service-Provider den Service nicht mehr anbietet.

Im letzten Schritt des Rekonfigurationsvorgangs abonniert die Client-Applikation den Anbieter für die Fallback-Instanz der Trajektorienplanung. Zeitgleich fordert die Client-Applikation den Service von der Fallback-Instanz an. Nachdem der Client das Ergebnis der Trajektorienplanung erhalten und verarbeitet hat, sendet er zur Steuerung der Bremse die entsprechenden Befehle an den CAN-Provider.

In Abbildung 6.15 ist das zum beschriebenen Rekonfigurationsvorgang gehörige Sequenzdiagramm dargestellt. In dem verwendeten Modellierungstool kann nun die Ausführungszeit für den Rekonfigurationsprozess berechnet und hinsichtlich der An-

forderungen überprüft werden. Nur wenn die Anforderungen erfüllt sind, sollten die nächsten Implementierungs- und Simulationsschritte ausgeführt werden.

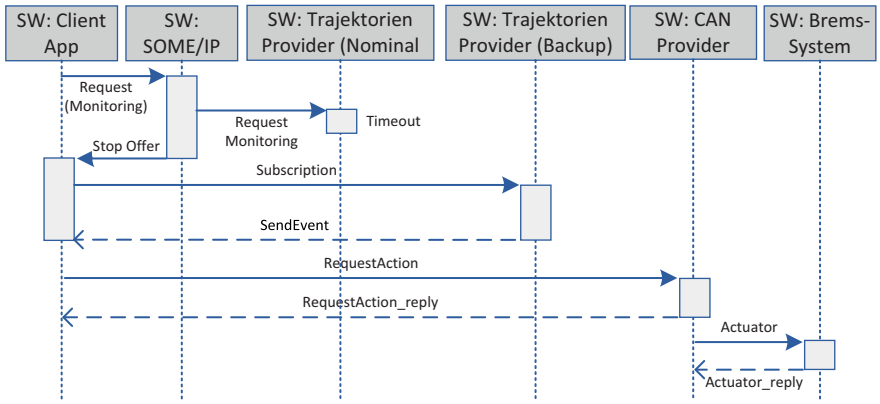

Abbildung 6.15: Sequenzdiagramm für den Rekonfigurationsvorgang nach [145]

Modellierung der Ende-zu-Ende-Latenz

Für die Beispielarchitektur wird eine Ende-zu-Ende-Latenzzeit festgelegt. Diese legt fest, dass die Bereitstellung der Trajektorienplanung, die Verarbeitung ihrer Daten und die Übermittlung der Steuerungsnachricht an den Bremsaktuator den Wert von $150\,\mathrm{ms}$ nicht überschreiten darf. Diese Anforderung muss auch für den Fall einer Rekonfiguration erfüllt werden, bei der der Nachrichtenpfad eine zusätzliche Service-Discovery-Prozedur umfasst. Das Nachrichtenprotokoll für den Fall einer Rekonfiguration ist in Abbildung 6.13 dargestellt.

Als Auslöser für die Rekonfiguration wird angenommen, dass die primäre Instanz des Sensor-Service beeinträchtigt ist. Die Degradations-Erkennung erfolgt auf dem Client durch den Monitoring-Task, der innerhalb eines bestimmten Zeitintervalls von $100\,\mathrm{ms}$ nicht die angeforderte Anzahl von Ereignissen empfängt. Im Folgenden wird ein Service-Erkennungsprotokoll ausgeführt bevor schließlich eine Nachricht an den Aktuator gesendet wird.

Systemarchitektur

Als Beispielarchitektur wird die Topologie in Abbildung 6.12 betrachtet. Die Topologie verfügt über serviceorientierte Funktionen innerhalb eines Ethernet-Netzwerks sowie über nicht-serviceorientierte Funktionen, die über den CAN-Bus kommunizieren. Das Bewertungsbeispiel besteht aus drei ECUs. Für die serviceorientierten Funktionen wird eine Client-Anwendung eingeführt, die zwei Instanzen eines Trajektorienplanungs-Service verwenden kann. Da kein Broker verfügbar ist, implementiert dieses System ein dezentrales Architekturmuster. Der Trajektorienplanungs-Service liefert mithilfe eines ereignisbasierten Interaktionsmusters Daten zu Hindernissen. Die Client-Anwendung wertet die Service-Informationen aus, um den Bremsaktuator zu steuern. Die Steuerbotschaften können die Bremse anweisen, zu bremsen, um einen Unfall zu vermeiden. In Bezug auf das Software-Architekturmodell wird für jede der Service-Instanzen sowie für die Client-Anwendung und den Bremsaktor eine Software-Komponente benötigt, die in einzelne Tasks heruntergebrochen wird.

6.3.3 Simulationsergebnisse und Diskussion

In diesem Unterabschnitt werden die Endergebnisse für die ermittelten Latenzen dargestellt. Dabei werden zum einen die Ergebnisse aus dem statischen Modell und zum anderen die aus der Simulation gewonnenen Resultate dargelegt.

Ende-zu-Ende-Bewertung der Latenz

Ergebnisse aus dem statischen Modell: In dem in Abbildung 6.15 beschriebenen Rekonfigurations-Szenario wird die Funktion *Rekonfigurationsprozess überprüfen* verwendet, um die Gesamtzeit zwischen dem Senden der Anforderung durch die Client-Anwendung und dem Empfangen der Nachricht durch die Bremssteuerung zu berechnen. Die für jede Task verwendete Ausführungszeit und die gesamte Rekonfigurationszeit sind in Tabelle 6.8 aufgeführt.

Die Gesamtzeit ist kürzer als die erforderliche Rekonfigurations-Zeit, sodass die statische Überprüfung für diesen Rekonfigurations-Prozess erfolgreich ist. Der nächste Schritt einer simulativen Bewertung kann somit durchgeführt werden.

Ergebnisse aus der Simulation: Die Beispielarchitektur wurde im Tool chronSIM simuliert, um die Ausführungszeiten der eingeführten Tasks abzuleiten. In dieser Hinsicht konnten die ermittelten maximalen Ausführungszeiten für einzelne Tasks abgeleitet werden. Für 100 % der Simulationsläufe wurden die Anforderungen an die Ende-zu-Ende-Latenz erfüllt. Insgesamt wurden mehr als 6000 Simulationsläufe ausgewertet.

Tabelle 6.8: Ausführungszeit für jeden Task und gesamte Rekonfigurationszeit

Task	Ausführungszeit [ms]
Anforderung (Monitoring-Task)	22.0
Rekonfiguration (Interrupt-Service-Routine)	70.0
Beenden des Ereignisses (Interrupt-Service-Routine)	70.0
Subscription-Task	1.0
Request-Action-Task	2.0
Aktuator-Task	4.2
Rekonfigurationszeit insgesamt	99.2

6.4 Methodik zur Entwurfsevaluierung

In diesem Abschnitt wird zunächst die Einbindung der bereits vorgestellten Markov-Ketten in die Monte-Carlo-Simulation gezeigt (Abschnitt 6.4.1), die verschiedenen Szenarien dafür aufgestellt (Abschnitt 6.4.2) und die erzielten Ergebnisse dargelegt (Abschnitt 6.4.3).

6.4.1 Monte-Carlo-Simulation - Evaluierung und Optimierung

Wie in der Einführung zur Systemmodellierung erläutert, sind die FIT-Raten mit Ungenauigkeiten verbunden. Dies resultiert aus den zu ihrer Bestimmung eingesetzten Methoden, den bei der Ermittlung vorhandenen Messfehlern, den statistischen Ungenauigkeiten sowie den Transformationsungenauigkeiten (vgl. Abschnitt 5.4.3).

Die Berechnung der FIT-Raten ist daher analytisch nicht möglich bzw. ist deren Ermittlung mit einem großen Aufwand verbunden. Aus diesem Grund soll für die Evaluierung und die Optimierung des Gesamtsystems nach einer weiteren Möglichkeit

gesucht werden, um die FIT-Raten bestimmen zu können. An dieser Stelle kommt die Monte-Carlo-Simulation zum Einsatz, um die Bestimmung der FIT-Raten mit Hilfe der Wahrscheinlichkeitstheorie numerisch zu lösen. Für die Darstellung der einzelnen Schritte wurde ein Ablaufdiagramm erstellt.

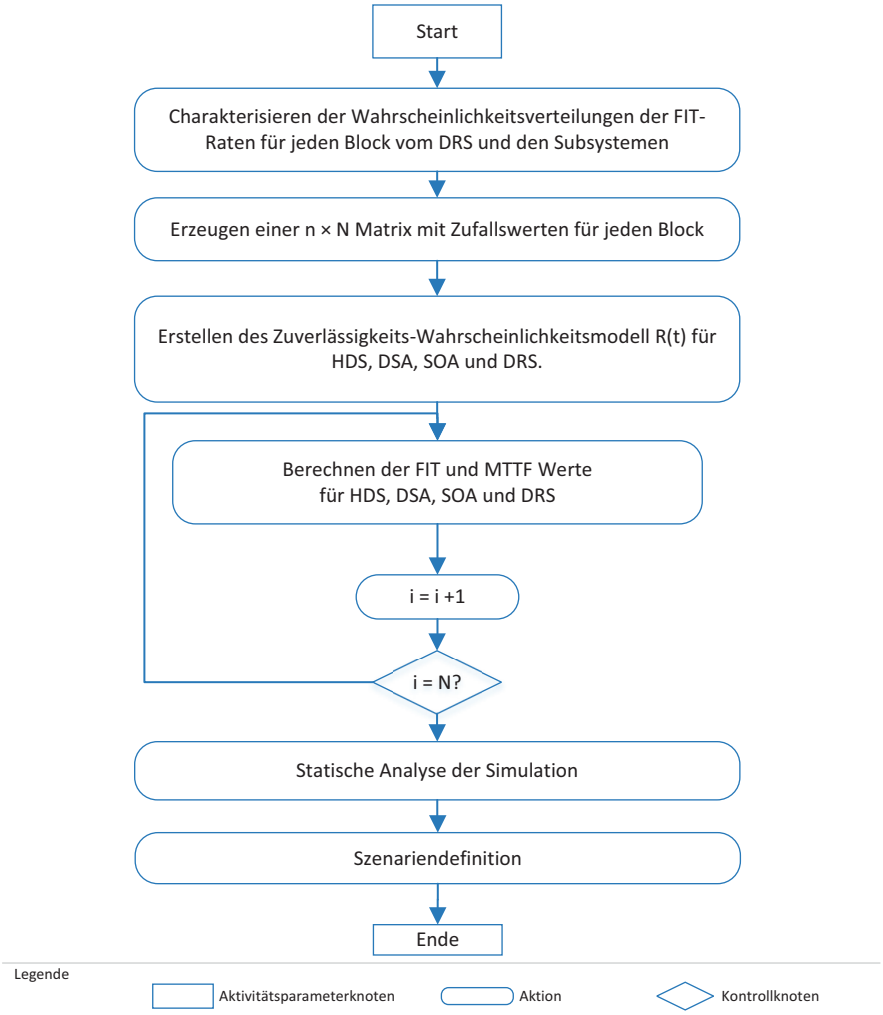

Abbildung 6.16: Ablaufdiagramm für die Implementierung der MCS nach [144]

Die Abbildung 6.16 zeigt das zugehörige Ablaufdiagramm für die Implementierung der MCS mit Matlab. Folgende Schritte wurden implementiert:

- Bestimmen der Ungenauigkeiten in den FIT-Raten durch Wahrscheinlichkeitsverteilung für jeden Block
- Erzeugen der Matrix n x N (n = Anzahl der verschiedenen Blöcke, N = Anzahl der Abtastwerte) von Zufallsabtastwerten gemäß der im vorhergehenden Punkt definierten Verteilung
- Ermittlung des Wahrscheinlichkeitsmodells für die Zuverlässigkeit $R(t)$ für HDS, DSA, SOA und DRS durch Markov-Ketten
- Berechnung des N-fachen der FIT-Rate und der MTTF für HDS, DSA, SOA und DRS durch Ersetzen von zufällig erzeugten FIT-Raten
- Durchführung einer statistischen Analyse der Simulationsergebnisse
- Ermitteln von Optimierungen und neuen Szenarien gemäß den Erkenntnissen zum vorherigen Punkt

Bei der Monte-Carlo-Simulation werden N-fach die Berechnungen durchgeführt. Zunächst muss dazu die Zustandsübergangsmatrix M aufgestellt werden (vgl. Abbildung 5.16). Beispielhaft ist die Zustandsmatrix M wie folgt implementiert:

Quelltext 6.1: Zustandsübergangsmatrix M

```
1      M = matrix(c(0.0, 0.5, 0.0, 0.0, 0.5,
2                   0.5, 0.0, 0.5, 0.0, 0.0,
3                   0.0, 0.5, 0.0, 0.5, 0.0,
4                   0.0, 0.0, 0.5, 0.0, 0.5,
5                   0.5, 0.0, 0.0, 0.5, 0.0),
6                 nrow=5, byrow=TRUE)
```

Dieses ist eine quadratische Matrix, deren einzelne Elemente jeweils die Wahrscheinlichkeit für den Übergang in den entsprechenden Zustand angeben.

Nach dem Aufstellen der Matrix M wird eine Iteration über die Anzahl der gewählten Samples N durchgeführt und somit die jeweiligen FIT-Raten und MTTF-Werte ermittelt. Dabei beinhaltet jedes Matrixelement die Wahrscheinlichkeit für den Übergang in den nachfolgenden oder vorangegangen Zustand.

Die Umsetzung der Iteration durch die einzelnen Samples ist im folgenden Quelltextausschnitt dargestellt und ist wiederum beispielhaft zu sehen:

Quelltext 6.2: Iteration durch die einzelnen Samples

```
n = 5000 # Anzahl Samples
x = numeric(n)
x[1] = 1 # Zustand auf 1 setzen fuer den Beginn
for (i in 2:n) {
  x[i] = sample.int(5, size=1, prob=M[x[i-1],])
  # beziehen des naechsten Zustands
  # fuer die Integer 1 bis 5 mit der Wahrscheinlichkeit
  # aus der Zustandsuebergangsmatrix M, basierend
  # auf dem vorangegangen Wert von X.
}
```

Basierend auf diesem einfachen Beispiel werden die entsprechenden FIT- und MTTF-Werte unter Verwendung von Formel 5.1, Formel 5.2 und Formel 5.3, sowie Formel 2.6 bestimmt [24]. Mit diesen Formeln wird der Random-Walk Metropolis-Hastings Sampler aufgebaut und für die Berechnungen verwendet.

Der nächste Quelltextausschnitt zeigt für das hier verwendete Beispiel, wie die Funktion zur Berechnung des MTTF-Werts des HDS aufgestellt ist:

Quelltext 6.3: Berechnung der MTTF-Werte für das HDS

```
mttf_hds = function(P1[t], P2[t], P3[t], P4[t], P5[t], R[t], lambda_R,
↪  lambda_S, lambda_B){
  # Zustandsuebergangsmatrix:
  M = matrix(c(-(lambda_B + lambda_S + lambda_R), 0, 0, 0, 0,
  lambda_B, -(lambda_R + lambda_S), 0, 0, 0,
  lambda_R, 0, -(lambda_B + lambda_S), 0, 0,
  lambda_S, 0, 0, -(lambda_R + lambda_S), 0,
  0, (lambda_R + lambda_S), (lambda_B + lambda_S), (lambda_R +
  ↪  lambda_S), 0),
  nrow=5, byrow=TRUE)
  # Aufstellen des Vektors:
  Y = [P1; P2; P3; P4; P5]
  # Bilden der Matrix:
  S = M * Y
  # Setzen der Anfangswerte
  P1[0] = 1
  P2[0] = 0
```

```
16        P3[0] = 0
17        P4[0] = 0
18        P5[0] = 0
19        # Berechnen der Zuverlaessigkeitsfunktion:
20        R[t] = sum(P[t])
21        # Berechnen der MTTF:
22        MTTF = int(R, 0, Inf)
23      }
```

Dieser Codeausschnitt zeigt wichtige Schritte für die Berechnung der MTTF-Werte und wird in den folgenden Szenarien für jeden einzelnen Block des Gesamtsystems durchgeführt. Für die Durchführung des Samplings als Schritt bei der Monte-Carlo-Simulation ist im nächsten Quelltextausschnitt das entsprechende Beispiel dargestellt.

Quelltext 6.4: Berücksichtigung der Wahrscheinlichkeitsverteilung

```
1       mh = function(){
2         # Schritt 1, Initialisierung:
3         N_ROWS = 10000000
4         A = 1 + (10-1)*rand(N_ROWS,1)
5         A(:,1) = []
6         A
7         # Schritt 2, Iteration
8         for (row in 1:N_ROWS){
9          # Schritt 2a
10         # Einen Samplekandidaten beziehen mit der
11         # gewaehlten Wahrscheinlichkeitsverteilung:
12         apu_cand = rnorm(n=1, mean= 5, sd= 10)
13         # Schritt 2b
14         mttf_hds_cand = mttf_hds(P1(t), P2(t), P3(t), P4(t), P5(t), R(t),
           ↪  lambda_R, lambda_S, lambda_B)
15         # Ergebnisse sammeln
16         mttf_hds_out[i] = mttf_hds_cand
17        }
18      }
```

Alle berechneten Werte werden in jedem einzelnen Iterationsschritt in einer Datei abgelegt. Auf diese Weise kann nach der gesamten Berechnung eine Korrelation der Werte durchgeführt werden, um auf das nächste Szenario zu schließen (vgl. Abbildung 6.18 und Abbildung 6.21).

Durch genaue Kenntnisse des Systems und der Zuhilfenahme der Korrelation werden Optimierungsmöglichkeiten identifiziert. Dazu können unter anderem aus der Bibliothek der Lösungselemente (vgl. Abbildung 4.1), die gemäß der Vorgaben der ISO 26262 im Laufe der Entwicklungserfahrungen immer weiter mit Lösungselementen gefüllt wird, Kandidaten zur Optimierung herangezogen werden.

6.4.2 Szenarien

Die FIT-Raten für jeden Systemblock werden von proprietären FMEDA-Tools ermittelt, die auf der quantitativen Ermittlung aller elektronischen Komponenten im Hinblick auf permanenten und vorübergehenden Hardware-Fehlern basieren. Daher können bereits zwei verschiedene Szenarien je nach Fehlertyp für die Analyse identifiziert werden. Darüber hinaus basiert die FMEDA-Analyse auf der erwarteten Lebensdauer der Fahrzeugelektronik von 15 Jahren und 8000 Betriebsstunden pro Jahr.

Die Referenzwerte für das HDS werden durch firmeninterne FMEDA-Analysen ermittelt. Sie umfassen nicht nur das Steuergerät mit seinen eingebauten redundanten Kernen, sondern auch seine Schnittstellen, Peripheriegeräte und Speicher. Der gleiche Ansatz wird für die SOA verwendet, um die FIT-Rate für die APU und die zugehörigen Komponenten zu ermitteln. Schließlich ist für die DSA die Bestimmung der FIT-Rate komplexer, da sie aus zwei isolierten Leistungsbereichen besteht, dem LPD und der PL. In dieser Hinsicht umfasst die RPU-FIT-Rate den ARM-Cortex-R5 im Lockstep-Modus, einschließlich des Speichers und des Interrupt-Controllers.

Die übrigen Komponenten wie PMU und CSU sowie I/O und Interconnects sind in den K-Modulen enthalten. In ähnlicher Weise wird auf der PL-Domäne die Implementierung des MB isoliert, die durch den Bereich der PL dargestellt wird. Diese umfasst die Logikschaltungen, den Block-RAM (BRAM), den Konfigurations-RAM (CRAM), den Ultra-RAM (URAM), das Advanced Extensible Interface (AXI) und den Interrupt-Controller. Die verbleibenden Module und PL-Funktionen sind in den K-Modulen zusammengefasst, dazu gehören das Atom-Modul, das Status-Modul, das Rücksetzsystem und der AXI-Interrupt-Controller.

Die für das MCS gewählten statistischen Verteilungen sind Normalverteilungen (N) für Werte im Bereich von QM[5] bis ASIL C. Die Referenzwerte werden also zentriert und durch den Mittelwert (x) charakterisiert. Die Ungenauigkeiten werden durch die Standardabweichung (σ) beschrieben. Für niedrige FIT-Ratenwerte (FIT<1) ist die gewählte statistische Verteilung die Beta-Verteilung (B) (a, b), wobei die Unsicherheiten durch das Intervall der durch a und b definierten Kurve gekennzeichnet sind. Diese Werte sind auf die Referenzwerte zentriert.

Referenzszenario

Die in Tabelle 6.9 zusammengefassten Referenzwerte geben die Block-FIT-Raten für zufällige Hardwarefehler wieder. Dies bedeutet, dass nur irreparable Szenarien für jeden Block und jedes Subsystem berücksichtigt werden. Dieses Szenario stellt die Referenz dar und steht in direktem Zusammenhang mit der Implementierung von ISO 26262 und IEC 61508.

Tabelle 6.9: Szenario 1: Referenzszenario und Werte

Block	$\mathbf{ECU_B}$	$\mathbf{ECU_S}$	$\mathbf{ECU_R}$	**K-Module**	**MB**	**RPU**	**APU**
FIT	700	700	700	16.97	0.0357	0.13	50
MCS	Samples: 1×10^6						
Verteilung	N	N	N	N	B	B	N
Parameter	$\bar{\chi} = 700$	$\bar{\chi} = 700$	$\bar{\chi} = 700$	$\bar{\chi} = 16.97$	a = 81	a = 81	$\bar{\chi} = 50$
	$\sigma = 140$	$\sigma = 140$	$\sigma = 140$	$\sigma = 3.4$	b = 2265	b = 619	$\sigma = 10$

Die verschiedenen Modellierungsszenarien spiegeln einen inkrementellen Ansatz wider. In dem zuvor erwähnten ersten Szenario sind die Referenzwerte bestimmt worden und eine vorläufige Ergebnisanalyse wurde durchgeführt. Die folgenden Szenarien wurden unter Berücksichtigung neuer Optimierungen zur Verbesserung der Zuverlässigkeit abgeleitet, ausgehend von einem mehr oder weniger pessimistischen Ansatz. Bei der Formulierung dieser Szenarien wurden die Monte-Carlo-Ergebnisse, die CCF-Analyse und die Betrachtungen bei der Design Space Exploration (DSE[6]) berücksichtigt.

[5] Qualitätsmanagement
[6] Design Space Exploration

Alternative Szenarien

Das erste alternative Szenario wird im Folgenden als Szenario 2 weitergeführt. Die DSA ist eine Kernkomponente für mögliche zukünftige Implementierungen und eignet sich daher ideal für Optimierungen. Die für dieses Szenario verwendeten Werte sind in Tabelle 6.10 angegeben.

Tabelle 6.10: Szenario 2: Pessimistischer Ansatz für DSA

Block	$\mathbf{ECU_B}$	$\mathbf{ECU_S}$	$\mathbf{ECU_R}$	**K-Module**	**MB**	**RPU**	**APU**
FIT	700	700	700	55.41	4.79	0.885	50
MCS	Samples: 1×10^6						
Verteilung	N	N	N	N	N	B	N
Parameter	$\bar{\chi} = 700$	$\bar{\chi} = 700$	$\bar{\chi} = 700$	$\bar{\chi} = 55.41$	$\bar{\chi} = 4.79$	a = 44	$\bar{\chi} = 50$
	$\sigma = 140$	$\sigma = 140$	$\sigma = 140$	$\sigma = 11$	$\sigma = 0.9$	b = 50	$\sigma = 10$

Transiente Fehler werden in diesem Szenario als permanente Fehler betrachtet, das dazu führt, dass sich die FIT-Rate stark erhöht. Aus diesem Grund wird dieses Szenario als pessimistischer Ansatz bezeichnet, da transiente Fehler per Definition normalerweise durch Redundanz in jedem Block korrigiert oder minimiert werden. Des Weiteren wird in diesem Szenario angenommen, dass die Wiederherstellungsraten nicht ausreichend sind und der jeweilige Redundanz- oder Wiederherstellungsmechanismus immer ausfällt.

Tabelle 6.11: Szenario 3: Optimierung für das HDS-System

Block	$\mathbf{ECU_B}$	$\mathbf{ECU_S}$	$\mathbf{ECU_R}$	**K-Module**	**MB**	**RPU**	**APU**
FIT	50	50	50	16.97	0.0357	0.13	50
MCS	Samples: 1×10^6						
Verteilung	N	N	N	N	B	B	N
Parameter	$\bar{\chi} = 70$	$\bar{\chi} = 70$	$\bar{\chi} = 70$	$\bar{\chi} = 16.97$	a = 81	a = 81	$\bar{\chi} = 50$
	$\sigma = 10$	$\sigma = 10$	$\sigma = 10$	$\sigma = 3.4$	b = 2265	b = 619	$\sigma = 10$

Das dritte betrachtete Szenario bezieht sich auf die Optimierung des HDS. Es besteht darin, die FIT-Rate jedes ECU-Blocks zu senken. Die dafür verwendeten Werte sind in Tabelle 6.11 angegeben. Das Szenario repräsentiert die benutzerdefinierte Architektur und das benutzerdefinierte Modell, bei dem die FIT-Raten der HDS-Steuergeräte für

die Gesamtzuverlässigkeit optimiert sind. Dieses Szenario basiert auf einer vorläufigen Analyse, bei der die Ergebnisse darauf hindeuteten, dass das HDS eine Kernkomponente für die gesamte DRS-MTTF und damit für den FIT- und ASIL-Wert ist.

Das vierte Szenario ergibt sich aus den Ergebnissen der MCS-Korrelation nach der Optimierung des HDS. In Tabelle 6.12 sind die Werte für dieses Szenario aufgeführt.

Tabelle 6.12: Szenario 4: Optimierung des SOA- und HDS-Subsystems

Block	$\mathbf{ECU_B}$	$\mathbf{ECU_S}$	$\mathbf{ECU_R}$	**K-Module**	**MB**	**RPU**	**APU**
FIT	50	50	50	16.97	0.0357	0.13	7
MCS	Samples: 1×10^6						
Verteilung	N	N	N	N	B	B	N
Parameter	$\bar{\chi} = 50$	$\bar{\chi} = 50$	$\bar{\chi} = 50$	$\bar{\chi} = 16.97$	a = 81	a = 81	$\bar{\chi} = 7$
	$\sigma = 10$	$\sigma = 10$	$\sigma = 10$	$\sigma = 3.4$	b = 2265	b = 619	$\sigma = 1.4$

Das fünfte Szenario leitet die Möglichkeit her, die auf der PL implementierten Blöcke zu optimieren. Die FIT-Raten jedes Blocks hängen von dem Bereich ab, den er auf der PL einnimmt. Die Vivado Design Suite HLS bietet Optimierungsmöglichkeiten bei der Synthese und Implementierung. Tabelle 6.13 zeigt die Werte für den PL-Bereich und die Taktoptimierung.

Tabelle 6.13: Szenario 5: DSE-Optimierungen für PL-Fläche und Takt

Block	$\mathbf{ECU_B}$	$\mathbf{ECU_S}$	$\mathbf{ECU_R}$	**K-Module**	**MB**	**RPU**	**APU**
PL-Fläche	$\bar{\chi} = 700$	$\bar{\chi} = 700$	$\bar{\chi} = 700$	$\bar{\chi} = 16.971$	a = 0.0334	a = 135	$\bar{\chi} = 50$
Takt	$\sigma = 700$	$\sigma = 700$	$\sigma = 700$	$\sigma = 16.972$	b = 0.0336	b =0.135	$\sigma = 50$

6.4.3 Simulationsergebnisse und Diskussion

Im Folgenden wird der vorgeschlagene Entwurfsansatz auf eine beispielhafte Fahrzeugfunktion in Form einer automatisch gesteuerten Bremse angewendet.

Szenario 1

Das linke Diagramm in Abbildung 6.17 zeigt, dass die Zuverlässigkeitsfunktion $R(t)$ wie erwartet exponentiell abnimmt. Aus der rechten Darstellung ist ersichtlich, dass nach 15 Jahren Dauerbetrieb eine Zuverlässigkeit von $90.5\,\%$ vorliegt.

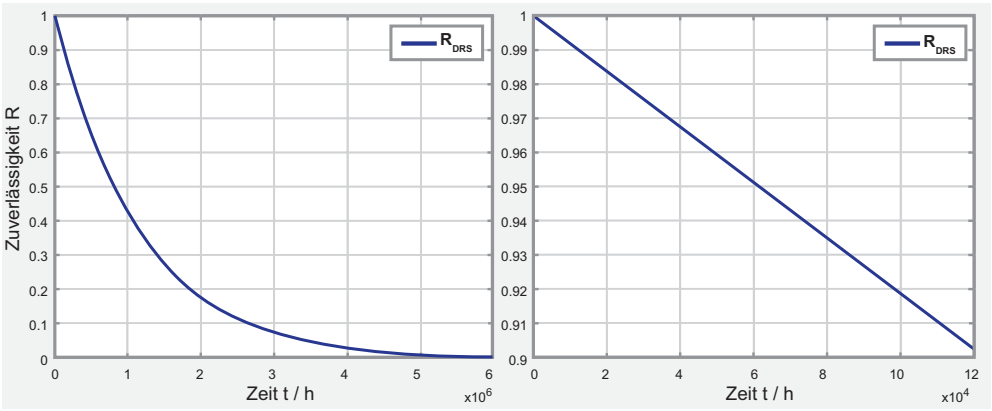

Abbildung 6.17: DRS-Zuverlässigkeitsfunktion nach [144]

Aus den in Tabelle 6.14 zusammengefassten Ergebnissen erschließt sich, dass das DRS gemäß der FIT-Rate für dieses Szenario als ASIL A eingestuft wird. Des Weiteren sind

Tabelle 6.14: Szenario 1: Quantitative Ergebnisse für DRS, HDS und DSA

	MTTF	**R(15 J.)**	**λ**	**FIT**	**ASIL**
DRS	$1.18 \cdot 10^6$ h	0.905	$8.48 \cdot 10^{-7}$ h	848.0	A
HDS	$1.19 \cdot 10^6$ h	0.98	$8.40 \cdot 10^{-7}$ h	840.0	A
DSA	$5.89 \cdot 10^7$ h	0.99	$1.69 \cdot 10^{-8}$ h	16.9	B/C

dort ebenfalls die berechneten Werte für das HDS, die DSA und das DRS dargestellt. Aus diesen wird ersichtlich, dass das DRS bezüglich der FIT-Rate nahe an den Werten des HDS liegt.

Führt man im nächsten Schritt eine Pearson Korrelation auf den Ergebnissen der MCS durch, kann abgeleitet werden, dass der Parameter mit dem größten Einfluss auf die FIT-Rate des DRS das HDS ist. Diese Erkenntnis deckt sich mit dem Ergebnis der sehr nahe beieinanderliegenden FIT-Werte von DRS und HDS. Dieser Zusammenhang wird auch aus der Abbildung 6.18 deutlich.

Im Vergleich zu anderen Architekturen wie der in [86] dargestellten k-aus-n-Architektur kann eine Verbesserung des DRS erzielt werden, indem die FIT-Raten der einzelnen Elemente reduziert wird.

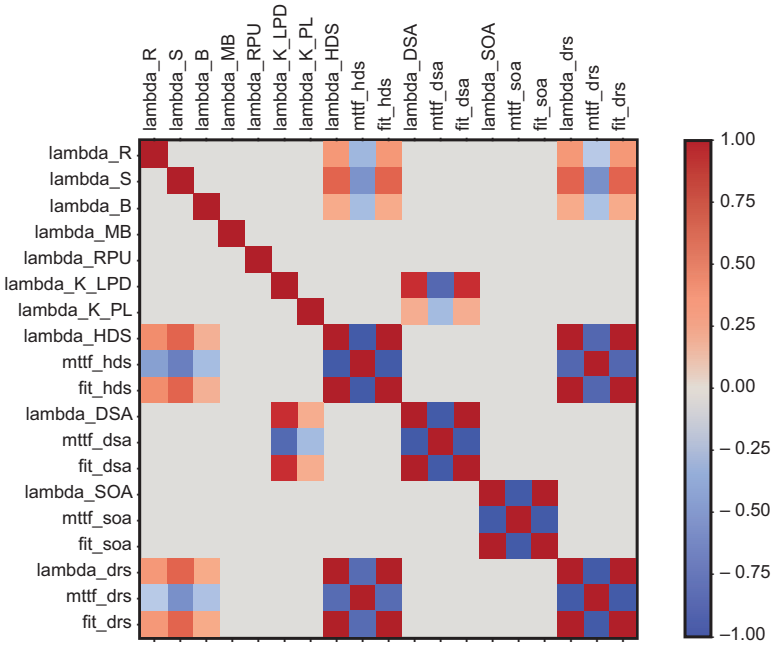

Abbildung 6.18: MCS-Korrelationsdiagramm für das Referenzszenario nach [144], Farbe der Rechtecke symbolisieren Stärke und Richtung der Zusammenhänge.

Eine weitere Möglichkeit besteht darin, dass das HDS in der FTA-Hierarchie geändert wird. Dies wird ermöglicht, indem entweder das HDS-System unter jeder DSA dupliziert wird oder Steuerungsfunktionen des HDS auf die PL übertragen werden, um eine zentralisierte Architektur zu erstellen, wie es bei der in [42] vorgestellten DSA der Fall ist.

Szenario 2

Die resultierende Zuverlässigkeitsfunktion $R(t)$ ist in Abbildung 6.19 dargestellt. Durch Betrachtung des Graphen kann auf eine Verschlechterung der DSA-Zuverlässigkeit geschlossen werden. Die Auswirkungen auf das DRS sind jedoch im Vergleich zum Referenz-Szenario nicht signifikant.

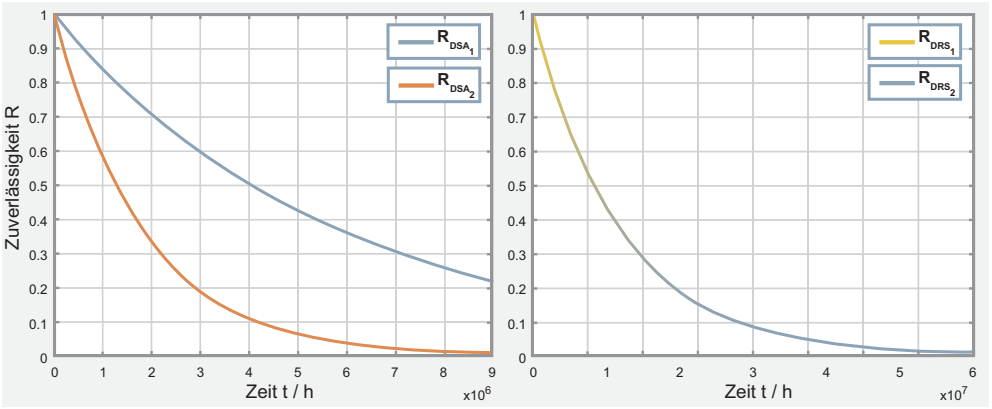

Abbildung 6.19: Vergleich der Zuverlässigkeit zwischen dem Szenario 2 und dem Referenzszenario für DSA (links) und DRS (rechts) nach [144]

Es kann auf eine Erhöhung der DSA-FIT-Rate um ca. $180\,\%$ geschlossen werden, während sich die FIT für das DRS nur um ca. $1.3\,\%$ erhöht. Beide Systeme behalten ihr ASIL-Niveau und ihre Zuverlässigkeit bei 15 Jahren bei, was wiederum die Bedeutung der MTTF für diese Optimierung herabsetzt.

Tabelle 6.15: Szenario 2: Quantitative Ergebnisse für DRS- und DSA-System

	MTTF	R(15 J.)	λ	FIT	ASIL
DRS	$1.16 \cdot 10^6$ h	0.905	$8.59 \cdot 10^{-7}$ h	859.26	A
DSA	$1.80 \cdot 10^7$ h	0.995	$5.55 \cdot 10^{-8}$ h	55.54	B/C

Die Ergebnisse für dieses Szenario sind in Tabelle 6.15 zusammengefasst, in der die Resultate für das DRS und die DSA dargestellt sind.

Szenario 3

Erwartungsgemäß verbessert die Erhöhung des HDS auf ASIL B/C-Niveau die Zuverlässigkeit in beiden Systemmodellen erheblich. Die Ergebnisse für das HDS-Subsystem werden in allen Aspekten verbessert, wobei die Verbesserung der FIT-Rate und daher die Verbesserung von ASIL A auf ASIL B/C am ausschlaggebendsten ist. Die daraus resultierenden Zuverlässigkeiten $R_{HDS_2}(t)$ und $R_{DRS_2}(t)$ sind in Abbildung 6.20 dargestellt.

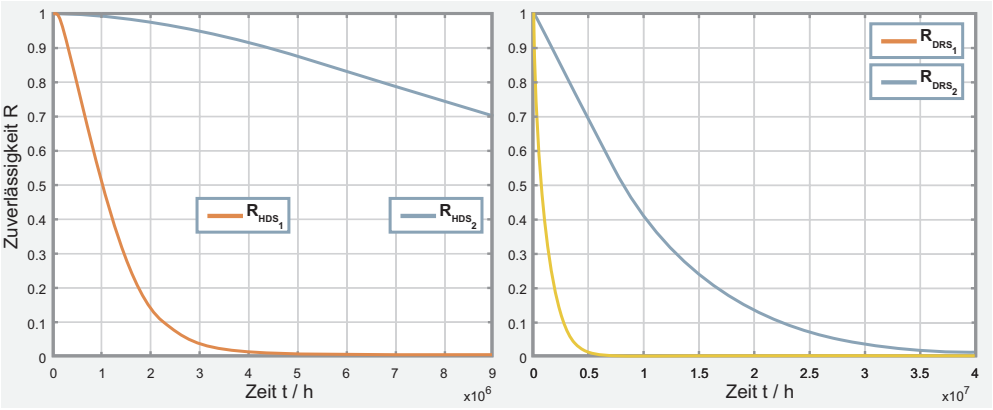

Abbildung 6.20: Vergleich der Zuverlässigkeit zwischen dem Szenario 3 und dem Referenzszenario für HDS (links) und DRS (rechts) nach [144]

Aus den in Tabelle 6.16 zusammengefassten Ergebnissen erschließt sich, dass das DRS gemäß der FIT-Rate für dieses Szenario als ASIL A eingestuft wird. Weiterhin sind die berechneten Werte für das HDS, die DSA und das DRS in Tabelle 6.16 dargestellt:

Tabelle 6.16: Szenario 3: Quantitative Ergebnisse für HDS und DRS

	MTTF	**R(15 J.)**	**λ**	**FIT**	**ASIL**
HDS	$1.66 \cdot 10^7$ h	0.99	$6.00 \cdot 10^{-8}$ h	60	B/C
DRS	$1.06 \cdot 10^7$ h	0.993	$9.44 \cdot 10^{-8}$ h	94.37	A

Trotz dieser Verbesserungen steigt die Zuverlässigkeit für 15 Jahre im Vergleich zu den Referenzwerten des HDS nur um 1 %. Die größeren Verbesserungen treten jedoch beim DRS auf, bei dem die ASIL- und FIT-Verbesserung durch die Erhöhung des

$R(15Jahre)$ um 90 % des Referenzszenarios auf 99 % gesteigert wird. In Abbildung 6.21 ist die Pearson-Korrelation für dieses Szenario dargestellt.

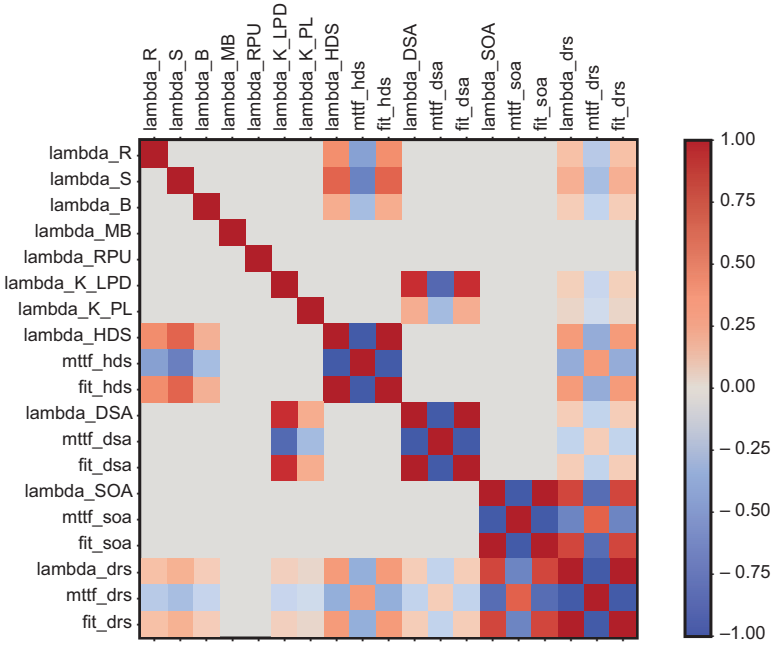

Abbildung 6.21: MCS-Korrelationsdiagramm für das HDS-Optimierungsszenario nach [144]

Aus dem Diagramm kann auf eine mögliche Kernkomponente für das nächste Szenario geschlossen werden. Im Vergleich mit dem MCS deutet alles auf das SOA-Subsystem hin, das für die DRS-FIT von erheblicher Bedeutung ist.

Szenario 4

Im Szenario 4 wird das Ergebnis aus Szenario 3 aufgegriffen, das SOA-System zu einem ASIL D-System zu verbessern. Die aus diesem Szenario resultierende Zuverlässigkeit von $R(t)$ ist in Abbildung 6.22 dargestellt.

Die Verbesserung des SOA-Systems auf ein ASIL D hat jedoch nicht die gewünschten Auswirkungen auf das Gesamtsystem. Es müssen demzufolge weitere Szenarien und

Optimierungen berücksichtigt werden, um für das Gesamtsystem von einem ASIL B auf ein ASIL D zu kommen.

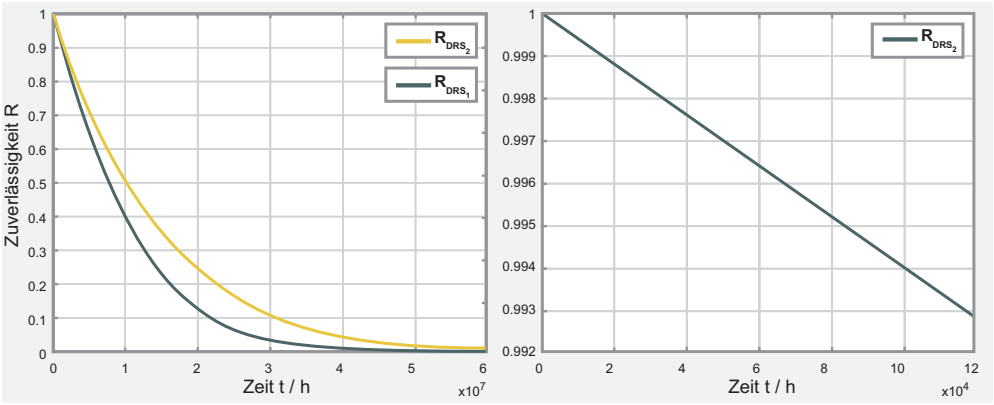

Abbildung 6.22: Vergleich der Zuverlässigkeit zwischen Szenario 4 und dem Referenz-Szenario (links), sowie die Zuverlässigkeit im Szenario 4 für 15 Jahre nach [145].

Für das Szenario 4 sind die Ergebnisse in Tabelle 6.17 zusammengefasst. Dort finden sich die Resultate für die SOA- und HDS-Optimierung.

Tabelle 6.17: Szenario 4: SOA- und HDS-Optimierung

	MTTF	**R(15 J.)**	**λ**	**FIT**	**ASIL**
Resultate	$1.45 \cdot 10^7$ h	0.99	$6.87 \cdot 10^{-8}$ h	68.71	B/C

Szenario 5

Die Ergebnisse dieser Optimierung sind für die Verbesserung der FIT für die DSA oder das DRS nicht relevant. Die verwendete Fläche beträgt ca. 2 % der gesamten PL. Somit ist die belegte Fläche durch die Implementierung sehr klein und hat damit kaum Einfluss. Daher ist die Gesamtverbesserung mit ca. 1 % gering, wenn beide Optimierungen in Bezug auf die genutzte Fläche auf dem PL verglichen werden. Das Optimierungsszenario kann jedoch in Betracht gezogen werden, wenn die Implementierung auf andere Funktionalitäten erweitert wird oder weitere redundante Funktionen auf der PL angewendet werden.

7 Schlussfolgerung und Ausblick

Die in der Dissertation erzielten Methoden und Ergebnisse werden in diesem Kapitel zusammengefasst. Abschließend wird ein Ausblick auf mögliche aufbauende Arbeiten und Erweiterungen gegeben.

7.1 Zusammenfassung und Schlussfolgerung

Mit den in dieser Arbeit entwickelten Methoden und Konzepten wird die Möglichkeit gegeben, dynamische Rekonfiguration im automobilen Umfeld zu entwickeln. Dabei werden insbesondere sicherheitskritische Funktionen mit einer Anforderung an ein Fail-Operational-Verhalten berücksichtigt. Die entwickelte HW-/SW-Bibliothek wurde zusammen mit dem erstellten Methodenbaukasten in den Standardentwicklungsprozess der Automobilindustrie eingeordnet. Auf diese Weise wird es ermöglicht, eine Adaption zur Lösung an eine konkret gestellte Aufgabenstellung vorzunehmen.

In einem ersten Schritt wurde eine umfassende Analyse zum Stand der Technik im Bereich der dynamisch rekonfigurierbaren Systeme erstellt und eine Kategorisierung mithilfe von Abstraktionsebenen durchgeführt. Diese Ebenen wurden um die fehlende Ebene des System-of-Systems erweitert, die in zukünftigen automobilen Systemen eine zunehmend dominierende Stellung einnehmen wird.

Als Nächstes wurden Anforderungen abgeleitet, die unter anderem aus den identifizierten Limitierungen im Kapitel „Stand der Technik und Forschung“ entstanden sind. Dabei wurde der Bedarf erkannt, Auslegungsparameter an dynamisch rekonfigurierbare Systeme zu ermitteln. Hierfür schuf diese Arbeit die Herleitung aus Normen und Standards. Dabei wurde die ISO 26262-Definition für die FTTI um den Begriff der Rekonfigurationszeit erweitert. Dieser wichtige Wert für die Systemauslegung wurde

systematisch weiterverwendet, um zunächst eine Methode zu erhalten, die den maximal zulässigen Wert für diese Rekonfigurationszeit ermittelt. An einem Beispielaufbau in Form eines BMW-Simulators wurde gezeigt, wie dieser Wert experimentell bestimmt werden kann. Auf diese Weise ist es nun möglich, auch für andere in dieser Arbeit nicht gezeigte Anwendungsfälle einen solchen Wert zu erfassen. Die Rekonfigurationszeit wurde ebenfalls im darauffolgenden Schritt zur Timing-Evaluierung herangezogen.

Aufbauend auf dem modellbasierten Entwurf wurde eine Möglichkeit geschaffen, das erstellte System hinsichtlich seiner Anforderungen zu überprüfen und zu evaluieren. Dafür wurde die maximale Rekonfigurationszeit herangezogen und verwendet. Beispielhaft erfolgte der Aufbau einer serviceorientierten Architektur und Untergliederung in einzelne SW-Komponenten und Tasks. Dabei wurden den jeweiligen Tasks Timing-Anforderungen zugewiesen, die mithilfe einer statischen und simulativen Modellierung nun die Möglichkeit bieten, bereits in einer frühen Entwicklungsphase das System hinsichtlich seiner maximalen Ausführungszeiten zu evaluieren und so die Basis für eine spätere Zertifizierung gemäß der ISO 26262 zu bilden.

Mit dem Einsatz der serviceorientierten Architektur wurden verschiedene Protokolle bezüglich ihres Einsatzes für dynamisch rekonfigurierbare Systeme im sicherheitskritischen Umfeld für automobile Anwendungen untersucht. Das Protokoll SOME/IP wurde dabei exemplarisch herausgegriffen und im Demonstrator umgesetzt. Die erstellte serviceorientierte Architektur wurde auf zwei MPSoC-Boards umgesetzt, die aufgrund ihrer Eigenschaften zukunftsweisend sind: Sowohl die Multicore-Fähigkeit als auch die vorhandene Unterteilung in High-Performance Cores und Real-Time Cores sowie das Vorhandensein eines FPGAs, um beispielsweise HW-Beschleuniger zu untersuchen, machen diese Boards ideal für den Einsatz in dieser Dissertation. Dabei wurde der FPGA nicht nur als Fallback für den Real-Time Core eingesetzt, sondern es konnte auch eine Absicherung der CAN-Kommunikation implementiert werden, die als Service gestaltet ist. Auf diese Weise konnte eine Cross-Layer-Rekonfiguration gezeigt werden. Bei der Umsetzung wurden verschiedene Quality of Service-Level eingeführt, um die Rekonfiguration steuern zu können.

Die beiden MPSoC-Boards sind über Ethernet miteinander verbunden und sind auf diese Weise prädestiniert für die Veranschaulichung der systemübergreifenden Rekonfiguration. Es wurde in diesem Zusammenhang gezeigt, dass eine Rekonfiguration sowohl innerhalb eines MPSoC-Boards erfolgt, aber auch über die Systemgrenzen hin-

aus. Somit spiegelt der Versuchsaufbau ein verteiltes System innerhalb eines Fahrzeugs wider als auch die Möglichkeit der Rekonfiguration über die Systemgrenzen hinaus. Mit den gezeigten Konzepten und Methoden können für diese Anwendungsfälle die entsprechenden Grundlagen gelegt werden.

Für eine Evaluierung eines komplexen Systems hinsichtlich seiner ASIL-Fähigkeit wurde eine Methode ausgearbeitet. Des Weiteren können mit dieser Methode Optimierungspotenziale ermittelt und bezüglich ihrer Auswirkung auf das ASIL überprüft werden. Hierzu wurden in dieser Arbeit umfassende Szenarien erstellt und diesbezüglich überprüft. In dieser Arbeit wurde das Gesamtsystem Fahrzeug betrachtet und ein deutlicher Mehrwert durch die erstmalige konsequente Herangehensweise an das Thema geschaffen, das gesamthaft betrachtet und auf eine systematische Grundlage gestellt wurde. In dieser Arbeit wurden die beiden Fragestellungen aus dem Kapitel „Einleitung“ beantwortet.

Zum einen wurden die notwendigen Methoden für den Einsatz von dynamischer Redundanz für echtzeitfähige Systeme im Fail-Operational-Kontext aufgezeigt. Mit den vorgestellten Methoden wird für zukünftige E/E-Architekturen Entwurfszeit gewonnen, die Qualität nachhaltig gesteigert und eine mittel- bis langfristige Kostensenkung durch die wiederverwendbaren Entwicklungsartefakte, die in die Bibliothek mit Lösungselementen mit einfließen erreicht. Weiterhin wurde eine Analysefähigkeit vorgestellt und die Beherrschbarkeit von Komplexität gefördert.

Zum anderen wurden die Vorteile von dynamischer Redundanz herausgearbeitet, die sich in erster Linie dadurch ergeben, dass zur Laufzeit nur eine Variante betrieben werden muss. Aus diesem Grund muss gegenüber der statischen Redundanz nicht aus mehreren Ergebnissen ausgewählt werden. Mit Hilfe des vorgestellten Ansatzes ist es nun möglich Systeme mit einem Fail-Operational-Kontext basierend auf dynamischer Rekonfiguration zuverlässig zu spezifizieren, zu evaluieren, zu optimieren, umzusetzen und zu zertifizieren bzw. zu homologieren.

Die Arbeit stellt exemplarisch die Anwendung der erarbeiteten Methoden dar und zeigt darüber hinaus auch eine allgemeine Anwendbarkeit auf. Quantitativ ist auf Systemebene mit Teilen des Lösungsbaukastens eine deutliche Kosteneinsparung möglich, eine exakte Bewertung ist dafür in dieser Arbeit nicht durchgeführt worden. Weiterhin sind auf Systemebene auch Gewichtseinsparungen möglich, die jedoch ebenfalls nicht im

Fokus dieser Arbeit standen. Eine qualitative Bewertung der Ergebnisse dieser Arbeit schließt sich insofern aus, da die Funktionserfüllung stets gegeben sein muss.

Im Automobilbereich wird die Entwicklung von hoch- und vollautomatisierten Fahrzeugen weiterhin verfolgt und aktiv vorangetrieben. Die dafür notwendigen echtzeitfähigen und Fail-Operational-fähigen Systeme werden bis zu der Ersteinführung und nach der ersten Serieneinführung weiter verfeinert und optimiert. Für den Entwicklungsprozess werden die geeigneten Methoden und für die Systeme die richtigen Lösungen benötigt. Diese Arbeit hat exakt das erarbeitet und bietet darüber hinaus die Möglichkeit die Lösungen und Methoden zu parametrieren, zu erweitern und modular einzusetzen.

Zusammenfassend präsentiert die vorliegende Dissertation ein umfassendes Paket aus neuen Methoden und Konzepten, die ebenenübergreifend die Entwicklung von dynamisch rekonfigurierbaren Systemen für sicherheitskritische Anwendung, deren Modellierung und Evaluierung ermöglichen, eingeordnet in den Entwicklungsprozess der Automobilindustrie.

7.2 Ausblick

In dieser Dissertation wurde in weiten Teilen die Basis für den Einsatz von dynamisch rekonfigurierbaren, sicherheitskritischen Echtzeitsystemen im automobilen Umfeld gelegt.

Das Projekt XANDAR, das zusammen in der Kooperation zwischen dem KIT/ITIV und der BMW Group sowie weiteren namhaften europäischen Forschungs- und Industriepartnern entstanden ist, trägt dazu bei, die hier entwickelten Konzepte weiter auszubauen. Dabei werden zusätzliche bedeutende und notwendige Bausteine insbesondere der automatisierten Code-Generierung zur Fehlerreduktion sowie die Berücksichtigung wichtiger Security-Aspekte angegangen.

Der Fokus dieser Arbeit lag auf der Erarbeitung von Methoden und Konzepten sowie der prototypischen Umsetzung mit einer serviceorientierten Architektur. Die dynamische Rekonfiguration im Sinne einer zu Laufzeit zu ändernden Logikprogrammierung eines FPGAs stand nicht im Mittelpunkt der Betrachtung. Vielmehr unterstützen die erarbeiteten Methoden und Konzepte jedoch den möglichen Einsatz einer solchen partiellen Rekonfiguration von FPGAs.

Für die Bearbeitung der vielversprechenden Anwendung von dynamischer Rekonfiguration auf der Ebene der System-of-Systems werden grundsätzliche konzeptionelle Ideen durchdacht, für eine Umsetzung eines solchen äußerst komplexen Systems ist aber eine weitere Vernetzung zu anderen Projekten notwendig, um das entsprechende Know-how aus diesen Bereichen einbinden zu können. Ein Beispiel dafür ist die durchgängige, ebenenübergreifende Modellierung, die Berücksichtigung von Security-Aspekten, die automatische Code-Generierung und die Kommunikationstechnologie, wie zum Beispiel der Einsatz von 5G.

Der Grundgedanke, die Systemkosten für sicherheitskritische Anwendungen insbesondere für die Anwendung bei autonomen Fahrzeugen, wurde zwar berücksichtigt, letztendlich ist eine detaillierte Kostenbetrachtung ausstehend.

Der Trend hin zu einer zentralisierten E/E-Architektur mit Zentralplattformen, die schon intrinsisch die Anforderungen für Fail-Operational-Funktionen erfüllen, wurde in Ansätzen unter zu Zuhilfenahme der MPSoC-Boards bereits aufgezeigt. Sobald diese Plattformen jedoch zur Verfügung stehen, sollte auf diesen ein MVP realisiert werden, bei dem die dynamische Rekonfiguration zum Einsatz kommt.

Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

Quelltextverzeichnis

Abkürzungsverzeichnis

APU	Application Processing Unit
ARAMiS	Automotive, Railway and Avionics Multicore System
ASIL	Automotive Safety Integrity Level
AutoKonf	Automatische rekonfigurierbare Aktorikansteuerungen für ausfallsichere automatisierte Fahrfunktionen
AUTOSAR	AUTomotive Open System ARchitecture
BMW	Bayerische Motoren Werke
CAN	Controller Area Network
COTS	Commercial Off The Shelf
CPS	Cyber Physical Systems
DDS	Data Distribution Service
DRS	dynamisch rekonfigurierbares System
DSA	Dynamic Simplex Architecture
E/E	Elektrik/Elektronik
ECU	Electronic Control Unit
EPS	Electric Power Steering
FIT	Failure In Time
FMEDA	Failure Modes, Effects and Diagnostic Analysis
FPGA	Field Programmable Gate Array

FRT	Fault Reaction Time
FTA	Fault Tree Analysis
FTTI	Fault Tolerant Time Interval
GuR	Gefahren- und Risiko
HDS	Hardware Dependant E/E System
HW	Hardware
I/O	Input/Output
IEC	International Electrotechnical Commission
IPC	Inter Processor Communication
ISO	International Organization for Standardization
ITIV	Institut für Technik der Informationsverarbeitung
Kfz	Kraftfahrzeug
KPI	Key-Performance-Indicator
LPD	Low-Power Domain
MAC	Media Access Control
MB	MicroBlaze
MCMC	Markov-Ketten Monte Carlo-Methode
MCS	Monte-Carlo-Simulation
MPSoC	Multiprocessor System on Chip
MTTF	Mean Time To Failure
MVP	Minimum Viable Product
PL	Programmable Logic
PMU	Platform Management Unit
PS	Processing System
QoS	Quality of Service
RPU	Real-Time Processing Unit
SAE	Society of Automotive Engineers
SOA	Service-Oriented Architecture
SoC	System on Chip
SOME/IP	Scalable Service-Oriented Middleware over IP
SW	Software
vgl.	vergleiche
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VMM	Virtual Machine Monitor

Glossar

ACC	aktive, abstandsabhängige Geschwindigkeitsregelung
Aktor/Aktuator	Bewegung, Tätigkeit
ASIC	anwendungsspezifische integrierte Schaltung, eine als integrierter Schaltkreis realisierte, nicht mehr veränderbare elektronische Schaltung
CANcaseXL	Das CANcaseXL ist ein USB-Interface mit einem leistungsfähigen 32Bit 64MHz Mikrocontroller und zwei CAN-Controllern zur Verarbeitung von CAN-Botschaften
CANoe	CANoe ist eine Entwicklungs- und Test-Software der Vector Informatik GmbH
CAPL	Communication Access Programming Language ist eine von Vector Informatik entwickelte Programmiersprache, die in den weit verbreiteten Software-Werkzeugen CANoe und CANalyzer zur Verfügung steht
Dekomposition	Die Aufteilung oder Zerlegung eines Systems in Komponenten wird als Partitionierung oder Dekomposition bezeichnet [118, S. 129].
Die	Würfel oder Plättchen, ungehaustes Stück eines Halbleiter-Wafers
ereignisgesteuertes System	ist ein System, dem die Erzeugung, Erkennung, die Verwendung und die Reaktion auf Ereignisse zugrunde liegt
FlexRay	serielles, deterministisches und fehlertolerantes Feldbussystem für den Einsatz im Automobil
Funktionsbeitrag	Durch eine Softwarekomponente umgesetzter und zur Erfüllung einer (verteilten) Funktion notwendiger Bestandteil
Gesamtsystem	Kombination von interagierenden Komponenten, die organisiert sind, um eine oder mehrere festgelegte Bestimmungen zu erfüllen [7, S. 13], [76].
Lidar	ist ein Verfahren zur Entfernungsmessung, bei dem das Ziel mit Laserlicht beleuchtet und die Reflexion mit einem Sensor gemessen wird

MIPS	Mikroprozessor ohne verschränkte Pipeline-Stufen ist eine Befehlssatzarchitektur im RISC-Stil, die ab 1981 von John L. Hennessy und seinen Mitarbeitern an der Stanford-Universität entwickelt wurde.
OEM	Originalausrüstungshersteller, Erstausrüster; dieser ist im Automobilbereich der Automobilhersteller selbst, z. B.: BMW, Daimler, Volkswagen, etc.
PCAN	CAN auf Ethernet Gateway
Radar	ist ein Erkennungssystem, das mit Hilfe von Funkwellen die Entfernung, den Winkel oder die Geschwindigkeit von Objekten bestimmt
Sensor	Empfindung, Gefühl
Tier	Ebene; Bezeichnung als Tier-1, Tier-2 als Systemlieferant, Komponentenlieferant etc.
Trajektorie	Bahn oder Bewegungspfad eines Objektes
UPC	ist definiert als eine Reihe von Aktionen, die das Netzwerk zur Überwachung und Kontrolle des Datenverkehrs durchführt
zeitgesteuertes System	ist ein Computersystem, das eine oder mehrere Gruppen von Aufgaben nach einem vorher festgelegten und eingestellten Aufgabenplan ausführt

Literatur- und Quellenverzeichnis

[1] Akkaya, I., P. Derler, S. Emoto und E. A. Lee. „Systems Engineering for Industrial Cyber–Physical Systems Using Aspects“. In: *Proceedings of the IEEE* 104.5, 2016, S. 997–1012. ISSN: 0018-9219. DOI: 10.1109/JPROC.2015.2512265.

[2] Angermeier, D., C. Bartelt, O. Bauer et al. *V-Modell XT. Das deutsche Referenzmodell für Systementwicklungsprojekte*. Hrsg. von Verein zur Weiterentwicklung des V-Modell XT e.V. Version 2.3. 2019. [Zugriff am: 25. 06. 2019]. Verfügbar unter: https://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/vmodell_xt_node.html.

[3] Armbruster, M., L. Fiege, G. Freitag et al. „Ethernet-Based and Function-Independent Vehicle Control-Platform: Motivation, Idea and Technical Concept Fulfilling Quantitative Safety-Requirements from ISO 26262“. In: Meyer, G., Hrsg. *Advanced Microsystems for Automotive Applications 2012*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, S. 91–107. ISBN: 978-3-642-29672-7. DOI: 10.1007/978-3-642-29673-4_9.

[4] Aslansefat, K., S. Kabir, Y. Gheraibia und Y. Papadopoulos. „Dynamic Fault Tree Analysis: State-of-the-Art in Modelling, Analysis and Tools“. In: Taylor and Francis, Hrsg. *Reliability Management and Engineering: Challenges and Future Trends*. 11.

[5] Avizienis, A., J.-C. Laprie, B. Randell und C. Landwehr. „Basic concepts and taxonomy of dependable and secure computing“. In: *IEEE Transactions on Dependable and Secure Computing* 1.1, 2004, S. 11–33. ISSN: 1545-5971. DOI: 10.1109/TDSC.2004.2.

[6] Azarian, A. und M. Ahmadi. „Reconfigurable computing architecture survey and introduction“. In: Li, W., Hrsg. *2nd IEEE International Conference on Computer Science and Information Technology* (*ICCSIT*). Proceedings (Peking, China, 8.–11. Aug. 2009). Institute of Electrical and Electronics Engineers. Piscataway, NJ, USA: IEEE, 2009, S. 269–274. isbn: 978-1-424-44519-6. doi: 10.1109/ICCSIT.2009.5234721.

[7] Bach, J. „Methoden und Ansätze für die Entwicklung und den Test prädiktiver Fahrzeugregelungsfunktionen“. Institut für Technik der Informationsverarbeitung (ITIV). Dissertation. Karlsruhe: Karlsruher Institut für Technologie, 2018.

[8] Bak, S., D. K. Chivukula, O. Adekunle et al. „The System-Level Simplex Architecture for Improved Real-Time Embedded System Safety“. In: *15th IEEE Real-Time and Embedded Technology and Applications Symposium* (*RTAS*). Proceedings (San Francisco, CA, USA, 13.–16. Apr. 2009). IEEE Computer Society Technical Committee on Real-Time Systems. Place of publication not identified: IEEE Computer Society Press, 2009, S. 99–107. isbn: 978-0-7695-3636-1. doi: 10.1109/RTAS.2009.20.

[9] Baleani, M., A. Ferrari, L. Mangeruca et al. „Fault-tolerant platforms for automotive safety-critical applications“. In: Moreno, J., Hrsg. *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems* (San Jose, CA, USA). ACM Special Interest Group on Microarchitectural Research and Processing. New York, NY, USA: ACM, 2003, S. 170. isbn: 1-5811-36765. doi: 10.1145/951710.951734.

[10] Balzert, H. und P. Liggesmeyer. *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. 3. Aufl. Lehrbücher der Informatik. Heidelberg: Spektrum Akademischer Verlag, 2011. isbn: 978-3-8274-2246-0. doi: 10.1007/978-3-8274-2246-0.

[11] Bapp, F. K. „Adaptives Monitoring für Mehrkernprozessoren in eingebetteten sicherheitskritischen Systemen“. Institut für Technik der Informationsverarbeitung (ITIV). Dissertation. Karlsruhe: Karlsruher Institut für Technologie, 2018.

[12] Bapp, F. K., T. Dörr, T. Sandmann et al. „Towards Fail-Operational Systems on Controller Level Using Heterogeneous Multicore SoC Architectures and Hardware Support“. In: *WCX SAE World Congress Experience* (10. Apr. 2018).

SAE Technical Paper Series. 400 Commonwealth Drive, Warrendale, PA, USA: SAE International, 2018. DOI: 10.4271/2018-01-1072.

[13] Basagiannis, S. und F. Gonzalez-Espin. „Towards Verification of Multicore Motor-Drive Controllers in Aerospace“. In: Koornneef, F. und C. van Gulijk, Hrsg. *International Conference on Computer Safety, Reliability, and Security*. proceedings (Delft, Niederlande, 22. Sep. 2015). Bd. 9338. Lecture notes in computer science 9338. Cham u. a.: Springer, 2015, S. 190–200. ISBN: 978-3-319-24248-4. DOI: 10.1007/978-3-319-24249-1.

[14] Bauer, H., Hrsg. *Kraftfahrtechnisches Taschenbuch*. 25., überarb. und erw. Aufl. Wiesbaden: Vieweg, 2003. 1232 S. ISBN: 3-528-23876-3.

[15] Bayerische Motoren Werke Aktiengesellschaft. *Die fünf Stufen bis zum autonomen Fahren*. Hrsg. von Bayerische Motoren Werke Aktiengesellschaft. 2019. [Zugriff am: 18. 03. 2019]. Verfügbar unter: https://www.bmw.com/de/automotive-life/autonomes-fahren.html.

[16] Beck, K. und C. Andres. *Extreme programming explained. Embrace change*. 2. Aufl. The XP series. Boston: Addison-Wesley, 2007. 189 S. ISBN: 978-0-321-27865-4.

[17] Becker, J. *ARAMiS II. Development Processes, Tools, Platforms*. 2017. [Zugriff am: 09. 08. 2017]. Verfügbar unter: https://www.aramis2.org/.

[18] Becker, J. *Automotive, Railway and Avionics Multicore Systems. ARAMiS*. 2017. [Zugriff am: 08. 08. 2017]. Verfügbar unter: http://www.projekt-aramis.de.

[19] Becker, J., M. Hubner, G. Hettich et al. „Dynamic and Partial FPGA Exploitation“. In: *Proceedings of the IEEE* 95.2, 2007, S. 438–452. ISSN: 0018-9219. DOI: 10.1109/JPROC.2006.888404.

[20] Becker, K. „Software Deployment Analysis for Mixed Reliability Automotive Systems“. Dissertation. München: Technische Universität München, 2017.

[21] Bereisa, J. „Applications of Microcomputers in Automotive Electronics“. In: *IEEE Transactions on Industrial Electronics* IE-30.2, 1983, S. 87–96. ISSN: 0278-0046. DOI: 10.1109/TIE.1983.356715.

[22] Bertelo, R. „Evaluation Methodologies of Dynamically Reconfigurable Systems in the Automotive Industry“. Departamento de Engenharia Eletrotécnica. Masterarbeit. Porto: Instituto Superior de Engenharia do Porto, 2019.

[23] Binfet-Kull, M. *Entwicklung einer Steer-by-wire-Architektur nach zuverlässigkeits- und sicherheitstechnischen Vorgaben*. 1. Aufl. Aachen: Mainz, 2001. 204 S. isbn: 3-89653-909-4.

[24] Birolini, A. *Reliability Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. isbn: 978-3-642-39534-5. doi: 10.1007/978-3-642-39535-2.

[25] Boahen, K. A. „A retinomorphic vision system". In: *IEEE Micro* 16.5, 1996, S. 30–39. issn: 0272-1732. doi: 10.1109/40.540078.

[26] Boch, R., Hrsg. *Geschichte und Zukunft der deutschen Automobilindustrie. Tagung im Rahmen der Chemnitzer Begegnungen 2000*. Stuttgart: Steiner, 2001. 290 S. isbn: 978-3-515-07866-5.

[27] Boehm, B. W. „A spiral model of software development and enhancement". In: *Computer* 21.5, 1988, S. 61–72. issn: 0018-9162. doi: 10.1109/2.59.

[28] Borgeest, K. *Elektronik in der Fahrzeugtechnik. Hardware, Software, Systeme und Projektmanagement*. ATZ/MTZ-Fachbuch. Wiesbaden: Springer Fachmedien Wiesbaden und Imprint Springer Vieweg, 2008. 1495 S. isbn: 978-3-834-81642-9. doi: 10.1007/978-3-8348-2145-4.

[29] Borgeest, K. *Elektronik in der Fahrzeugtechnik. Hardware, Software, Systeme und Projektmanagement*. 2. ATZ/MTZ-Fachbuch. Wiesbaden: Springer Fachmedien, 2010. 398 S. isbn: 978-3-8348-9337-6.

[30] Braun, J. und J. Mottok. „Fail-safe and fail-operational systems safeguarded with coded processing". In: Kuzle, I., Hrsg. *IEEE EUROCON* (Zagreb, Kroatien, 1.–4. Juli 2013). Institute of Electrical and Electronics Engineers. Piscataway, NJ, USA: IEEE, 2013, S. 1878–1885. isbn: 978-1-467-32232-4. doi: 10.1109/EUROCON.2013.6625234.

[31] Braun, L., F. Gauterin und E. Sax. *Experteninterview zur Anforderungsanalyse heutiger und zukünftiger E/E Architekturen im Kraftfahrzeug. Abschlussbericht, 27. April 2016*. 2016. doi: 10.5445/IR/1000054216.

[32] Broy, M., I. H. Krüger und M. Meisinger. „A formal model of services". In: *ACM Transactions on Software Engineering and Methodology* 16.1, 2007, 5–es. issn: 1049331X. doi: 10.1145/1189748.1189753.

[33] Broy, M., I. H. Krüger und M. Meisinger. *Model-Driven Development of Reliable Automotive Services*. Bd. 4922. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. isbn: 978-3-540-70929-9. doi: 10.1007/978-3-540-70930-5.

[34] Calabrese, R. L. „Half-center Oscillators Underlying Rhythmic Movements“. In: Arbib, M. A., Hrsg. *The handbook of brain theory and neural networks*. 1. Aufl. A Bradford book. Cambridge, MA, USA: MIT, 1998, S. 444–447. isbn: 0-262-51102-9.

[35] CMMI Institute LLC. *Introducing CMMI V2.0. CMMI Capability Maturity Model Integration*. 2019. [Zugriff am: 26. 06. 2019]. Verfügbar unter: https://cmmiinstitute.com/cmmi.

[36] Cubo, J. und E. Pimentel. „DAMASCo: A Framework for the Automatic Composition of Component-Based and Service-Oriented Architectures“. In: Crnkovic, I., V. Gruhn und M. Book, Hrsg. *Software Architecture*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 388–404. isbn: 978-3-642-23798-0.

[37] Culurciello, E., R. Etienne-Cummings und K. A. Boahen. „A biomorphic digital image sensor“. In: *IEEE Journal of Solid-State Circuits* 38.2, 2003, S. 281–294. issn: 0018-9200. doi: 10.1109/JSSC.2002.807412.

[38] Debouk, R., T. Fuhrman und J. Wysocki. „Architecture of By-Wire Systems Design Elements and Comparative Methodology“. In: *WCX SAE World Congress Experience* (3. März 2003). SAE Technical Paper Series. 400 Commonwealth Drive, Warrendale, PA, USA: SAE International, 2003. doi: 10.4271/2003-01-1291.

[39] Delbrück, T. und S.-C. Liu. „A silicon early visual system as a model animal“. In: *Vision Research* 44.17, 2004, S. 2083–2089. issn: 0042-6989. doi: 10.1016/j.visres.2004.03.021.

[40] Diorio, C., D. Hsu, M. Figueroa und R. O'Donnell. „Prolog to adaptive CMOS: from biological inspiration to systems-on-a-chip“. In: *Proceedings of the IEEE* 90.3, 2002, S. 343–344. issn: 0018-9219. doi: 10.1109/JPROC.2002.993401.

[41] Dörr, T. „Entwicklung einer heterogenen Hardware/Software-Architektur für Fail-Operational-Systeme“. Institut für Technik der Informationsverarbeitung (ITIV). Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2017.

[42] Dörr, T., T. Sandmann, F. Schade et al. „Leveraging the Partial Reconfiguration Capability of FPGAs for Processor-Based Fail-Operational Systems“. In: Hochberger, C., Hrsg. *Applied reconfigurable computing: 15th International Symposium, ARC 2019, Darmstadt, Germany, April 9-11, 2019, proceedings / Christian Hochberger, Brent Nelson, Andreas Koch, Roger Woods, Pedro Diniz (eds.)*

Bd. 11444. LNCS sublibrary: SL1 - Theoretical computer science and general issues 11444. Cham, Schweiz: Springer, 2019, S. 96–111. ISBN: 978-3-030-17226-8. DOI: 10.1007/978-3-030-17227-5.

[43] Drössler, S., M. Eichhorn, S. Holzknecht et al. „A Real-Time Capable Virtualized Information and Communication Technology Infrastructure for Automotive Systems“. In: Chakraborty, S. und J. Eberspächer, Hrsg. *Advances in Real-Time Systems*. Berlin: Springer, 2012. ISBN: 978-3-642-24349-3. DOI: 10.1007/978-3-642-24349-3_14.

[44] Dubrova, E. *Fault-Tolerant Design*. New York: Springer New York, 2013. 194 S. ISBN: 978-1-461-42113-9.

[45] Ebert, C. und J. Favaro. „Automotive Software“. In: *IEEE Software* 34.3, 2017, S. 33–39. ISSN: 0740-7459. DOI: 10.1109/MS.2017.82.

[46] Echtle, K. *Fehlertoleranzverfahren*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990. ISBN: 978-3-642-75765-5. DOI: 10.1007/978-3-642-75765-5.

[47] Eckstein, L. „Future Trends for Automotive Steering Systems“. In: *JTEKT engineering journal : JEJ / English edition* 1013E, 2016, S. 2–7. ISSN: 1881-4093.

[48] Ehrlenspiel, K., A. Kiewert, U. Lindemann und M. Mörtl. *Kostengünstig Entwickeln und Konstruieren. Kostenmanagement bei der integrierten Produktentwicklung*. 6. Aufl. VDI-Buch. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2007. ISBN: 978-3-540-74223-4. DOI: 10.1007/978-3-540-74223-4.

[49] Eilers, D. *SafeAdapt. Safe Adaptive Software for Fully Electric Vehicles*. 2017. [Zugriff am: 25. 11. 2019]. Verfügbar unter: https://www.safeadapt.eu.

[50] Epping, T. *Kanban für die Softwareentwicklung*. Informatik im Fokus. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2011. ISBN: 978-3-642-22595-6. DOI: 10.1007/978-3-642-22595-6.

[51] Flühr, H. *Avionik und Flugsicherungstechnik. Einführung in Kommunikationstechnik, Navigation, Surveillance*. 2. Aufl. Berlin, Heidelberg: Springer Vieweg, 2012. 369 S. ISBN: 978-3-642-33576-1. DOI: 10.1007/978-3-642-33576-1.

[52] Frigerio, A., B. Vermeulen und K. Goossens. „A Generic Method for a Bottom-Up ASIL Decomposition“. In: Hoshi, M. und S. Seki, Hrsg. *Developments in language theory: 22nd International Conference (DLT). Proceedings* (Tokyo, Japan, 10.–14. Sep. 2018). Bd. 11088. LNCS sublibrary. SL 1, Theoretical computer

science and general issues 11088. Cham, Schweiz: Springer, 2018, S. 12–26. ISBN: 978-3-319-98653-1. DOI: 10.1007/978-3-319-99130-6_2.

[53] Fürst, S. und M. Bechter. „AUTOSAR for Connected and Autonomous Vehicles: The AUTOSAR Adaptive Platform“. In: *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Proceedings* (Toulouse, Frankreich, 28. Juni–1. Juli 2016). Institute of Electrical and Electronics Engineers, International Federation for Information Processing und Computer Society. Piscataway, NJ, USA: IEEE, 2016, S. 215–217. ISBN: 978-1-509-03688-2. DOI: 10.1109/DSN-W.2016.24.

[54] Gasser, T. M., C. Arzt, M. Ayoubi et al. *Rechtsfolgen zunehmender Fahrzeugautomatisierung*. Hrsg. von Bundesanstalt für Strassenwesen. Bergisch Gladbach, 2012. [Zugriff am: 08. 09. 2020]. Verfügbar unter: https://www.bast.de/BASt_2017/DE/Publikationen/Berichte/unterreihe-f/2013-2012/f83.html.

[55] Gebert, J. „Automatische ReKonfiguration (AutoKonf). Teilvorhabenbeschreibung der BMW AG“. In: 2016.

[56] GENIVI Alliance, Hrsg. *GENIVI Development Platform (GDP)*. 2018. [Zugriff am: 01. 02. 2019]. Verfügbar unter: https://at.projects.genivi.org/wiki/pages/viewpage.action?pageId=11567210.

[57] Geyer, F., U. Köhler, P. Grabs et al. „Automatische rekonfigurierbare Aktorikansteuerungen für ausfallsichere automatisierte Fahrfunktionen. Gesamtvorhabensbeschreibung“. AutoKonf. 2016.

[58] Ghadhab, M., M. Kuntz, D. Kuvaiskii und C. Fetzer. „A Controller Safety Concept Based on Software-Implemented Fault Tolerance for Fail-Operational Automotive Applications“. In: Artho, C. und P. C. Ölveczky, Hrsg. *Formal Techniques for Safety-Critical Systems 4th International Workshop (FTSCS)* (Paris, Frankreich, 6.–7. Nov. 2015). 1. Aufl. Bd. 596. Communications in Computer and Information Science. Cham, Schweiz: Springer International Publishing, 2016, S. 189–205. ISBN: 978-3-319-29509-1. DOI: 10.1007/978-3-319-29510-7_11.

[59] Grabs, P. *AutoKonf. Automatische rekonfigurierbare Aktorikansteuerungen für ausfallsichere automatisierte Fahrfunktionen*. 2018. [Zugriff am: 05. 02. 2018]. Verfügbar unter: http://www.autokonf.de/.

[60] Hanssen, G. K., T. Stålhane und T. Myklebust. *SafeScrum® - Agile Development of Safety-Critical Software*. Cham, Schweiz: Springer International Publishing, 2018. 3 S. isbn: 978-3-319-99334-8. doi: 10.1007/978-3-319-99334-8.

[61] Hedenetz, B. und R. Belschner. „Brake-by-Wire Without Mechanical Backup by Using a TTP-Communication Network". In: *International Congress & Exposition* (23. Feb. 1998). SAE Technical Paper Series. 400 Commonwealth Drive, Warrendale, PA, USA: SAE International, 1998. doi: 10.4271/981109.

[62] Heerwagen, M. *Entwicklung im Wandel Agile Methoden auf dem Vormarsch*. Hrsg. von ATZ elektronik. PII: 20. 2018. doi: 10.1007/s35658-018-0020-2. [Zugriff am: 18. 07. 2019]. Verfügbar unter: https://www.springerprofessional.de/entwicklung-im-wandel-agile-methoden-auf-dem-vormarsch/15813848.

[63] Heimbold, T. *Einführung in die Automatisierungstechnik. Automatisierungssysteme, Komponenten, Projektierung und Planung*. München: Hanser, 2015. Online-Ressource. isbn: 978-3-446-44690-8.

[64] Henshaw, M., C. Siemieniuch, M. Sinclair et al. „Systems of Systems Engineering: A research imperative". In: Szakál, A., Hrsg. *IEEE International Conference on System Science and Engineering* (*ICSSE*) (Budapest, Hungary, 4.–6. Juli 2013). Institute of Electrical and Electronics Engineers. Piscataway, NJ, USA: IEEE, 2013, S. 389–394. isbn: 978-1-479-90009-1. doi: 10.1109/ICSSE.2013.6614697.

[65] Herkersdorf, A. *IT_Motive 2020 - Real-Time Capable Virtualized Information and Communication Technology Infrastructure for Automotive Systems*. 2012. [Zugriff am: 25. 11. 2019]. Verfügbar unter: https://www.ei.tum.de/lis/forschung/abgeschlossene-projekte/it-motive/.

[66] Hillenbrand, M. *Funktionale Sicherheit nach ISO 26262 in der Konzeptphase der Entwicklung von Elektrik/Elektronik Architekturen von Fahrzeugen*. Bd. 4. Steinbuch series on advances in information technology. Hannover und Karlsruhe: Technische Informationsbibliothek u. Universitätsbibliothek und KIT Scientific Publishing, 2012. 422 S. isbn: 978-3-866-44803-2.

[67] Hitt, E. F. und D. Mulcare. „Fault-Tolerant Avionics". In: Spitzer, C. R., Hrsg. *Avionics: development and implementation*. 2. Aufl. Digital avionics handbook. Boca Raton: CRC Press, 2007. isbn: 978-0-849-38442-4.

[68] Hofmann, M. „HiBord. Hoch zuverlässige und intelligente Bordnetztopologien für automatisierte Fahrzeuge“. In: 2016.

[69] Hohlfeld, B. „Vorlesung - Automotive Software Engineering“. In: 2015.

[70] INCHRON GmbH. *INCHRON Tool-Suite*. Version 2.9.0.125 Build 20181108. INCHRON GmbH, 2018.

[71] Indiveri, G. „Neuromorphic selective attention systems“. In: *IEEE International Symposium on Circuits and Systems (ISCAS)* (Bangkok, Thailand, 25.–28. Mai 2003). IEEE, Circuits and Systems Society Staff. Piscataway: IEEE, 2003, S. 770–773. ISBN: 0-7803-7761-3. DOI: 10.1109/ISCAS.2003.1205133.

[72] Industrieanlagen-Betriebsgesellschaft mbH. *V-Modell 97*. Hrsg. von Bundesministerium des Innern. 2016. [Zugriff am: 26. 06. 2019]. Verfügbar unter: https://v-modell.iabg.de/index.php/vm97-uebersicht.

[73] International Electrotechnical Commission. IEC 61508, *Functional safety of electrical/electronic/programmable electronic safety-related systems (IEC 61508:2010)*. Genf, Schweiz, 2010.

[74] International Organization for Standardization. ISO 3888, *Passenger cars – Test track for a severe lane-change manoeuvre – Part 2: Obstacle avoidance (ISO 3888-2:2011)*. Genf, Schweiz, 2011.

[75] International Organization for Standardization. ISO/IEC 33001, *Information technology – Process assessment – Concepts and terminology (ISO/IEC 33001:2015)*. 2015.

[76] International Organization for Standardization. ISO/IEC/IEEE 24765, *Systems and software engineering - Vocabulary (ISO/IEC/IEEE 24765:2017)*. Piscataway, NJ, USA, 2017.

[77] International Organization for Standardization. ISO 26262, *Road Vehicles – Functional Safety (ISO 26262:2018)*. Genf, Schweiz, 2018.

[78] Iwai, A., N. Oohashi und S. Kelly. „Experiences with Automotive Service Modeling“. In: *Proceedings of the 10th Workshop on Domain-Specific Modeling*. DSM '10. New York, NY, USA: ACM, 2010, 1:1–1:6. ISBN: 978-1-450-30549-5. DOI: 10.1145/2060329.2060337.

[79] Jaensch, M. *Modulorientiertes Produktlinien Engineering für den modellbasierten Elektrik/Elektronik-Architekturentwurf*. Bd. 8. Steinbuch series on advances in information technology. Hannover und Karlsruhe: Technische Informationsbibliothek u. Universitätsbibliothek und KIT Scientific Publishing, 2012. 2 S. isbn: 978-3-866-44863-6.

[80] Johnson, B. „Fault-Tolerant Microprocessor-Based Systems“. In: *IEEE Micro* 4.6, 1984, S. 6–21. issn: 0272-1732. doi: 10.1109/MM.1984.291277.

[81] Kaiser, R. „Virtualisierung von Mehrprozessorsystemen mit Echtzeitanwendungen“. Dissertation. Universität Koblenz-Landau, Campus Koblenz, Universitätsbibliothek, 2009. XII, 222.

[82] Kästner, D. und C. Ferdinand. „Safety Standards and WCET Analysis Tools“. In: *Embedded Real Time Software and Systems* (*ERTS2012*). Feb, Toulouse, France, 2012.

[83] Kim, J., G. Bhatia, R. Rajkumar und M. Jochim. „SAFER: System-level Architecture for Failure Evasion in Real-time Applications“. In: Kato, S., Hrsg. *IEEE 33rd Real-Time Systems Symposium* (*RTSS*) (San Juan, Puerto Rico, 4.–7. Dez. 2012). Institute of Electrical and Electronics Engineers und IEEE Computer Society. Piscataway, NJ, USA: IEEE, 2012, S. 227–236. isbn: 978-1-467-33098-5. doi: 10.1109/RTSS.2012.74.

[84] Kim, J., R. Rajkumar und M. Jochim. „Towards dependable autonomous driving vehicles“. In: *ACM SIGBED Review* 10.1, 2013, S. 29–32. doi: 10.1145/2492385.2492390.

[85] Kohn, A., M. Kasmeyer, R. Schneider et al. „Fail-operational in safety-related automotive multi-core systems“. In: *10th IEEE International Symposium on Industrial Embedded Systems* (*SIES*). *Proceedings* (Siegen, 8.–10. Juni 2015). Hrsg. von Obermaisser, R. und R. Passerone. Institute of Electrical and Electronics Engineers und IEEE Industrial Electronics Society. Piscataway, NJ, USA: IEEE, 2015, S. 1–4. isbn: 978-1-467-37711-9. doi: 10.1109/SIES.2015.7185051.

[86] Kohn, A., R. Schneider, A. Vilela et al. „Markov Chain-based Reliability Analysis for Automotive Fail-Operational Systems“. In: *SAE International Journal of Transportation Safety* 5.1, 2017. issn: 2327-5634. doi: 10.4271/2017-01-0052.

[87] Koren, I. und C. M. Krishna. *Fault-tolerant systems*. Amsterdam: Elsevier Morgan Kaufmann, 2007. isbn: 978-0-120-88525-1.

[88] Krüger, I. „Rich Services — A SOA Pattern for Dynamic Change in Cyber-physical Systems“. In: *it - Information Technology* 55.1, 2013, S. 10–18. issn: 1611-2776. doi: 10.1524/itit.2013.0002.

[89] Kugele, S., D. Hettler und S. Shafaei. „Elastic Service Provision for Intelligent Vehicle Functions“. In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)* (Maui, HI, USA, 4.–7. Nov. 2018). Institute of Electrical and Electronics Engineers und IEEE Intelligent Transportation Systems Society. Piscataway, NJ, USA: IEEE, 2018, S. 3183–3190. isbn: 978-1-728-10321-1. doi: 10.1109/ITSC.2018.8569374.

[90] Kugele, S., P. Obergfell, M. Broy et al. „On Service-Orientation for Automotive Software“. In: *IEEE International Conference on Software Architecture (ICSA)*. Proceedings (Göteborg, Schweden, 3.–7. Apr. 2017). Institute of Electrical and Electronics Engineers. Piscataway, NJ, USA: IEEE, 2017, S. 193–202. isbn: 978-1-509-05729-0. doi: 10.1109/ICSA.2017.20.

[91] Lanigan, P. E., S. Kavulya, P. Narasimhan et al. „Diagnosis in Automotive Systems: A Survey“. 2011.

[92] Laprie, J.-C. „Dependable Computing and Fault Tolerance : Concepts and Terminology“. In: *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Highlights from twenty-five years* (Pasadena, CA, 27.–30. Juni 1995). IEEE Computer Society und Laboratorie d'automatique et d'analyse des systemes. Los Alamitos, CA, USA: IEEE Computer Society Press, 1995, S. 2. isbn: 0-8186-7150-5. doi: 10.1109/FTCSH.1995.532603.

[93] Laprie, J.-C., Hrsg. *Dependability: basic concepts and terminology. In English, French, German, Italian, and Japanese*. Bd. 5. Dependable computing and fault-tolerant systems. Wien: Springer-Verlag, 1992. 265 S. isbn: 3-211-82296-8.

[94] Lauber, A., H. Guissouma und E. Sax. „Virtual Test Method for Complex and Variant-Rich Automotive Systems“. In: *IEEE International Conference on Vehicular Electronics and Safety (ICVES)* (Madrid, Spanien, 12.–14. Sep. 2018). Institute of Electrical and Electronics Engineers. Piscataway, NJ, USA: IEEE, 2018, S. 1–7. isbn: 978-1-538-63543-8. doi: 10.1109/ICVES.2018.8519599.

[95] Liu, B., V. P. Betancourt, T. Glock et al. „Model-Driven Design of Tools for Multi-Domain Systems with Loosely Coupled Metamodels“. In: *Annual IEEE International Systems Conference* (Orlando, FL, USA, 8.–11. Apr. 2019). 2019, S. 1–7. doi: 10.1109/SYSCON.2019.8836952.

[96] Lorenz, M. „Adaption und Implementierung einer Fail-Operational-Architektur für eingebettete Systeme im Automobilsektor“. Institut für Technik der Informationsverarbeitung (ITIV). Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie, 2018.

[97] Maier, M. W. „Architecting principles for systems-of-systems“. In: *Systems Engineering* 1.4, 1998, S. 267–284. issn: 1098-1241. doi: 10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D.

[98] Mariani, R., T. Kuschel und H. Shigehara. „A flexible microcontroller architecture for fail-safe and fail-operational systems“. In: European Network on High Performance and Embedded Architecture and Compilation, Hrsg. *The 5th International Conference on High Performance and Embedded Architectures and Compilers. The 2nd HiPEAC Workshop on Design for Reliability* (Pisa, Italien, 24. Jan. 2010). 2010.

[99] Matheis, J. *Abstraktionsebenenübergreifende Darstellung von Elektrik/ Elektronik-Architekturen in Kraftfahrzeugen zur Ableitung von Sicherheitszielen nach ISO 26262*. 1. Aufl. Berichte aus der Informationstechnik. Aachen: Shaker, 2010. 217 S. isbn: 978-3-832-28968-3.

[100] Mattson, T. *The Future of Many Core Computing: A tale of two processors*. 2010.

[101] Meroth, A. und B. Tolg. *Infotainmentsysteme im Kraftfahrzeug*. 1. Aufl. Vieweg Praxiswissen. Vieweg, 2008. 380 S. isbn: 978-3-834-89430-4.

[102] Mueller-Glaser, K. D., E. Sax, W. Stork et al. „Design and simulation of heterogeneous embedded systems“. In: Monteiro, J. C., R. A. d. L. Reis und W. van Noije, Hrsg. *13th Symposium on Integrated Circuits and Systems Design. Proceedings* (Manaus, Brasilien, 18.–24. Sep. 2000). Simpósio de Concepção de Circuitos Integrados u. a. Los Alamitos, CA, USA: IEEE Computer Society, 2000, S. 385–390. isbn: 0-7695-0843-X. doi: 10.1109/SBCCI.2000.876059.

[103] Nelson, V. P. „Fault-tolerant computing. Fundamental concepts“. In: *Computer* 23.7, 1990, S. 19–25. issn: 0018-9162. doi: 10.1109/2.56849.

[104] Nolting, S., G. Payá-Vayá, F. Giesemann et al. „Dynamic self-reconfiguration of a MIPS-based soft-core processor architecture“. In: *Journal of Parallel and Distributed Computing*, 2017. ISSN: 07437315. DOI: 10.1016/j.jpdc.2017.09.013.

[105] Oertel, M. „Architectures of High Performance Computing“. In: *9th Vector Congress 2018*. Stuttgart, 2018.

[106] Pischinger, S. und U. Seiffert. *Vieweg Handbuch Kraftfahrzeugtechnik*. 8. Aufl. ATZ / MTZ-Fachbuch. Wiesbaden: Springer Vieweg, 2016. 1425 S. ISBN: 978-3-658-09528-4. DOI: 10.1007/978-3-658-09528-4.

[107] Prasad, V. B. „Fault tolerant digital systems“. In: *IEEE Potentials* 8.1, 1989, S. 17–21. ISSN: 0278-6648. DOI: 10.1109/45.31576.

[108] Profanter, S., A. Tekat, K. Dorofeev et al. „OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols“. In: *IEEE International Conference on Industrial Technology (ICIT)* (Melbourne, Australia, 13.–15. Feb. 2019). IEEE, 2019, S. 955–962. ISBN: 978-1-538-66376-9. DOI: 10.1109/ICIT.2019.8755050.

[109] Reif, K., C. Kannenberg und E. Lange, Hrsg. *Batterien, bordnetze und vernetzung*. 1. Auflage. Bosch Fachinformation Automobil. Wiesbaden: Vieweg+Teubner, 2010. 1225 S. ISBN: 978-3-834-89713-8. DOI: 10.1007/978-3-8348-9713-8.

[110] On-Road Automated Driving (ORAD) committee, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. 400 Commonwealth Drive, Warrendale, PA, USA, 2018.

[111] Royce, W. W. „Managing the Development of Large Software Systems: Concepts and Techniques“. In: *Proceedings of the 9th International Conference on Software Engineering*. ICSE '87. Los Alamitos, CA, USA: IEEE Computer Society Press, 1987, S. 328–338. ISBN: 0-89791-216-0.

[112] Saighi, S., Y. Bornat, J. Tomas und S. Renaud. „Neuromimetic ICs and system for parameters extraction in biological neuron models“. In: *IEEE International Symposium on Circuits and Systems. Proceedings* (Kos, Griechenland, 21.–24. Mai 2006). IEEE Circuits and Systems Society. Piscataway, NJ, USA: IEEE Service Center, 2006, S. 5. ISBN: 0-7803-9389-9. DOI: 10.1109/ISCAS.2006.1693557.

[113] Sangiovanni-Vincentelli, A. und M. Di Natale. „Embedded System Design for Automotive Applications“. In: *Computer* 40.10, 2007, S. 42–51. issn: 0018-9162. doi: 10.1109/MC.2007.344.

[114] Sattler, K. „Methodik für den Systemtest in der integralen Fahrzeugsicherheit“. Fakultät für Elektrotechnik und Informationstechnik. Dissertation. Magdeburg: Otto-von-Guericke-Universität, 2015. XIX, 181 S.

[115] Savaresi, S. M. und M. Tanelli. *Active braking control systems design for vehicles*. Advances in industrial control. London: Springer, 2010. isbn: 978-1-849-96350-3.

[116] Sax, E., Hrsg. *Automatisiertes Testen eingebetteter Systeme in der Automobilindustrie*. München: Hanser, 2008. 219 S. isbn: 978-3-446-41635-2. doi: 10.3139/9783446419018.

[117] Schäuffele, J. „E/E Architectural Design and Optimization using PREEvision“. In: *SAE World Congress and Exhibition* (12. Apr. 2019). SAE Technical Paper Series. 400 Commonwealth Drive, Warrendale, PA, USA: SAE International, 2016. doi: 10.4271/2016-01-0016.

[118] Schäuffele, J. und T. Zurawka. *Automotive Software Engineering. Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen*. 6. Auflage. ATZ / MTZ-Fachbuch. Wiesbaden: Springer Vieweg, 2016. 359 S. isbn: 978-3-658-11814-3. doi: 10.1007/978-3-658-11815-0.

[119] Schnellbach, A. „Fail-operational automotive systems“. Institute of Automotive Engineering. Dissertation. Graz: University of Technology, 2016.

[120] Scholz, P. *Softwareentwicklung eingebetteter Systeme*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. isbn: 978-3-540-23405-0. doi: 10.1007/3-540-27522-3.

[121] Schwaber, K. und J. Sutherland. *The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game*. 2017. [Zugriff am: 27. 06. 2019]. Verfügbar unter: https://www.scrumguides.org/.

[122] Shalf, J., K. Asanovic, D. Patterson et al. „The MANYCORE Revolution: Will HPC lead or follow?“ In: *SciDAC Review* 2009.Fall, 2009, S. 40–49.

[123] Simoni, M. F., G. S. Cymbalyuk, M. E. Sorensen et al. „A multiconductance silicon neuron with biologically matched dynamics“. In: *IEEE Transactions on Biomedical Engineering* 51.2, 2004. Feb, S. 342–354. ISSN: 0018-9294. DOI: 10.1109/TBME.2003.820390.

[124] Sinha, P. „Architectural design and reliability analysis of a fail-opera-tional brake-by-wire system from ISO 26262 perspectives“. In: *Reliability Engineering & System Safety* 96.10, 2011, S. 1349–1359. ISSN: 09518320. DOI: 10.1016/j.ress.2011.03.013.

[125] Sommer, S., A. Camek, K. Becker et al. „RACE. A Centralized Platform Computer Based Architecture for Automotive Applications“. In: *IEEE International Electric Vehicle Conference (IEVC)* (Santa Clara, CA, USA). IEEE, 2013, S. 1–6. ISBN: 978-1-479-91451-7. DOI: 10.1109/IEVC.2013.6681152.

[126] Staron, M. *Automotive software architectures. An introduction*. Cham, Schweiz: Springer, 2017. 237 S. ISBN: 978-3-319-58609-0.

[127] Streichert, T. und M. Traub. *Elektrik/Elektronik-Architekturen im Kraftfahrzeug. Modellierung und Bewertung von Echtzeitsystemen*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. 1293 S. ISBN: 3-642-25478-0. DOI: 10.1007/978-3-642-25478-9.

[128] Temple, C. und A. Vilela. „Fehlertolerante Systeme im Fahrzeug – von fail-safe zu fail-operational“. In: *Elektronik automotive* 2014, 2014. ISSN: 1614-0125.

[129] The Object Management Group, Hrsg. *Data Distribution Service (DDS)*. USA, 2015. [Zugriff am: 01. 02. 2019]. Verfügbar unter: https://www.omg.org/spec/DDS/1.4/.

[130] Traub, M., A. Maier und K. L. Barbehön. „Future Automotive Architecture and the Impact of IT Trends“. In: *IEEE Software* 34.3, 2017, S. 27–32. ISSN: 0740-7459. DOI: 10.1109/MS.2017.69.

[131] Varanasi, B. „Continuous Integration“. In: Apress, Hrsg. *Introducing Gradle*. Berkeley, CA: Apress, 2015, S. 111–128. ISBN: 978-1-4842-1032-1. DOI: 10.1007/978-1-4842-1031-4_9.

[132] Vector Informatik GmbH. *PREEvision – Modellbasierte Elektrik-/Elektronik-Entwicklung*. 2018. [Zugriff am: 07. 02. 2018]. Verfügbar unter: https://vector.com/vi_preevision_de.html.

[133] Völker, L. *Scalable service-Oriented MiddlewarE over IP*. 2018. [Zugriff am: 01. 02. 2019]. Verfügbar unter: http://some-ip.com/.

[134] Wang, Q., J. Mao und H.-y. Wei. „Reliability Analysis of Multi-rotor UAV Based on Fault Tree and Monte Carlo Simulation". In: Tan, J., F. Gao und C. Xiang, Hrsg. *Advances in Mechanical Design: Proceedings of the 2017 International Conference on Mechanical Design (ICMD2017) / Jianrong Tan, Feng Gao, Changle Xiang*. Bd. 55. Mechanisms and Machine Science 55. Singapur: Springer, 2017, S. 1525–1534. ISBN: 978-9-811-06552-1. DOI: 10.1007/978-981-10-6553-8_100.

[135] Wanner, D., A. Stensson Trigell, L. Drugge und J. Jerrelind. „Survey on fault-tolerant vehicle design". In: *26th Electric Vehicle Symposium 2012. Los Angeles, California, USA, 6 - 9 May 2012*. Red Hook, NY: Curran, 2013. ISBN: 978-1-622-76421-1.

[136] Weber, J., Hrsg. *Automotive Development Processes*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. ISBN: 978-3-642-01252-5. DOI: 10.1007/978-3-642-01253-2.

[137] Weber, S. *Agile in Automotive – State of Practice 2015*. Hrsg. von KUGLER MAAG CIE GmbH. Kornwestheim: Sergej Weber, 2015. Verfügbar unter: https://www.kuglermaag.com/fileadmin/05_CONTENT_PDF/2-22_agile-automotive_survey-2015.pdf.

[138] Winzker, M. *Elektronik für Entscheider. Grundwissen für Wirtschaft und Technik*. Wiesbaden: Springer Fachmedien Wiesbaden, 2017. 1223 S. ISBN: 978-3-8348-0616-1.

[139] Xilinx. *Considerations Surrounding Single Event Effects in FPGAs, ASICs, and Processors*. WP402. Version 1.0.1. 2012. [Zugriff am: 12. 02. 2021]. Verfügbar unter: https://www.xilinx.com/support/documentation/white_papers/wp402_SEE_Considerations.pdf.

[140] Xilinx. *Device Reliability Report. First Half 2019*. UG116. Version V10.11. 2019. [Zugriff am: 01. 11. 2019]. Verfügbar unter: https://www.xilinx.com/support/documentation/user_guides/ug116.pdf.

[141] Xilinx. *Zynq UltraScale+ Device. Technical Reference Manual*. UG1085. Version 2.2. 2020. [Zugriff am: 16. 02. 2021]. Verfügbar unter: https://www.xil

inx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf.

[142] Yuyan, C., L. Ting, W. Jian et al. „Fuzzy dynamic fault tree analysis for electro-mechanical actuator based on algebraic model with common-cause failures“. In: *Automatic Control and Computer Sciences* 50.2, 2016, S. 80–90. issn: 0146-4116. doi: 10.3103/S0146411616020024.

[143] Zou, Q., Y. Bornat, J. Tomas et al. „Real-time simulations of networks of Hodgkin–Huxley neurons using analog circuits“. In: *Neurocomputing* 69.10, 2006. Computational Neuroscience: Trends in Research 2006, S. 1137–1140. issn: 0925-2312. doi: 10.1016/j.neucom.2005.12.061.

Eigene Veröffentlichungen

Konferenz- und Journalbeiträge

[144] Oszwald, F., R. Bertelo, M. Gericota und J. Becker. „Evaluation Methodologies in the Development of Dynamically Reconfigurable Systems in the Automotive Industry“. In: *SAE International Journal of Advances and Current Practices in Mobility* 2.5, 2020. ISSN: 2641-9637. DOI: 10.4271/2020-01-1363.

[145] Oszwald, F., P. Obergfell, B. Liu et al. „Model-Based Design of Service-Oriented Architectures for Reliable Dynamic Reconfiguration“. In: *SAE International Journal of Advances and Current Practices in Mobility* 2.5, 2020. ISSN: 2641-9637. DOI: 10.4271/2020-01-1364.

[146] Obergfell, P., J. Becker, F. Oszwald und E. Sax. „Dynamic Aspects of Centralized Automotive Software and System Architectures“. In: *Future Automotive HW/SW Platform Design (Dagstuhl Seminar 19502)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, S. 45–46.

[147] Orlov, S., M. Korte, F. Oszwald und P. Vollmer. „Automatically reconfigurable actuator control for reliable autonomous driving functions (AutoKonf)“. In: Pfeffer, P. E., Hrsg. *10th International Munich Chassis Symposium 2019. Chassis.tech plus* (München, 25.–26. Juni 2019). 1. Aufl. Proceedings. Springer Vieweg, 2020, S. 355–368. ISBN: 978-3-658-26435-2. DOI: 10.1007/978-3-658-26435-2_26.

[148] Bayerische Motoren Werke. *AutoKonf - Automatische rekonfigurierbare Aktorikansteuerungen für ausfallsichere automatisierte Fahrfunktionen - Teilprojekt: Anforderungen und Vernetzungskonzept für eine rekonfigurierbare Steuerungsplattform. Schlussbericht AutoKonf : Laufzeit: 01.10.2016-31.09.2019*. München, 2019. DOI: 10.2314/KXP:1736141406.

[149] Chamas, M., W. Hopfensitz, P. Obergfell et al. „Entwicklung eines Informationsmodells für eine Hochintegrationsplattform im Automotive-Umfeld“. In: Schulze, S. O., C. Tschirner, R. Kaffenberger und S. Ackva, Hrsg. *Tag des Systems Engineering* (München, 6.–8. Nov. 2019). Gesellschaft für Systems Engineering, 2019. isbn: 978-3-981-88055-7.

[150] Oszwald, F., P. Obergfell, M. Meseth et al. „Establishing and Enhancing Agility with Model-based Systems Engineering“. In: *Proceedings of the 20th International Congress Electronics in Vehicles (ELIV)* (Bonn, 16.–17. Okt. 2019). 2019. doi: 10.5445/IR/1000130224.

[151] Oszwald, F., P. Obergfell, M. Traub und J. Becker. „Reliable Fail-Operational Automotive E/E-Architectures by Dynamic Redundancy and Reconfiguration“. In: *Proceedings of the 32nd IEEE International System-on-Chip Conference (SOCC)* (Singapur, 3.–6. Sep. 2019). 2019, S. 203–208. doi: 10.1109/SOCC46988.2019.1570547977.

[152] Obergfell, P., F. Oszwald, M. Traub und E. Sax. „View-point-Based Methodology for Adaption of Automotive E/E-Architectures“. In: *IEEE International Conference on Software Architecture Companion (ICSA-C)* (Seattle, WA, USA, 30. Apr.–4. Mai 2018). Piscataway, NJ, USA: IEEE, 2018, S. 128–135. isbn: 978-1-538-66585-5. doi: 10.1109/ICSA-C.2018.00041.

[153] Obergfell, P., M. Traub, F. Oszwald und E. Sax. „Agiles Automotive E/E-Systems Engineering“. In: VDI Wissensforum GmbH, Hrsg. *ELIV-MarketPlace. 8. Internationaler VDI-Kongress: weltweit einzigartiger Elektronik-Kongress für Pkw-, Nfz- und Off-Highway-Anwendungen*. E/E in PKW, Nutzfahrzeugen und mobilen Arbeitsmaschinen (Baden-Baden, 16.–17. Okt. 2018). VDI-Berichte 2338, CD-ROM. Düsseldorf: VDI Verlag GmbH, 2018, S. 47–58. isbn: 978-3-18-092338-3.

[154] Oszwald, F., J. Becker, P. Obergfell und M. Traub. „Dynamic Reconfiguration for Real-Time Automotive Embedded Systems in Fail-Operational Context“. In: *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (Vancouver, BC, Kanada, 21.–25. Mai 2018). 2018, S. 206–209. isbn: 978-1-538-65555-9. doi: 10.1109/IPDPSW.2018.00039.

[155] Oszwald, F., P. Obergfell, M. Traub und J. Becker. „Using Simulation Techniques within the Design of a Reconfigurable Architecture for Fail-Operational Real-Time Automotive Embedded Systems“. In: *IEEE International Symposium*

on Systems Engineering (Rom, Italien, 1.–3. Okt. 2018). 2018, S. 1–3. ISBN: 978-1-538-64446-1. DOI: 10.1109/SysEng.2018.8544451.

Patentanmeldungen

[P1] Schreckling, D. und F. Oszwald. „Verfahren und Vorrichtung zur Selbstdiagnose eines Umfeldsensors“. B60W 50/02 (2012.01). DE 10 2019 127 050 A1 (DE). Bayerische Motoren Werke Aktiengesellschaft, 80809 München, DE. 8. Apr. 2021.

[P2] Orlov, S., J. Eckstein, N. Decius et al. „Kontrollsystem für ein Kraftfahrzeug und Verfahren zur Fehlerdiagnose bei einem Kontrollsystem“. B60W 50/02 (2012.01). DE 10 2018 213 182 A1 (DE). Bayerische Motoren Werke Aktiengesellschaft, 80809 München, DE, HELLA GmbH & Co. KGaA, 59557 Lippstadt, DE, Intedis GmbH & Co. KG, 97084 Würzburg, DE und Robert Bosch GmbH, 70469 Stuttgart, DE. 13. Feb. 2020.

[P3] Oszwald, F. „Method and system for Fail-Operational Handover of Service During Vehicle Operation“. G05D 1/02 (2020.01). EP 3 726 330 A1. Bayerische Motoren Werke Aktiengesellschaft, 80809 München, DE. 21. Okt. 2020.

[P4] Oszwald, F. „Method and System for Preserving Consistency of States During Fail-Operational Handover“. G06F 11/07 (2006.01). EP 3 726 384 A1. Bayerische Motoren Werke Aktiengesellschaft, 80809 München, DE. 21. Okt. 2020.

[P5] Oszwald, F. und J. Gebert. „Predictive reconfiguration for fail-operational automotive applications“. 19-1641 PIF. Oszwald, F. 2020.

[P6] Schreckling, D. und F. Oszwald. „Verfahren und Steuereinheit zum Betreiben einer Fahrfunktion“. B60W 50/029 (2012.01). DE 10 2019 113 963 A1 (DE). Bayerische Motoren Werke Aktiengesellschaft, 80809 München, DE. 26. Nov. 2020.

[P7] Orlov, S., J. Eckstein, N. Decius et al. „Kontrollsystem für ein Kraftfahrzeug, Kraftfahrzeug, Verfahren zur Kontrolle eines Kraftfahrzeugs, Computerprogrammprodukt und computerlesbares Medium“. B60W 50/023 (2012.01). DE 10 2018 112 254 A1 (DE). Bayerische Motoren Werke Aktiengesellschaft, 80809 München, DE, HELLA GmbH & Co. KGaA, 59557 Lippstadt, DE, Intedis GmbH & Co. KG, 97084 Würzburg, DE und Robert Bosch GmbH, 70469 Stuttgart, DE. 28. Nov. 2019.

Weitere Veröffentlichungen ohne direkten Bezug

[W1] Oszwald, F., R. Debon, T. Barmeyer und M. Walessa. „Verfahren zur Fahrerinformation“. B60R 1/00 (2006.01). WO2011047756 (DE). Bayerische Motoren Werke Aktiengesellschaft, 80809 München, DE. 13. Mai 2015.

[W2] Oszwald, F. und S. Spiekermann. *MIDIAS: Integriertes optisches Mikrosystem mit dreidimensionaler Informationsverarbeitung für Fahrerassistenzsysteme. Abschlussbericht*. 2010. doi: 10.2314/GBV:667091777.

[W3] Spiekermann, S., F. Oszwald, A. Zeller und D. Rossberg. „Ein Kamerasystem der neuen Generation. Bildgebung und Entfernungsmessung in einem 2D/3D Kamerasystem“. In: Pischinger, S., Hrsg. *19. Aachener Kolloquium Fahrzeug- und Motorentechnik* (Aachen, 4.–6. Okt. 2010). 2. Aachen, 2010, S. 1677–1689.

[W4] Ulrich, M. „Identifikation und Analyse von Funktionen zur Unterstützung des Fahrers, basierend auf Außenspiegelkameras“. Masterarbeit. München: Hochschule München, 2010.

[W5] Ehmanns, D., J. Aulbach, T. Strobel et al. „Aktive Sicherheit und Fahrerassistenz“. In: *Der neue BMW 7er. Entwicklung und Technik*. 1. Aufl. ATZ/MTZ-Typenbuch. Wiesbaden: Vieweg + Teubner, 2009, S. 150–158. isbn: 978-3-834-80773-1.

[W6] Ehmanns, D., J. Aulbach, T. Strobel et al. „Active Safety and Driver Assistance. The new BMW 7 Series offers a comprehensive package of active safety.“ In: *ATZ-Extra worldwide* 2008, 11 2008: *ATZextra worldwide. The New BMW 7 Series*, S. 114–119. doi: 10.1365/s40111-008-0114-6.

[W7] Ehmanns, D., J. Aulbach, T. Strobel et al. „Aktive Sicherheit und Fahrerassistenz. Der neue BMW 7er bietet ein umfangreiches Paket an Systemen der aktiven Sicherheit und Fahrerassistenz, das neue Maßstäbe setzt.“ In: *ATZ-Extra* 2008, 08 2008: *ATZextra. Der neue BMW 7er*, S. 114–119. doi: 10.1365/s35778-008-0174-2.

[W8] Wahl, E., F. Oszwald, A. Russ et al. „Evaluation of automotive vision systems: Innovations in the development of video-based adas“. In: International Federation of Automotive Engineering Societies, Hrsg. *FISITA World Automotive Congress;* (München, 14.–19. Sep. 2008). Hrsg. von ATZ / ATZautotechnology. 1. Aufl. Testing & Simulation 7. Wiesbaden: Springer Automotive Media/GWV Fachverl., 2008.

[W9] Zeller, A., S. Wilhelm, C. Reuter et al. „Blick für das Besondere. Multikamera-System sorgt für Überblick in Park- und Rangiersituationen“. In: *Elektronik automotive* 2008, 2008. issn: 1614-0125.

[W10] Spiekermann, S. „Tiefenbildvisualisierung für eine kombinierte 2D-/3D-Kamera in der automotiven Verwendung“. Institut für Computervisualistik. Diplomarbeit. Koblenz: Universität Koblenz-Landau, 2007.

[W11] Wessely, T. „Untersuchungen zum Einfluss von Videokompressionsverfahren auf die Qualität ausgewählter Methoden der Bildauswertung“. Elektro- und Informationstechnik. Diplomarbeit. Amberg-Weiden: FH Amberg-Weiden, 2006.

[W12] Oszwald, F., A. Wedler und A. Schiele. „Development of a Bioinspired Robotic Insect Leg“. In: ESA/ESTEC, Hrsg. *8th ESA Workshop on Advanced Space Technologies for Robotics and Automation. ASTRA 2004 Workshop* (2.–4. Nov. 2004). Hrsg. von Schiele, A. Noordwijk, Niederlande, 2004. doi: 10.13140/RG.2.2.23333.76009.

STEINBUCH SERIES ON ADVANCES IN INFORMATION TECHNOLOGY

Institut für Technik der Informationsverarbeitung
Karlsruher Institut für Technologie (KIT) | ISSN 2191-4737

Band 1 OLIVER SANDER
Skalierbare adaptive System-on-Chip-Architekturen für Inter-Car und Intra-Car Kommunikationsgateways.
ISBN 978-3-86644-601-4

Band 2 BENJAMIN GLAS
Trusted Computing für adaptive Automobilsteuergeräte im Umfeld der Inter-Fahrzeug-Kommunikation.
ISBN 978-3-86644-602-1

Band 3 RALF NÖRENBERG
Effizienter Regressionstest von E/E-Systemen nach ISO 26262.
ISBN 978-3-86644-135-4

Band 4 MARTIN HILLENBRAND
Funktionale Sicherheit nach ISO 26262 in der Konzeptphase der Entwicklung von Elektrik/Elektronik Architekturen von Fahrzeugen.
ISBN 978-3-86644-803-2

Band 5 MATTHIAS HEINZ
Modellbasierte Entwicklung und Konfiguration des zeitgesteuerten FlexRay Bussystems.
ISBN 978-3-86644-816-2

Band 6 Nicht erschienen

Band 7 CHRISTOPH SCHMUTZLER
Hardwaregestützte Energieoptimierung von Elektrik/Elektronik-Architekturen durch adaptive Abschaltung von verteilten, eingebetteten Systemen.
ISBN 978-3-86644-875-9

Band 8 MARTIN JAENSCH
Modulorientiertes Produktlinien Engineering für den modellbasierten Elektrik/Elektronik-Architekturentwurf.
ISBN 978-3-86644-863-6

Band 9 MAHTAB NIKNAHAD
Using Fine Grain Approaches for highly reliable Design of FPGA-based Systems in Space.
ISBN 978-3-7315-0038-4

Band 10 NICO ADLER
Modellbasierte Entwicklung funktional sicherer Hardware nach ISO 26262.
ISBN 978-3-7315-0442-9

Die Bände sind unter www.ksp.kit.edu als PDF frei verfügbar oder als Druckausgabe bestellbar.

Band 11 FLORIAN OSZWALD
Dynamische Rekonfigurationsmethodik für zuverlässige, echtzeitfähige Eingebettete Systeme in Automotive.
ISBN 978-3-7315-1102-1

STEINBUCH SERIES ON ADVANCES
IN INFORMATION TECHNOLOGY

Karlsruher Institut für Technologie (KIT)
Institut für Technik der Informationsverarbeitung

In der Fahrzeugentwicklung sind die drei Größen Kosten, Gewicht und verfügbarer Bauraum entscheidende Stellhebel. Dadurch vollzieht sich in der Fahrzeugtechnik ein Wechsel weg von hardware- hin zu softwareorientierten Lösungen. Des Weiteren werden im Bereich des hoch- und vollautomatisierten Fahrens Systeme benötigt, die eine Anforderung an ein Fail-Operational-Verhalten erfüllen. Derzeit kommen dynamisch rekonfigurierbare Systeme im Automobil noch nicht zum Einsatz und es gibt auch kein Vorgehensmodell, wie diese Systeme zu entwickeln sind. Aus diesem Grund liegt der Schwerpunkt dieser Dissertation auf der Erforschung von Methoden und Ansätzen für die Entwicklung von dynamisch rekonfigurierbaren Systemen.

Die Herausforderung ergibt sich aus der Kombination von zwei dominierenden Architekturtreibern. Einer davon ist das autonome Fahren als ein gesetztes Ziel in der Automobilindustrie, ein weiterer ist die funktionale Hochintegration auf zentralen Rechner-Plattformen. Unter Berücksichtigung dieser beiden Architekturtreiber wird die dynamische Rekonfiguration eingeordnet und erforscht.

ISSN 2191-4737
ISBN 978-3-7315-1102-1