Modellbasierte Entwicklung und Optimierung flexibler zeitgesteuerter Architekturen im Fahrzeugserienbereich

Julian Broy

Karlsruher Institut für Technologie
Fakultät für Elektro- und Informationstechnik
Institut für Technik der Informationsverarbeitung

Modellbasierte Entwicklung und Optimierung flexibler zeitgesteuerter Architekturen im Fahrzeugserienbereich

Zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEURS

von der Fakultät für

Elektrotechnik und Informationstechnik

des Karlsruher Institut für Technologie

genehmigte

DISSERTATION

von

Dipl. Inf. (Univ.) Julian Broy

geb. in: Dachau

Tag der mündlichen Prüfung:	3. März 2010
Hauptreferent:	Prof. Dr.-Ing. Klaus D. Müller-Glaser
Korreferent:	Prof. Dipl.-Ing. Dr. Wolfgang Pree

Für

meine Frau Mariana

meine Eltern Karin und Manfred

meine Großmutter Theresia

Danksagung

Diese Arbeit entstand im Rahmen einer dreijährigen Forschungsarbeit mit dem Institut für Technik der Informationsverarbeitung der Fakultät für Elektrotechnik der Universität Karlsruhe und dem Forschungs- und Entwicklungszentrum Weissach der Dr. Ing. h.c. F. Porsche AG. Ich möchte mich herzlich bei meinem betreuenden Doktorvater und Aufgabensteller Prof. Klaus D. Müller-Glaser für die Übernahme des Erstgutachtens dieser Forschungarbeit bedanken. Sein exzellentes Fachwissen und die ausgezeichnete Betreuung haben äußerst zum Gelingen der Arbeit beigetragen. Weiterhin danke ich Prof. Wolfgang Pree für die Übernahme des Zweitgutachtens. In gleicher Weise gebührt mein Dank meinem vormaligen Abteilungsleiter Hans Weiner und meinem Fachabteilungsleiter Rüdiger Roppel für die Bereitschaft diese Forschungstätigkeit im Bereich der Serienentwicklung der Vernetzung mit einem Stipendium zu fördern und auszubauen.

Die Vollendung der Arbeit wäre ohne die hilfreiche Unterstützung von einer Reihe an Leuten nicht möglich gewesen. Im Speziellen danke ich dafür Jürgen Bortolazzi, Martin Döring, Klaus Echtle, Georg Färber, Daniel Gebauer, Thorsten Koch, Matthias Kühlewein, Dietmar Luz, Paul Milbredt, Nicolas Navet, Clemens Reichmann und Rolf Zöller.

Im Besonderen möchte ich an dieser Stelle meiner Mutter Karin für das Korrekturlesen der Arbeit und meinem Vater Manfred für die aufgebrachte Geduld in unzähligen gemeinsamen Diskussionen im Umfeld des Automotive Software Engineering innerhalb der letzten Jahre danken. Sein einzigartiges Fachwissen hat mich in meiner Tätigkeit bestärkt und meiner Forschungsarbeit immer wieder neue Impulse verliehen. Mein Dank gilt auch den Doktorandenkollegen bei Porsche, insbesondere Carsten Bock, Jan Meinel und Matthias Wiese. Aus anfänglicher Kollegschaft wurde Freundschaft.

Wertvolle Beiträge haben die Studenten Marcel Flammer, Thomas Mair, Thibault Schneider und Thomas Surmann geleistet. Vielen Dank für den unermüdlichen Einsatz und die fantastische Zusammenarbeit.

Mehrere Firmen haben die Arbeit in unterschiedlichen Bereichen unterstützt. Mein Dank gilt Aquintos GmbH, dSpace GmbH, Mathworks Inc., TTTech-Automotive GmbH und Vector Informatik GmbH.

Abschließend möchte ich mich nochmals bei meinem engsten Umfeld bedanken für die Bereitschaft mich in jeder Phase der Arbeit zu unterstützen. Im Besonderen danke

ich meiner lieben Frau Mariana für die ungebrochene Bereitschaft unser Privatleben in den letzten Jahren in weiten Bereichen dem Interesse dieser Forschungsarbeit unterzuordnen. Gleichzeitig bedanke ich mich bei meinen Eltern Karin und Manfred, dass sie mir meine Ausbildung ermöglicht haben und mir gemeinsam mit meinen Schwestern Verena und Nora sowie meiner Großmutter Theresia in allen Phasen der Promotion die Daumen gedrückt haben. An dieser Stelle gebührt der Dank auch allen weiteren Personen, die zum Gelingen der Arbeit beigetragen haben.

Zusammenfassung

Der kontinuierliche Anstieg des Umfangs elektrisch/elektronischer Funktionen im Automobil zur Realisierung von steuerungs- oder regelungstechnischen Systemen stellt eine komplexe Herausforderung an einen Fahrzeughersteller dar. Der Prozess der Substitution mechanischer durch elektrische Komponenten hat in den letzten Jahren zu einem kontinuierlich wachsenden Anteil an Software-Funktionalität im Fahrzeug geführt. Diese ist verteilt auf eine Menge an interagierenden Elektronikkomponenten, genannt Steuergeräte.

Die Systemvernetzung bietet ein etabliertes technisches Mittel zur Verknüpfung von Steuergerätefunktionen. Bisher wurde dieser Bereich vorrangig durch ereignisgesteuerte Kommunikationskonzepte auf Basis der Feldbustechnologie CAN realisiert. Durch die Definition und den geplanten Umstieg auf hochperformante flexible zeitgesteuerte Netzwerktechnologien ergeben sich zukünftig Möglichkeiten um verteilte regelungstechnische Systeme hochpräzise zu realisieren.

Diese Arbeit fokussiert sich auf die Analyse und den Entwurf einer adaptierten Entwicklungsmethodik zur Integration verteilter Realzeitsysteme mit flexibler zeitgesteuerter Bussystemarchitekturen im Automobil. Der Schwerpunkt liegt auf der Integration FlexRay-basierter Architekturen unter Berücksichtigung serienrelevanter Anforderungen des Fahrzeugherstellers. Auf der Basis der Analyse existierender Ansätze wird die neue modellbasierte Entwicklungsmethode FlexZOOMED zur vollständigen Spezifikation von FlexRay-Systemen innerhalb zu entwickelnder Elektrik-/Elektronikarchitekturen vorgestellt. Mithilfe dieses Ansatzes können die Eigenschaften eines FlexRay-Busses hinsichtlich allgemeiner Gütekriterien einer Elektrik-/Elektronikarchitektur und in Bezug auf busspezifische Qualitätsmerkmale analysiert werden. Ein weiterer Vorteil der Formalisierung von FlexRay durch Metamodellierung zur Optimierung des komplexen Netzwerkkonfigurationsprozesses wird dabei herausgearbeitet. Die Tauglichkeit der entwickelten Designmethodik wird durch Applikation auf Basis einer praxisnahen Fallstudie validiert.

Abstract

The continuous increase of functionality in the field of electrical/electronic applications by closed loop control systems for automotive systems poses a complex challenge to vehicle manufacturers. In recent years the process of substituting mechanical by electrical components led to a steadily growth of software based functionality in vehicles with solutions distributed over networks of interacting electronic components, the control units.

Fieldbus systems provide a powerful concept in creating distributed applications for electronic control units (ECUs). Initially, such networks of ECUs were based on the event-triggered CAN fieldbus technology. New solutions and functionalities in terms of high-performance closed loop control systems are currently under development. They are enabled by the planned next step to high-performance flexible time-triggered network technologies as specified by the automotive network communication protocol under development by the FlexRay Consortium.

This work focuses on the analysis and design of an adapted development methodology for the integration of distributed real-time systems with flexible time-triggered bus architectures into automotive systems. The emphasis is on the integration of FlexRay based electrical electronic architectures by taking into account the constraints and requirements of the vehicle manufacturing process. Based on the analysis of existing approaches, the new methodology FlexZOOMED is introduced to support the comprehensive specification and design of FlexRay systems as part of electrical/electronic automotive architectures. This approach supports the analysis of the specific factors and properties of FlexRay bus systems in electrical electronic architectures regarding general quality criteria. The advantage of the formalization of FlexRay by meta modeling concepts is evaluated in terms of an optimization strategy and a semi-automated network configuration process. The suitability of the described design methodology is validated by a practical case study.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Konzept der Arbeit

Im Zentrum der Untersuchung der Anforderungen an zukünftige vernetzte eingebettete Systeme im Automobil werden die Leistungsfähigkeit und die Grenzen aktuell eingesetzter Vernetzungstechnologien analysiert.

Erweitert wird die Untersuchung um das flexibel zeitgesteuerte Bussystem FlexRay. In dem Zusammenhang erfolgt eine Erfassung der individuellen Anforderungen dieses flexiblen zeitgesteuerten Bussystems hinsichtlich Entwicklung und Pflege im Serieneinsatz.

Identifizierte Lücken im Entwurfsprozess von FlexRay-Netzwerken und deren Integration innerhalb von E/E-Systemarchitekturen werden in einer modellgestützte Methode für die FlexRay-Systementwicklung geschlossen.

Die Tragfähigkeit des entwickelten Konzepts wird durch Implementierung und Analyse einer seriennahen Fallstudie validiert.

Stichwörter:

{*Requirements Engineering, System Design, System Maintenance, Optimization, Automotive Bus Systems, Modelbased Development, Domain Specific Modeling, EE Architecture, FlexRay*}

Erweiterte Zusammenfassung:

Die modellbasierte Entwicklung flexibler zeitgesteuerter Architekturen ist ein Ansatz zum integrierten Entwurf, der Analyse und Optimierung von zeitgesteuerten Feldbussystemen innerhalb der Serienfahrzeug-Elektrik/Elektronik. Dieses Vorgehen verknüpft dabei die Vorteile der modellbasierten Elektrik/Elektronikentwicklung im Fahrzeug mit den spezifischen Aufgaben der Integration einer flexiblen zeitgesteuerten Netzwerktechnologie im Fahrzeug. Dazu zählt die durchgängige integrierte Modellierung der Systemanforderungen bezogen auf Fahrzeugausstattungsmerkmale, der elektronischen Fahrzeugfunktionen, den Komponenteneigenschaften (Hard-/Software), speziellen Fahrzeugvorgaben (Geometrie und Energieversorgung) und deren Einbettung in unterschiedlichen Fahrzeugvarianten.

FlexRay bildet eine neue leistungsfähige zeitgesteuerte Feldbustechnologie zur Vernetzung von Steuergeräten im Fahrzeug. Neben der methodischen Bewältigung der Zeitsteuerung stellt der komplexe Konfigurationsprozess eines FlexRay-Systems den Automobilhersteller vor neue Herausforderungen.

Die modellbasierte Entwicklung hat sich im Automobilbereich als standardisiertes Vorgehen im Bereich der modularen Entwicklung von Fahrzeugfunktionen etabliert. Aktuell zeigt sich der Trend zur Erweiterung des modellbasierten Entwurfs in Bezug auf eine integrierte Entwicklung von Elektrik-/Elektronikarchitekturen. Im Fokus steht dabei in erster Linie die Modellierung und die Bewertung von Elektrik/Elektronikkonzepten. Die Integration technischer Anforderungen für die Verifikation der Architekturen gegenüber der Funktionsmodulentwicklung bildet einen weiteren Schritt in Richtung der integrierten Architekturentwicklung.

Die Arbeit untersucht welchen Beitrag die modellgestützte Architekturentwicklung zur technischen Integration flexibler zeitgesteuerter Netzwerkarchitekturen leisten kann. Am Beispiel FlexRay zeigt sich ein Zusatznutzen bei der anforderungsgerechten Definition und Objektivierung des komplexen Parametersatzes. Im Serienbereich ergibt sich ein Vorteil bei der statischen Analyse der zeitgesteuerten Elektrik-/Elektronik-Architektur durch die Anwendung einer frühzeitigen *virtuellen Integration* von FlexRay-Steuergeräten im Fahrzeugentwicklungsprozess. Die Methode ist dabei als Erweiterung zu standardisierten Ansätzen wie Simulation, Rapid Prototyping und Hardware-in-the-Loop zu betrachten. Durch den integrierten Ansatz lassen sich zeiteffizient und unabhängig von der Auslieferung von Musterständen der Spielraum für Entwicklungserweiterungen und die Grenzen der Fahrzeugarchitektur untersuchen. Probleme bei der Integration neuer Funktionskomponenten auf einem FlexRay-System und bei der Entwicklung von FlexRay-Plattformbausteinen lassen sich dabei bereits in der Analysephase identifizieren, bewerten und lösen. Das *virtuelle Testen* erhöht zudem die Testtiefe des Gesamtsystems, da auch abstrakte Analysen, beispielsweise im Zeitbereich, durchgeführt werden können.

Die Arbeit grenzt die Potentiale der statischen Architekturanalyse beim modellbasierten FlexRay-Systementwurf von den Möglichkeiten der dynamischen Architekturanalyse ab. Offene Bereiche, welche durch Simulation als Weg für die dynamische Architekturanalyse abgedeckt werden können, werden zur Vervollständigung der Methode ergänzend skizziert und exemplarisch für das Aktivitätsverhalten des Netzwerks dargelegt.

Um die Eignung der modellbasierten Architekturentwicklung im Bezug auf die FlexRay-Systementwicklung zu bewerten, werden

(1) sämtliche Anforderungen des FlexRay-Systementwurfs zur Abstraktion der komplexen Systemkonfiguration zusammengefasst und strukturiert auf geeignete Modellierungsdomänen übertragen,

(2) die Einbettung des Konzepts in die Entwicklungsphasen des Fahrzeugentwicklungsprozesses hinsichtlich Daten- und Modellverfügbarkeit zur Entwurfsraumexploration untersucht,

(3) und resultierende Anforderungen hinsichtlich einer vollständigen technischen Unterstützung der erarbeiteten Methode in einer gekoppelten Werkzeugkette zur Entwicklung dargestellt.

Die „*virtuelle Fahrzeugentwicklung*“ ist aufgrund der Abstraktion von realen Hard--/Softwarelösungen in ihrer Aussagekraft limitiert. Die Tragfähigkeit des Vorgehens und dessen Grenzen werden im Kontext der FlexRay-Systementwicklung reflektiert. Eine Validierung wird als statische Analyse im Bereich der Designdefinition der elektrischen Bitübertragungsschicht des Protokolls und als dynamische Analyse zur Simulation des Zustandsverhalten der Ablaufkontrolle des FlexRay-Controllers durchgeführt.

Ein weiteres Ergebnis dieser Arbeit bildet die Zusammenfassung zweckmäßiger Einsatzbereiche modellbasierter Architekturentwicklung im Entwicklungsprozess für FlexRay-Systeme. Dafür dient eine erstellte seriennahe Fallstudie, welche werkzeuggestützt mit der vorgestellten Methode durchgeführt wird. Die Konformität des Anwendungsbeispiels mit den dargelegten Anforderungen einer FlexRay-Entwicklung wird dargestellt und die Qualität der Ergebnisse dem erbrachten Aufwand gegenübergestellt.

Abschließend wird die Leistungsfähigkeit der modellbasierten Architekturentwicklung für FlexRay-Systeme bewertet sowie Einschränkungen und Grenzen kritisch analysiert. Ein Ausblick für potentielle Erweiterungen im Bereich der Methodik, deren Umsetzung und Anwendung sowie Schwerpunkte der weiteren Entwicklungen werden skizziert.

KAPITEL 1

Einführung

Dieses Kapitel gibt einen Einstieg in das Umfeld der Vernetzung im Automobil als zentrale Grundlage dieser Arbeit. Es wird auf die Motivation zur Einführung flexibler zeitgesteuerter Bussysteme in der Fahrzeugserienentwicklung eingegangen und die damit verbundenen Problemstellungen und Herausforderungen werden dargestellt. Als Konsequenz werden Qualitätsmerkmale einer systematischen Entwicklungsmethodik für die aufgezeigten Anforderungen formuliert.

Rechnergestützte Systeme bilden gegenwärtig einen wesentlichen Bestandteil vieler Abläufe im Alltag. Neben der Schaffung vollkommen neuartiger Kommunikationsformen und Arten der Informationsverknüpfung in allen Anwendungen bis in den Privatbereich hinein, beispielsweise das Internet oder die E-mail, zeigt sich die Bedeutung auch anhand der heute anzutreffenden Vielzahl an elektrisch/elektronischen Systemen in Gebrauchsgegenständen, den *eingebetteten Systeme*. Darunter fallen kleinere Geräte wie mp3-Abspielgeräte oder Mobiltelefone, aber auch größere Systeme, zu denen unter anderem das Automobil zählt. Die Funktionalität von Computersteuerungen wird dem Nutzer aufgrund fehlender oder nur rudimentär ausgeprägter Schnittstellen, zum Beispiel in Form eines Schalters, implizit bereitgestellt. Dadurch wird die starke Verbreitung komplexer eingebetteter Systeme im Alltag nicht sofort offensichtlich. Da sich der Benutzer oftmals der Technik hinter diesen Systemen nicht bewusst ist, müssen zu jedem Zeitpunkt auch kritische oder unerwartete Systemeingaben bedient werden können. Daraus folgt, dass beliebig komplexe Systemzustände abbildbar werden und die Software jeglichen durch den Benutzer gesteuerten Zustandswechsel behandeln und auch ungültige Eingaben autonom abfangen können muss /[Sim99]/. Gleichzeitig dürfen durch die Interaktion mit einem Benutzer als potentieller Fehlerquelle in keinem Fall menschengefährdende Situationen verursacht werden /[Ber02]/. Diese Problematik kann bereits an einfachen Systemen wie Aufzug- oder Garagensteuerungen nachvollzo-

gen werden, bei denen beispielsweise ein Einklemmschutz in gefährdenden Situationen eingreift und dadurch Risiken für Mensch oder Umwelt minimiert.

Die Entwicklung von eingebetteter Software im Automobil, der so genannten *Automotive Software*, hat in der Fahrzeugbranche in den letzten Jahren mehr und mehr Zuwachs verzeichnet und bildet mittlerweile einen wesentlichen Faktor zur Bewertung der Produktqualität sowie den Schlüssel für das Gelingen eines entwickelten Autos. Dies liegt nicht zuletzt an der Tatsache, dass durch die bereitgestellten Mittel der Elektro- und Informationstechnik sowie der Informatik neue Geschäftsfelder erschlossen werden können, welche einem Hersteller neue Möglichkeiten bieten individuelle Fahrzeugfunktionen zu entwickeln, um sich gegenüber der Konkurrenz zu differenzieren. Dies liegt an den zahlreichen Verknüpfungen unterschiedlicher Fachdisziplinen, die im Gesamtsystem Fahrer, Fahrzeug, Umwelt ihre Anwendung finden.

Ein anschauliches Beispiel ist ACC (*Adaptive Cruise Control*), ein elektronisches Fahrerassistenzsystem zur adaptiven Abstandsregelung, um die Distanz zu einem vorausfahrenden Fahrzeug autonom konstant zu halten. In diesem System kommen Radarsensoren, Benutzerinformationsanzeigen sowie Motor und Bremse zum Einsatz, die im Zusammenspiel aufeinander abgestimmt sein müssen. Einerseits gilt es die gewünschte Fahrzeugdistanz zum vorausfahrenden Fahrzeug zu garantieren. Andererseits müssen die Regelsystemeingriffe so gestaltet sein, dass die Abstandskorrekturen der hintereinander fahrenden Autos aufeinander abgestimmt bleiben. So lässt sich vermeiden, dass sich Bremsmanöver eines einzelnen Fahrzeugs als Kettenreaktion auf nachfolgende Autos verstärkt propagieren, was eventuell zum Stillstand eines Verkehrteilnehmers in dieser Kette führt /[Raj05]/.

Aus diesem Anwendungsszenario wird offensichtlich, dass sich Ingenieure unter anderem mit der physikalischen Eigenschaft eines Radarsensors, einem optimalen Visualisierungskonzept gegenüber dem Fahrer und der elektrischen Vernetzung der mechatronischen Komponenten beschäftigen müssen. Diese vielschichtigen Disziplinen erfordern daher ein hohes Maß an Fachwissen, wohldefinierte Schnittstellen zwischen den entsprechenden Fachabteilungen eines Automobilherstellers und einen passend abgestimmten Entwicklungsprozess.

1.1. Vernetzung im Automobil

Die Einführung von vernetzten Steuergeräten hat in der letzten Dekade die entscheidende Weiche zur Schaffung verteilter eingebetteter Systeme im Fahrzeug gestellt. Mehrere Standardisierungsgremien (vgl. Abs. 2.2.6) haben diese Entwicklung durch die Schaffung einheitlicher Konzepte und Standards begleitet, um den schnell wachsenden Bereich der Fahrzeugelektrik/-elektronik (E/E) kontrolliert umsetzen zu können.

Aktuell dominieren im Wesentlichen die Bussysteme CAN (*Controller Area Network*, LIN (*Local Area Interconnect*) und MOST (*Media Oriented Systems Transport*) als herstellerübergreifende Standards die Fahrzeugvernetzung. Während das CAN-Bussystem den größten Anteil der Fahrzeug-E/E in den Bereichen Antrieb, Fahrwerk, Karosserie, Komfort- und Fahrerassistenzsysteme vernetzt, wird LIN zur einfachen, kostenoptimierten Vernetzung von Systemen mit geringerer Komplextät, beispielsweise intelligente Sensoren, eingesetzt. MOST ergänzt das gesamte Bordnetz mit der Integration verteilter Multimediasysteme, etwa in Form von Audio- und Navigationsgeräten.

FlexRay spezifiziert ein Hochgeschwindigkeitsbussystem, dessen aktuellen Einsatz-

gebiete im Bereich der Fahrerassistenz- und Fahrdynamiksysteme liegen. Dabei stellt diese Technologie eine Alternative zu den CAN-vernetzten Systemen in den E/E-Domänen mit hohem Kommunikationsbedarf dar[1].

Während sich die Technologien CAN, LIN und MOST heute bereits in der Fahrzeugtechnik etabliert haben, zeigt sich bei FlexRay eher verhaltenes Engagement im Bereich der breitflächigen Fahrzeugserienentwicklung. Das kann partiell auf den Kostennachteil gegenüber CAN aufgrund eines höheren Preises von Hardwarebausteinen zurückgeführt werden. Allerdings könnten bei hinreichend intensiver Auslastung mithilfe von FlexRay mehrere CAN-Segmente substituiert werden, was auf Architekturebene langfristig mit Kostenvorteilen verbunden wäre. Die konservative Zurückhaltung gegenüber einer FlexRay-Serieneinführung liegt in der hohen Komplexität dieser Bustechnologie und der Vorsicht der Fahrzeughersteller gegenüber dem revolutionären technischen Fortschritt, der mit dem Einsatz eines zeitgesteuerten Kommunikationsprotokolls in einem Serienfahrzeug verbunden ist. Speziell der Integrationsaufwand beim FlexRay-Serieneinsatz erfordert von den Entwicklern ein wesentlich detaillierteres Technikverständnis. Das lässt sich unter anderem auf das vom ereignisorientierten CAN-Busprotokoll fundamental zu differenzierende zeitgesteuerte Grundkonzept eines FlexRay-Feldbussystems zurückführen. Dieser Unterschied und dessen signifikante Auswirkungen auf den Entwicklungsprozess finden sich breitflächig in den Konzepten dieser Arbeit wieder (s. Kap. 3). Eine zusätzliche integrale Anforderung, die mit dem Paradigmenwechsel auf ein zeitgesteuertes Bussystem einhergeht, resultiert aus der wichtigen Schnittstelle zwischen Fahrzeughersteller und Zulieferer im Entwicklungsprozess. Die Nachhaltigkeit einer FlexRay-Entwicklung lässt sich in dem Zusammenhang durch hinreichend präzise formulierte Vorgaben des Fahrzeugherstellers an seine Zulieferer entscheidend beeinflussen.

Einschränkungen für eine Serienentwicklung

Trotz der intensiven Entwicklung des FlexRay-Standards innerhalb des FlexRay-Konsortiums haben sich die Fahrzeughersteller bisher im Bereich der Serienentwicklung aus den genannten Gründen noch zurückhaltend gezeigt. Mit /[Sch07], [HKL07]/ gibt es bisher von zwei großen Fahrzeugherstellern aus dem Premiumsegment offizielle Ankündigungen einer Serieneinführung innerhalb der ersten Dekade der FlexRay-Standardisierung. Folgende Gründe für die zögerliche Einführung erklären diese Entwicklung:

1. **Reifegrad:** Verzögerte Fertigstellungen und Überarbeitungen der offiziellen FlexRay-Spezifikationen haben gleichermaßen zu notwendigen Neuentwicklungen im Bereich der Entwicklungs- und Analysewerkzeuge wie auch der einsetzbaren Hard- und Softwarekomponenten geführt. Durch diese Entwicklung ist es für einen Hersteller schwer abschätzbar, welche Version eines Standards sich langfristig für ein Fahrzeugprojekt etablieren wird.

2. **Kosten:** Trotz der fortschreitenden Entwicklungserfahrung und steigender Nachfrage sind FlexRay-Halbleiterbauteile aktuell teurer als vergleichbare CAN-Komponenten. Da sich die im Vergleich zu CAN deutlich höhere Komplexität der

[1] Eine Übersicht zur technischen Abgrenzung der beschriebenen Vernetzungstechnologien wird in Abs. 2.4 gegeben.

FlexRay-Technologie auch auf die benötigte Chipflächengröße von FlexRay-Komponenten auswirkt, ergeben sich aktuell Kostenvorteile beim Einsatz CAN-basierter Steuergeräte. Da sich der Markt an Anbietern von FlexRay-Kommunikationssoftware überschaubar gestaltet, wirkt sich der Kostenvorteil auch in diesem Bereich zu Gunsten der CAN-Technologie aus.

3. **Anwendungsdomänen:** Die ursprüngliche Intension eines Systemeinsatzes im Bereich sicherheitskritischer Anwendungen, wie der elektrischen Bremse oder Steuerung (Brake-/Steer-by-Wire (vgl. /[AZL+06]/) hat sich im Laufe der Zeit verschoben. Einerseits sind die Entwicklungsaufwände und die Absicherungen eines derartigen sicherheitskritischen Systems kostspielig und risikobehaftet. Andererseits trägt das Kommunikationssystem nur partiell zur Gesamtsystemzuverlässigkeit bei, welches auf ergänzende Lösungen im Bereich der Hard-/Softwareentwicklung und des Entwicklungsprozesses angewiesen ist /[Kop97]/. Im Ersteinsatz FlexRay zeigt sich eine starke Fokussierung auf kommunikationslastige Systeme im Bereich der Fahrerassistenz und -dynamik /[Sch07], [HKL07]/.

4. **Komplexität:** Der technische Umfang und die Komplexität des Bussystems FlexRay ist signifikant höher im Vergleich zu den Erfahrungen im Bereich CAN. Diese Feststellung resultiert aus der verstärkten logischen Abhängigkeit der zeitgesteuerten Bussysteme von der zu entwickelnden Anwendungssoftware und dem in der Automobilbranche nicht verfügbaren adäquaten Entwicklungsprozess /[Rin02]/. Diese Problematik verstärkt sich durch die teilweise mehrjährige Entwicklungserfahrung und evolutionär fortgeschrittene Entwicklung von Fahrzeugfunktionen, etwa im Bereich der Motorelektronik oder der elektronischen Stabilitätskontrolle, die sich nur mit hohem Aufwand auf ein zeitgesteuertes Konzept übertragen lassen. Auch die physikalischen Anforderungen an ein hochfrequentes elektrisches Bussystem erfordern kostspielige Absicherungsmaßnahmen während der Entwicklung der technischen Bordnetzarchitektur. Verstärkt werden diese Themen durch die komplexe Parametrierung des FlexRay-Busses, der ein präzises Verständnis der zu entwickelnden Systemarchitektur auf einer Vielzahl der Schichten des ISO/OSI-Modells /[Zim88]/ erfordert. Aus diesem Grund ist bis heute kein Anwendungsdokument (*Application Note*) für die systematische Erarbeitung eines zielgerechten FlexRay-Systemdesigns verfügbar.

Motivation für eine Serienentwicklung

FlexRay bietet zum aktuellen Zeitpunkt neben den aufgeführten Einschränkungen auch einige Vorteile im Serieneinsatz:

1. **Reifegrad:** Im Rahmen der Weiterentwicklung der FlexRay-Spezifikationen wird darauf geachtet Fehler zu beseitigen und Funktionen zu ergänzen, die abwärtskompatibel mit existierenden Lösungen aus den ersten Fahrzeugprojekten vereinbar sind. Gleichermaßen werden die Entwicklungszyklen des Konsortiums vergrößert, um kostspielige Hard- und Softwareentwicklungen bei beteiligten Drittzulieferern zu begrenzen. Die Migration der FlexRay-Protokollspezifikation von Version 2.1 auf 3.0 benötigte eine Entwicklungsarbeit von vier Jahren.

2. **Kosten:** Zur Neutralisierung der hohen Hard-/Softwarekosten werden Analysen und Modifikationen auf Architekturebene notwendig. Durch die hohe Performanz

des FlexRay-Busses gibt es zwei Maßnahmen, die sich bei einer Neupartitionierung von Softwarefunktionen anbieten. Prinzipiell lässt sich eine Vielzahl buslastiger Komponenten auf einem Bus verbinden. Bei der CAN-Technologie sind dabei mehrere Bussysteme, die über ein Fahrzeuggateway verknüpft werden, erforderlich. Durch das Replizieren von Signalen durch ein Gateway auf mehrere CAN-Busse relativiert jeder zusätzliche CAN-Bus seinen Kostenvorteil aufgrund entstehender Bandbreiteneinbußen. Zusätzlich lassen sich auf dem FlexRay durch die hochperformante Busanbindung deutlich leistungsfähigere Steuergeräte entwickeln, die eine Vielzahl an Funktionen mehrerer CAN-Steuergeräte integrieren. Im Extremfall führt eine Verschmelzung einzelner Komponenten zur Möglichkeit des Wegfalls einer Komponente, beispielsweise eines Sensors.

3. **Anwendungsdomänen:** Der starke Anstieg der Datenkommunikation im Fahrzeug lässt sich mit der evolutionären Erweiterung durch neue CAN-Bussysteme nur mittelfristig kompensieren. Auf lange Sicht wird ein Kommunikationssystem erforderlich werden, das den Anforderungen an schnelle und umfangreiche Datenübertragung gerecht wird. Konkrete Anwendungsbereiche sind die Domänen Fahrerassistenz und Fahrdynamik, die aufgrund ihrer fortgeschrittener Sensorik und Datenfusion genau diesem Anforderungsprofil entsprechen. Da FlexRay bereits eine intensive mehrjährige Aufmerksamkeit im Bereich der Protokollentwicklung und der Bordnetzabsicherung genossen hat, wird sich dieser Standard bedingt durch seine höhere Performanz etablieren.

4. **Komplexität:** Die sukzessive Migration der Fahrzeugelektronik auf ein zeitgesteuertes Bussystem führt zu notwendigen Adaptionen im Entwicklungsprozess, die für den Fahrzeughersteller eine große Herausforderung darstellen. Durch die teilweise langjährige Weiterentwicklung einzelner Steuergeräte bei den Zulieferern generiert eine Umstellung auf synchrone Kommunikation zusätzliche Kosten, die aktuell in einem kostenorientierten Markt schwer argumentierbar sind. Erst wenn sich ein FlexRay-Bus am Rande seiner Kapazität befindet, werden entsprechende Maßnahmen eingeleitet werden. Die Eigenschaften der Busphysik und die Hardwarekomponenten sind mittlerweile intensiv untersucht worden. Allein die Güte verschiedener Busparametrierungen ist weiterhin ein offenes Feld. Mit der Entscheidung der beiden Erstanwender im Fahrzeugserienbereich eigene Parametersätze zu konzipieren, ist es versäumt worden das eigenständige komplexe Themenfeld einer sinnvollen FlexRay-Konfiguration einheitlich zu bearbeiten. Die dadurch erforderliche Systematik zur Festlegung eines FlexRay-Systems stellt einen Schwerpunkt dieser Arbeit dar.

1.2. Problemstellung: Integration flexibler zeitgesteuerter Architekturen

Das Kernproblem in der Anwendung flexibler zeitgesteuerter Architekturen bezieht sich auf das unzureichend vorhandene integrale Verständnis des Gesamtkonzepts der zu entwickelnden E/E-Architektur. Der Einsatz der etablierten CAN-Technologie erfordert keine komplexe Systemparametrierung, die eine dedizierte Berücksichtigung mehrdimensionaler Systemaspekte vorsieht. Dazu zählen beispielsweise die physikalischen Eigenschaften des Netzwerks, etwa Kabelspezifikation und -längen, temporale Eigenschaf-

ten in der Anwendung, etwa der Datendurchsatz pro Zeiteinheit und Latenzzeiten bei der Datenkommunikation. Zusätzliche Komplexität ergibt sich durch Anforderungen einer modular konfigurierbaren Ausprägung aller Netzwerkvarianten durch Hinzunahme und Weglassen von Steuergeräten auf physikalischer Netzwerk- und Fahrzeugfunktionen auf logischer Funktionsebene. Speziell die flexible Integration neuer Steuergeräte in den Systemverbund in späten Entwicklungsphasen des Gesamtsystems führt bei den statisch konfigurierten zeitgesteuerten Architekturen zu Schwierigkeiten. Diese Problematik ist aus den bisher eingesetzten CAN-Bussen weitgehend unbekannt, da niedrigere Übertragungsfrequenzen die Bordnetzauslegung weniger heikel beeinflussen und das ereignisgesteuerte Kommunikationsprinzip keine statisch exakte Einpassung von Übertragungszeitpunkten erfordert.

Als Konsequenz lassen sich folgende Aspekte als Problemstellungen bei der Entwicklung flexibler zeitgesteuerter Architekturen feststellen:

- Unterstützung sämtlicher physikalischer Bordnetzkonfigurationen
- Flexible Möglichkeiten zur Systemerweiterung
- Zuverlässige und robuste Systemoperation
- Maximierter Datendurchsatz bei minimierten Übertragungszeiten
- Effiziente Systemauslastung
- Zweckmäßig garantierte Systemlebensdauer

Aufgrund der hohen Komplexität einer Systementwicklung mit diesem heterogenen Anforderungsprofil ergibt sich der Bedarf nach einer hinreichenden Entwicklungssystematik in der Systementwurfsphase. Vergleichbare Problemstellungen haben sich in der Vergangenheit im Bereich des methodischen Hardware-Entwurfs in der Halbleiterentwicklung, insbesondere beim Hardware-/Software-Codesign gezeigt /[VG92], [Gup95]/. In diesem Entwicklungsgebiet müssen beim Systementwurf ähnliche Nebenbedingungen, beispielsweise Systemkosten, Echtzeitanforderungen, Durchsatz und Chipflächengröße, berücksichtigt werden. Dabei haben sich mittlerweile ausgereifte Entwicklungsmethoden etabliert, die vom Einsatz geeigneter Werkzeuge und verwendeter Heuristiken profitieren.

Aktuell wird ein methodischer E/E-Architekturentwurf im Bereich der Fahrzeugserienentwicklung kaum oder unzureichend umgesetzt. Der erfolgreiche Umstieg und die Integration eines flexiblen zeitgesteuerten Bussystems erfordert jedoch Weiterentwicklungen in diesem Bereich, um die stetig wachsende Komplexität der eingesetzten E/E-Architekturen zukünftig beherrschbar umsetzen zu können. Die nachfolgende Abschnitte dieser Arbeit beschreiben das entwickelte Konzept FlexZOOMED (s. Abs. 1.4), eine Entwicklungsmethodik für flexible zeitgesteuerte Architekturen im Fahrzeugserienbereich, die einen Beitrag zur Verbesserung der Entwicklungssystematik in der E/E-Architekturentwurfsphase leistet.

1.3. Ziel der Entwicklungsmethodik für flexible zeitgesteuerte Architekturen

Das Ziel der Arbeit besteht in der Konzeption einer Entwicklungsmethodik zum Entwurf von verteilten eingebetteten Realzeitsystemen unter Verwendung einer flexiblen zeitgesteuerten Netzwerkarchitektur. Als Netzwerktechnologie bildet FlexRay als neuartiges gemischt zeit-/ereignisgesteuertes Kommunikationssystem für hochperformante Applikationen das zugrunde liegende Feldbussystem. Folgende Prämissen gilt es im Fahrzeugserienbereich zu beherrschen, um FlexRay zielgerichtet applizieren zu können.

Flexibilität: Die Definition einer verteilten Anwendung in einem FlexRay-Netzwerk erfordert hohe Disziplin im Bereich System-Design. Dies kann durch den bekannten Designfluss des V-Modells 97 /[KNR05]/ nur eingeschränkt erfolgen. Das zu entwickelnde Konzept zielt auf eine flexible aber verzahnt nebenläufige Entwicklung (*Concurrent Engineering*) ab, welche zusätzlich Iterationsschleifen im Anwendungsprozess toleriert.

Strukturierbarkeit: Um die hohe Komplexität einer Vielzahl an verteilten Funktionen über FlexRay zu beherrschen, muss es möglich sein Teilaufgaben zu identifizieren (*Separation Of Concerns*), welche modular und nebenläufig abgearbeitet werden können (gemäß dem Prinzip „*Divide et impera*“). Im Raum steht dabei die Frage nach Systemoptimierung, Wiederverwendbarkeit, Variantenbildung, Entwicklungszeitverkürzung und die Integration in einen bestehenden Prozess. Ohne eine Dekomposition der Systemteile wird eine Zertifizierbarkeit für sicherheitskritische Anwendungen unmöglich.

Abstraktion: Neben der Dekomposition stellen geeignete, eventuell hierarchische, Verfeinerungsstufen (*Refinement Steps*) ein wichtiges Mittel, um ein verteiltes Gesamtsystem zu spezifizieren und die Komplexität zu beherrschen. Im Mittelpunkt steht die Definition einer FlexRay-basierten Plattform, die den Prozess der Integration von Systemkomponenten vereinfacht. Durch Anwendung des Geheimnisprinzips bei der Softwareentwicklung (*Information Hiding*) muss das geistige Eigentum Dritter (*Intellectual Property*) geschützt oder von einem Informationsmangel abstrahiert werden können.

Formalisierung: Die Substitution informeller Spezifikationen durch formale oder halbformale Beschreibungsmittel stellt die Grundlage dar für die korrekte Systeminterpretation und Implementierung. Das mächtige Mittel der Simulation kann damit stärker eingesetzt werden. Durch konkrete Modellbildung können auch die Bereiche Funktionsspezifikation, Architekturstruktur und Kommunikationsverhalten integriert entwickelt werden /[FSV99]/.

Anwendbarkeit: Das entwickelte Konzept muss anwendbar (*Applicability*) bleiben und prozessfähig durchlaufen werden können. Dabei ist es erforderlich entwickelte Teilkonzepte geschickt zu verknüpfen, um Teilergebnisse inkrementell zusammenzuführen. Eine skalierbar geführte wechselseitige Entwicklung zwischen Systemintegrator und Komponentenlieferant stellt einen wichtigen Aspekt dar, der beispielsweise durch globales und lokales Spezifizieren angestrebt werden kann. Spezifische Lösungen sollen generell ausgeschlossen bleiben, damit ein allgemeines

Einsatzspektrum für Komponentenapplikationen in verschiedenen Projekten gewährleistet wird /[FSV99]/.

Vollständigkeit: Nicht zuletzt muss ein integrierter Ansatz den Anspruch an Vollständigkeit erfüllen, um eine hinreichend hohe Qualität zu generieren, da das Gesamtsystem eine starke Bindung zu seinen einzelnen Komponenten über das Netzwerk aufweist. Das gilt für nichtfunktionale und funktionale Anforderungen. Dazu können auch zusätzliche Informationen mithilfe von Schablonen (*Patterns*) beitragen.

1.4. Herausforderungen bei der Entwicklung eingebetteter Systeme im Fahrzeug

Eine methodische integrierte Entwicklung von vernetzten eingebetteten Systemen vollstreckt sich über eine Vielzahl an Arbeitsschritten, welche verschiedenen Phasen zugeordnet werden können (s. Abb. 1.1).

Knotendesign & Simulation: Jede Funktion eines Steuergeräts wird innerhalb der Knotendesignphase[2] spezifiziert. Für die Modellierung gemischt diskret/kontinuierlicher Regelungs- oder diskreter Steuerungsaufgaben werden technisch etablierte Beschreibungstechniken und zugehörige Werkzeuge zur Simulation und Codegenerierung herangezogen /[BS06, The07a, The07b]/.

Aus Applikationssicht werden die einzelnen (Teil-)Funktionen selbst wiederum innerhalb von Funktionsnetzen zur statischen Strukturmodellierung der Softwarearchitektur verbunden. Da die Funktionsmodellierung keine explizite Unterstützung zur Beschreibung der statischen lokalen Funktionsnetzarchitektur vorsieht, konzentrieren sich verschiedene Hersteller auf eine entsprechende Werkzeugentwicklung /[BFM05], [dSp07], [HW04]/. In diesem Bereich hat sich jedoch bisher kein einheitlicher Ansatz zur Architekturmodellierung des Funktionsnetzes global durchsetzen können.

Ein Konzept zur Architektursimulation wird in /[Sch06]/ vorgestellt, das sich aber bislang aufgrund mangelnder Werkzeugunterstützung nicht durchsetzen konnte.

Clusterdesign & Scheduling: Jede Buskommunikation lässt sich durch Qualitätsmerkmale, beispielsweise bezogen auf Antwortzeiten und deren Varianz, Datendurchsatz, Bandbreiteneffizienz oder Zuverlässigkeit beurteilen. Zur Berücksichtigung entsprechender Designziele müssen entsprechende Zielwerte identifiziert und in einer geeigneten Notation, z.B. als Anforderungsvorlagen (*Constraint Templates*), formal definiert werden.

Eine neuartige Fragestellung im Fahrzeugserienbereich resultiert aus dem sinnvollen Verpacken von Datenelementen (*Signalen*) zu Netzwerkbotschaften für die Verteilung in einem zeitgesteuerten Sendeplan des Bussystems (*Busschedule*). Dessen Ausprägung leitet sich dabei auch von den benötigten Kommunikationsbeziehungen im Netzwerk ab.

In Abstimmung mit dem Busschedule werden knoteninterne Systemprozesse (*Tasks*) separat für jeden Knoten sequentiell angeordnet (*Scheduling*) und deren Zeitverhalten bezogen auf maximale Ausführungszeiten (*Worst-Case-Execution-Time Analysis*) /[KP05]/ bestimmt.

[2] Ein Knoten bildet dabei die logische Abstraktion eines Steuergeräts von dessen Hardware.

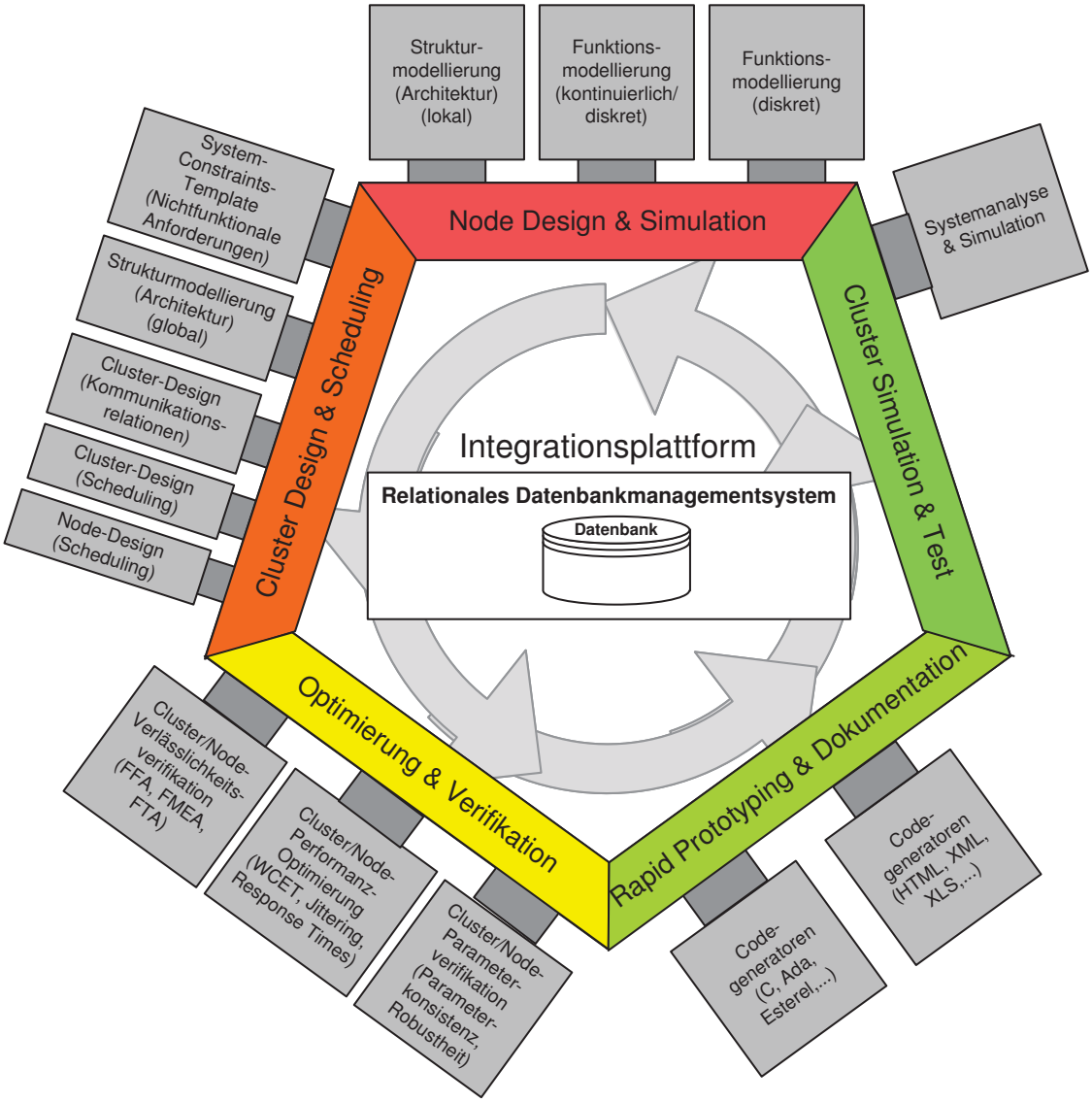


Abbildung 1.1.: Integrationsplattform zur Kopplung von Werkzeugen für eine durchgehende modellbasierte FlexRay-Applikationsentwicklung

Optimierung & Verifikation: Zur Einhaltung der definierten Qualitätsmerkmale sind dedizierte Optimierungsverfahren anzuwenden. Diese entsprechen oftmals den Kriterien allgemeiner mathematischer Mehrzieloptimierungsprobleme. Partielle komplexe Entwicklungsaufgaben, etwa beim *Scheduling*, die sich nicht in polynomieller Zeit rechnergestützt lösen lassen (*NP-vollständige Probleme*), werden dabei oftmals regelbasiert unter Hinzunahme von Informationen (*Heuristiken*) gelöst. Optimierungsprobleme im Gebiet der flexiblen zeitgesteuerten Systeme referenzieren auf das Systemzeitverhalten (*Timing*). Dabei gilt es die Performanz der Systemantwortzeiten (*Response Times*) global auf die maximal tolerierbare Zeitverzögerungen aus funktionaler Sicht (*Deadlines*) auszulegen. Diese Zeiten setzen sich dabei zum Teil aus der WCET (*Worst-Case-Execution-Time*) und dem *Jitter*-Verhalten bei der Buskommunikation zusammen (s. Abs. 3.1.5.1, 2.3.4).

Die Prüfung des Systems auf funktionale Korrektheit (*Verifikation*) wird dabei aus Sicht der Systemzuverlässigkeit und zur Freigabe des Parametersatzes relevant und lässt sich im Anschluss eines Optimierungsvorgangs durchführen. Dazu zählen deduktive Verfahren zur Bestimmung der Ausfallwahrscheinlichkeit (Fehlerbaumanalyse (*Fault Tree Analysis*)), die funktionalen Fehleranalyse (FFA) und die Auswirkungsanalyse (*Failure Mode and Effects Analysis*) zur Identifikation von Fehlerursachen /[PASH01], [MMP01]/.

Im Bezug auf die zeitgesteuerten Architekturen muss die Parameterkonsistenz des Feldbusses sowie dessen Toleranzbereiche verifiziert werden, um Aussagen über Robustheit und Lebensdauer des vernetzten Systemverbunds abzuleiten.

Rapid Prototyping & Dokumentation: Mit der Methode des *Rapid Prototyping* lassen sich frühe Entwicklungsstände aus der Hard- und Softwareentwicklung zur praktischen Analyse auf Demonstratoren untersuchen /[Spr96], [Sur08]/. Dabei steht unter anderem die Überführung und Anpassung der simulierten Funktionsmodelle auf ausführbaren Code für eine zugrundeliegenden Hardwareplattform im Vordergrund. Ergänzend zum Rapid Prototyping lassen sich auch vorhandene Hardwareeigenschaften und -ressourcen miteinbeziehen, was einen essentiellen Schritt bei der Codegenerierung für Seriensysteme darstellt. Bei den zeitgesteuerten Bussystemen interessiert dabei vorrangig die Analyse der busspezifischen hardwarenahen Softwareschichten (*Standardsoftware*), etwa der FlexRay-Treiber, das Modul zur Ablaufkontrolle oder Transport- und Netzwerkmanagementprotokolle /[Fla07]/. Im Idealfall lassen sich applikationsspezifische Softwaremodule, beispielsweise eine Dämpferregelung, zusammen mit der Standardsoftware unter Verwendung der spezifizierten Busparameter testen. Wesentlicher Bestandteil des Rapid Prototypings ist der Einsatz von Codegeneratoren für Hochsprachen (bspw. C, Ada, Esterel). Zusätzlich werden sämtliche erzeugten Artefakte in individueller Ausprägung dokumentiert. Dazu zählen informelle textuelle oder tabellarische Dokumente, aber auch strukturierte Datenzwischenformate in allgemein etablierten Standards der Informationsverarbeitung (HTML, XML). Bei zeitgesteuerten Bussystemen lässt sich der Schedule beispielsweise tabellarisch visualisieren und dessen Inhalte für die Konfiguration der Standardsoftware strukturiert ablegen (ASAM FIBEX.xml /[ASA08]/).

Clustersimulation & Test: Zur Ergänzung einer singulären Rapid Prototyping Studie wird eine Cluster Simulation (Restbussimulation) zur Analyse einer isolierten Netzwerkkomponente in virtueller Umgebung des restlichen Netzwerks umgesetzt. Nur in dieser Verbundanalyse lässt sich das Verhalten des Gesamtsystems in sukzessiven Erweiterungen auf Architekturebene analysieren. Entsprechend aufgebaute virtuelle Umgebungen eignen sich dabei in erster Linie für die schrittweise Überführung eines Verbunds von Demonstratoren auf seriennahe Steuergeräteprototypen. Demnach wird der Übergang zwischen der simulativen Entwicklung und dem konkreten Test eines Prüflings fließend gestaltet.

Die dargestellten Aufgabengebiete im Systementwicklungsprozess einer verteilten flexibel zeitgesteuerten Fahrzeugarchitektur beinhalten eine große Anzahl komplexer Artefakte, die im Laufe der Entwicklung entstehen. Ideal wäre eine Bündelung dieser Ergebnisse in einer zugrunde liegenden Plattform, über deren Schnittstellen dedizierte kommerzielle und nichtkommerzielle Entwicklungswerkzeuge durch Kopplung integriert werden können.

1.5. Struktur der Arbeit

Dieser Abschnitt gibt als *Einführung* eine Übersicht zu den aktuellen Vernetzungstechnologien im Fahrzeug. Dabei werden Problemstellungen im Zusammenhang mit der Entwicklung flexibler zeitgesteuerter Architekturen im Fahrzeugserienbereich identifiziert. Ergänzend erfolgt eine initiale Auflistung der Aufgabenbereiche für die verteilte Entwicklung zeitgesteuerter eingebetteter Systeme, aus denen sich grundlegende Anforderungen an eine dedizierte Entwicklungsmethodik ableiten lassen.

In Kapitel 2 werden die *Grundlagen der E/E-Entwicklung* im Kontext der Automobilentwicklung betrachtet. Dazu zählen neben dem gelebten Entwicklungsprozess auch eingesetzte Methoden und Werkzeuge sowie angewendete Standards. Die Eigenschaften einer verteilten Echtzeitentwicklung, die Grundlagen der Fahrzeugvernetzung und der Stand der Technik im modellbasierten E/E-Architekturentwurf akzentuieren die für diese Arbeit signifikanten Teilbereiche der E/E-Entwicklung im Fahrzeug.

Das wiederkehrende zentrale Themengebiet der *flexiblen zeitgesteuerten Architekturen* wird in einem eigenständigen Kapitel 3 behandelt. Diese inhaltliche Fokussierung folgt aus dem eigenständigen Charakter zeitgesteuerter Architekturen in Bezug auf deren technische Funktionsweise und der Implementierung der dafür relevanten Entwicklungsmethoden. Die spezifischen Eigenschaften der in dieser Arbeit betrachteten flexibel zeitgesteuerten Bustechnologie *FlexRay* stehen dabei im Vordergrund.

Sämtliche Erkenntnisse aus den vorhergehenden Abschnitten fliessen in die Definition der Entwicklungsmethodik *FlexZOOMED* (*Flex*Ray *O*bject *O*riented *M*odel *E*nhanced *D*esign) ein, welche in Kapitel 4 konzeptionell vorgestellt wird. Die *Methodik* wird formal spezifiziert und die Verknüpfung mit domänenspezifischer objektorientierter Modellierung hergestellt. In zwei separaten Abschnitten werden Entwicklungsaspekte für flexible zeitgesteuerte Architekturen am Beispiel FlexRay zusammengefasst und Ansätze für Optimierungen im Systementwurf aufgezeigt.

In Kapitel 5 wird die Tauglichkeit des erarbeiteten Konzepts durch eine partielle prototypische Implementierung der FlexZOOMED-Methodik demonstriert. Möglichkeiten und Grenzen potentieller Optimierungen zeigen sich bei der Analyse einer spezifisch entwickelten Fallstudie.

Im letzten Abschnitt werden die erzielten Ergebnisse reflektiert und Schlüsse für die zukünftige Entwicklung der flexiblen zeitgesteuerten Architekturen im Fahrzeug gezogen. Als Ergänzung dient der Ausblick mit Ideen zum weiteren Ausbau und Vertiefung der in dieser Arbeit beschriebenen Entwicklungsmethodik.

KAPITEL 2

Grundlagen

Zum Verständnis der nachfolgenden Abschnitte werden in diesem Kapitel die allgemeinen Grundlagen des Arbeitsumfelds beschrieben. Dazu zählt der aktuell gelebte Entwicklungsprozess zur E/E-Entwicklung und dessen zugrunde gelegte Methoden, Werkzeuge und Standards. Vertieft werden die Aspekte der verteilten eingebetteten Realzeitentwicklung und die dabei eingesetzten Netzwerktechnologien. Abschließend werden aktuelle Entwicklungen im Bereich des modellbasierten E/E-Architekturentwurfs charakterisiert, die in späteren Kapiteln dieser Arbeit Bedeutung erlangen.

In der Automobilentwicklung hat sich mittlerweile aufgrund des hohen Softwareanteils in den E/E-Systemen eines Fahrzeug die Notwendigkeit ergeben die Entwicklungsprozesse der E/E-Entwicklung an etablierte Vorgehen in der allgemeinen Softwareentwicklung zu adaptieren. Dieser Trend wird ersichtlich durch die Themenbereiche der eingesetzten Vorgehensmodelle, Methoden, Werkzeuge und Standards. Im Wesentlichen sind diese Entwicklungen durch die Phasen der Steuergerätespezifikation, -implementierung und -test geprägt. Speziell die Bereiche der verteilten Realzeitentwicklung mit spezifischen Anforderungen sowie der strukturierten modellbasierten E/E-Architekturspezifikation und -prüfung sind allerdings aktuell noch in der Initialphase im Bereich der Serienentwicklung. Diese erwähnten Themen werden in den nächsten Abschnitten behandelt.

2.1. Elektrik/Elektronik im Fahrzeugumfeld

Zur Abgrenzung der Zieldomäne der E/E-Entwicklungsmethoden im Fahrzeugumfeld wird initial eine Übersicht zum allgemeinen Fahrzeugentwicklungsprozess eines Fahrzeugherstellers gegeben. Weiterhin wird die Bedeutung der Domäne der E/E-Architektur im Fahrzeug herausgearbeitet und die Eigenschaften E/E-unterstützter Funktionali-

tät im Fahrzeug auf Systemebene zusammengefasst.

2.1.1. Fahrzeugentwicklungsprozess

Jeder Fahrzeughersteller operiert nach einem für sich individuell konzipierten Fahrzeugentwicklungsprozess (s. Abb. 2.1).

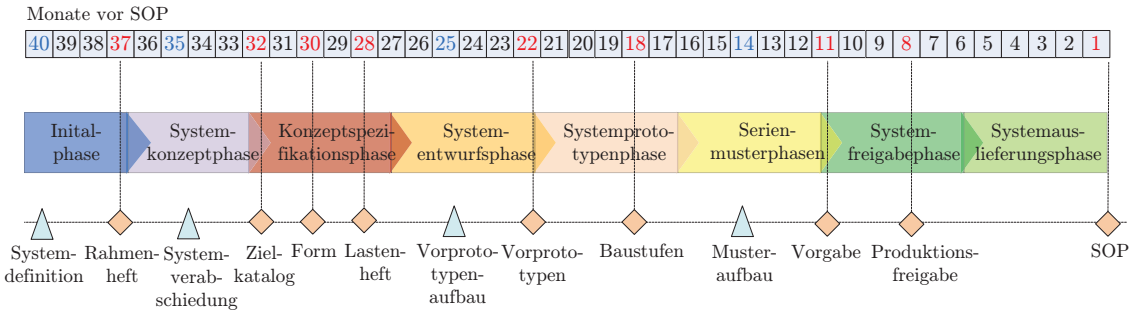


Abbildung 2.1.: Vereinfachte Darstellung eines strukturierten Fahrzeugentwicklungsprozesses

Dadurch lassen sich die unterschiedlichen Entwicklungsdomänen je nach Strukturierungsform (bspw. E/E, Antrieb, Karosserie, etc.) aber auch die unterschiedlichen Organisationseinheiten prozesstechnisch verknüpfen. Der Fahrzeugentwicklungsprozess stellt einen fundamentalen Bestandteil der Fahrzeugentwicklung dar, da einerseits verschiedene Fachbereiche unterschiedliche Entwicklungszeiten und -etappen benötigen und andererseits Entwicklungsergebnisse zu dedizierten Zeitpunkten (*Quality Gates*) synchronisiert werden müssen. Zu diesen wichtigen Entwicklungszeitpunkten lassen sich frühzeitig und iterativ Schwierigkeiten einer Fahrzeugentwicklung evident nachweisen. Potentielle Fehler werden analysiert und gegebenenfalls Abhilfemaßnahmen eingeleitet.

Konkret lässt sich der Fahrzeugentwicklungsprozess in zwei Phasen untergliedern. Während am Anfang in der Vorentwicklung die Produktplanung (Konzeptfindung, -verabschiedung und -spezifikation) im Vordergrund steht, so liegt in der eigentlichen Entwicklungsphase die Konzentration auf der iterativen Entwicklung und Prüfung von Musterständen.

Da der Fahrzeugentwicklungsprozess sämtliche Organisationseinheiten eines Unternehmens in ihrer Operation betrifft und einen mehrjährigen Entwicklungszeitraum umfasst, ist eine strikte Einhaltung der dort definierten Entwicklungsschritte essentiell für alle Fachbereiche.

2.1.2. Komplexitätszuwachs in der E/E-Architektur

Aufgrund der kontinuierlich gestiegenen technischen Gesamtkomplexität in Serienfahrzeugen hat sich die Anzahl von Rückrufaktionen seitens der Fahrzeughersteller im Laufe des letzten Jahrzehnts (s. Abb. 2.2) erhöht.

Beim Vergleich der Statistiken zur Anfälligkeit der Fahrzeugelektronik in Bezug auf Fahrzeugbetriebsausfälle zeigt sich ein gemischtes Bild. Einerseits lassen sich nach /[Kra05]/ Rückrufaktionen zu zwei Drittel der Fahrzeugmechanik zuordnen (s. Abb. 2.3). Beim Blick auf die bei Fahrzeugpannen beteiligten Baugruppen verteilen sich jedoch 40% auf Elektronikkomponenten (s. Abb. 2.4), wobei die Hauptursachen nicht im-

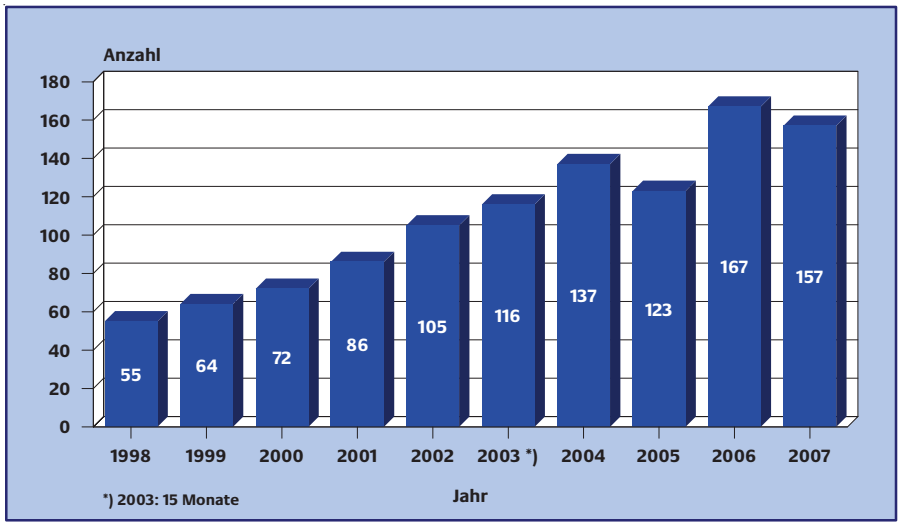


Abbildung 2.2.: Entwicklung der Anzahl der Rückrufaktionen von 1998 bis 2007 /[Kra07]/

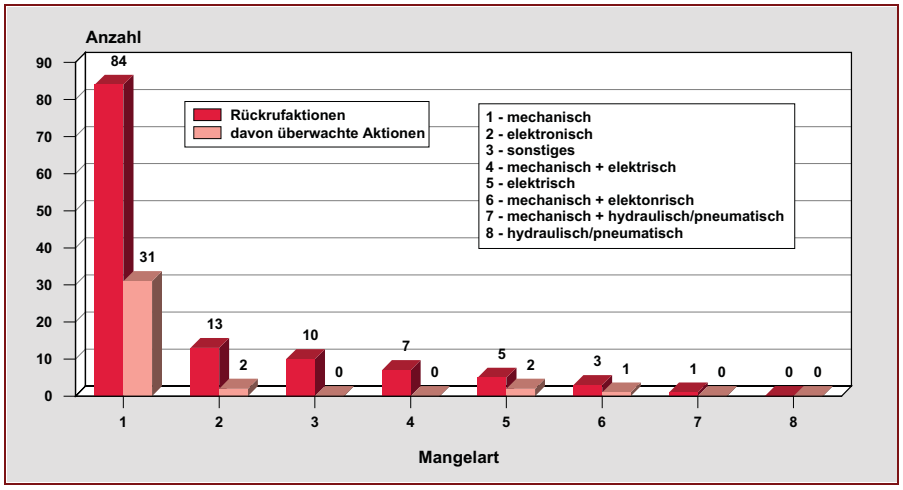


Abbildung 2.3.: Mängelartbezogene Verteilung der Rückrufaktionen aus Sicht des Kraftfahrt-Bundesamts /[Kra05]/

mer im Softwarebereich zu suchen sind, sondern sich vorrangig der Batterie und dem Kabelstrang, insbesondere dessen Steckverbindern, zuordnen lassen.

Insgesamt lässt sich folgern, dass E/E-Probleme einen signifikanten Einfluss in der Zuverlässigkeit des Fahrbetriebs leisten und die Anzahl der dort zu erwartenden Probleme in den kommenden Jahren ansteigen wird.

2.1.3. Systemklassifikation automotiver E/E-Systeme

Allgemein wird im Umfeld des Einsatzes der E/E-Systeme im Fahrzeug von *reaktiven Systemen* gesprochen. Diese lassen sich von den *interaktiven* und den *transformierenden Systemen* abgrenzen /[Ber89], [ELLSV02], [HCRP94]/.

Reaktive Systeme sind gekennzeichnet durch eine kontinuierliche Interaktion mit einer physikalischen Umgebung, deren Verhalten asynchron zu dem ablaufenden System

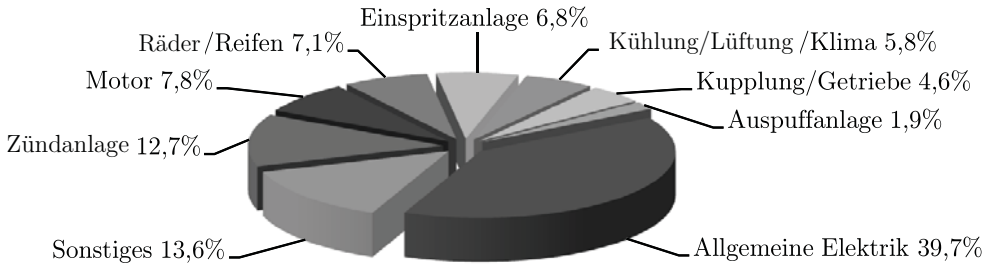


Abbildung 2.4.: Daten zu der Verteilung der an Fahrzeug-Ausfällen beteiligten Baugruppen /[ADA07]/

verläuft. Der Prozess der Interaktion zwischen System und Systemumgebung ist dabei in aller Regel nichtterminierend. Ein Systemstopp wird daher in aller Regel als Betriebsausfall oder Störung aufgefasst. Kritische Ereignisse, die aus der Systemumgebung in das System kommuniziert werden, erfordern eventuell eine umgehende Behandlung im System. Die Umgebung induziert in dem Zusammenhang die maximal akzeptierbaren Antwortzeiten für die Reaktion auf ein Ereignis, beispielsweise die Selektion des passenden Gangs in einem Automatikgetriebe auf Basis einer an der Motorsensorik gemessenen Drehzahl. Als Konsequenz einer kurzfristigen Systemreaktion kommt es zu potentiellen Veränderungen innerhalb des Systemzustands. Eine Systemantwort ermöglicht die Aktivierung, Verstärkung oder Unterdrückung im Verhalten des eingebetteten Systems und dessen Kommunikation zur Systemumgebung.

Transformierende Systeme sind geprägt durch die Berechnung mit zu Programmbeginn vorliegenden Eingabedaten und der mit der Datenrückgabe verbundenen Programmterminierung. Im Gegensatz zu den reaktiven Systemen wird der interne Ablauf eines transformierenden Systems nicht durch das Auftreten externer Ereignisse aus der Umwelt während der Ausführungszeit beeinflusst.

Bei den interaktiven Systemen wird das ausführende System mit dessen Systemumgebung synchronisiert, beispielsweise für die Entwicklung von Betriebssystemen. Im Fahrzeug bietet sich ein derartiges Szenario im Zusammenhang mit der Mensch-Maschine-Interaktionsschnittstelle. Beispielhaft lässt sich dafür die Bedienung eines Einparkassistenten unter Einbezug der Komponenten Lenkrad, Getriebewählhebel, Gaspedal, Bremse, Infotainmentsystem und Kombiinstrument nennen. Da eine Interaktion stets ein a priori definiertes deterministisches Bedienszenario voraussetzt, lässt sich die Abgrenzung zu den zur Systemumgebung asynchron ablaufenden, reaktiven Systemen nachvollziehen. Aufgrund der allgemeinen technischen Nähe zwischen den interaktiven und den reaktiven Systemen werden deren Ansätze auch integriert aufgefasst /[Wie02]/.

Zusammenfassend werden die Kernelemente der drei Systemarten dargestellt (s. Tab. 2.1).

2.2. Automotive Systems Engineering

Im Fahrzeugbereich hat sich mit der Zeit eine eigenständige *System Engineering*-Domäne entwickelt, mit teilweise spezifischen Architekturen, Methoden, Prozessen und Standards. Dieser Abschnitt erfasst dabei die wesentlichen Merkmale des *Automotive Systems Engineerings* aus Sicht des E/E-Bereichs.

	Reaktive Systeme	Interaktive Systeme	Transformierende Systeme
Interaktion	Sehr hoch	Hoch	Gering
Terminierung	Unendlicher Prozess	Endlicher Prozess	Endlicher Prozess
Reaktion	Unterbrechbar	Unterbrechbar	Nicht Unterbrechbar
Zustand	Zustandsabhängig	Zustandsabhängig	Zustandsunabhängig
Umgebungsorientierung	Ja	Ja	Gering
Ausführung	Parallel	Gemischt	Sequentiell
Zeitanforderung	(Harte) Echtzeit	Echtzeit	Unabhängig
Synchronisation	Asynchron	Synchron	(Keine)Asynchron

Tabelle 2.1.: Abgrenzung der reaktiven Systeme von weiteren Systemkonzepten

2.2.1. Überblick und Entwicklungstrends

Die Anzahl an E/E-Systemen im Fahrzeug hat sich in den letzten Jahren kontinuierlich erhöht. Dieser Trend zeigte sich zuerst an einzelnen Komponenten wie zum Beispiel der Motorsteuerung oder des elektrischen Fensterhebers bis zu den aktuellen vollvernetzten Fahrzeugen mit mehreren Feldbussystemen und intelligenten Fahrerassistenzsystemen (s. Abb. 2.5). Beschleunigt wurde die Entwicklung aufgrund der Substitution mechanischer durch elektromechanische Komponenten. Mithilfe von Software als Teil der Komponenten lässt sich beispielsweise in Fahrwerkssystemen wie der adaptiven Dämpfersteuerung eine verbesserte Fahrzeugdynamik umsetzen bei gleichzeitig sinkenden Gewichts- und Kostenaspekten.

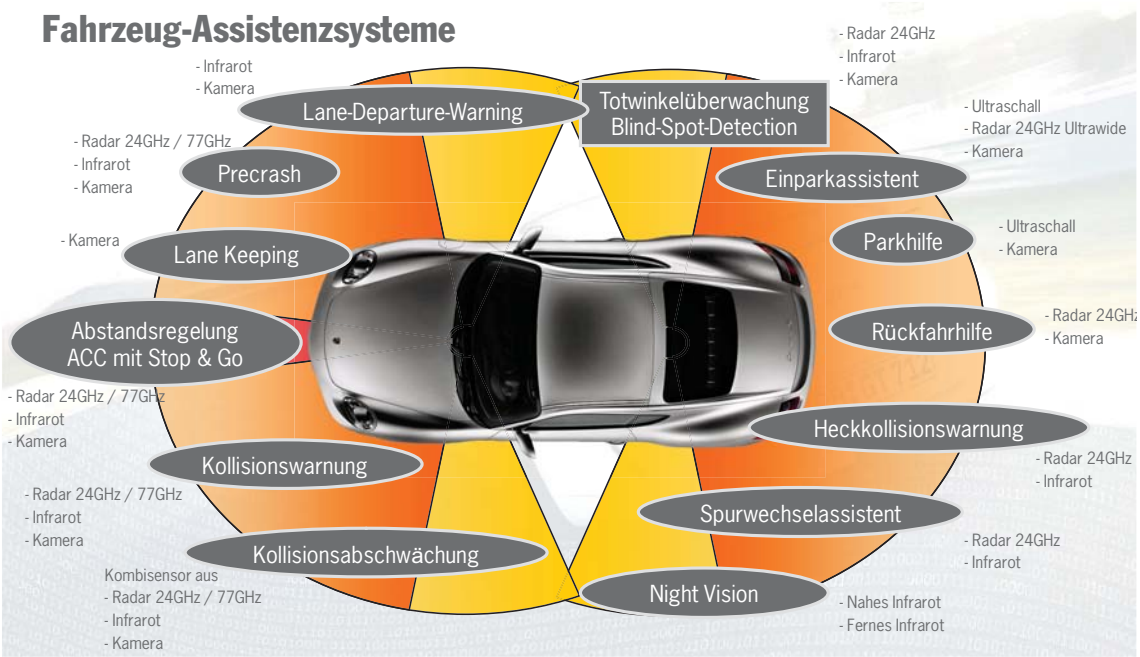


Abbildung 2.5.: Intelligente Fahrerassistenzsysteme zur Umfelderkennung im Überblick /[Mic07]/

Durch die sukzessive Einführung von Software im Automobil entstehen neue Entwicklungsaufgaben, die es für die traditionell maschinenbauorientierte Fahrzeugentwicklung zukünftig zu beherrschen gilt. Speziell die Möglichkeiten der Produktdifferenzierung über Softwarefunktionen und deren Qualität bleiben unterschätzt, wenn Par-

allelen zur Wertschöpfung im Vergleich von Hard- und Software aus der allgemeinen Rechnerentwicklung im Privat- und Geschäftsbereich gezogen werden /[NR05]/.

Die Ausprägung der Integration softwarebasierter E/E-Funktionalität zeigt sich momentan im Fahrzeugkontext stark gekennzeichnet durch Einzelkomponentenlösungen. Diese werden auf Basis einer verteilten Steuergeräteentwicklung mit einer Vielzahl an Zulieferern umgesetzt. Dieser Entwicklungsstil lässt sich auf die interne dezentrale Organisationsstruktur eines Automobilherstellers zurückführen, welche zu einer komponentenorientierten Systementwicklung passt. Nach /[BGG+04]/ liegt beim Vergleich der lokalen komponenten- mit einer globalen systemorientierten Betrachtungsweise der Fokus verstärkt auf der Komponentensicht (61% zu 39%). Unabhängig davon ist es aktuell aus Komplexitäts- und Aufwandsgründen nicht mehr möglich, die gesamten Systemfunktionen zentral im Fahrzeug in einer Systemkomponente integriert zu entwickeln. Ein verteiltes Systemkonzept begründet sich in der Dekomposition der Gesamtsystemgröße und der Reduzierung der Gesamtsystemkomplexität. Die räumliche Verteilung von Komponenten im Fahrzeug, die Integration einer Vielzahl an Entwicklern sowie die Beherrschung sicherheitsrelevanter Anforderungen bedingen eine verteilte Systementwicklung. Zusätzlich zu der hohen Systemkomplexität und der verteilten Entwicklung wird die Fertigungstiefe zum Ziel einer kosteneffizienten Produktgenerierung niedrig gehalten und in Zukunft eher ab- als zunehmen /[IKB03]/.

Als Folge der verteilten Komponentenentwicklung in der Fahrzeug-E/E sehen sich Fahrzeughersteller verstärkt in der Rolle des Systemintegrators, der eine steigende Anzahl an unabhängigen Zulieferfirmen in seiner verteilten Systementwicklung berücksichtigen muss, was die Gestaltung des Prozesses zum Systementwurf und dessen Implementierung maßgeblich beeinflusst. Mit der Verfolgung eines komponentenorientierten Entwurfs eines verteilten Gesamtsystems steigt die Signifikanz von etablierten Standardisierungen im Bereich der Soft- und Hardwareentwicklung. Konsequenterweise reduziert sich der Lösungsraum für Systemarchitekturalternativen, was im Automobil zu stabilen aber sich nur langsam entwickelnden Gesamtsystemarchitekturen führt.

Um den stetigen Wachstum der E/E und deren Software innerhalb des verteilten Gesamtsystems in Zukunft besser zu beherrschen und verstehen zu können, muss auch beim Fahrzeughersteller als Systemintegrator stärker auf eine Gesamtsystementwicklung hingearbeitet werden. Die Summe eines lokal optimierten Systems unterliegt dabei im Allgemeinen der Qualität eines global optimierten Systems. Diese Ideen sind allgemein mit dem Begriff des *Systems Engineerings* verbunden. Dieser Ansatz wurde in der Raumfahrt geprägt, um komplexe, interdisziplinäre Entwicklungen optimiert zu koordinieren /[R. 95]/. Beim Entwurf eines passenden Prozesses müssen Geschäftsinteressen des Unternehmens bewahrt sowie Vorstellungen auf Kundenseite getroffen werden. Erfolgsfaktoren sind in diesem Zusammenhang eine hohe Qualität, Vertrauenswürdigkeit, Kosteneffizienz sowie eine passende Ablaufplanung während des Gesamtlebenszyklus eines zu entwickelnden Systems /[INC07]/.

In der komponentenorientierten Systementwicklung im automotiven Umfeld hat sich wie in weiteren Sektoren, beispielsweise dem Avionikbereich oder der Automatisierungstechnik, eine Entwicklung hin zu einem effizienten, fehlerminimierenden System Engineering-Prozess durch IT-basierte Werkzeuge bewährt. Diese CAE-Werkzeuge (*Computer Aided Engineering*) bieten Lösungen für eine kontinuierliche Unterstützung des Systementwurfs und der Systemverifikation, die einerseits durch modellbasierte formalisierte Systemspezifikationen präzises Systemdesign gestatten und oftmals auch mächtige Hilfsmittel zur Simulation und Analyse bereitstellen. Die Entwicklungsverfah-

ren MIL (*Model in the Loop*), SIL (*Software in the Loop*), HIL (*Hardware in the Loop*) sind als Stand der Technik etabliert und tragen dazu bei, das Potential an menschlichen Fehlern frühzeitig im Entwicklungsprozess zu diagnostizieren und zu reduzieren /[BS06]/. Gleichzeitig wird der Ansatz des *Rapid Prototyping* mithilfe leistungsfähiger Codegeneratoren unterstützt, um ein modelliertes System pragmatisch auf dedizierter Hardware zu testen /[Spr96]/.

In der globalen Entwicklung mit gesteigertem Bedarf an Mobilität, Ressourcen, Information und Kommunikation sowie Schutz und Sicherheit entstehen maßgebliche Einflussfaktoren für den Automobilbau. Entsprechend stellen Mobilität, Sicherheit, Komfort und Konnektivität wichtige Eckpunkte der Fahrzeugentwicklung dar.

Abbildung 2.6.: Vision zukünftiger Fahrzeugapplikationen aus Zulieferersicht /[BHWJ07]/

Diese Aspekte finden sich wieder in den Bereichen Emissionsreduzierung, Umfelderkennung, modularisierter Systemaufbau und Systemintegration/-interaktion (s. Abb. 2.6). Aus der Sicht eines Zulieferers stehen dabei die Ziele der Unfallreduzierung (durch erweiterte Fahrerassistenzsysteme), die Verbrauchsreduzierung (durch verbesserte Einspritztechnik, Hybrid-Antriebe, *X-by-Wire*), die Konnektivitätserhöhung (durch *Car-to-Car*-Kommunikation (vgl. Abs. 2.3.1), Integration elektronischer Geräte der Fahrzeugbenutzer) sowie die Komplexitätsreduzierung in der E/E-Entwicklung (durch modulare Baukastensysteme und Plattformsteuergeräte[1] (vgl. Abs. 2.2.3)) im Vordergrund.

2.2.2. Systemarchitekturen

Die Systemarchitektur eines heutigen Premiumfahrzeugs enthält bis zu 70 Steuergeräte, die zum Teil über Netzwerke miteinander per Nachrichtenaustausch kommunizieren, allerdings auch jeweils Aufgaben autonom abarbeiten (z.B. Steuerungs- und Regelungsaufgaben) (s. Abb. 2.7). Durch die Interaktionsmöglichkeit zwischen den Steuergeräten werden Anwendungen auf verteilten Systemen realisiert, deren hoher Komplexitätsgrad durch verschiedene Abstraktionssichten im Entwicklungsprozess entscheidend verringert werden muss (s. Abb. 2.8).

[1] In diesem Beispiel bezieht sich das Baukastensystem auf die modulare Fahrzeugcockpitarchitektur CESAR (*Cockpit Electro-Mechanical System Architecture*) und das Plattformsteuergerät auf ein zentrales Karosseriesteuergerätekonzept (*Body Control Unit*) /[BHWJ07]/.

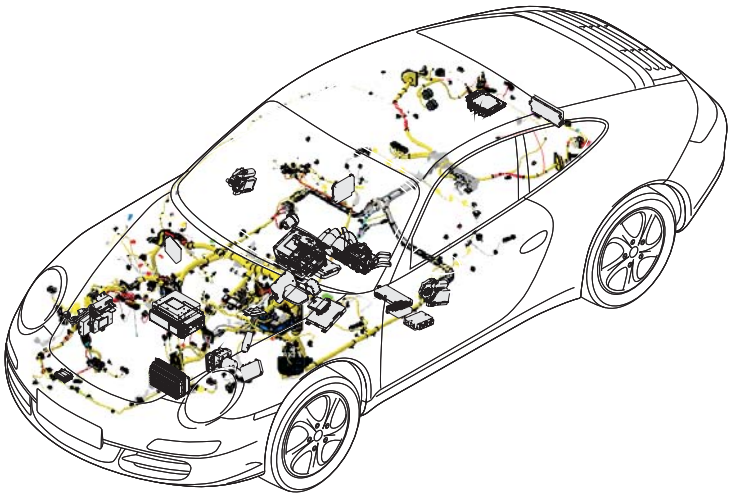


Abbildung 2.7.: Steuergeräteverteilung innerhalb des Porsche Carrera Typ 997

E/E-Systemarchitektur: Allgemein wird dieser Begriff unscharf, und oftmals rein Hardware bezogen, benutzt. Im Allgemeinen wird darunter ein Verbund an Einzelsystemen, den Steuergeräten, verstanden, die über Bussysteme vernetzt sein können. Unterschiedliche Sensoren werden zur Datenerfassung und Aktoren zur Ansteuerung mechanischer Komponenten über Netzwerke oder konventionell über analoge oder digitale Verbindungen angebunden. Im Folgenden sind die Begriffe E/E-Systemarchitektur und Systemarchitektur als synonym zu betrachten.

Im Allgemeinen wird bei Systemarchitektur zwischen zwei Sichtweisen auf das Gesamtsystem /[SZ03], [Bee07]/ unterschieden.

Logische Systemarchitektur: Zum einem existiert ein funktionales Modell der verteilten Anwendung, das *Fahrzeugfunktionsnetzwerk*, welches die Funktionen und ihre Kommunikationskanäle beschreibt. Auf oberster Ebene werden die technischen Fahrzeugfunktionen in abstrakter Form als kundenerlebare Funktionen (*Features*) aufgefasst. Durch Verfeinerung von einzelnen Funktionen der logischen Systemarchitektur lassen sich über mehrere Hierarchiestufen hinweg weitere konkrete Funktionen ableiten, welche die eigentliche algorithmische Datentransformation kapseln. Funktionen beinhalten dabei grundlegende Rechenoperationen, die sich als Kombinationen von unterschiedlichen Berechnungsblöcken auffassen lassen. Diese können modellgestützt als Element von Datenflussdiagrammen oder als Teil eines Zustandsautomaten dargestellt werden oder durch eine textuelle Berechnungsvorschrift gegeben werden. Oftmals werden logische Systemarchitekturen nicht explizit aufgeführt, da kein allgemeines Verständnis über die Konkretisierung von Semantik und Syntax für den Zweck geeigneter Notationsformen gegeben ist.

Technische Systemarchitektur: Die zweite Betrachtungsebene, die technische Systemarchitektur, leitet sich als technische Implementierung der logischen Systemarchitektur unter Berücksichtigung existierender physikalischer Einflussfaktoren ab. Dazu gehört auf oberster Ebene der Systemverbund des Steuergerätenetzwerks und dessen Verbindungsausprägung (*Netzwerktopologie*). Weiterhin werden die spezifischen Eigenschaften

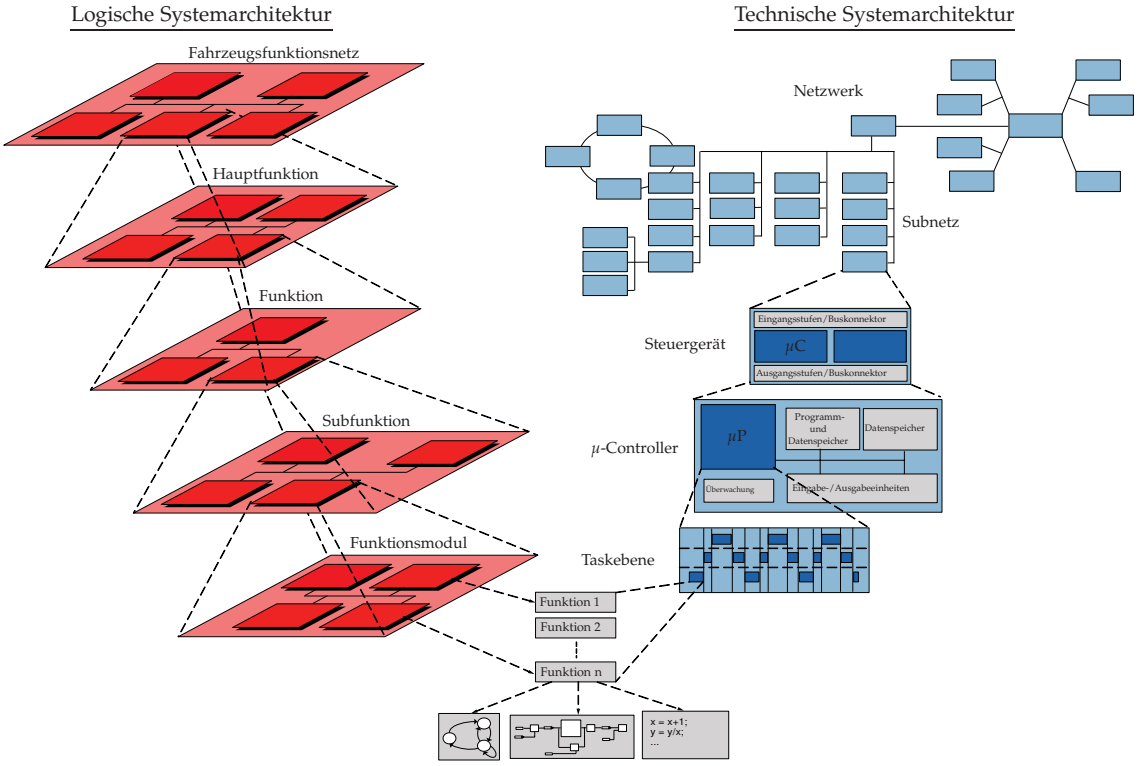


Abbildung 2.8.: Abhängigkeiten und Hierarchien in logischen und technischen Systemarchitekturen

eines Steuergeräts, dessen Verarbeitungsgeschwindigkeit, gemessen an der Taktrate vorhandener Mikrocontroller, und dessen Ressourcen in Form von Daten- oder Programmspeichergrößen berücksichtigt. Der Zusammenhang zu den logischen Funktionen vollzieht sich auf der Ebene von Betriebssystemtasks, welche, abgeleitet von der Betriebssystemebene, als Prozesse eine sequentielle Ausführung von Funktionsteilen beschreiben. Tasks sind abhängig von technischen Bauteilen, dem Mikroprozessor und dessen Leistung und werden daher der technischen Systemarchitektur zugerechnet.

Auf Basis dieser beiden Sichtweisen erfolgt der Entwurf jedes einzelnen Steuergeräts. Es wird zwischen Standard- und Applikationssoftware unterschieden. Unter Standardsoftware werden plattformspezifische Software-Anteile bezeichnet, die für alle Steuergeräte weitestgehend identisch übernommen werden. Die Applikationssoftware beinhaltet hingegen die eigentliche Steuergerätefunktionalität, in der die Logik des steuerungs- oder regelungstechnischen Systems umgesetzt wird.

Der Systemintegrator definiert allgemein seine Anforderungen in Form von *Lastenheften*. Dort werden die nichtfunktionalen und funktionalen Anforderungen bauteilbezogen (inklusive Applikationsebene) und plattformbezogen (ausschließlich Systemebene) in separaten Dokumenten versioniert gepflegt. Durch den Einsatz von dedizierter Software werden die Dokumente an Entwicklergruppen innerhalb von Datenbanken bereitgestellt und miteinander verknüpft. Analog zur allgemeinen Softwareentwicklung birgt die Anforderungserhebung, -spezifikation und -verfeinerung (*Requirements Engineering*) eine hochkomplexe, eigenständige Entwicklungsphase für ein Fahrzeugprojekt /[WW03], [Gri03]/.

Die Schnittstelle zum Zulieferer erfolgt in der Phase der Spezifikation von Hard-/-

Software jedes Steuergeräts. Während der Fahrzeughersteller die Systemsoftware auf Basis standardisierter Softwarekomponenten in der Serienentwicklung weitestgehend selbst vorgibt, wird die Applikationssoftware in der Regel direkt bei den Zulieferern umgesetzt. Neben dem zu dem Lastenheft obligatorisch vom Zulieferer entwickelten *Pflichtenheft* erfolgt die Applikationsentwicklung in Koordination mit dem Fahrzeughersteller. Dabei geht der Trend zu einer aktiven Einbindung des Fahrzeugherstellers in der Rolle des Systemintegrators, der sich selbst durch beigesteuerte Softwareanteile vom Markt differenzieren möchte. Jedoch beschränken strenge Haftungs- und Verantwortungsvorgaben die Bereitschaft eines Automobilherstellers noch intensiver aktiv in die Entwicklung einer Systemkomponente einzugreifen.

Demzufolge müssen auch die *Komponenten-* und *Integrationstests* der Software auf einem Steuergerät von dem jeweils zuständigen Zulieferer erfolgen, da ausschließlich bei ihm die Gesamtfunktionalität der Software eines Bauteils im Quellcode vorliegt. Um die Abnahme des Herstellers für ein Steuergerät auf einem qualitativ hochwertigen Leistungsstand zu ermöglichen, muss herstellerseitig ein konsequentes *Qualitätsmanagement* geführt werden. Dabei haben die Zulieferer die Kriterien eines Qualitätssteckbriefs im Sinne einer Vorqualifikation zu erfüllen. Falls sich der Hersteller die Integration der verschiedenen Systemkomponenten nicht zumutet, kann auch einer der Zulieferer eines Subsegments der technischen Systemarchitektur mit dieser Aufgabe betraut werden. Nachgelagerte und für die Serienproduktion relevante Aufgaben müssen jedoch wieder vom Systemhersteller übernommen werden wie beispielsweise die Kalibrierung der Software wie auch der *System-* und *Akzeptanztest* zur Verifikation und Validation des Gesamtsystems.

2.2.3. Plattformkonzepte und Variantenmanagement

Um den enormen Kostendruck in der Fahrzeugserienentwicklung zu begegnen hat sich herstellerübergreifend das Bestreben zur Entwicklung von Plattformkonzepten durchgesetzt. Diese Idee bezieht sich auf die Wiederverwendung funktionsgleicher Bauteile in verschiedenen Fahrzeugderivaten, den Baureihen, um den Anteil an Parallel- oder unnötiger Neuentwicklungen zu minimieren. Dieses Vorhaben fundiert nicht nur auf der globalen E/E-Architekturebene über die einzelnen Bussegmente hinweg, sondern bezieht sich gleichermaßen auf einzelne Steuergeräte und deren Subkomponenten. Plattformen beziehen sich dabei auf mechanische und elektrisch/elektronische Fahrzeugkomponenten.

In der E/E-Architekturentwicklung lässt sich der Plattformgedanke vorrangig auf Steuergeräte anwenden, bei denen ein Basisanteil, etwa die Hardware, identisch gehalten wird, wobei fahrzeugspezifische Umfänge (oftmals als freigeschaltete Softwarefunktionen) variantenabhängig umgesetzt werden. Aus der Sicht der E/E-Entwicklung existiert dabei ein intensives Bestreben einen hohen Grad an diesen Plattformkomponenten zu entwickeln, falls der Einsatz identischer Steuergeräte in verschiedenen Baureihen nicht möglich ist. Je nach Quantität und Vielfalt der zu entwickelnden Fahrzeugbaureihen eignen sich Baukastenkonzepte, beispielsweise für Premium-, Mittelklasse- und Kleinwagen. Allgemein basieren die Baukästen auf einheitlichen E/E-Architekturstrukturen mit identischen Fahrzeugdomänen (in der gröbsten Form klassischerweise mit der Unterteilung in den Komfort-, Antriebs- und Telematikbereich). Jede Fahrzeugdomäne wird dabei in einem Baukasten auf einen oder mehrere Bussysteme mit allgemein einheitlichen Bustechnologien verteilt.

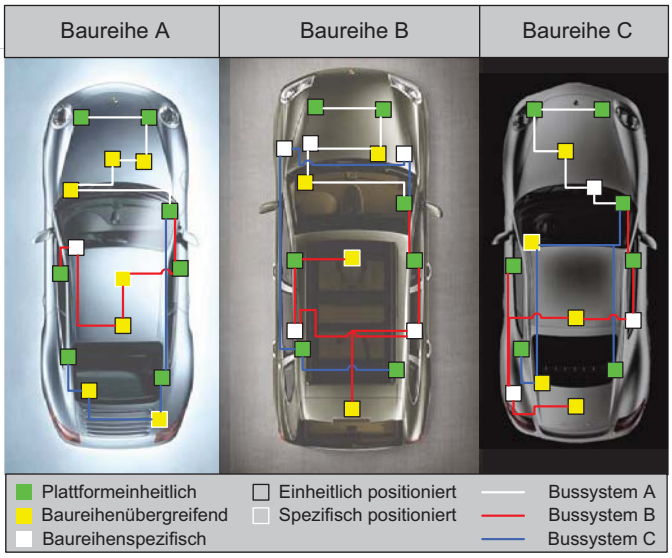


Abbildung 2.9.: Gegenüberstellung der physikalischen E/E-Architekturvarianten in den verschiedenen Fahrzeugbaureihen

Wie in Abb. 2.9[2] dargestellt, erhöht sich die Komplexität der E/E-Architekturen in Baukästen aufgrund regelmäßig auftretender proprietärer Abweichungen einer Baureihe von der Plattform. So können teilweise baureihenspezifische Komponenten, auch komplette Steuergeräte, vorkommen.

Als Zwischenstufe gilt es baureihenübergreifende Komponenten zu identifizieren, die jedoch nicht in allen Fahrzeugbaureihen vorhanden sein müssen (in dem Beispiel sind spezielle Heckklappensteuergeräte in Baureihe B und C identisch, wobei in Baureihe A auf diese Komponente verzichtet werden kann).

Selbst bei den in allen Baureihen identischen Steuergeräten, den plattformeinheitlichen Bauteilen, kann es vorkommen, dass aus Paketierungsgründen eine unterschiedliche Positionierung im Fahrzeug notwendig ist. Im Extremfall können Einschränkungen aus der Kabelbaumentwicklung zu einer Umpartitionierung eines einheitlichen Steuergeräts auf ein alternatives Bussegment führen. Derartige Einschränkungen erschweren den allgemeinen Ansatz der Plattformentwicklung.

Variantenmanagement

Aktuell erfolgt die Analyse und Pflege der Fahrzeugplattform und -varianten größtenteils in Form von eigenständigen Dokumenten. Dabei werden die Fahrzeugausstattungen den Funktionen oder Steuergeräten zugeordnet und diese in Tabellen mit den verschiedenen Fahrzeugbaureihen verknüpft. Falls die Systemarchitektur bei der Plattformentwicklung bereits in einem elektronischen Modell vorliegt, so lässt sich das Variantenmanagement feingranularer betreiben. Das in Kapitel 5 umgesetzte Konzept der in dieser Arbeit definierten Entwicklungsmethodik beruht auf einem derartigen Variantenkonzept /[Aqu07]/. Dabei liegt die Herausforderung in der sinnvollen Erfassung und Gruppierung von Architekturelementen in Gruppen (*Modellgruppen*). In der Regel werden dabei je nach Klassifikationsmethode disjunkte Mengen entstehen. Unab-

[2]Die dargestellten Topologien sind stark vereinfacht und mit rein fiktiven Charakter.

hängig von dem Modell sind aus den Eigenschaften der E/E-Architektur Konzepte zu definieren (*Konzeptvorlagen*), die es erlauben technische Alternativen auszuwählen (*Konzeptinstanzen*). Eine Strukturierung und Gruppierung der Konzeptvorlagen erfolgt über das Bilden von *Konzeptgruppen*. Die Zuordnung der technischen Konzepte zu konkreten Fahrzeugausstattungen vollzieht sich durch die Definition von *Ausstattungsvorlagen*. Die Ausstattungsmöglichkeiten werden in dieser Arbeit mit einer Paketierung der verschiedenen definierten Ausstattungsmerkmale der E/E-Architektur in klassische Alternativen, etwa Basis, Medium und Premium, überführt. Die *Architekturvariante* lässt sich nun durch die Zuordnung einer eindeutigen Ausstattungsinstanz zu jeweils einer eindeutigen Konzeptinstanz aus den spezifizierten Konzeptvorlagen erzeugen. Die Anwendung des Variantenmanagements wird in Abs. 5.4 dokumentiert.

2.2.4. Entwicklungsprozess

Der Einzug der Software in die elektronischen Systeme im Fahrzeug und deren starker Anstieg in den letzten Jahren ging konform mit den Aussagen von G. Moore, dessen populäres *Moore's Law* 1965 einen Wachstum an integrierten Schaltkreisen innerhalb von elektronischen Komponenten mit dem Faktor 2 pro Jahr prophezeite /[Moo65]/. Um diesen enormen Komplexitätszuwachs in einem für die Automobilbranche noch jungen Segment stemmen zu können, musste zwangsläufig der Gesamtfahrzeugentwicklungsprozess auch auf die Umfänge des Systems Engineering mit hohen Software-Anteilen abgestimmt werden /[SZ03]/.

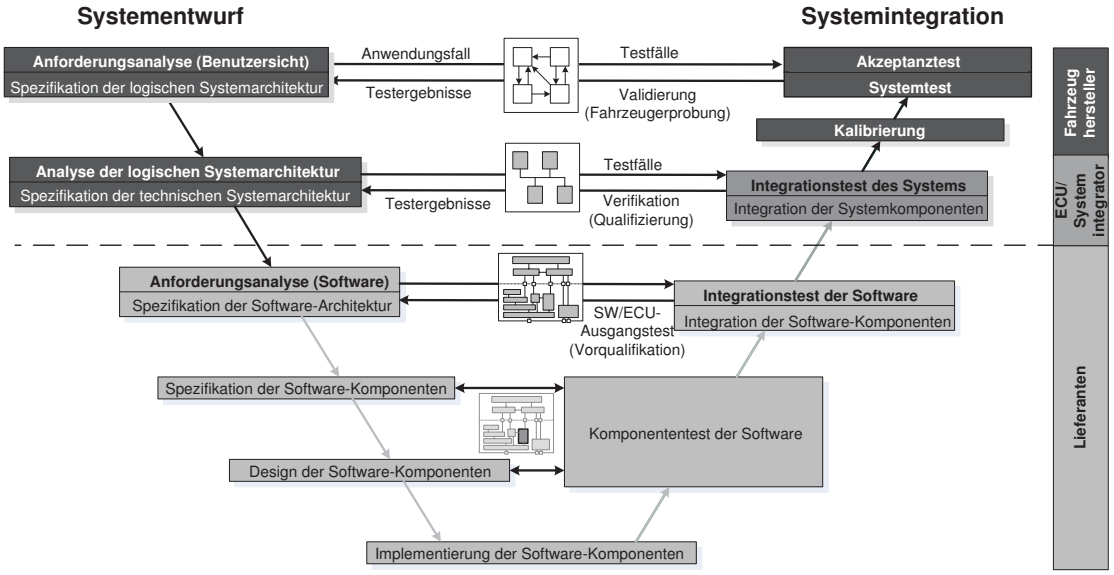


Abbildung 2.10.: Kernprozesse in der Softwareentwicklung nach dem V-Modell

Durch die starke Einbindung von Lieferanten und Sublieferanten in die Fahrzeugentwicklung hat sich ein iterativer Entwicklungsprozess nach dem Vorgehensmodell *V-Modell 97* für softwarebasierte Systeme etabliert (s. Abb. 2.10). Die zu durchlaufenden Kernprozesse in diesem Konzept scheinen geeignet zur Spezifikation, Design und Implementierung von eingebetteten Systemen durch die in jeder Entwicklungsphase mögliche Prüfung der Korrektheit (*Verifikation*) von (Teil-)Ergebnissen. In Projektentwicklun-

gen wird durch iteratives Ausführen des Vorgehensmodells die Produktfunktionalität in skalierbarer Form inkrementell erweitert und deren Qualität über entsprechende Musterphasen verbessert. Auch zur parallelen Entwicklung von Hard- und Software kann dieses Vorgehensmodell geeignet implementiert werden /[SZ03], S. 148/.

Mittlerweile gilt das V-Modell 97 als überholt, da neben den Vorteilen auch einige Schwächen identifiziert worden sind. Im speziellen fehlt eine Skalierbarkeit auf den spezifischen Einsatzzweck, da Projekte im Allgemeinen stark unterschiedliche Anforderungen aufweisen können. Da die Phasen des V-Modell 97 strikt sequentiell durchlaufen werden, ist ein agiler Ansatz in Bezug auf zeitliche Terminierungen in den Projektphasen nur beschränkt umsetzbar. Auch das Zusammenspiel zwischen dem Fahrzeughersteller in der Rolle des Auftraggebers und dem Zulieferer als Auftragnehmer lässt sich nur unsauber auf die vorgeschriebenen Projektphasen abbilden. So wird in der Automobilbranche die Schnittstelle zwischen Systemintegrator und Systemzulieferer horizontal im V-Modell vollzogen (s. Abb. 2.10). Späte Korrekturen können dabei ein hohes Projektrisiko darstellen. Zusätzlich können beim Übergang zwischen den einzelnen Projektphasen Informationen durch unsaubere Schnittstellen verloren gehen. Die dabei entstehenden Fehler lassen sich in der Regel auf menschliche Ursachen zurückführen, da die Informationsüberführung und -interpretation oftmals durch unabgestimmte heterogene und proprietäre Werkzeugketten erfolgt, welche potentielle Fehlerquellen darstellen[3].

Interessanterweise unterlaufen die meisten Softwarefehler ($> 40\%$) bei der Definition der Systemspezifikationen bzw. des Systemdesigns /[Spr96], [Gri03]/ und nicht wie häufig angenommen während der Implementierungsphase. Diese Erkenntnis lässt sich bei der Betrachtung des Entstehungsprozesses eines E/E-Fahrzeugbauteils nachvollziehen. Dabei stellt sich beispielsweise die Frage wie die Korrektheit der Funktionalität einer Komponente gegenüber partiell informeller und partiell halb-formaler Spezifikationen, in Form von gering strukturierten Lastenheften des Auftraggebers, verifiziert werden soll.

Gesamtheitlich betrachtet resultiert aus den aufgezeigten Unzulänglichkeiten des aktuellen Entwicklungsprozesses der Bedarf einer deutlichen Verbesserung in der funktionalen Gesamtspezifikation eines E/E-Fahrzeugsystems. Diese sollte sich in Bezug auf die in den Komponenten ablaufenden Applikationen abstimmen und verfeinern lassen (*System Refinement*). Zusätzlich gilt es den Automatisierungsgrad der Übergänge an den Schnittstellen der Projektphasen sukzessive zu erhöhen. Gleichzeitig muss die Schaffung einer skalierbaren, integriert durchgehenden Entwicklungsumgebung das Ziel des Systemintegrators bilden.

Es stellt sich zwangsläufig die Frage inwieweit das klassische V-Modell 97 auf die Entwicklung von verteilten eingebetteten Systemen in Bezug auf die Automobilindustrie unter Hinzunahme einer Vielzahl an Projektbeteiligten aus verschiedensten Organisationen passt. Verbesserte Konzepte wie beispielsweise das *V-Modell XT* adressieren genau diese Fragestellung /[KNR05]/. Es sollte dabei nicht vergessen werden, dass ein Vorgehensmodell eher eine strukturelle Vorgabe für den Gesamtproduktentwicklungsprozess darstellt und keine konkrete technische Umsetzung der einzelnen Kernprozesse vorgegeben wird. Im Folgenden wird die Applizierung des V-Modells 97 im Kontext der E/E-Entwicklung im Automobil beschrieben.

[3]Eine *heterogene* Werkzeugkette besteht aus einer Kombination von Entwicklungswerkzeugen mit unterschiedlichen Eigenschaften und Anwendungsgebieten, die unabhängig voneinander entwickelt werden. *Proprietäre* Werkzeugketten beinhalten Softwareeigenentwicklungen mit einem dedizierten beschränkten Anwendungsgebiet.

2.2.4.1. Anforderungsanalyse

Das Ziel der Anforderungsanalyse basiert auf der Identifikation zwangsläufig zu erfüllender Eigenschaften einer zu entwickelnden E/E-Architektur, um den Bedarf an kundenerlebbaren Funktionen des Fahrzeugs vollständig abzudecken.

In der Praxis bezieht sich die Anforderungsanalyse auf den Prozess der Erstellung eines Lastenhefts. Dieses bezieht sich dabei konkret auf physikalische oder logische Komponenten eines Fahrzeugs und wird allgemein in textueller Form spezifiziert. Probleme dieser Darstellungsform wurzeln in fehlenden oder ungenau darstellbaren Normen und Standards zur

- Strukturierung komplexer Inhalte,
- Spezifikation von Bedingungen, Attribute, Einschränkungen oder gesamter Algorithmen,
- Spezifikation der Abhängigkeiten beinhalteter Artefakte,
- Zuordnung beinhalteter Artefakte zu implementierbaren Komponenten oder Modulen.

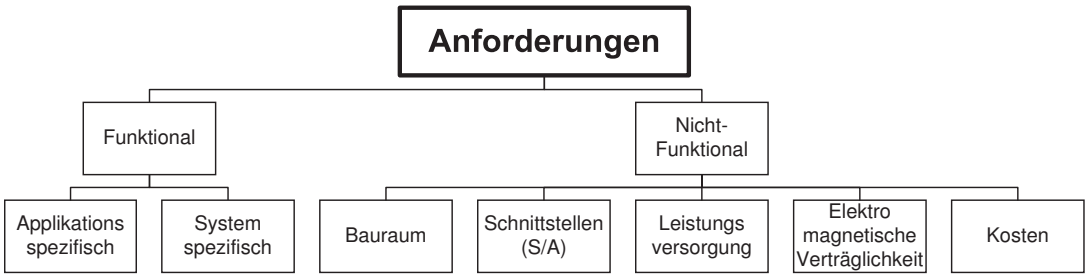


Abbildung 2.11.: Anforderungsklassifikation für eingebettete Systeme in der Fahrzeugentwicklung

Bei der Entwicklung im Elektronikbereich des Fahrzeugs ist eine Mischung aus nichtfunktionalen und funktionalen Anforderungen zu berücksichtigen. Nichtfunktionale Anforderungen resultieren aus Einschränkungen

- im Bauraum (z.B. Gewichts-, Temperatureffekte, Einflüsse durch vorhandene chemikalische Flüssigkeiten oder physikalische Kräfte, Crash-Zonen und maximale Leitungslängen),
- in der Energieversorgung,
- durch die Anschlussstellen für Sensorik und Aktorik,
- in der elektromagnetischen Verträglichkeit (EMV),
- durch die Komponentenkosten der Steuergeräte (Prozessor, Speicher, E/A-Schnittstellen).

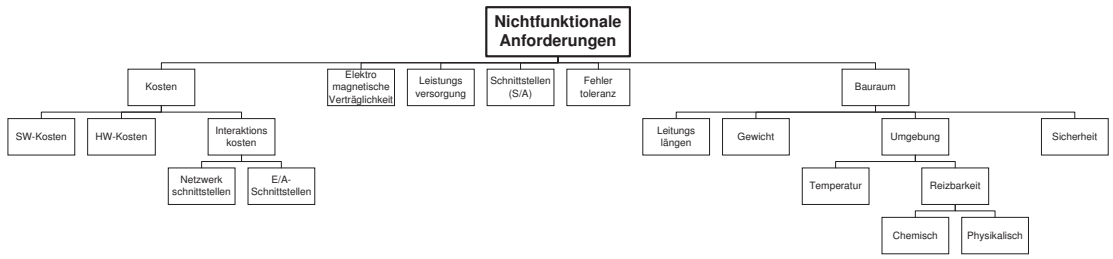


Abbildung 2.12.: Beispielhafte Verfeinerung der Struktur für nichtfunktionale Anforderungen an E/E-Systeme im Automobil

Je nach Betrachtungsweise können auch Kosten für SW-Komponenten (Entwicklungskosten, Lizenzierungen) oder auch Fehlertoleranzeigenschaften eines Steuergeräts in den nichtfunktionalen Anteil eingehen.

Funktionale Anforderungen gliedern sich pro Steuergerät in einen applikationsspezifischen Anteil, der die für den Fahrer nutzbare Funktionalität, beispielsweise einen Parkassistenten, beinhaltet und einen systemspezifischen Anteil, der die Basisfunktionalität eines Steuergeräts, etwa die hardwarenahen Treiber, die Kommunikationsschichten und die Systemdienste, abdeckt. Dazu können nutzerneutrale Funktionen bzgl. des Systemmanagements, der Systemaktualisierung, der Systemdiagnose, der Systemkonfiguration oder des Komponentenschutzes eines Steuergeräts gezählt werden. Zusätzlich müssen Systemdienste definiert werden, welche teilweise applikativ, etwa bei höheren Datenprotokollen auf dem Feldbus zur Übertragung von Bedien- und Anzeigeinformationen für das Cockpit des Fahrzeugs /[WW04]/, oder systembezogen beispielsweise zum Softwareupdate eines Steuergeräts oder zur dessen Kalibrierung implementiert werden.

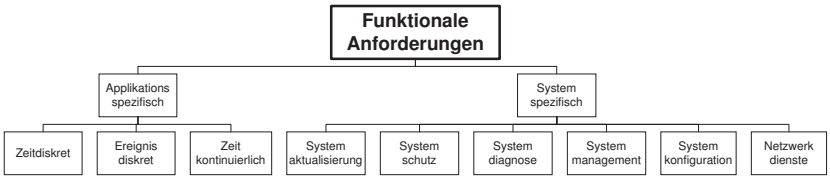


Abbildung 2.13.: Verfeinerung der funktionalen Anforderungsstruktur für E/E-Systeme im Automobil

Obwohl die beiden Klassifikationen der Anforderungen zwei grundsätzlich unterschiedliche Aspekte der Systementwicklung spezifizieren, so gibt es trotzdem wechselseitige Wirkungen zwischen den beiden Ästen, die den Lösungsspielraum einschränken. Während der Systemlieferant in der Regel sein Wissen auf der funktionalen Seite zur Implementierung einsetzt, beschäftigt sich der Systemintegrator vor allem mit den nichtfunktionalen Anforderungen, da nur ihm die Informationen über das Gesamtsystem, den Systemverbund, vorliegen. Aktuell hat sich der Einsatz von Dokumentenverwaltungssystemen mit Verweisen zur Nachverfolgbarkeit (*Traceability*) von Abhängigkeiten durch Verknüpfungen etabliert /[VHHM03], [Tel06], [IBM06]/. Allerdings können da-

bei in der Regel nur informelle Textblöcke angelegt werden. Eine Formalisierung der Objekte und ihres Inhalts wird nicht unterstützt, was die interpretationsfreie Umsetzung einer Spezifikation erschwert. Halbformale Beschreibungsmittel werden dabei zur Veranschaulichung herangezogen, beispielsweise in Form eines vereinfachten Zustandsautomaten. Die Qualität einer formal präzisen Spezifikation auf Basis einer definierten Grammatik lässt sich damit nicht erreichen. Teilweise werden die Anforderungen auch sehr abstrakt definiert (*„... die Funktion xy muss die Qualität besitzen, um den Status „Best in Class" erreichen zu können ...*"), die sich nur schwer in einem Pflichtenheft konkretisieren lassen.

Um die Schnittstelle zwischen Anforderungs- und Systemspezifikation möglichst sauber und verlustfrei zu spezifizieren, eignen sich formalisierte Notationsformen, die präzise Semantikeigenschaften aufweisen und auf einer eindeutig definierten Syntax basieren (vgl. Abs. 2.3.2). Die Umsetzung einer modellgetriebenen Entwicklung bietet die Möglichkeit zur Formalisierung von rein textuell beschriebenen Spezifikationen. In /[BFM05]/ werden die Anforderungen klassifiziert in Schichten abgebildet, welche formal in Modellen gepflegt werden. Dazu gehören Ebenen zur Funktions-, Vernetzungs- und Bauraumspezifikation. Ergänzend sind Sichten zur Funktionstypisierung in Bibliotheken, zur Leistungsversorgung, zur Leitungsverteilung und für den Komponentenaufbau addiert worden (s. Abs. 2.5.2)[4] /[AQU06]/. Der Vorteil dieses Ansatzes liegt in der unabhängigen Spezifikation der Anforderungen in verschiedenen Ebenen, deren wechselseitigen Abhängigkeiten frühzeitig inkonsistente Anforderungen aufdecken. Zusätzlich werden eine Variantenpflege und der Vergleich unterschiedlicher Lösungen unterstützt und ein Konzept zum nebenläufigen Entwickeln (*Concurrent Engineering*) angeboten.

Im Gegensatz zur dezentralen verteilten Systemspezifikation in Form von mehreren Lastenheften erfordert der modellgetriebene Ansatz eine verzahnte Schnittstelle zwischen allen beteiligten Entwicklern. Gleichzeitig muss ein durchgehend einheitliches Verständnis für eigenständige Notationsformen sowie deren Zusammenspiel, Syntax und Semantik vorhanden sein.

Die in dieser Arbeit definierte Entwicklungsmethodik für flexible zeitgesteuerte Systeme basiert partiell auf der Basis eines modellbasierten E/E-Architekturentwicklungswerkzeugs /[Aqu07]/. Dabei wird in diesem Ansatz die Interpretation der entwickelten Anforderungen und deren Relation zu den Produktfeatures und Fahrzeugfunktionen aus einem modellbasierten Ansatz gewonnen. Die Grundidee basiert auf der Extraktion aller Ereignisse aus dem Lastenheft und deren Zuordnung zu einer Funktionalität. Die Funktionalität selbst wird mit konkreten Objekten verknüpft, die die technischen Inhalte der Funktionalität erfüllen. Durch diesen Ansatz ergibt sich ein Szenario der Definition eines Dreitupels aus dem anfordernden Objekt, der Funktionalität und dem erfüllenden Objekt (*Requestor, Function, Fulfiller*). Diese RFF-Kombination wird als *Wirkkette* bezeichnet. Jede Wirkkette spezifiziert dabei die Ausprägung eines Ausstattungsmerkmals. Da eine Funktionalität mehrere Ausstattungsmerkmale eines Fahrzeugs abdecken kann, existiert die Möglichkeit auf Basis der Menge aller Wirkketten sämtliche Ereignisse und die korrespondierenden erfüllenden Objekte einer spezifischen Funktionalität zu identifizieren. Diese Abbildung wird in einer spezifischen Darstellung erfasst und in Abs. 5.1.2 exemplarisch für eine Fallstudie dokumentiert.

[4] Einen weiteren modellbasierten Ansatz bietet die *SysML*-Spezifikation, die sich ebenfalls auf eine modellbasierte Anforderungsdefinition stützt (s. Abs. 2.2.4.2).

Spezifikation kundenorientierter Produktsubstanzen

Beim Entwurf und der Pflege einer E/E-Architektur steht die Erfüllung der Anforderungen an die Fahrzeugelektronik aus Kundensicht im Vordergrund. Um diese Eigenschaften des zu entwickelnden Produkts bestmöglich zu erfassen, muss in erster Linie eine vollständige, abstrakte Modellierung der Systemfeatures erfolgen. Darunter verstehen sich die Produkteigenschaften, die sich dem Kunden als erlebbarer Teil des Gesamtsystems präsentieren, beispielsweise der Einklemmschutz eines elektrischen Fensterhebers. Ein Feature darf dabei nicht zwangsläufig mit der Ausprägung physikalischer Systemkomponenten in Form eines Steuergeräts gleichgesetzt werden, auch wenn Features inhärent als Teil einer Systemkomponente existieren können. Speziell hochwertige Produktsubstanzen werden über die integrierte Kombination mehrerer Teilfunktionen erzielt, welche sich nicht immer eindeutig einzelnen Fahrzeugdomänen wie dem Antrieb oder dem Infotainment zuordnen lassen. Die Idee der *„aktiven Fahrsicherheit“* lässt sich dabei exemplarisch nennen. Wie in Abb. 2.14 dargestellt wird die *„Aktive Fahrsicherheit“* als integriertes Feature erzeugt.

Zum aktuellen Zeitpunkt gilt die strukturierte Featureentwicklung noch als Forschungsthema, da die Signifikanz dieser Entwicklungsphase industrieweit aufgrund deren abstrakten Komplexität in der Vergangenheit aus Fahrzeugherstellersicht unterbewertet wurde. Teilweise wird die Featureentwicklung in Form rudimentärer Tabellen und Listendarstellungen durchgeführt. Es erfolgt dabei eine unscharfe Trennung zwischen dem Begriff einer technischen Fahrzeugfunktion und deren physikalischen Implementierung im Steuergerät. Eine Strukturierung erfolgt maximal durch einfache Hierarchiebildung der Features innerhalb von in Katalogform angelegten Funktionsgruppierungen. Dabei geht die Möglichkeit der Modellierung domänenübergreifender Features sowie die Darstellung der wechselseitigen Abhängigkeiten atomarer Features verloren.

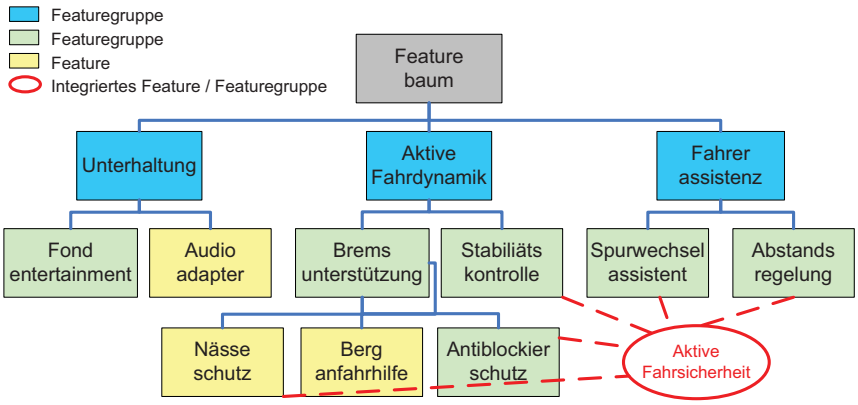


Abbildung 2.14.: Ausschnitt aus einer Featuredarstellung in Baumstruktur zur Strukturierung von Systemfeatures

In Abb. 2.14 wird ein Teilausschnitt einer potentiellen Featuremodellierung in Baumstruktur anhand einer fiktiven vereinfachten E/E-Systemarchitektur dargestellt. Dabei lässt sich eine Hierarchiebildung unterschiedlicher Featuregruppen und -untergruppen

erreichen. Auch die Modellierung atomarer Features an den Baumblättern wird dabei unterstützt. Interessant wird das Einfügen der erwähnten integrierten Features, hier in Form der „*aktiven Fahrsicherheit*“. Es wird ersichtlich, dass eine vollständige sequenzielle Abbildung der Systemfeatures in einem einzelnen baumförmigen Graphen nicht zweckmäßíg darstellbar ist. Speziell das Erfassen von normalisierten Anforderungen innerhalb atomarer, wiederverwendbarer Features, die Featurebaumgenerierung sowie die Hinzunahme von Entwicklungsschablonen (*Templates*) zur Vervollständigung der einzelnen Features im Sinne der Vertiefung des Domänenwissens wird in /[HFP+06]/ behandelt. Dabei wird auch die Abbildung semantischer Aspekte zwischen den Elementen eines Featurebaums innerhalb eigenständiger Semantikbäumen erläutert.

2.2.4.2. Systementwurf

Aufgrund unterschiedlicher Strategien zwischen Fahrzeugherstellern und der hohen Komplexität im Systemverbund existieren bis heute im E/E-Bereich noch keine einheitlichen Standards zur Systemspezifikation unter Berücksichtigung der nicht-/funktionalen Anforderungen (s. Abs. 2.2.4.1). Allgemein lassen sich die Themen nach verschiedenen Kriterien klassifizieren:

- Netzwerkebene und Komponentenebene
- Plattform und Varianten
- Applikationsebene und Systemebene

Dem Systemintegrator muss klar sein, welche Aspekte sich auf das Netzwerk auswirken oder durch die Spezifikation des Netzwerks beeinflusst werden und welche Anforderungen sich nur auf die einzelnen Komponenten beziehen. Diese Differenzierung ist entscheidend, um eine verbesserte Flexibilität des Netzwerks in Bezug auf zusätzliche Systemintegrationen zu erreichen und einen potentiellen Reengineering-Aufwand zu reduzieren. Die Spezifikation auf Komponentenebene muss konsequenterweise entkoppelt von der Netzwerkebene erfolgen. Falls aus Kostengründen eine Gleichteilestrategie für verschiedene Fahrzeugbaureihen existiert, so wirkt sich diese Vorgabe stark auf den Systems Engineering-Prozess aus. Es gilt zu analysieren welche Komponenten (Hard-/Software) Variantencharakter aufweisen und welche Teile plattformübergreifend implementiert werden können.

Während sich die Netzwerk-/Komponentenunterscheidung der globalen Dekomposition zuwendet, zielt die Differenzierung zwischen System- und Applikationsebene auf die funktionalen Trennung innerhalb einer Komponente. Wie in Abschnitt 2.2.4.1 beschrieben können die nutzerneutralen Funktionen in der Regel der Systemebene zugeordnet werden, da sich diese vorwiegend auf Systemdienste beschränken. Die kundenerlebbare Funktionalität wird innerhalb der Komponenten auf Applikationsebene fixiert. Aus Sicht des Systementwicklers würde sich noch eine weitere Aufgliederung der Funktionalität in Hardware- und Softwareanteile eignen. Bei komplexen Komponenten, wie zum Beispiel der zentralen Infotainmenteinheit *Head Unit* (etwa für Radio, Navigation, Telefonbetrieb), werden zum Teil Lösungen unter Verwendung von konfigurierbaren FPGA-Hardwarebausteinen appliziert, die sich auf die Konzepte des Hardware/Software-Codesigns stützen. Dabei können Basisumfänge des Steuergeräts, etwa Hardwareabstraktionsschichten oder Graphikberechnungen, einheitlich für verschiedene Anwender eingesetzt werden, indem entwickelte logische Modelle (bspw. als VHDL-Code)

auf Hardwarebausteine abgebildet werden. Eine kundenspezifische Produktdifferenzierung lässt sich dabei individuell durch Adaptionen der Grundfunktionalität im Basismodell erreichen. Aus Sicht des Fahrzeugherstellers wird diese Technik bisher eingeschränkt angewendet, da der Großteil der in Steuergeräten verbauten Chips aus anwendungsspezifischen integrierten Schaltungen (ASIC) besteht. Diese werden kosteneffizient und proprietär hergestellt und widersprechen im Fundament dem Plattformkonzept. Allerdings bieten sich hierbei entscheidende Vorteile im Bereich der bei der Produktion anfallenden Hardwarekosten.

Durch den starken Anstieg der Software-Komplexität durch die sukzessive Substitution mechanischer durch elektrische Systemlösungen auf aktuell bis zu 10 Millionen LoCs (*Lines of Code*) pro Fahrzeug und ca. 100 Millionen LoCs in der nächsten Generation /[Bro03], [Bro06]/ fließt dem Software Engineering eine immer größer werdende Bedeutung zu. Zum Beispiel kommen zur Beschreibung des Systemverhaltens dedizierte modellbasierte Werkzeuge zur Spezifikation von steuerungs-/regelungstechnischen Algorithmen zum Einsatz (s. Abs. 2.2.5.2).

Für die Spezifikation und Pflege von Softwarearchitekturen hat sich im Gegensatz dazu noch kein einheitlicher Standard durchgesetzt. Dies liegt zum einen an den nicht unmittelbar in den Automotive Bereich zu übertragenen klassischen Modellierungstechniken der Softwaretechnik. Viele Versuche die standardisierten Notationsformen für Objektorientierung aus der allgemeinen Softwaretechnik auf den Automobilbereich zu übertragen, beispielsweise mit der Modellierungssprache UML (*Unified Modeling Language*), haben sich bisher als nicht ausreichend praktikabel erwiesen. Diese Sprache berücksichtigt nicht die Eigenschaften von Realzeitsystemen, beispielsweise deren *Timing*-Anforderungen, und es fehlten bis vor kurzer Zeit noch Möglichkeiten zur Hierarchiebildung und Kapselung von Komponenten /[JHQ+03]/. Zusätzlich ergeben sich Einschränkungen aufgrund eines fehlenden Schnittstellenkonzepts zwischen den Komponenten. Aus technischer Sicht ist beispielsweise keine exakte Beschreibung von Konnektoren zwischen Komponenten möglich, da asynchroner Signalaustausch zwischen Komponenten auf Basis von Protokollen nicht berücksichtigt wird. Diesen Einschränkungen hat sich die UML für Echtzeitanwendungen (*UML-RT*) gewidmet, die sich durch die Option zur Modellierung der Verhaltensspezifikation für nebenläufige Systeme auszeichnet /[Krü01]/.

Da sich UML-RT aufgrund fehlender Architektursichten und der Abwesenheit einer formalen Überprüfbarkeit auch nicht optimal für die Entwicklung eingebetteter Systeme eignet, hat sich in den vergangenen Jahren eine Initiative zur Spezifikation einer standardisierten Modellierungssprache für technische Systemarchitekturen (*SysML*) gebildet /[Sys06], [Wei06]/. Dieser Standard konzentriert sich auf die domänenspezifische Modellierung von Systemen unter Verwendung von Systemeinschränkungen (*Constraints*) und unterstützt eine Trennung der logischen und technischen Aspekte in der Systemmodellierung. Zusätzlich existiert eine eigenständige Modellierungsform für Anforderungen, welche sich jeweils zu Testfällen zuordnen lassen (*Traceability*), um die vollständige Verifikation der Systemanforderungen im Konzept zu garantieren. Trotz der vielseitigen Einsatzmöglichkeiten muss sich die SysML noch in der Praxis etablieren und intensive Werkzeugunterstützung erhalten.

2.2.4.3. Implementierung

Die Implementierungsphase bezieht sich auf die Umsetzung der spezifizierten Fahrzeugfunktionen in Steuergerätesoftware für dedizierte Steuergeräteplattformen auf Basis vorliegender Lastenhefte oder entwickelter Funktionsmodelle. Grob lassen sich drei Aufgabenbereiche abgrenzen. Einerseits werden die kundenerlebbaren Funktionen in Modelle umgesetzt und Seriencode generiert oder konventionell in einer Hochsprache programmiert. Diese Funktionsmodule werden in einem weiteren Schritt mit den Hardware-, Systemkonfigurations-, Betriebssystem- und Bussystemspezifischen Softwaremodulen zu einer Gesamtsoftwarearchitektur verbunden. Dabei müssen die Hardwarekonfigurationen und deren Auswirkungen, etwa den Restriktionen beim Laufzeit und Ressourcenverbrauch, als separate Aufgabe analysiert werden. Ein entsprechend einsatzorientiertes Hardwarelayout ist dabei zu entwerfen, welches Spielraum für Systemerweiterungen im Softwarebereich über den gesamten Steuergeräteentwicklungszyklus bietet. Gleichzeitig sollte eine Überdimensionierung der Steuergerätehardware vermieden werden, um den stringenten Kostenanforderungen im Fahrzeugserienbereich gerecht zu werden.

2.2.4.4. Verifikation und Validierung

Im Bereich der rechten Seite des V-Modells wird grundlegend zwischen rein theoretischer und praktisch kombinierter Verifikation und Validierung (*Testen*) unterschieden. Die theoretische Verifikation stützt sich auf formalisierte Modelle des Gesamtsystems, welche mithilfe automatischer Theorembeweiser auf Korrektheit untersucht werden. Obwohl die theoretische Verifikation, im Vergleich zum praktischen Test, eine kosteneffiziente Variante zur frühzeitigen Prüfung von Entwicklungsergebnissen im Entwicklungsprozess darstellt, wird diese Art der Prüfung im Fahrzeugserienbereich nur eingeschränkt berücksichtigt. Dabei birgt speziell die getrennte Entwicklung von logischer und technischer Systemarchitektur mit anschließender manueller Partitionierung die Gefahr eines Fehlverhaltens oder der Verletzung von Vorbedingungen im Zeitbereich /[BGH+06]/. Je später derartige Fehler im Entwicklungsprozess identifiziert werden, desto größer wird dabei der Behebungsaufwand, was die Relevanz der frühzeitigen theoretischen Verifikation auf Systemspezifikationsebene unterstreicht. Mit der Entwicklung allgemein zugänglicher standardisierter formaler Modelle könnte dieser Methode zukünftig mehr Aufmerksamkeit zukommen. In /[BBG+05]/ wird das Konzept der formalen Verifikation der niederen Schichten eines eingebetteten System im Fahrzeug auf Basis der FlexRay-Technologie erläutert.

Aktuell wird in der Fahrzeugserienentwicklung dem isolierten und dem gruppenorientierten Steuergerätetest die größte Bedeutung zugemessen. Dabei wird die eigentliche Netzwerkfunktionalität getrennt von den Fahrzeugapplikationen (HIL-Test (*Hardware-In-The-Loop*)) getestet. Partiell wird die Funktionstüchtigkeit auch direkt im Fahrzeug untersucht (*Validierung*), was einerseits verlässliche Ergebnisse zulässt, aber andererseits mit hohen Kostenaufwänden verbunden ist.

2.2.5. Methoden und Werkzeuge

Vorrangig für die Umsetzung von kundenrelevanten Funktionen haben sich mittlerweile ausgereifte Methoden der modellbasierten Softwareentwicklung und -generierung in der Automobilindustrie zum Stand der Technik entwickelt. Aber auch im Sinne der

Prozessautomatisierung werden für die Zwecke der Anforderungsentwicklung und -revision, der Standardsoftwareentwicklung und der Testdurchführung dedizierte Werkzeuge eingesetzt. Im Folgenden werden die wichtigsten Vertreter aus dem Bereich des Systemdesigns und -entwurfs zur Übersicht vorgestellt.

2.2.5.1. Klassifikation

In der vernetzten Fahrzeugelektronik spiegeln sich klassischerweise zwei verschiedene Systemtypen wider, deren Ausprägungen definiert werden müssen. Der eine Anteil basiert auf stark regelbasierten Konzepten, bei denen Ist-Werte, gemessen durch Sensoren, mit durch von Sollwertgebern definierten Soll-Werten verglichen werden und potentielle Abweichungen durch Aktoren, die über eine spezifizierte Strecke regeln können, ausgeglichen werden /[SZ03]/. Eine zweite Klasse beschreibt die Systeme, welche einen stark ereignisorientierten reaktiven Charakter aufweisen, der stark von den Benutzereingaben durch Taster und Schalter des Benutzers abhängig ist. Während die Regel-/Steuerungssysteme vorwiegend in der Fahrdynamik eingesetzt werden, beschreiben die ereignisorientierten Systeme Funktionalität im Komfortsegment eines Fahrzeugs.

Die Grundeigenschaften der verschiedenen Systemarten werden in Abs. 4.2.1 tiefergehend betrachtet.

2.2.5.2. Notationsformen und Werkzeuge

Im Bereich der Entwicklung eingebetteter Systeme kommen mittlerweile für verschiedenste (Teil-)Aufgaben etablierte Werkzeuge mit hohem Reifegrad zum Einsatz, deren Qualität aufgrund einer kontinuierlichen Weiterentwicklung und einer hohen Marktdurchdringung als Entwicklungsstandards sukzessive verbessert wird /[SHH+03]/. Um den Anteil an aufwändigen proprietären Systemlösungen gering zu halten und die Anzahl an manuellen Fehlern zu minimieren, werden die Werkzeuge eingesetzt, um die Entwicklungszeit bei gleichzeitig steigender Qualität zu verkürzen. Dabei gilt es zu beachten ob Werkzeuge den Entwicklungsprozess und seine Artefakte besser strukturieren, organisieren und versionieren sollen, beispielsweise im Bereich der Anforderungsanalyse, oder ob damit konkrete Systemdesigns spezifiziert bzw. implementiert werden sollen. Der *Systemintegrator* konzentriert sich dabei in der Regel verstärkt auf technische Werkzeuge zur *Anforderungsanalyse*, *Systemspezifikation* und der *Gesamtsystemvalidierung* während der *(Teil-)Systementwickler* vorwiegend konkrete *Systementwurfswerkzeuge* einsetzt.

Da sich die Vorgehensmodelle zur Systementwicklung bei den Systemintegratoren an den Konzepten des V-Modell 97 orientieren werden folgende Entwicklungsphasen berücksichtigt:

Anforderungsspezifikation

Im Bereich Anforderungsanalyse und -management werden die *nichtfunktionalen* und *funktionalen Anforderungen* zumeist von getrennt operierenden Entwicklern des Systemintegrators durch die Erstellung von Lastenheften spezifiziert. Die Lastenhefte können dabei für gekapselte elektrische Teilsysteme, etwa ein Steuergerät, oder übergreifend für Querschnittssysteme, beispielsweise der FlexRay-Vernetzung, angelegt werden. Diese Lastenhefte werden aufgrund fehlender Standards und zur Verständniserleichterung

größtenteils textuell spezifiziert, gemischt mit Abbildungen, Tabellen und teilweise halbformalen Notationstechniken. Diese setzen sich in der Regel aus schlichten Zustandsmaschinen ohne einheitliche, vollständige Semantiken und mit zumeist textueller Syntax zusammen. Im regelbasierten Bereich werden partielle Systemkomponenten, beispielsweise ein Softwaremodul für die Schaltstrategie im Getriebe, durch Modelle spezifiziert.

Als etabliertes textuelles datenbankbasiertes Werkzeug zur Ablage und Verknüpfung von Anforderungsspezifikationen (*Traceability*) lässt sich in dem Zusammenhang *Telelogic Doors* /[Tel06]/ (oder alternativ auch *IBM Rational RequisitePro* /[Int08]/) aufführen.

Steuer- und Regelbereich

Die modellbasierte Abbildung von Regelungssystemen mit automatischen, leistungsfähigen Codegeneratoren gilt heute als Stand der Technik /[BS06, The07a, The07b]/. Die Unterstützung des Entwicklungsprozesses wird in dem Zusammenhang durch die Notationsformen der rechnergestützten Werkzeuge zur Modellierung gemischt diskret-/kontinuierlicher Systeme, den *Hybridsystemen*, maßgeblich beeinflusst. So bietet die verbreitete Entwicklungsmethodik auf Basis der CAE-Werkzeugkette MATLAB/Simulink/Stateflow exakte Vorschriften zur Spezifikation und Partitionierung zeit-/ereignisdiskret gesteuerter Systeme auf der Grundlage von Zustands-/Transitionsbasierten Komponenten oder zum Entwurf von kontinuierlicher Dynamik bzw. Gesetzen zur Steuerung des Systems /[Sta01]/.

Für die Entwicklung reaktiver Systeme, die einen stark ereignisgesteuerten Charakter aufweisen, etwa einem Sitzsteuergerät, werden Notationsformen herangezogen, die auf der Idee von Zustandsmaschinen aufsetzen. Ebenso wie für die regelungstechnischen Systeme gilt auch hier, dass die Notation eine halbformale Beschreibung darstellt, welche die Anbindung von Codegenerierungswerkzeugen ermöglicht. Während für Einzelkomponenten durch diese Entwicklung ein großer Fortschritt erzielt worden ist, steht ein vergleichbarer Fortschritt für verteilte Systeme mit mehreren physikalisch getrennten Komponenten noch aus. Dies kann auf die hochkomplexen Eigenschaften eines Systemverbunds zurückgeführt werden. So müssen unterschiedliche Ressourcen und Performanzaspekte wie auch komplett unterschiedliche HW/SW-Lösungen von Systemkomponenten berücksichtigt werden. In dem Zusammenhang gilt es verschiedenste Schnittstellen zu bedienen und alternative Systemausprägungsmöglichkeiten miteinzubeziehen.

In der Regel unterscheiden sich Steuergeräte massiv in ihrem HW/SW-Design und welche Funktionen in Hard- oder Software gelöst werden. Durch Simulationen können bei Umwandlung zwischen diesen beiden Typen beispielsweise Rundungsfehler auftreten, welche eventuelle zu einer numerischen Instabilität führen. Speziell bei der Verwendung von Gleitpunktarithmetik kommt es aufgrund von Fehlerfortpflanzungen zu Effekten mit potentiellen Stellenauslöschungen und aufgeschaukelten Berechnungsfehlern, deren Auswirkungen sich mit frühzeitigen Simulationen in der Entwicklung analysieren lassen. Funktionen zur Abbildung von gemischt diskret-/kontinuierlichen Reglermodellen können mithilfe von datenflussbasierten Notationen wie den Blockdiagrammen, welche aus der Regelungstechnik stammen, rechnergestützt modelliert werden, welche die Anschlussmöglichkeit von leistungsfähigen C-Codegeneratoren bieten.

Die *Matlab/Simulink*-Entwicklungsumgebung /[The07a, The07b]/ mit ergänzender Hardware und Codegeneratoren dritter Anbieter (z.B. *dSpace* /[BS06]/) lässt sich vorrangig als wichtiger Vertreter im Bereich der Entwicklungswerkzeuge nennen. Als Al-

ternative existiert auch die *ETAS ASCET*-Produktpalette, welche ein ähnliches Anwendungsspektrum bedient.

Analyse stochastischer Prozesse

Ergänzend zur den geschlossenen regelungstechnischen Systemen (*Closed Loop Control*) treten im Fahrzeugkontext eine Vielzahl an Ereignissen auf, etwa die Betätigung eines Fensterhebers, deren Existenz wahrscheinlichkeitsabhängig betrachtet werden muss. Derartige Ereignisse lassen sich am ehesten innerhalb von stochastischen Prozessen betrachten und analysieren. In /[Bor94]/ werden für diese Bereiche stochastische Modelle verwendet, die sich werkzeuggestützt simulieren lassen. Als Werkzeugumgebung eignet sich hierfür beispielsweise die *SES/Workbench* /[BTB91]/ oder *Foresight* /[For02]/. Die Modellierung und Simulation diskret stochastischer Prozesse hat sich im Automobilbereich bis heute jedoch nur eingeschränkt verbreitet.

2.2.6. Normen und Standards

Der fortschreitende Einzug von E/E-Systemen im Fahrzeug ist in den letzten 20 Jahren durch verschiedene Meilensteine beschleunigt worden, die heute branchenweit als Industriestandards anerkannt und eingesetzt werden.

Ergänzend zu dem technischen Fortschritt mit der verbreiteten Einführung der CAN-Technologie ist mit dem geschaffenen OSEK/VDX-Standard (*Offene Systeme und deren Schnittstellen für die Elektronik im Fahrzeug*) ein entscheidender Fortschritt in der Entwicklung der Fahrzeug-E/E /[OSE07]/ gelungen. Dieses herstellerübergreifende Konsortium hat Standards in den Bereichen Steuergerätebetriebssysteme, Software-Kommunikationsschichten zwischen System- und Applikationsebene sowie Netzwerkmanagement (kurz NM) etabliert. Weiterhin ist eine Spezifikation zur fehlertoleranten Kommunikation (FTCOM) verabschiedet worden, die sich unter anderem mit der Organisation einer redundanten Datenübertragung bezüglich replizierter Signale oder Botschaften beschäftigt.

Parallel werden im HIS-Konsortium /[HIS07]/ prozessorientierte und produktspezifische Themen adressiert. Dabei ist ein Ziel harmonische Standards bezüglich der Anforderungen und Schnittstellen zu schaffen, um den Bemühungen der Zulieferer den Anforderungen der Fahrzeughersteller möglichst effizient gerecht zu werden nachzukommen. Dabei werden auch klassische Themen der Softwaretechnik, beispielsweise die Prozessbewertung (*Process Assessment*) zur Leistungs- und Reifegradbestimmung behandelt. Zur Bewertung der Zulieferer wird das in der Herstellerinitiative *AutomotiveSPICE* /[Aut06]/ entwickelte Prozessreferenz- und Prozessbewertungsmodell herangezogen, welches sich an den Standards ISO/IEC15504 (SPICE) und CMM(I) orientiert. Die Entwicklungen rund um *AutomotiveSPICE* beruhen auf der Erkenntnis, dass allgemeine Prozessreifegradmodelle zur Verbesserung der Softwaresystementwicklung in der Automobilbranche teilweise nur unzureichend berücksichtigt werden. Viele Hersteller haben beispielsweise den Reifegrad SPICE-Level 2 noch nicht erreichen können (s. Abb. 2.15). Weiterhin wurde in HIS ein Referenzmodell zum Software-Test erarbeitet und ein eigener Satz an Kodierrichtlinien definiert, der sich aus einer Submenge der MISRA C-Regeln /[MIS04]/ zusammensetzt. Ergänzend existieren Standardisierungen in den Bereichen Simulation und Werkzeuge, Flashprogrammierung in Steuergeräten sowie Standardsoftware, etwa CAN-Treiber- oder Bootloaderspezifikationen.

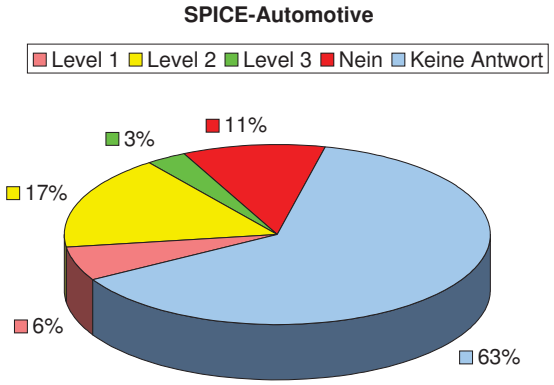


Abbildung 2.15.: Spice-Reifegradlevel in der Fahrzeugindustrie /[Har06]/

Aktuell stehen im Mittelpunkt die Arbeiten rund um die internationale AUTOSAR-Initiative (*Automobile offene Systemarchitektur*) /[AUT07a]/. Auf die Ziele dieses Konsortiums wird in Abs. 2.2.6.3 eingegangen.

2.2.6.1. CAN

Das CAN-Feldbussystem /[Rob91]/ verkörpert einen der wichtigsten Meilensteine in der modernen E/E-Entwicklung im Automobil. Seit seinem ersten Serieneinsatz 1991 in einem Premiumfahrzeug (Mercedes Benz S-Klasse) hat sich die Technologie als Standard weltweit im Bereich der Vernetzung für Antriebs-, Fahrwerks- und Komfortelektronik durchgesetzt. Das Konzept ist 1983 von der Firma Bosch mit dem Ziel der Vernetzung von Steuergeräten in Automobilen initiiert worden. Als Treiber dieser Entwicklung gilt vorrangig der gewachsene Anteil an Kabelverbindungen im Fahrzeug, der sich kosten- und gewichtsmäßig negativ bei der Fahrzeugserienentwicklung auswirkt. Die technischen Eigenschaften der CAN-Technologie und die Erkenntnisse der Integration von Bussystemen in die Fahrzeugserienentwicklung werden in Abs. 2.4.1 erläutert.

2.2.6.2. OSEK/VDX

Im Sinne einer ersten Standardisierung von Softwarearchitekturen im Fahrzeug erwies sich die 1993 gegründete Initiative der deutschen Automobilbranche OSEK als wegweisend in der E/E-Entwicklung. Durch die Ergänzung von OSEK durch den VDX-Ansatz (Vehicle Distributed eXecutive) aus der französischen Automobilbranche ist ein harmonisierter Standard definiert worden, der internationale Akzeptanz genießt. Ziel der Vereinigung war die Schaffung von Kompatibilität zwischen Steuergeräten unterschiedlicher Hersteller und die Reduktion der Kosten für nicht applikationsspezifische Softwareentwicklung in Steuergeräten. OSEK/VDX umfasst übergreifend drei Kernbereiche:

Datenkommunikation zwischen Steuergeräte: Die Basis der netzwerkbasierten Datenkommunikation zwischen Steuergeräten stellt eine einheitlich spezifizierte Netzwerkschnittstelle. Diese Schnittstelle kann aber auch innerhalb eines Steuergeräts angewendet

werden. Insgesamt bildet das Datenkommunikationsmodul demzufolge eine Interaktionsschicht im Steuergerät mit Anforderungen und Schnittstellen zur Sicherungsschicht, dem Netzwerkmanagementmodul und der Applikationssoftware.

Netzwerkmanagementkonzept: Wechselseitige Einflüsse und Abhängigkeiten zwischen den Netzwerkteilnehmern erfordern ein plattformunabhängiges Management des Netzwerks. Per Datenaustausch über das Netzwerk können Systemzustände der angeschlossenen Steuergeräte überwacht und notwendige Maßnahmen zur Garantie von Zuverlässigkeit und Sicherheit des verteilten Gesamtsystems ergriffen werden.

Betriebssystemkonzept für Echtzeitsysteme: Das OSEK/VDX-OS spezifiziert Dienste und Prozesse für eine nebenläufige Ausführung mehrerer Applikationen in Echtzeit. Drei wesentliche Ausführungslevel spielen dabei eine Rolle. Dazu gehört der Umgang mit Prozessunterbrechungen, den allgemeinen Prozessen (Tasks) und den betriebssystemspezifischen Aktivitäten, beispielsweise dem Ereignis und Taskmanagement, der Verwaltung der Betriebsmittel oder der Fehlerbehandlung.

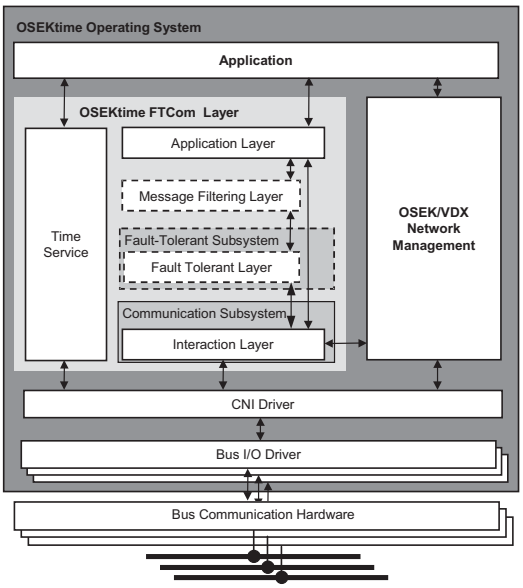


Abbildung 2.16.: Softwarearchitektur auf Basis des OSEK-Konzepts /[OSE07]/

Als Erweiterung von OSEK/VDX zählt der OSEKtime-Ansatz, ein Konzept zur Erfüllung potentieller zeit- und sicherheitskritischer Anforderungen in Steuergeräten. Zu OSEKtime zählen folgende Module:

Betriebssystemkonzept für harte Echtzeitanwendungen: Das OSEKtime-Betriebssystem stellt notwendige Dienste zur Unterstützung fehlertoleranter verteilter Applikationen mit erhöhten Zuverlässigkeitsanforderungen. Die a priori und statisch zur Laufzeit festgelegte Systemkonfiguration bietet dabei fehlertolerante Eigenschaften im Bereich des Systemstartvorgangs, der Nachrichten-, Unterbrechungs- und Fehlerbehandlung sowie einer Schnittstelle zur Bereitstellung von Zustandsinformationen. Eine Kernidee in OSEKtime ist die zeitliche Synchronisation der internen Prozesse mit einer vorgegeben globalen Zeit im System.

Fehlertolerante Kommunikationsschicht FTCom: FTCom beschreibt mehrere Softwareschichten auf der Applikationsebene und für die Interaktion zwischen den Steuergeräten. Entscheidend ist die Hinzunahme einer fehlertoleranten Schicht zum Umgang mit mehreren Instanzen von Nachrichten bei der Anwendung von redundanter Datenübertragung. Als Ergänzung dient ein weiterer optionaler Mechanismus zur Nachrichtenfilterung.

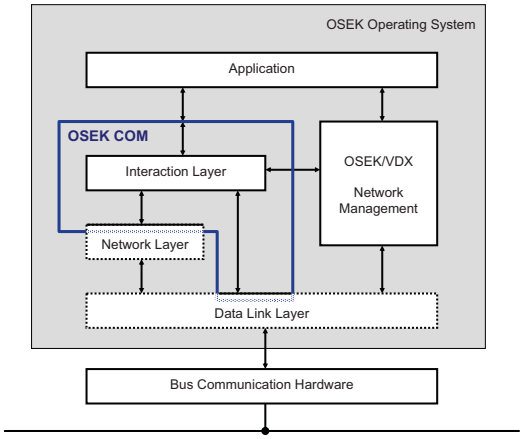


Abbildung 2.17.: Softwarearchitektur auf Basis des OSEKtime-Konzepts /[OSE01]/

In den letzten Jahren waren in erster Linie die Konzepte des allgemeinen OSEK/-VDX-Standards im Fahrzeug etabliert. Aufgrund der eingeschränkten Anwendungsmöglichkeiten der fehlertoleranten OSEKtime-Variante, sind diese Konzepte verstärkt im Forschungsbereich innerhalb der E/E-Architekturen von PKWs appliziert worden. Diese Entwicklung lag nicht zuletzt an den bisher nicht verfügbaren serienreifen fehlertoleranten Netzwerktechnologien für das Fahrzeug[5]. Mittlerweile werden etablierte OSEK/VDX-kompatible Softwaremodule durch adjazente Gruppen an Softwaremodulen des AUTOSAR-Standards sukzessive ersetzt.

2.2.6.3. AUTOSAR

Die größte internationale Standardisierungsinitiative der letzten Jahre im Bereich der E/E-Entwicklung im Automobilbereich bildet das AUTOSAR-Konsortium (Automotive Standard Software Architecture) /[AUT07a]/. Basierend auf dem Erfolg des OSEK-Standards in der Entwicklung von eingebetteten Systemen im Fahrzeug hat sich AUTOSAR zum Ziel gesetzt einen allgemeinen E/E-Standard zur vereinfachten Integration und Austauschbarkeit von Softwaremodulen zu etablieren /[H. 06]/. Die Entwicklung des Standards umfasst dabei folgende Kernbereiche:

Modularer Aufbau von Applikationssoftware: Aufgrund des hohen Innovationspotentials kundenorientierter Softwarefunktionen soll zukünftig eine stärkere Trennung von Steuergerät- und Funktionsentwicklung möglich werden. Mittelfristig werden Funktionsmodule als Bibliotheken separat entwickelt, welche individuell in verschiedensten

[5]Mit dem TTCAN-Standard /[Rob02]/ existiert zwar eine erweiterte Spezifikation des CAN-Standards zur Abdeckung der fehlertoleranten Anforderungen im Zeitbereich. Dieser Standard hat bis heute jedoch in der Praxis keine signifikante Bedeutung erlangt.

Steuergeräten verbaut werden können. Ziel dieses Ansatzes ist die weitgehende Unabhängigkeit bei der Entwicklung von Applikationssoftware von dem zugrunde liegenden Steuergerät und in der Vereinfachung der Partitionierung. So kann beispielsweise eine Fensterheberfunktion unabhängig von dem Lieferanten eines Türsteuergeräts vom Fahrzeughersteller erworben und in die E/E-Architektur integriert werden.

Basis für diesen Ansatz ist die Idee eines virtuellen Kanals VFB (*Virtual Functional Bus*) im Fahrzeug zum Austausch von Funktionssignalen. Dieser residiert auf dem Konzept einer einheitlichen Laufzeitumgebung RTE (*Runtime Environment*), welche in AUTOSAR-konformen Steuergeräten implementiert wird (s. Abb. 2.18).

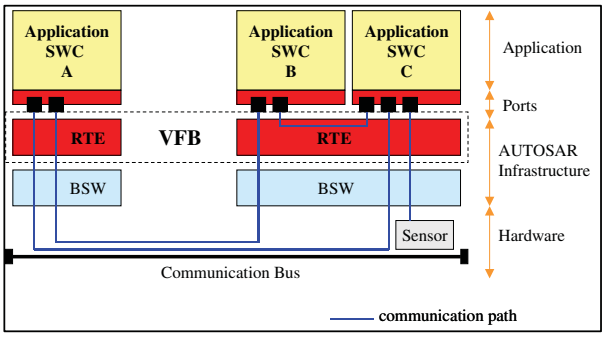


Abbildung 2.18.: Hardwareabstraktes Konzept des virtuellen Übertragungskanals für Applikationssoftwarefunktionen /[H. 06]/

Modularer Aufbau von Basissoftware: Zur Komplettierung der Applikationssoftware gilt es analog einen Standard zu entwerfen, der die Architektur grundlegender Softwarefunktionen und -dienste spezifiziert. Zu diesem Gebiet zählen hardwarespezifischer Softwaremodule wie Speicherdienste, Hardwaretreiber und -abstraktionsschichten sowie Treiber und Dienste für Netzwerktechnologien (CAN, FlexRay) (s. Abb. 2.19).

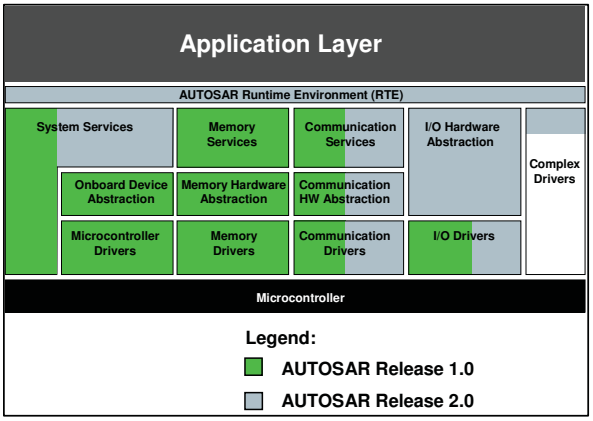


Abbildung 2.19.: Elemente einer AUTOSAR Basissoftwarearchitektur /[H. 06]/

Besondere Aufmerksamkeit erfährt in diesem Kontext die Spezifikation und Entwicklung der Netzwerkkommunikationssoftwaremodule (s. Abb. 2.20). Daher wird der Ausdruck Basissoftware in der Regel mit den spezifischen Modulen zur Netzwerkkommunikation verbunden.

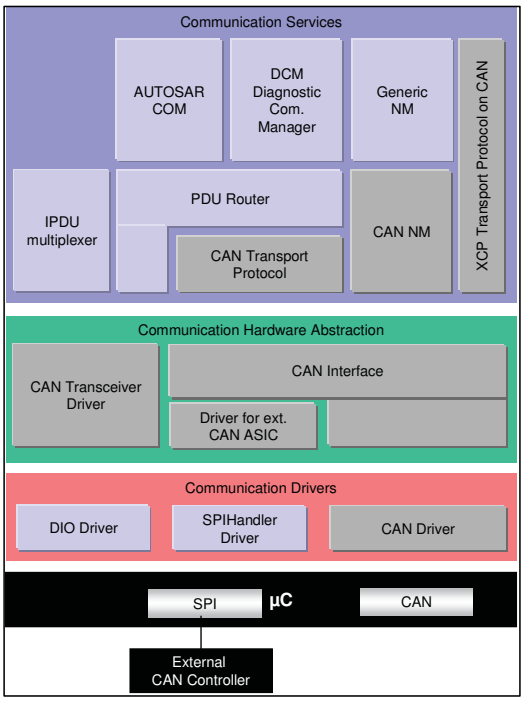


Abbildung 2.20.: Elemente einer AUTOSAR CAN-Basissoftwarearchitektur /[H. 06]/

Entwicklungsmethodik: Zur Anwendbarkeit der entwickelten Softwarearchitektur ist eine konkrete Entwicklungsmethodik notwendig. AUTOSAR versteht dabei die Definition eines Prozesses und entsprechender Schnittstellenspezifikationen hinsichtlich der Abdeckung der Softwarekonfiguration und anschließender Codegenerierung (s. Abb. 2.21).

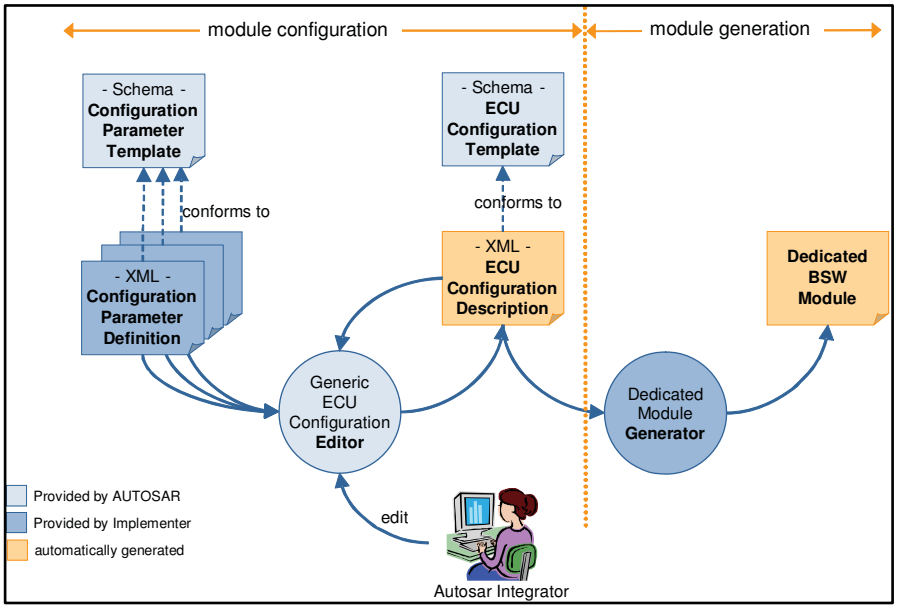


Abbildung 2.21.: Prozessschaubild und Schnittstellen der AUTOSAR-Entwicklungsmethodik /[H. 06]/

Die Komplexität und der Umfang der Inhalte in der AUTOSAR-Standardisierung sind allgemein sehr hoch einzustufen. Erschwerend muss dabei die Tatsache miteinbezogen werden, dass keine spezifischen Vorgaben hinsichtlich der Definition der konkreten Werkzeugentwicklung zur Unterstützung des Standards existieren.

Um den allgemeinen Anstieg der Komplexität in der E/E-Entwicklung langfristig begegnen zu können müssen jedoch weitere Entwicklungen im Bereich der Beschreibungssprachen zum Entwurf, Integration und Verifikation von Softwarefunktionen ergänzt werden. Zusätzlich müssen korrelierende Anforderungen hinsichtlich des Zeitverhaltens und der Systemzuverlässigkeit von eingebetteten Softwarefunktionen konkretisiert werden.

2.3. Entwicklungsmethoden der verteilten Realzeitentwicklung

In diesem Abschnitt wird der Systembegriff innerhalb dieser Arbeit mit seinen Schnittstellen zur Systemumwelt informell erläutert. Dabei werden Einprozessor- von den Multiprozessorsystemen abgegrenzt. Auf Basis der Systemeinordnung lässt sich der Zusammenhang zu Systemspezifikationsebenen ziehen. Zusätzlich werden die beiden signifikanten Kernthemen *Zeitverhalten* und *Zuverlässigkeit* im Bereich der Systementwicklung zusammengefasst.

2.3.1. Grundbegriffe und Systemkontext

Um die Anforderungen und Bedeutung des Designs eines verteilten eingebetteten Echtzeitsystems eindeutig darzustellen, wird im Folgenden die Bedeutung der einzelnen Teilbegriffe definiert:

System: Nach /[JKR63]/ besteht ein System aus einer Reihe von Komponenten, dessen Design zur Erfüllung eines speziellen Ziels passend zum Plan dient. Grundsetzlich entsteht ein System bei der Anordnung von miteinander in Beziehung stehenden Entitäten, die von ihrer Umgebung abgegrenzt werden können und sich als strukturierte Einheiten auffassen lassen /[DIN97], [ISO93]/. /[Mil73]/ bezieht den Systembegriff auf einen Satz an Konzepten und/oder Elementen zur Anforderungserfüllung. Die Definition von /[Wym76]/ konkretisiert den Begriff in technischer Hinsicht. Aus seiner Sicht gehört zu einem System notwendigerweise Eingabe-, Zustands- und in einigen Fällen auch Ausgabedefinitionen. Der Zusammenhang zwischen Systemzustand und Eingabewerte ist elementar zur Beschreibung des Systemverhaltens. Jeder Systemzustand muss jederzeit über alle relevanten Informationen zur Berechnung der gewünschten Systemausgabe verfügen. Der sehr allgemeine Begriff *System* wird in /[MG06]/ noch weiter vertieft.

Verteiltes System: Ein verteiltes System beschreibt eine Menge unterschiedlicher örtlich getrennter Prozesse, die miteinander über den Austausch von Nachrichten kommunizieren können /[Lam78]/./[CDK01]/ greift den Begriff technischer auf, indem nicht wie bei /[Lam78]/ von Prozessen, sondern von verteilten Hard- und Softwarekomponenten innerhalb von Rechnernetzen gesprochen wird, die nachrichtenbasiert kommunizieren und ihre Aktionen koordinieren. Aus Sicht eines Benutzers sieht /[TS06]/ ein verteiltes System als Menge unabhängiger Computersysteme, die dem Anwender den Eindruck suggerieren als würde es sich um einen

einzelnen Computer handeln. Tiefere Einblicke werden in /[VR01], [Web94]/ gegeben.

Eingebettetes System: /[Kop97]/ charakterisiert ein eingebettetes System als permanentes Teilelement eines gut spezifizierten größeren Systems, welches als intelligentes Produkt bezeichnet wird. Dieses besteht aus einem mechanischen Subsystem, den kontrollierenden eingebetteten Computer und meistens einer Mensch-Maschine-Schnittstelle (*Human Machine Interface*).

Echtzeitsystem: Ein Echtzeitcomputersystem ist ein Computersystem, in dem die Korrektheit des Systemverhaltens nicht nur auf logischen Berechnungsergebnissen basiert, sondern auch von der physikalischen Instanz abhängt, welche die Ergebnisse produziert. Entscheidend für die einwandfreie Funktionsweise zeigt sich, neben der Qualität und Korrektheit von Ergebnissen, der Zeitpunkt, wann diese Ergebnisse erzeugt werden /[Som01], [Kop97]/.

Wenn die verschiedenen Teilbegriffe eines verteilten eingebetteten Echtzeitsystems zusammengefasst werden, lässt sich demzufolge exakt das komplexe heterogene Umfeld erfassen, dem die Fahrzeug-E/E zugerechnet wird. Das Gesamtsystem Fahrzeug liegt dabei eingebettet in dem wechselseitig abhängigen Echtzeitsystem, bestehend aus Fahrer, Fahrzeug und Umwelt (s. Abb. 2.22).

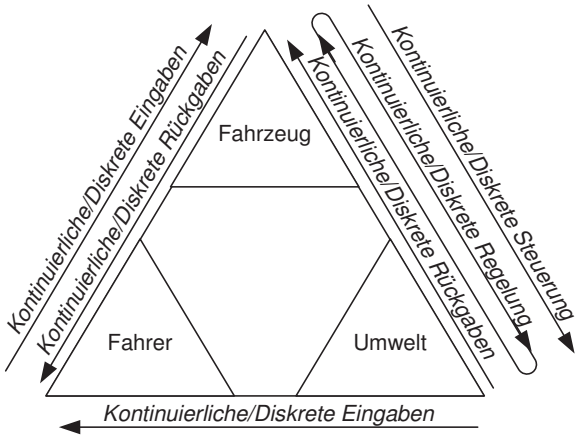


Abbildung 2.22.: Wirkungsdreieck Fahrer, Fahrzeug und Umwelt

Fahrer: Der Fahrer erhält über die Umwelt diskrete oder kontinuierliche Eingaben, beispielsweise Temperatur, Helligkeit, Geräusche, Verkehrsverhalten, welche direkt sein Verhalten beeinflussen. Diese Vorgaben in Kombination mit seinen umweltunabhängigen subjektiven Zielen führen zu den Eingaben, welches das Fahrzeug über den Fahrer erhält. Dazu gehören diskrete Aktionen, beispielsweise mittels Taster oder Schalter und kontinuierliche Eingaben, z.B. der Lenkwinkel oder die Pedalstellung, welche auf den Gesamtzustand Fahrzeug wirken.

Fahrzeug: Anhand der Fahrereingaben kann das Fahrzeug entsprechend reagieren, indem diese Werte an die Eingangsschnittstellen der Steuer- und Regelsystemen gelegt werden. Bei den Regelsystemen werden zusätzlich die Ausgangsgrößen an

die Steuereinheit zurückgeführt, um einen Vergleich mit dem erwünschten Soll-Wert ziehen und das System in einem gewünschtem Zustand erhalten zu können. Zusätzlich leitet das Fahrzeug über Sensorik autonom Informationen aus der Umwelt ab, beispielsweise mit Regen-, Licht- oder Beschleunigungssensorik und übergibt diese als Eingaben an seine elektronischen Systeme. Als Schnittstelle zum Fahrer werden Informationen kontinuierlich zum Beispiel in Form einer aktiven Lenküberlagerung oder diskret durch visuelle/akustische Ausgaben im Cockpit zurückgegeben.

Umwelt: Die Schnittstelle zum Fahrer lässt sich allgemein als unidirektional beschreiben, da der Fahrer, abgesehen von dessen Gestik und Mimik, nicht direkt auf die Umwelt einwirkt, sondern nur über das Fahrzeug als Zwischenschnittstelle. Das Fahrzeug selbst kann im Gegensatz dazu mit der Umwelt interagieren. Als Beispiel könnte wie zu Beginn das ACC genannt werden, welches sich bei zweckmäßigem Design auf das Verkehrsverhalten adaptiert ohne dabei kettenreaktionsartige Einflüsse auf die ACC-Systeme nachfolgender Fahrzeuge zu erzeugen.

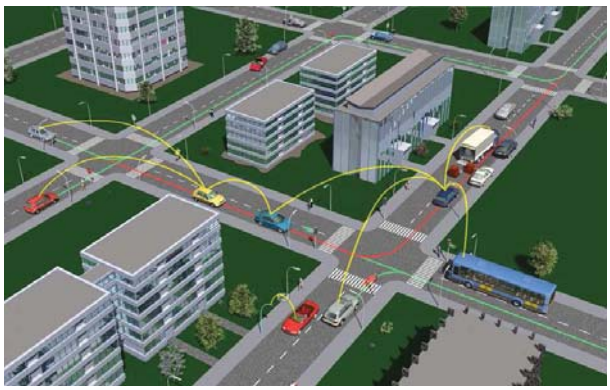


Abbildung 2.23.: Szenario eines *Car2Car*-Kommunikationsnetzwerks /[Car06]/

Ein weitreichendes Beispiel bildet die Fahrzeug-Zu-Fahrzeug-Kommunikation (*Car2Car-Communication*), bei der sich die Fahrzeuge und Umwelt mithilfe von multihop Adhoc-Netzwerken gegenseitig verständigen können /[Fre05]/.

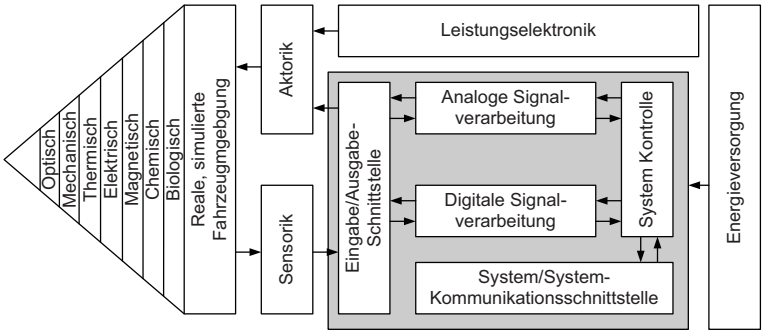


Abbildung 2.24.: Steuergerätkomponenten eingebettet in die Fahrzeugumgebung als *heterogenes System*

Wenn der Ansatz auf ein Steuergerät als Standardkomponente der Fahrzeug-E/E verfeinert wird, ergibt sich ein heterogenes System /[Bor94]/. Diese Bezeichnung entstammt der Mischung aus digitalen elektronischen Komponenten mit dazugehöriger Software und analogen Elektronik- sowie nichtelektronischer Sensorik-/Aktorikkomponenten.

Funktion: Jedes elektrisch/elektronische System eines Fahrzeugs wird durch die Implementierung eindeutiger Funktionen gekennzeichnet. Dabei propagiert sich der Begriff Funktion über die verschiedenen Fahrzeugdomänen und scheint oftmals nicht sauber definiert zu sein. Zum Beispiel ist zu unterscheiden, ob eine Funktion sich auf das Interaktionsdreieck Fahrer/Fahrzeug/Umwelt oder aber auf eine Systemkomponente und dessen Ein- und Ausgabeschnittstellen bezieht. /[Gri01]/ bezeichnet eine Funktion in einem eingebetteten System als das Ergebnis des Zusammenspiels von Rechner und Umgebung. /[SZ03]/ versteht unter Funktion die Funktionsmerkmale eines Fahrzeugs, die durch den Benutzer, in der ersten Linie durch den Fahrer des Fahrzeugs, direkt oder indirekt wahrgenommen werden und für ihn einen Wert oder Nutzen darstellen. In dieser Arbeit bezieht sich die Funktion in gleichem Sinne auf das technisch oder durch den Menschen wahrnehmbare zeitabhängige Verhalten einzelner oder mehrerer sich in Interaktion befindlicher Komponenten des Fahrzeugs. Eine Abstraktionsstufe höher müssen Funktionen über Funktionen zum Beispiel zur Verknüpfung von Fahrzeug- oder Fahrerassistenzsystemen angesiedelt werden. Das bedeutet, dass Funktionen als logische Komponenten betrachtet werden können, die die Konzepte der Hierarchisierung, Modularisierung, Strukturbildung und Atomarisierung unterstützen. Funktionen sollten klar durch ihre Anforderungen und nicht durch ihre Lösungsvarianten gekennzeichnet sein. So spielt es auf dieser Ebene keine Rolle, ob eine Benutzerinteraktion akustisch oder visuell zu erfolgen hat. Eine derartige Differenzierung in der Funktionsrealisierung muss sich demzufolge auf einer Designebene vollziehen, die getrennt vom Kontext der zugrunde liegenden Funktion betrachtet wird /[FSV99]/. Neben der informellen Darstellung lässt sich der Funktionsbegriff in der Fahrzeug-E/E auch rein mathematisch interpretieren (s. Abs. 4.2).

Die erläuterten Teilbegriffe müssen in dem Zusammenhang unter den spezifischen Gesichtspunkten der *Automotive Embedded Systems* betrachtet werden mit einem hohen Grad an Innovationen und Veränderungen in der Fahrzeugentwicklung, die im Gegensatz zu anderen Branchen mit zusätzlichen Anforderungen im Prozess der Wertschöpfung und für die Erfüllung gesetzlicher Vorschriften zu kämpfen hat. Das spiegelt sich einerseits in der starken Einschränkung an technischen Ressourcen wie Speichergröße und Rechenleistung im Elektronikbereich durch den Einsatz von kostenoptimierten Hardware-Komponenten und zum anderen in Anforderungen an Echtzeitfähigkeit in sicherheitskritischen Systemen des Fahrzeugs wieder /[FBR+06]/. Außerdem gibt es in wenigen Branchen eine derartige Vielfalt an unterschiedlichsten Zulieferer, welche zu dem Gesamtprodukt beitragen.

2.3.2. Einprozessor- und Multiprozessorsysteme

Die Konzentration der Systementwicklung hat sich in den vergangenen Jahren vorrangig auf die Entwicklung von Komponentenlösungen auf Basis von Steuergeräten (s. Abb. 2.25 und vgl. /[SZ03]/) beschränkt. Steuergeräte haben einen gemischt diskret-/kontinuierlichen Charakter, da kontinuierliche Datenflüsse aus analogen Sensoren

empfangen und gleichzeitig durch einen Mikroprozessor oder dedizierten digitalen Signalprozessor (DSP) verarbeitet werden müssen. In diesem Ansatz beziehen sich Einprozessorsysteme auf Steuergeräte in Form technisch gekapselter Einheiten. Obwohl in speziellen Fällen auch zwei Prozessoren auf einem Steuergerät verbaut werden, motiviert durch sicherheits- oder performanztechnische Gründe, wird bei Multiprozessorsystemen in diesem Zusammenhang von physikalisch verteilen Komponenten gesprochen, die innerhalb eines Netzwerks verbunden sind.

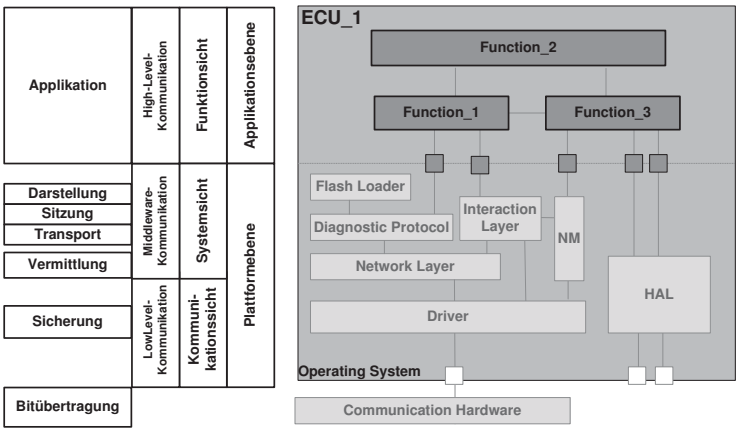


Abbildung 2.25.: Modularer Aufbau eines OSEK-konformen Steuergeräts mit geschichteten Abstraktionsstufen

Um die Komplexität eines Steuergeräts zu beherrschen und dessen Software-Komponenten getrennt entwickeln und pflegen zu lassen, hat sich ein monolithischer Ansatz als inadäquat erwiesen. Durch den OSEK-Standard (s. Abs. 2.2.6), ist ein Schritt in Richtung modulare Architektur erzielt worden. Mit der Dekomposition der Software in Module lassen sich auch verschiedene Abstraktionsstufen für ein Steuergerät deklarieren. Neben dem ISO/OSI-Ansatz lässt sich das Steuergerät prinzipiell in drei Stufen zerlegen. Die *LowLevel-Kommunikation* bezeichnet die Funktionalität der Module, welche an den Schnittstellen zu den physikalischen Netzwerk-Controllern oder weiteren Ein-/Ausgängen liegen (*Hardware Abstraction Level*). Die *Middleware-Kommunikation* deckt die Module ab, welche System-/Kommunikationsdienste den höheren Schichten bereitstellen, damit diese möglichst unabhängig von den technischen Komponenten entwickelt werden können. Darum wird allgemein bei LowLevel- und Middleware-Software integriert von Basissoftware gesprochen. Aus der Sicht eines Echtzeitbetriebssystems werden die abzuarbeitenden Aufgaben, in /[HOP05]/ als *Jobs* bezeichnet, in Pakete gebündelt und innerhalb von Tasks durchgeführt. *Kommunikationstasks* werden nach dem für die Koordination der Tasks erstellten Ablaufplan, dem *Task-Schedule*, implementiert. Diese speziellen Tasks übernehmen die Aufgaben des Beschreibens von Puffern des Netzwerkcontrollers und des Lesens der für die Netzwerkkommunikation allokierten Speicherplätze. Für typische Systemdienste, beispielsweise der Systemdiagnose, dem Netzwerkmanagement oder für höhere Transportprotokollaufgaben, eignet sich die Bezeichnung *Systemtasks*. Die eigentlichen technisch-funktionalen Aufgaben des Steuergeräts, etwa die Ausführung eines Regelungsblocks eines E/E-Systems, werden in Funktionsteilen als *Applikationstasks* abgearbeitet. Da ausschließlich die Funktionsblöcke der Applikationstasks ein Steuergerät charakterisieren, lassen sich Produktlinien anhand der spezifischen Applikations- oder Anwendungssoftware vom restlichen An-

teil an Plattformsoftware abgrenzen. Da die Bezeichnung Plattform speziell bei hoher Steuergerätestückzahl innerhalb verteilter Systeme appliziert wird, kommt der Begriff speziell bei den Mehrprozessorsystemen zum tragen.

Die Entwicklung von Einprozessorlösungen in Form eines einzelnen Steuergeräts bietet prozesstechnische Vorteile, da sich die Verteilung von Funktionalität, beispielsweise einem Steuer- oder Regelsystem, auf jeweils einen Zulieferer beschränkt und sich damit einfach in den Gesamtentwicklungsprozess integrieren lässt. In diesem Zusammenhang liegt die Anforderung für den Systemintegrator vor allem in der präzisen Beschreibung der Applikationsschnittstellen des Systems, während der Zulieferer selbst sein Wissen im Bereich der Algorithmik der zu implementierenden technischen Funktionen einsetzt. Dennoch spielt der Systementwurf der Funktionalität auf Anwendungsebene auch aus Systemintegratorsicht eine wichtige Rolle, da es in dessen Verantwortungsbereich liegt die komplexen Systeme präzise zu spezifizieren und zu verstehen. Dadurch ist er in der Lage die entwickelten Systemlösungen eigenständig zu testen, um diese fehlerfrei in seine E/E-Architektur zu integrieren.

Eine Schlüsselaufgabe für die Spezifikation des Einprozessorsystems liegt in der Beschreibung des funktionalen Verhaltens einer physikalischen Systemkomponente, eines Steuergeräts, durch entsprechende Spezifikationsformalismen. Dafür eignen sich drei verschiedene Sichten.

Statische Architektur: Aus Sicht der logischen Funktionen des Steuergeräts können Komponenten hierarchisch im Form von Strukturdiagrammen beschrieben werden. Die Komponenten werden dabei als Blöcke beschrieben, die über dedizierte Schnittstellen in Form von *Ports* kommunizieren können. Die Konnektoren zwischen den Ports basieren je nach Modellierungssprache aus uni- oder bidirektionalen Signalübertragungsstrecken, die je nach Ansatz bidirektionale Ports oder unidirektionale Ein-/Ausgabe-Ports zur Kommunikation mit einer Systemkomponente benötigen. Welches Kommunikationsprinzip den Konnektoren zugrunde liegt kann in der Notation berücksichtigt werden. /[AUT07a]/ unterscheidet beispielsweise zwischen Client/Server- und Sender/Receiver-Kommunikation.

Dynamisches Verhalten: Um das dynamische Verhalten der Funktionen zu beschreiben, eignen sich zwei etablierte Notationsformen, welche seit der letzten Dekade in der Automotivebranche zum Systementwurf herangezogen werden. Ereignisdiskrete Vorgänge orientieren sich in dem Zusammenhang an den Konzepten der endlichen Zustandsmaschinen, deren Transitionen abhängig von einem vorgegebenen Takt gegenüber einer Vorbedingung geprüft und gegebenenfalls ausgelöst werden. Zur präzisen Definition von Zustandsautomaten und zur Unterscheidung der Varianten Mealy und Moore wird auf /[Mea55], [Moo56]/ verwiesen. Eine weitverbreitete Darstellungsform für endliche Automaten ist von Harel entworfen worden. Seine *Statecharts* bieten Notationsformen zur hierarchischen Dekomposition von Automaten durch Unterautomaten und zur Komposition von parallelen ausführbaren Zustandsautomaten. Weiterhin lassen sich Zustandsübergänge mithilfe temporaler Logik ausdrücken. Eine Konzept dabei basiert auf der bedingten Erweiterung von Transitionen durch Konnektoren zur Abbildung situationsabhängiger Wechsel in einen von mehreren paarweise disjunkten Zielzuständen. Präziser werden diese und weitere syntaktische und semantische Eigenschaften von Statecharts in /[Har84], [Har87]/ vorgestellt.

Im gemischt diskret/kontinuierlichen Bereich haben sich signalflussorientierte Notationen etabliert, welche in Form von Blockdiagrammen dargestellt werden. Dabei können kontinuierliche, diskrete oder gemischte Systeme, wie sie in der Steuer- und Regelungstechnik auftauchen, so genannte *Hybride*, modelliert und simuliert werden. Zwischen dedizierten Ein- und Ausgangssignalen werden Blöcke modelliert, hinter denen verschachtelte Subdiagramme gelegt werden, mit denen sich diskrete und kontinuierliche Systeme abbilden lassen.

Interaktion: Während die statische Architektur die Systemstruktur und dessen Kommunikationsverbindungen beschreibt, stützt sich das dynamische Verhalten auf den direkten Einfluss von Eingaben auf Systemkomponenten und dessen Ausgaben. Zwischen diesen beiden Sichten fehlt eine dritte Sicht zur Beschreibung des Ablaufs der Interaktion zwischen Komponenten auf Basis von Signalflüssen. Mithilfe von Interaktionsszenarien lässt sich die zeitliche Abfolge von Nachrichten spezifizieren. Vergleichbare Notationsformen bestehen in den Sequenzdiagrammen der UML /[OMG06]/ oder den MSC (*Message Sequence Charts*) /[Int99a]/ aus dem Telekommunikationsbereich, welche in Kombination mit der Prozessbeschreibungssprache SDL (*Specification and Description Language*) /[Int99b]/ verwendet werden.

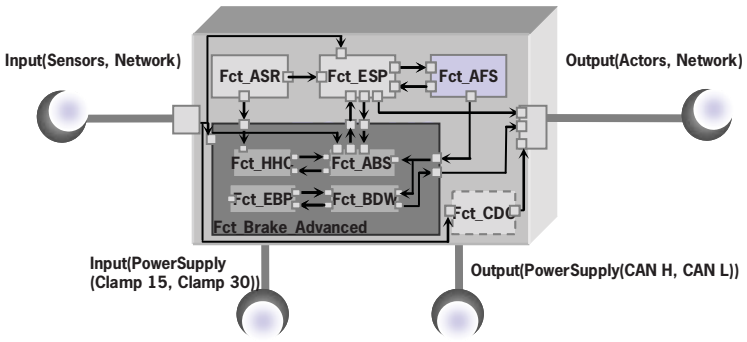


Abbildung 2.26.: Funktionale Architektursicht auf ein ESP-Steuergerät

Exemplarisch sollen die drei Sichten anhand eines vereinfachten Steuergeräts zur Stabilitätskontrolle ESP (*Electronic Stability Program*) (s. Abb. 2.26) dargestellt werden. Für diesen Zweck wird ein einfacher möglicher Querschnitt auf die statische Systemstruktur des ESP aus Sicht der funktionalen Architektur dargestellt. Dabei zeigt sich, dass verschiedene Funktionsmodule innerhalb dieser physischen Netzwerkkomponente verbaut sind und teilweise unterschiedliche Hierarchiestufen besitzen. So werden bremsenspezifische Funktionen wie die Berganfahrhilfe, das Antiblockiersystem, die Bremsscheibenwischer und die elektronische Parkhilfe in einem Modul *Advanced Brake* gebündelt. Am Eingang liegen die Daten, die über die Sensorik oder ein Netzwerk empfangen werden, in Form von Datenströmen an, welche in der Abbildung allgemein als Eingangskanal abstrahiert werden. Analog werden am Ausgang Datenströme auf die Aktoren und das Netzwerk gelegt. Separat werden die Ein- und Ausgänge zur Leistungsversorgung des Steuergeräts bzw. des Transceivers als Netzwerkschnittstelle dargestellt.

Eine verfeinerte Darstellung des Eingangskanals auf Signalebene (s. Abb. 2.27) veranschaulicht, dass bereits die wenigen Signale, welche die reine Stabilisierungsfunktion bedaten, zu einem wesentlich komplexeren Systemstrukturdiagramm führen. Daraus lässt

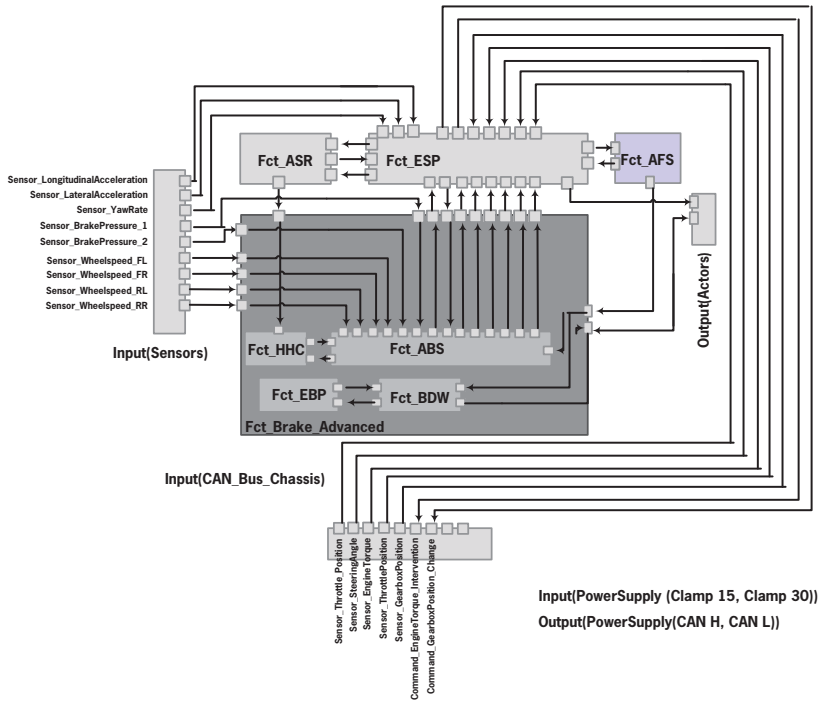


Abbildung 2.27.: Verfeinertes Systemstrukturdiagramm bezogen auf das ESP-Funktionsmodul im ESP-Steuergerät

sich ableiten, welche Bedeutung der *Hierarchiebildung* und der *Kapselung* von Funktionsblöcken in einem Modell zufließen. Ohne Abstraktionsstufen lässt sich die Komplexität eines ESP-Steuergeräts nicht mehr sauber abbilden.

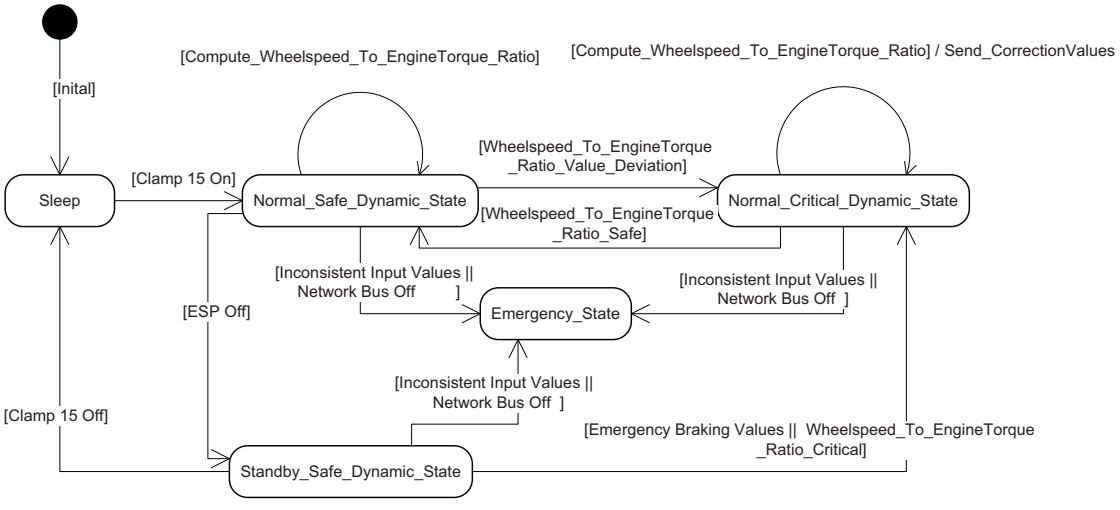


Abbildung 2.28.: Zustandsmaschine zur Darstellung der Systemmodi des ESP

Das dynamische Verhalten lässt sich in einen ereignisdiskreten Anteil gliedern, der die Systemzustände (*Modi*) des ESP abdeckt (s. Abb. 2.28). Für den gemischt diskret/-kontinuierlichen Anteil kommt der klassische ESP-Regler zum Einsatz, der heute mitt-

lerweile in fast allen Fahrzeugen angeboten wird. Da das vollständige System umfangreiche Strukturen aufweist und nicht öffentlich vorliegt, zeigt das Datenflussdiagramm in Abb. 2.29 lediglich den konzeptionellen Grundaufbau eines ESP-Reglers.

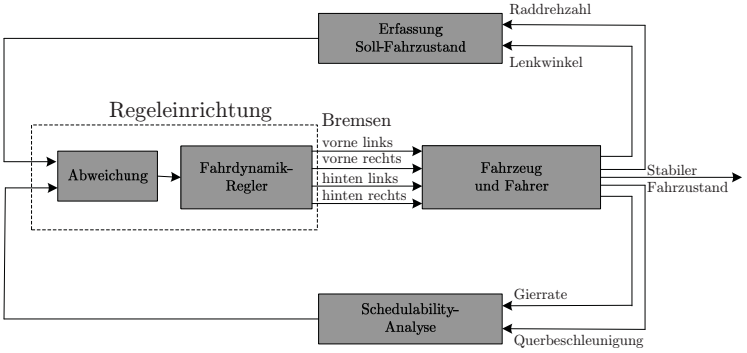


Abbildung 2.29.: Datenflussdiagramm des Schemas einer Fahrdynamikregelung als Teil eines ESP-Regelsystems /[Trö05]/

Die Signalflüsse sind unidirektional und innerhalb der Blöcke liegen Signalverknüpfungen vor in Form von Differentialgleichungssystemen für den kontinuierlichen Teil oder als Zustandsmaschine für den diskreten Teil.

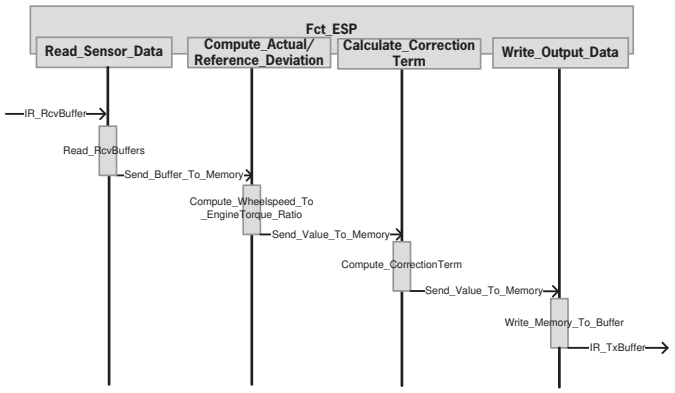


Abbildung 2.30.: Zeitlich geordnetes Interaktionsszenario innerhalb der ESP-Regelfunktion

Als Beispiel für die Interaktionssicht wird ein vereinfachter Kommunikationsablauf in Abb. 2.30 dargestellt. Ein Sensorwert wird per Unterbrechungsroutine (*Interrupt Service Routine*) an eine Vergleichsfunktion weitergereicht, die den aktuellen mit dem kalkulierten Soll-Wert vergleicht und den Differenzbetrag weitergibt. Dieser geht in die Berechnung des Korrekturwerts ein, der in einen Sendepuffer geschrieben und an einen Aktor übermittelt wird.

Der wesentliche Unterschied zwischen einem Einprozessor- und einem Multiprozessorsystem liegt in der Nebenläufigkeit von Prozessen und den getrennten Speicherbereichen in einem physikalisch verteilten System. Prinzipiell kann durch Softwareschichten ein System so gekapselt werden, dass sich ein physikalisch verteiltes System gegenüber einem lokalen System aus Sicht der Anwendungsfunktionen identisch verhält.

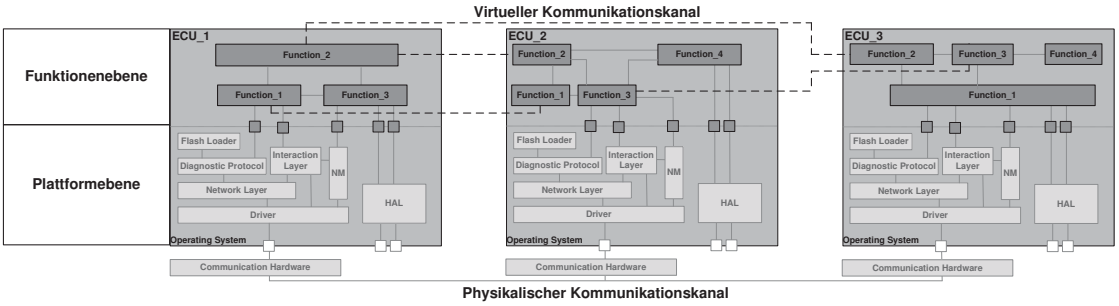


Abbildung 2.31.: Physikalisch verteiltes Mehrprozessorsystem auf Basis des OSEK-Standards

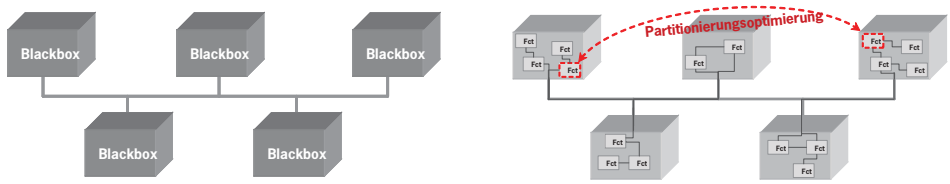
2.3.3. Systemspezifikationsebenen

Analog zu Einprozessorsystemen gilt auch bei Plattformkonzepten die Annahme, dass die Funktionen über einen virtuellen Kommunikationskanal unabhängig von ihrer technischen Komponentenzuordnung kommunizieren können. Folgende Aspekte eines verteilten plattformorientierten Multiprozessorsystems gilt es zu beachten:

Funktionsbeschreibung: Als Funktionsbeschreibung gelten die gleichen Konzepte, die bereits für Einprozessorsysteme dargestellt worden sind. Zusätzlich gilt es die Funktionen konsequent uniform zu kapseln, damit eine leichte Verschiebung innerhalb des verteilten Systems ermöglicht wird. Bei hinreichend guter funktionaler Dekomposition können *Funktionsbibliotheken* generiert werden, welche abgeschlossene Funktionalitäten wie zum Beispiel einen Fensterheber implementieren. Dieses Konzept ist als eines der Ziele im CARTRONIC-Ansatz verfolgt worden /[FGZA03]/. Eine Standardisierung eignet sich durchaus, da viele einfache Funktionen im Fahrzeug mittlerweile herstellerübergreifend nahezu identisch umgesetzt werden. Allgemein wird bei der verteilten Funktionssystembetrachtung von einer *logischen Systemarchitektur* /[SZ03]/ gesprochen.

Partitionierung: Die Partitionierung beschreibt die Anordnung von logischen Funktionen auf technischen Komponenten. Diese Zusammenführung zwischen logischer und technischer Systemarchitektur wird auch allgemein als (*Mapping*) bezeichnet. Die Verknüpfungsbeziehungen werden dabei in unterschiedlicher Art und Weise spezifiziert. Die einfachste Variante stellt die Zusammenführung in tabellarischer Form dar. Aber auch schablonenartige Ansätze eignen sich wie zum Beispiel die *Description Templates* zur System- und Steuergerätebeschreibung in AUTOSAR /[AUT07a]/ oder die *System Blueprints* von Marchetto /[Mar97]/. In /[FM98]/ setzen Faboul und Martin im Bereich Mapping auf ein so genanntes *Allocation Model*, welches die Zuordnung einer Partitionierung beschreibt. In /[SZ03]/ wird die Partitionierung im Automotive Segment beschrieben. Eine formale Beschreibung dazu wird in Abs. 4.2 aufgeführt.

Ressourcenbeschreibung: Zur Beschreibung der Ressourcen eignet sich die *technische Systemarchitektur*. Dazu gehören die konkret existierenden Komponenten, die Steuergeräte. Die wesentlichen Leistungsmerkmale eines Steuergeräts liegen in der zentralen Recheneinheit, gegeben durch die Performanz im Datendurchsatz und Verarbeitungszeit in einem μ-Prozessor, einem μ-Controller oder einem Logikbau-



(a) Heutige Komponentensicht und resultierende Einschränkungen für die Partitionierung

(b) Anzustrebende Systemsicht mit optimierbaren Funktionsmodellen

Abbildung 2.32.: Systempartitionierung auf Basis von abhängigen Funktionsmodellen

stein auf FPGA-Basis. Zusätzlich wird die Speicherleistung in Form von flüchtigen RAM- und beständigen ROM- oder Flash-Speichern beurteilt. Auch die technische Schnittstelle zur Kommunikation in einem Netzwerk, bestehend aus einem Interface, CCs (*Communication Controllers*), Konnektoren, in Form von Transceivern und analogen/digitalen Ein-/Ausgabeschnittstellen I/Os (*Input/Output*) wird dem Ressourcenumfang eines Steuergeräts zugeordnet.

Netzwerkbeschreibung: Die Konfiguration eines Netzwerks muss eindeutig zur fehlerfreien Kommunikation beschrieben sein. Dazu gehören die Datenkommunikation und die Sende-/Empfangsbeziehungen in Form eines Sendeschedules oder einer Kommunikationsmatrix der beteiligten Komponenten, eine Netzwerk- und Knotenkonfiguration des Kommunikationsprotokolls und die Spezifikation des physikalischen Kommunikationsmediums. In einer Kommunikationsmatrix wird der Signalaustausch zwischen den Knoten festgelegt und die Botschaften des Kommunikationsprotokolls spezifiziert. Dabei sind unterschiedliche Zykluszeiten für die Signalübertragung zu berücksichtigen, womit sich das Paketieren von Botschaften auf eine NP-harte Aufgabenstellung zurückführen lässt, die an das *Bin Packing*-Problem /[JDU+74]/ erinnert. Die Entwicklung von Algorithmen und Heuristiken zur möglichst niedrigen Belastung des Kommunikationsmediums stellt eine zentrale Herausforderung in der Fahrzeugvernetzung /[SN03], [SNA00]/ dar. Die formale Spezifikation der Kommunikation erfolgt wie in Abb. 2.33 exemplarisch aufgezeigt.

Das Netzwerk wird durch folgende Kommunikationsklassen bei der Signalübertragung belastet:

Zyklische Kommunikation: Diese Signale kommen im Allgemeinen von Funktionen, die den Zustand einer Komponente regelmäßig aktualisieren müssen. In /[Kop97]/ wird auch von Zustandssignalisierung gesprochen. Entscheidend für die Bandbreitenbelastung wirkt die Zyklus- und die Signallänge.

Spontane Kommunikation: Funktionen mit ereignisgesteuertem Charakter, etwa der Wert einer Schalterposition im Cockpit, übertragen ereignisdiskret die Schalterposition im Netzwerk, um eine Belastung durch unnötig zyklische Übertragung zu vermeiden. Für die Ermittlung der Busbelastung muss die relative Häufigkeit in Verbindung mit der Signallänge berechnet werden.

Serviceorientierte Kommunikation: Zu den Servicen zählen vorrangig höherwertige Datentransportprotokolle, die für die segmentierte Übertragung größerer Datenpakete eingesetzt werden. Wichtig für die Busbelastung ist die Häufig-

Änderungen | S - zyklisches Signal | S - zyklisches wenn aktiv | S - ereignisgesteuertes Signal | M - Motorola Format

				Antrieb									Diagnose		Infotainment					Komfort												
ID-Name	Byte, Bit Position	Signalname	Beschreibung	Airbag	Allrad	Stabilisator	ESP	Gateway_SG	Getriebe	Lenkwinkel	Motronic	Waehlhebel	Systemtester	Gateway_SG	Gateway_SG	Kombi	Parkassistent	Navigation	Radio	Benutzerzugang	Gateway_SG	Heck_SG	Climatronic	Sitzsteuergerät	Standheizung	Targa_SG	TSG_Beifahrer	TSG_Fahrer	Wertebereich	Normierung	Auflösung	Bemerkungen/ Zuordnungstabelle
ESP																																
BRAKE_1 ID: 0x3A1 15 ms Zyklisch DLC: 8	1.0	B_bract	Bremsmomenten-Eingriff				S	E	E		E				S																1.0	1=Bremsmomenten-Eingriff aktiv
	1.1	B_abs	ABS-Bremsung		E		S	E	E		E				S																1.0	1=ABS-Bremsung aktiv
	1.2	B_trc	Traktionskontrolle-Anforderung				S	E	E		E				S																1.0	1=Anforderung TRC
	1.3	B_esp	ESP-Eingriff		E		S	E	E		E				S																1.0	1=ESP-Eingriff aktiv
	…	…	…																										…	…	…	…
	8.0...8.7	loacc	Längsbeschleunigung		E		S	E	E		E				S	E		E											-1.27 .. 1.27 g	0.0-254.0	0.01 g	positiv = Beschleunigung negativ = Bremsung
BRAKE_2 ID: 0x3B1 20 ms Zyklisch DLC: 8	1.1…1.7	laacc	Querbeschleunigung		E		S	E	E	E	E				S	E		E											-1.27 .. 1.27 g	0.0-254.0	0.01 g	positiv = Linkskurve negativ = Rechtskurve
…	…	…	…																										…	…	…	…

Abbildung 2.33.: Klassische Darstellung der Kommunikationsbeziehungen eines Antriebs-/Fahrwerks-CAN-Bus in Matrixform

keit, in der diese Dienste abgerufen werden sowie der Zyklusabstand und die Länge der Botschaften innerhalb eines Services. Beispiele für diese Service wären Datenprotokolle für Displayinformationen oder der Diagnosebetrieb.

Eine besondere Aufmerksamkeit erfordert der Einsatz von verteilten Funktionen, falls Funktionsbefehle direkt über den Bus verschickt werden. Dabei lassen sich Funktionen direkt aufrufen (z.B. ABS-Bremsung). Eine zweite Variante bezieht sich auf den Fahrzustand, indem Werte über den aktuellen Fahrzeugzustand, beispielsweise *Passagiergewicht/Dämpferlast*, über das Netzwerk kommuniziert werden.

Die Netzwerk- und Knotenkonfiguration hängt stark von der jeweiligen Vernetzungstechnologie ab. Im Allgemeinen werden dort Einstellungen bzgl. der Netzwerkübertragungsrate und der Busphysik getroffen.

Die Netzwerktopologie beschreibt den konkreten Aufbau des Feldbussystems und seines Kabelbaums (*Wiring Harness*). Kabelbäume in heutigen Fahrzeugen haben eine hohe Komplexität, da starke Anforderungen an Robustheit und Signalqualität herrschen. Signalreflexionen müssen niedrig gehalten werden, indem eine zweckmäßige Topologie implementiert wird. Dabei muss auch das elektromagnetische Ein-/Abstrahlverhalten beachtet werden. Es gibt viele Möglichkeiten die Topologie zu gestalten, indem Sterne, Busse und Ringe konzipiert werden, die teilweise auch kaskadiert oder in hybrider Form verbaut werden. Die Signalqualität wird durch Drosseln, Kondensatoren, Schaltungen und Widerstände optimiert.

2.3.4. Zeitverhalten

Die Analyse des Systemverhaltens bezogen auf das uniforme Fortschreiten der physikalischen Zeit, trägt entscheidend zur Systemspezifikation eines verteilten eingebetteten Echtzeitsystems bei. Um diese Einflüsse im Systementwurf zu verstehen, werden die grundsätzlichen Systemqualitätskriterien im temporalen Bereich (*Timing*) vorgestellt.

Leistung (*Performance*) definiert die Verarbeitungsgeschwindigkeit von Eingabedaten bezogen auf festgelegte Zeiteinheiten. Je nach Betrachtung kann sich dies abstrakt auf eine logische Funktion oder konkret auf einen Prozessor oder ein Busprotokoll beziehen.

Latenz (*Latency*) bezeichnet das Zeitintervall zwischen dem Auftreten eines Ereignisses und dem Beginn einer dem Ereignis zuordbaren Systemreaktion. Aus technischer Sicht lassen sich Latenzen folglich als Summe an diversen Teilverzögerungen im System auffassen.

Verzögerungen (*Delays*) werden technisch aufgrund unterschiedlicher Einflüsse provoziert. Die Qualität eines Algorithmus und die maximale Rechengeschwindigkeit eines implementierten Mikrocontrollers werden für einen Netzwerkknoten erfasst. Der Vorgang der Datenserialisierung zur Nachrichtenübertragung und die Ausbreitungsverzögerung von Signalen auf einem Kommunikationsmedium werden nicht den physikalischen Einflüssen, sondern der Netzwerkkommunikation beim Nachrichtenaustausch zugerechnet.

Schwankungen (*Jitter*) bezeichnen die maximale Abweichung der nominalen Systemverzögerung bei der End-zu-End-Kommunikation, welche aufgrund der Summierung von variierenden Teilverzögerungen auf der Kommunikationsstrecke entsteht. Die Varianz der Teilverzögerungen bezieht sich auf dynamische Effekte, verursacht durch Systemauslastung oder externe Einflüsse.

Zeitmodell der elektrisch physikalischen Bitübertragungsschicht

Aus Sicht eines elektrischen Übertragungsmediums stellt der Einsatz eines hochfrequenten Bussystems bei 10Mbit/s eine Herausforderung im Sinne einer stabilen elektrischen Signalübertragung dar. Speziell der Kodier-/Dekodierprozess an den Kommunikationscontrollern (CC) sowie die Signalwandlung an den Transceiver-Schnittstellen erfordert ein präzises Verständnis während des Systemdesigns.

Im Rahmen der TBTF-Analyse (T*iming Budget Task Force*) des FlexRay-Konsortiums sind intensive Analysen durchgeführt worden, um die Qualität der End-To-End-Kommunikation im Netzwerk zwischen beliebigen Knoten in unterschiedlichen Topologien abzusichern (s. Abb. 2.34). Aktuell gibt es bei vereinzelten Topologien kritische Grenzen, über die hinaus eine fehlerfreie Bitübertragung nicht gewährleistet werden kann (s. Abs. 5.4).

Die wesentlichen Einflussfaktoren leiten sich dabei aus folgenden Effekten ab:

- Asymmetrische Verzögerung des Bussignals
- Ausbreitungsverzögerungen bei der Signalübertragung
- Flankenpegel
- Jittereffekte aufgrund von Quarzungenauigkeiten

Beispiel:

1. Die Länge eines übertragenen Bits beträgt 100ns bei 10Mbit/s Baudrate.

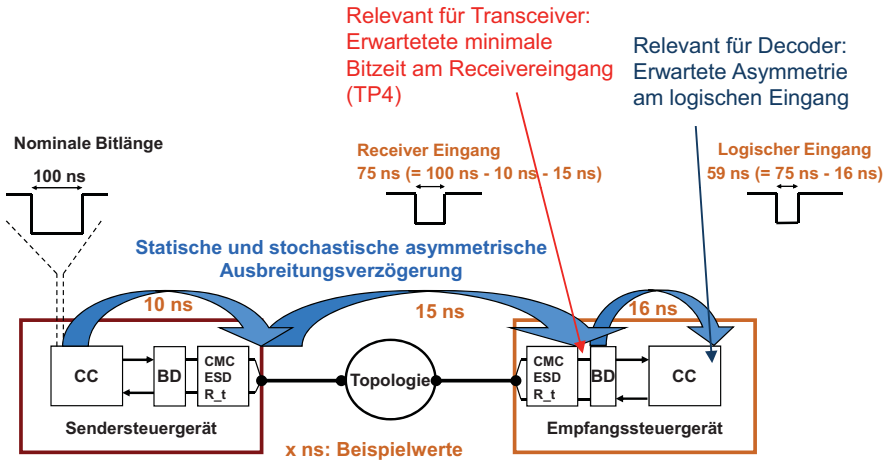


Abbildung 2.34.: Darstellung der zu analysierenden Aspekte bei der Bitübertragung im elektrischen Physical Layer (vgl. /[Fle06a] (adaptiert)/)

2. Da der Decoder am 5. Sample bei 8 Samplings pro Bitlänge den Bitwert abtastet, kann eine maximale Bitverkürzung um $3 * 12,5ns = 36,5ns$ toleriert werden.

3. Durch einen Beitrag des Bustreibers von 5ns und der ECU von 1ns abzüglich eines positiven Effekts bei der Umrechnung des Frame-Clock-Jitters $(2 * (-0,5ns))$ auf den Bit-Clockjitter $(2 * 0,1ns)$ von $0,9ns$ ergibt sich eine Toleranz von $36,5ns - 5ns - 1ns + 0,9ns = 31,4ns$ bei einem 80ns-FlexRay-Busreceiver.

In Tab. 2.2 wird eine reduzierte Übersicht zu den relevanten Parametern bei der Zeitmodellierung der elektrischen Bitübertragungsschicht bezogen auf die Bitlänge in einer Topologie dargestellt.

Zeitmodell der Protokollschicht

Bei zeitgesteuerter Kommunikation wird auch von „*synchroner Datenübertragung*“ innerhalb des verteilten Systems gesprochen. Informationen, gegeben als Signale innerhalb von Botschaften, werden in einem vordefinierten Takt ausgetauscht. Dadurch lassen sich nur endliche Berechnungsvorgänge (Operationen) innerhalb eines Taktes durchführen. Um das Verhalten eines Systems über Zeit zu planen, muss eine lokale Sequenzialisierung der Prozesse (*Tasks*) auf Knotenebene durchgeführt und das Ergebnis auf Eignung für die globale Netzwerkebene verifiziert werden (*Schedulability*-Analyse). Neben der Zuverlässigkeit gilt die Performanz als weiteres entscheidendes Qualitätsmerkmal in einem System als Teil einer E/E-Architektur eines Fahrzeugs. Für verteilte Multiprozessorsysteme teilt sich diese Eigenschaft zwischen der Leistungsfähigkeit zweier Komponenten und dessen zwischengeschalteter Netzwerkkommunikation.

Für die Betrachtung der maximalen Verzögerung t_{ges} müssen zu den reinen Netzwerkübertragungszeiten mehrere Verzögerungen an den verschiedenen Schnittstellen addiert werden:

$t_1 = t_{Sensor \rightarrow HostA}$: Beim Eintreffen eines Sensorwerts wird dieser in einer Systemtask

Verzögerungseffekte		statisch (ns)	stochastisch (ns)
Kommunikationscontroller			
I/O-Puffer		1	0,5
Leiterplatten		0,5	0
EMV	Transient CW/AM	0 0	0 0
Sicherheitsabstand		individuell	individuell
Bittakt (*Clock Tree*)	PLL keine PLL	0,5 0	0,16 0,16
Quarztoleranz		1,5	0
Bustreiber			
Chip		4	0
EMV		0	0
Flankensteilheiten/Schwellenwerte		spezifisch	spezifisch
Sicherheitsabstand		individuell	individuell
Netzwerkverbindungen			
Verkabelung	p2p Daisy Chain Maximum Net	0 0 0	4 8 8
Aktiver Stern			
Bustreiber		5	0
EMV		0	0
Sicherheitsabstand		individuell	individuell
Chiptyp	Monolithisch Diskret	3 5	0 0
ECU			
HW-Design	Layout Drossel ESD-Schutz	0,25	0,25
EMV	Transient CW/AM	0 0	0 0
Sicherheitsabstand		individuell	individuell

Tabelle 2.2.: Übersicht der elektrischen Physical Layer Timing-Parameter der TBTF-Gruppe

des Host-Betriebssystems gelesen und an eine Adresse im Speicher des Hosts A geschrieben.

$t_2 = t_{HostA \rightarrow CC-Tx}$: Innerhalb einer Kommunikationstask wird der aktualisierte Sensorwert an einen Sendepuffer des Kommunikationscontrollers zum Senden über das Netzwerk weitergereicht.

$t_3 = t_{CC-Tx \rightarrow CC-Rx}$: Der Sendekommunikationscontroller verschickt je nach Medienzugriffsverfahren den Sensorwert mit einer Verzögerung über das Netzwerk.

$t_4 = t_{CC-Tx \rightarrow CC-Rx}$: Beim Eintreffen des Sensorwerts am empfangenden Kommunikationscontroller entsteht eine zeitliche Verzögerung durch das Ausführen einer Kommunikationstask zum Auslesen des neuen Sensorwerts am Empfangspuffer des Kommunikationscontrollers und dem Schreiben an eine reservierte Adresse im Speicher des Hosts B.

$t_5 = t_{HostB}$: Der neue Sensorwert wird an eine Applikationstask und damit an den Algorithmus eines Steuer-/Regelsystems übergeben, der seinerseits einen neuen Wert für den reagierenden Aktor berechnet und diesen an eine reservierte Adresse im Speicher schreibt.

$t_6 = t_{HostB \rightarrow Aktor}$: Host B aktualisiert in einer Systemtask den Aktorwert anhand der neuen Daten an der definierten Adresse im Speicher des zugeordneten Aktors.

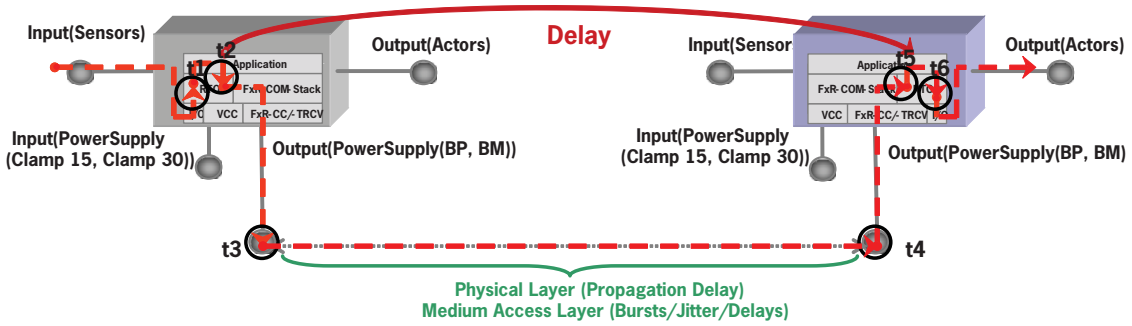
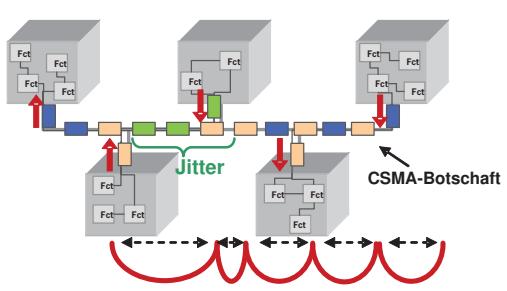
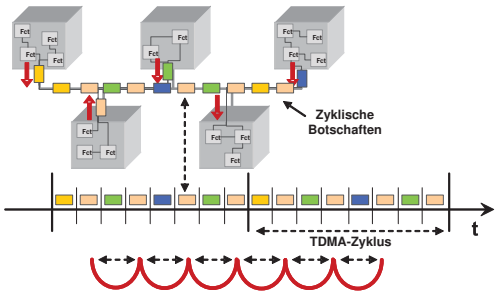


Abbildung 2.35.: Verzögerungspunkte für Laufzeit einer Sensor-Aktor-Kommunikationsstrecke

Bei Übertragung einer Information zwischen den Komponenten Host A und Host B entsteht eine netzwerkbedingte Sendelatenz. Die Gesamtverzögerung wird dabei physikalisch bedingt durch eine Verzögerung bei der Signalausbreitung auf dem Übertragungsmedium (*Propagation Delay*). Zusätzlich wirken sich Netzwerkjitter (*Network Delay Jitter*) aus, die vom Medienzugriffsverfahren des Netzwerkprotokolls abhängen (s. Abb. 2.36).



(a) Variable Jitter innerhalb eines CSMA-basierten Netzwerks



(b) Konstante Jitter innerhalb eines TDMA-basierten Netzwerks

Abbildung 2.36.: Unterschiedliches Zeitverhalten in ereignis- und zeitgesteuerten Netzwerkarchitekturen

Der ereignisgesteuerte CAN-Bus unterliegt im Fahrzeug diesbezüglich stärkeren Einschränkungen im Vergleich mit dem zeitgesteuerten FlexRay-Cluster. Durch das CSMA-Verfahren hängt der Buszugriff bei CAN stark von den im Netzwerk aktiven Komponenten ab. Durch eine Nachrichtenpriorisierung innerhalb der Arbitrierungsvorgänge auf dem Bus kann es zu unkalkulierbaren Verzögerungen kommen während das TDMA-Konzept des FlexRay-Systems konstante *Jitter* garantiert (vgl. Abb. 2.36(a) mit Abb. 2.36(b)).

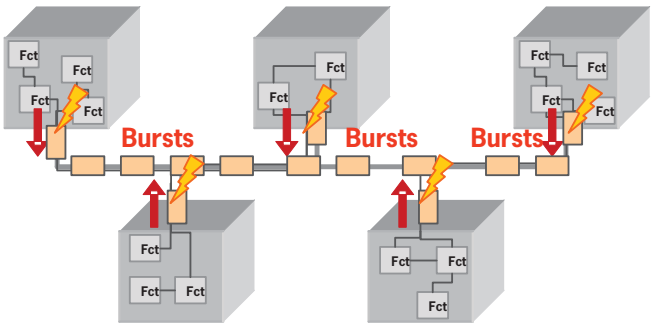


Abbildung 2.37.: Aufgeschaukelte Busbelastung durch Bursts beim parallelen Buszugriff

Speziell aufgeschaukelte Situationen, in denen simultan mehrere Buszugriffsanfragen von verschiedenen Knoten auftreten (*Bursts*), können eine Information innerhalb eines CAN-Busses massiv verzögern (s. Abb. 2.37). Diese Situationen entstehen oftmals beim Netzwerkhochlauf. Aufgrund des unzuverlässigen Verhaltens bei hohen Buslasten nutzen Fahrzeughersteller die Bandbreite eines CAN-Feldbussystem sicherheitsbedingt nur zu maximal 70% aus.

Zusammenfassend kann die Gesamtübertragungsdauer einer Sensor/Aktor-Systemreaktion wie folgt definiert werden:

$$t_{ges} = \sum_{i=1}^{6} t_i + t_{propagationdelay} + t_{jitter}$$

Eine formale Analyse des Zeitverhaltens erfordert aufgrund der Heterogenität des Systems einigen Aufwand. In /[RZE+02], [RE02]/ wird ein Ansatz vorgestellt, der das gesamte Zeitverhalten zerlegt, indem Architekturkomponenten im Gesamtsystem identifiziert werden, für die bereits Analysemethoden zum Zeitverhalten bekannt sind. Eine Komposition der Zeitverhalten der verschiedenen Komponenten ergibt ein komplexes, vollständiges Zeitverhalten des Gesamtsystems auf Systemebene. Das Konzept basiert grundlegend auf der Definition ausgehender Ereignisströme (*Event Streams*) von Sendekomponenten, die assoziierte Ereignismodelle (*Event Models*) von Empfängerkomponenten treffen (*matchen*). Die Ereignismodelle müssen dabei die empfangenen Ereignisströme verarbeiten können. Dadurch kann formal der Zusammenhang der Komposition von Architekturkomponenten und dem entstehenden zeitlichen Gesamtssystemverhalten geschlossen werden. Einschränkungen bietet der Ansatz, durch seine Standardereignismodelle. Diese stützen sich auf

- rein periodische,
- periodische mit *Jitter*,
- periodische mit *Bursts* und *Jitter*,
- rein sporadische

Übertragung. Die erzeugten Ereignisströme können das reale Verhalten lediglich approximieren. Um Fehler in einem analysierten System auszuschließen, lassen sich hiermit konservative Schranken durch eine Zeitanalyse definieren (*Worst Case Bounds*). Die Methodik wurde in dem Werkzeug *SymTA/S* implementiert /[HHJ+]/.

/[Kün06]/ setzt auf einem hybriden Ansatz aus simulativen und analytischen Verfahren auf, um die Performanz eines eingebetteten Systems zu bestimmen. Dabei liegt der Fokus darauf eine Mehrzieloptimierung durchzuführen, die der Entwickler mithilfe von Performanzindikatoren beeinflussen kann. Dies wurde mithilfe eines evolutionären Algorithmus entwickelt und durch ein Entwicklungsframework vervollständigt, um bei der Entwurfsraumexploration eine minimale Anzahl an Komponenten zu identifizieren, deren Software im Systementwurf implementiert werden müssen.

Während das TDMA-Verfahren in einer zeitgesteuerten Architektur (s. Abs. 3) konstant deterministische Verzögerungen zwischen Pufferaktualisierung und Netzwerkübertragung garantiert, treten in einer ereignisgesteuerten Kommunikation variable Verzögerungen auf. Bei eingeschränkt echtzeitfähigen Feldbussystemen wie dem CAN-Bus spielt die Netzwerkauslastung diesbezüglich eine wichtige Rolle, da Performanz und Zuverlässigkeit mit ihr gekoppelt sind.

Wichtig für die Berechnung der Netzwerkauslastung sind auch die Aktivitätszeitpunkte der Netzwerkteilnehmer, die durch die Klemmensteuerung oder externe Ereignisse bestimmt werden. Das selektive Abschalten von kommunikationsintensiven Knoten in bestimmten Situationen führt zu einem bandbreitenorientierten Teilnetzbetrieb, obwohl der Extremfall (*Worst Case*) einer maximalen Buslast in einem komplett aktiven Netzwerk berücksichtigt bleiben muss. Aktuell wird aufgrund des komplexeren Systemverhaltens ein Teilnetzbetrieb ausgeschlossen und lediglich das Weck-/Schlafverhalten einzelner Bussysteme im Sinne eines Netzwerk-/Energiemanagementkonzepts spezifiziert und angewendet.

Zusammenfassend lassen sich allgemein vier verschiedene Bereiche ableiten, die das Zeitverhalten des Gesamtsystems maßgeblich beeinflussen:

Knoten: Das dynamische Verhalten einer Netzwerkkomponente wird maßgeblich durch die Anzahl und Ausführungszeiten der ihr zugeordneten Prozesse und deren Ressourcenbedarf definiert. Dabei werden quantitative Aussagen über die Ausführungszeiten benötigt, welche zeitbezogen für den jeweils schlechtesten Fall geschätzt werden müssen (WCETs). WCET-Schätzungen stellen ein altbekanntes Problem in der Systementwicklung dar und sind bereits intensiv untersucht worden (s. Abs. 3.1.5.1).

Netzwerk: Das globale E/E-Architekturdesign wird durch die Ausprägung des Bordnetzes beeinflusst, beispielsweise durch die Einführung von Signal-Gateways und die Verwendung heterogener Bustechnologien mit unterschiedlichen Konfigurationen. Wesentliche Elemente in dieser Betrachtung referenzieren auf die Effekte durch Burst Conditions, Delays und Jittern.

Erweiterung: Da eine Systemerweiterung in der Regel zu Effekten mit veränderten Latenzen auf dem Bus (ereignisgesteuerte Kommunikation) und auf dem Knoten (Schedule) führt, gilt es die Anforderungen einer Kommunikationserweiterung im Lebenslauf der p2p-Kommunikationsstrecke genau zu beschreiben.

Funktionale Abhängigkeit: Bemerkenswert bei der Betrachtung der Zeitmodellierung ist der Aspekt, dass bei einer Adaption im Zeitverhalten, beispielsweise durch die Modifikation der angenommenen WCET-Zeiten einer Task oder der Übertragungszykluslängen im Netzwerk, von der funktionalen Abhängigkeit zweier Steuergeräte x und y in der Regel abstrahiert wird. Die Anzahl der beteiligten Komponenten in der Kommuni-

kationsstrecke verkörpert dabei eine wichtige Kennzahl für die Erfassung der Komplexitätstiefe einer E/E-Architektur.

2.3.5. Zuverlässigkeit

Für Systeme, die in einem sicherheitskritischen Kontext zur Umwelt stehen, gilt es Anforderungen hinsichtlich ihrer Verlässlichkeit zu erfüllen. Verlässlichkeit basiert auf der Vertrauenswürdigkeit eines Computersystems für den Anwender, der sich dessen Dienste bedient und auf dessen Funktionalität *verlässt*. Der Begriff der *Verlässlichkeit* ist dabei speziell bei der Entwicklung von Rechensystemen mit hohen *Zuverlässigkeits-* und *Sicherheitsanforderungen* geprägt worden. Nach /[Lap91]/ korreliert Verlässlichkeit mit der englischen Bezeichnung *Dependability*, die als Überbegriff die Themen *Verfügbarkeit* (*Availability*), *Wartbarkeit* (*Maintainability*) und *Zuverlässigkeit* (*Reliability*) umfasst. Im Bereich der Echtzeit-Computersysteme komprimiert sich die Terminologie auf den Begriff der Zuverlässigkeit, der daher oftmals in der Praxis auch mit dem Begriff der Verlässlichkeit synonym verwendet wird.

Begriffsdefinitionen

Nach /[DIN90]/ ist Zuverlässigkeit definiert als „*Beschaffenheit einer Einheit bezüglich ihrer Eignung, während oder nach vorgegebenen Zeitspannen bei vorgegebenen Anwendungsbedingungen die Zuverlässigkeitsforderung zu erfüllen*“. Im technischen Bereich lässt sich diese Aussage konkretisieren als die durchschnittliche Zeit zwischen den Ausfällen in einem System MTBF (*mean-time-between-failure*) mit zufällig auftretendem Ausfallverhalten.

Im Bereich der E/E-Architekturentwicklung konzentriert sich der Zuverlässigkeitsbegriff auch auf die Sichtweise für strukturierte Prozesse, einem durchdachten Systemdesign sowie einer engen Verzahnung aller beteiligten Entwickler /[SW07]/. Die Motivation des Fahrzeugherstellers in Fragen der Zuverlässigkeit des Gesamtsystems Fahrzeug mit hoher Priorität zu agieren, resultiert aus den gravierenden Auswirkungen im Falle von Fahrzeugdefekten. Besonders der komplette Verlust der Funktionstüchtigkeit eines Fahrzeugs, bei oder zwischen den Fahrbetriebszeiten beim Kunden, wirkt sich fatal auf das Qualitätsempfinden gegenüber einer Fahrzeugmarke aus und kann bei sicherheitskritischen Systemen auch zu juristischen Folgen führen. Eine Möglichkeit die Zuverlässigkeit zu verbessern besteht in der Erhöhung der *Ausfallsicherheit*, etwa durch die Implementierung von Systemredundanz. Dabei ist jedoch zu beachten, dass ein hinsichtlich der Ausfallsicherheit lokal optimiertes Steuergerät (z.B. Airbag) global betrachtet nicht zwangsläufig zu einem robusten Gesamtsystem führen muss.

Methoden zur Erhöhung der Systemzuverlässigkeit

Aus Sicht der angewandten Entwicklungsmethoden und der zugrunde gelegten Prozesse gibt es unterschiedliche Möglichkeiten die Zuverlässigkeit der E/E-Architektur zu erhöhen (in Anlehnung an /[SW07]/):

Design: Modulare Entwicklung mit Einführung von Funktionsredundanzen, Rückfallebenen, Fehlertoleranzen, Selbstüberwachungen, Fehlerbehandlungsstrategien, erweiterte Diagnosekonzepte,

Entstehungsprozess: Einhaltung von Designrichtlinien, Anwendung modellbasierter

Funktionsentwicklung und Simulation, Einsatz automatisierter Codegenerierung, Umstellung auf einen funktionsorientierten Entwicklungsprozess mit durchgängigen Werkzeugketten,

Komponenten/Kommunikation: Entwicklung von robusten Kommunikationsstrategien, Systemverständnis für Zeitverhalten und Steuergerätezustände bei verteilten Funktionen,

Standardisierung: Aufbau und Integration modularer Softwarekomponenten (Vorteil der Wiederverwendung und Austauschbarkeit), Einsatz standardisierter Basissoftwarearchitekturen und Laufzeitumgebungen,

Test: Etablierung geeigneter Organisationen mit ausgereiften Prozessen, Entwicklung einer fundierten Teststrategie, Einsatz von neuen Methoden und Technologien,

Entwicklungsprozess: Vertrauen auf standardisierte Verfahren und Prozesse, Berücksichtigung von Lernprozessen (*Lessons Learned*), Förderung der internen und herstellerübergreifenden Zusammenarbeit (Kollaborations-, Prozess-, Projekt- und Qualitätsmanagement).

Robustheit

Allgemein wird der Robustheitsbegriff im Zusammenhang mit der Fahrzeugentwicklung unscharf benutzt. Prinzipiell versteht sich darunter die Anfälligkeit von Systemkomponenten in Fehlerfällen im Serieneinsatz auszufallen. Besonders die Situationen wenn der Fahrzeugbenutzer von den Ausfällen bewusst betroffen wird, etwa durch Verlust der Funktionstüchtigkeit des Autos oder schlimmstenfalls durch das Entstehen von Sicherheitsgefährdungen, sind dezidiert in der Systemrobustheit zu analysieren.

/[Sus07]/ verbindet den Systemrobustheitsbegriff mit dem akzeptablen Verhalten innerhalb einer größeren als der ursprünglich von den Designern angenommenen Klasse von Situationen. In /[ISE93]/ wird Robustheit im Bezug auf den Softwarebereich konkretisiert mit[6]:

$$\text{Zuverlässigkeit} = \text{Korrektheit} + \text{Robustheit}$$

Unter Korrektheit versteht sich in dieser Darstellung das Maß für die Fehlerfreiheit der Umsetzung der Produktspezifikation. Weiterhin lassen sich fehlerfreie (korrekte) und robuste Systeme als zuverlässig in ihrer Operation deklarieren.

Zusammenfassend lässt sich somit die Robustheit als das Maß an ungeplanter, nicht verifizierter Einsatzbereiche eines Systems auffassen, in denen über eine Produktspezifikation hinaus eine zuverlässige und korrekte Funktionsweise gewährleistet bleibt. Die Ausprägung dieses Einsatzbereiches wird dabei als Toleranzbereich aufgefasst.

Mögliche beeinflussende Faktoren im Zusammenhang mit robusten Systemen werden in Abb. 2.38 zusammengefasst. Bei der Konkretisierung der mit der Systemrobustheit korrelierenden Bereiche der E/E-Architektur ergeben sich folgende Domänen, die

[6]Der Zuverlässigkeitsbegriff dient in diesem Zusammenhang lediglich der Einordnung des Robustheitsbegriffs. Das verbreitete Verständnis von *Zuverlässigkeit* bezieht sich klassischerweise auf die mittlere Lebensdauer einer Komponente MTTF (*mean-time-to-failure*).

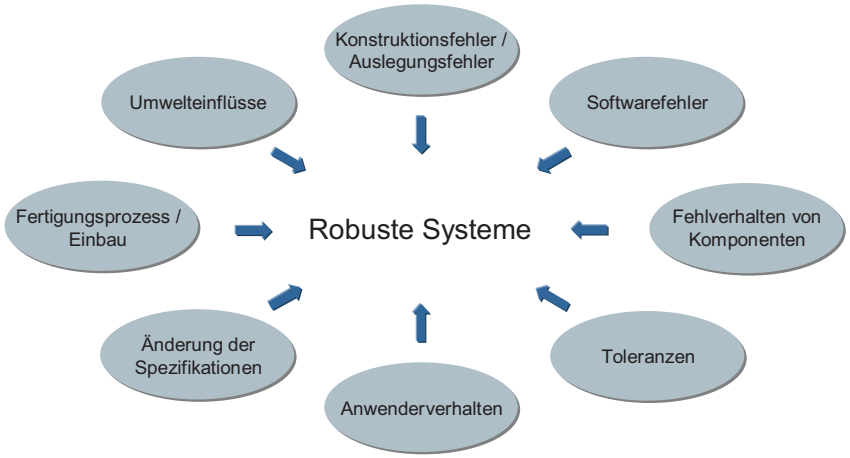


Abbildung 2.38.: Darstellung von Bereichen, die der Erhöhung der Robustheit von Systemen dienen /[SSV06]/

im Designprozess zu beachten sind (s. Abb. 2.39). Das Netzwerkarchitekturdesign und -management beeinflussen die Systemrobustheit. Dazu bieten sich mehrere Ansatzpunkte an /[SSV06]/:

- Einsatz zentraler Fahrzeuggateways zum Management des Schlaf- und Weckverhaltens der Feldbussysteme,
- Begrenzung der Netzwerkauslastung zur Segmentierung von Steuergerätegruppen auf Subbussystemen und eine geeignete Bündelung von Systemsignalen in Botschaften,
- Überprüfung der Einsatzmöglichkeit von Buswächtern (*Bus Guardians*) zur zusätzlichen Überwachung der Buskommunikation,
- Einsatz technologieunabhängiger Standards als Option zur Komplexitätsreduzierung von Netzwerkmanagementmodulen oder höheren Netzwerkprotokollen /[ISO04]/,
- Physikalische Trennung des zentralen Fahrzeugdiagnosezugangs vom allgemeinen Netzwerk,
- Fehlertolerante Botschaftsauswertung,
- Begrenzung der Weckereignisse des Gesamtsystems.

Alle Schnittstellen, die bei der Systemintegration zu berücksichtigen sind (Fahrer, Fahrzeug, Systeme, Kommunikation, Software, Hardware, Aktorik, Sensorik) müssen hinsichtlich der Robustheit analysiert werden.

Die beschriebenen Maßnahmen zur Robustheitserhöhung bringen teilweise auch in anderen Bereichen positive Nebeneffekte. Beispielsweise wird das Thema Energiemanagement zur Optimierung des gesamten Energieverbrauchs im Fahrzeug bereits seit Jahren restriktiv gehandhabt. Mithilfe eines intelligenten Weck-/Schlafkonzepts für die Steuergeräte wird neben der Systemrobustheit auch der durchschnittliche Ruhestrom abgesenkt und damit die Standzeiten des Fahrzeugs im Ruhezustand verlängert, was

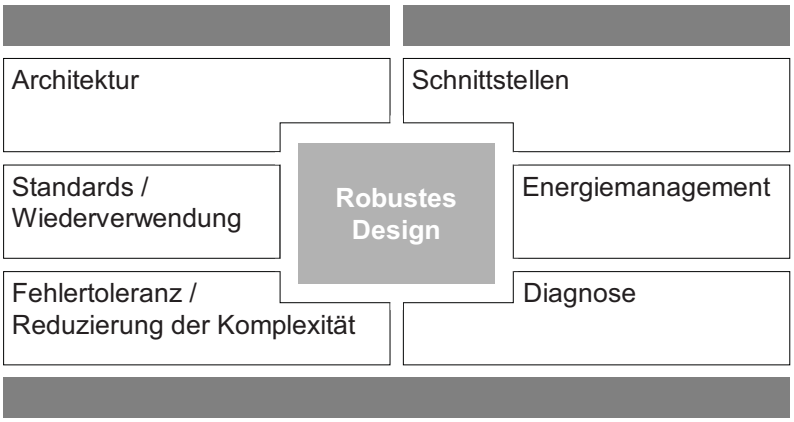


Abbildung 2.39.: Konkrete Schnittstellen der E/E-Architektur mit der Systemrobustheit /[SSV06]/

als Voraussetzung für die sukzessive Erhöhung der Steuergeräteanzahl in der E/E-Architektur gilt.

Um die Robustheit allgemein bereits im Entwicklungsprozess zu erhöhen, eignen sich folgende Vorgehen /[SSV06]/:

1. Überprüfung und Aussage zur Zuverlässigkeit über die ursprünglichen Spezifikationsgrenzen hinweg,
2. Berücksichtigung der notwendigen Robustheit beim Systementwurf und -design,
3. Logische Verknüpfungen in einer Architektur erzeugen bereits im frühen Entwicklungsstadium eine Robustheitserhöhung des Gesamtsystems.

Zuverlässigkeit in vernetzten Systemen

Die Zuverlässigkeit (*Reliability*) eines Netzwerks ist entscheidend für das Einsatzspektrum der zu übertragenden Funktionsdaten. Der Grad an Zuverlässigkeit ist mit der Qualität der Fehlertoleranz des Netzwerks gekoppelt. Zur Steigerung der Fehlertoleranz gibt es unterschiedliche Möglichkeiten /[Kop97], [Che08], [Ech90]/:

Distributed Membership: Mithilfe eines Mitgliedschaftsdiensts (*Membership Service*) verfügt jeder Netzwerkteilnehmer über eine Liste, in der alle weiteren fehlerfrei kommunizierenden Netzwerkknoten verzeichnet sind. Entdeckt ein Knoten Inkonsistenzen zwischen seiner eigenen *Membership*-Liste und der Liste der anderen Netzwerkteilnehmer, so wird sein Aktivitätszustand eigenständig auf passiv zurückgesetzt. Im passiven Zustand „hört“ ein Netzwerkknoten den Busverkehr mit, kann aber selbst nicht aktiv in das System senden. Über die zyklisch mit der Übertragung aktualisierte *Membership*-Liste lassen sich auch korrupte Knoten identifizieren, die fehlerhafte Botschaften versenden. Derartige Knoten werden aus der Mitgliedschaft ausgeschlossen. Mit niedriger Wahrscheinlichkeit können durch Cliquenbildung in der zeitgesteuerten Kommunikation Fehler beim Knotenmitgliedschaftsdienst entstehen /[MSH08]/. Dabei identifizieren sich die Teilnehmer innerhalb von mindestens zwei Gruppen untereinander als fehlerfrei, wodurch praktisch mehrere disjunkte Teilsysteme im Netzwerk existieren, die aus-

schließlich untereinander kommunizieren. In diesem Fall muss dafür gesorgt werden, dass lediglich die größte Clique die Netzwerkkommunikation fortsetzt. Die Knoten aller anderen Cliquen müssen in den Passivmodus wechseln, um sich anschließend wieder auf die größte Clique synchronisieren zu können.

Redundanz: Der Redundanzbegriff lässt sich grundlegend auf drei Bereiche verfeinern. *Informationsredundanz* bezieht sich auf die Ablage von identischen Informationen in unterschiedlichen unabhängigen Speicherbereichen (Informationsreplikation). Im abstrakten Fall reicht auch eine CRC-Verschlüsselung zur redundanten Absicherung einer Information aus. Zeitliche Redundanz bezieht sich auf einen mehrfachen Austausch der identischen Informationen zu unterschiedlichen Zeitpunkten, etwa in zwei verschiedenen Botschaften. Örtliche oder *strukturelle Redundanz* bezieht sich auf die Verteilung der Information über mehrere physikalisch unabhängige Kanäle, beispielsweise der Versand zweier identischer Botschaften auf mehreren unabhängigen angebundenen CAN-Bussen.

N-Version-Programming: *N-version Programming* (NVP) /[Avi95]/ bildet eine Methode zur Entwicklung einer Vielzahl funktionsäquivalenter Softwarekomponenten auf Basis einer identischen Spezifikation. Dabei werden die Funktionskomponenten vollständig und unabhängig voneinander in verschiedenen Entwicklerteams erarbeitet, um die Wahrscheinlichkeit des Entstehens identischer implementierungsbedingter Softwarefehler zu vermeiden. Durch dieses Verfahren lässt sich die Zuverlässigkeit eines Systems verbessern, wobei der verursachte doppelte Entwicklungsaufwand mit hohen Kosten verbunden ist.

Fehlertoleranter Physical Layer: Die physikalische Bitübertragungsschicht trägt je nach Ausprägung zur Erhöhung der Systemzuverlässigkeit bei. Dazu zählen einerseits Möglichkeiten den partiellen Ausfall von Kabelleitungen zu kompensieren, beispielsweise wenn ein Feldbussystem statt im Zweidraht auch im Eindrahtbetrieb die Funktionstüchtigkeit aufrecht erhält (s. Abs. 2.4.1). Alternativ lässt sich auch die Idee des „aktiven Sternkopplers“ anführen, der bei partiellen Defekten im Netzwerk einzelne Bereiche der Netzwerktopologie vom Rest des Netzwerks physikalisch trennt. Dadurch wird verhindert, dass sich ein Fehler über einen gewissen Bereich des Netzwerks hinaus ausbreiten kann (*Error Containment*).

Byzantinisch robuste Fehlertoleranz: Ein wesentliches Kriterium zur Definition der Anzahl startup- und syncfähiger Netzwerkteilnehmer resultiert aus dem Konzept zur Vermeidung byzantinischer Fehler bei der Synchronisation. Ein byzantinischer Fehler bezeichnet die Existenz temporärer schädlicher Netzwerkteilnehmer (*Malign Nodes*), deren korruptes Verhalten ein hohes Gefährdungspotential für die Netzwerksynchronisation und deren Stabilität darstellt. Das grundsätzliche Problem wird allgemein als *byzantinisches Generalsproblem* bezeichnet und wird in /[LSP82]/ behandelt. Die Grundregel $n = 3k + 1$ ist dabei zu erfüllen, falls in einem Cluster mit n Synchronisationsknoten k byzantinische Fehler toleriert werden sollen.

2.4. Grundlagen der Fahrzeugvernetzung

In diesem Abschnitt werden alle aktuell in der Serienproduktion eingesetzten Netzwerktechnologien technisch erläutert. Dazu zählen die Technologien CAN, LIN, MOST und FlexRay. Mittelfristig bildet auch die Ethernet-Technologie eine Alternative in der Fahrzeugvernetzung. Da sich Ethernet mit dem TCP/IP-Protokoll bisher vorrangig als hochperformantes Anbindungskonzept für die Fahrzeugdiagnose anbietet, wird diese Technik in dieser Arbeit nicht weiter betrachtet. Für weitere Details wird auf einschlägige Literatur weiterverwiesen /[Bil08], [Tan00]/.

2.4.1. Controller Area Network

Ein wichtiger Meilenstein ist mit der Implementierung der CAN-Datenbusspezifikation (*Controller Area Network*) /[Rob91]/ im Fahrzeug erreicht worden. Dieses Feldbusprotokoll erfüllt die essentielle Anforderung nach Echtzeitfähigkeit für performante Kontroll-, Steuer- und Regelsysteme im Fahrzeug. Ein weiteres Erfolgskriterium resultiert aus der Schlichtheit des Protokolls, welches akzeptable Freiheitsgrade für Steuergeräteintegrationen bietet und mit einer Datenabsicherung auf physikalischer Ebene Vorteile beim Einsatz im Fahrzeug bringt. Innerhalb der letzten Dekade haben sich die Anforderungen an die Vernetzung im Bereich der Kosten und der technischen Eigenschaften der Zielsysteme (Performanz, Übertragungsfrequenz, Datendurchsatz) weiter verschärft, wodurch neue Netzwerktechnologien entstanden sind. Dazu gehören die bereits in der Fahrzeugentwicklung etablierten Kommunikationssysteme LIN (*Local Area Interconnect*) /[LIN03]/ und MOST (*Media Oriented Systems Transport*) /[MOS04]/.

Grundsätzlich bestehen CAN-Bussysteme aus den in Hardware umgesetzten CAN-*Controllern*, die durch die Verwendung von CAN-*Transceivern* wertdiskrete und zeitkontinuierliche Signale erzeugen. Diese Signale werden über zwei Busadern (*High* und *Low*) an die am selben Bus angeschlossenen CAN-Empfangssteuergeräte übertragen. Die physikalische Übertragungsform ist elektrisch, wobei sich zwei Baudraten etabliert haben (125kbit/s für den Komfortbereich[7] und 500kbit/s für den Antriebs- und Fahrwerksbereich). Entsprechend existieren mittlerweile mehrere CAN-Bussegmente innerhalb einer E/E-Architektur, da die verfügbare Bandbreite eines einzelnen Busses nicht mehr ausreichend für den Datenaustausch aller Steuergeräte ist. Zur Verknüpfung dieser mehreren Bussysteme wird in der Regel ein zentrales Steuergerät (*Gateway*) eingesetzt, welches die Botschaften entsprechend zwischen den Bussen umleitet (*Routing*). Die Routing-Beziehungen aller Steuergeräte werden in einer Tabelle im Gateway-Steuergerät abgelegt. Eine aktuelle CAN-basierte E/E-Architektur wird in Abb. 2.40 beispielhaft dargestellt. Folgende technische Eigenschaften zeichnen das Bussystem speziell für den Serienentwicklung und -einsatz aus:

Übertragungssicherheit: Durch die Möglichkeit Fehler auf Netzwerkebene zu detektieren und zu signalisieren eignet sich das CAN-Protokoll speziell für Echtzeitanwendungen. Durch das CSMA/CR-Verfahren bietet der Bus die Möglichkeit einer zuverlässigen Übertragung durch Kollisionserkennung auf Botschaftsebene. Durch den Einsatz des Bitstopfens, einem extra eingeführten Füllbit (*Stuff-Bit*) mit komplementären Wert bei fünf aufeinanderfolgenden gleichwertigen Bits im Bitstrom, bleibt die Synchronisati-

[7] Aufgrund der gestiegenen Performanzanforderungen wird auch in den klassischen *Lowspeed*-Segmenten mittlerweile mit 500kbit/s übertragen.

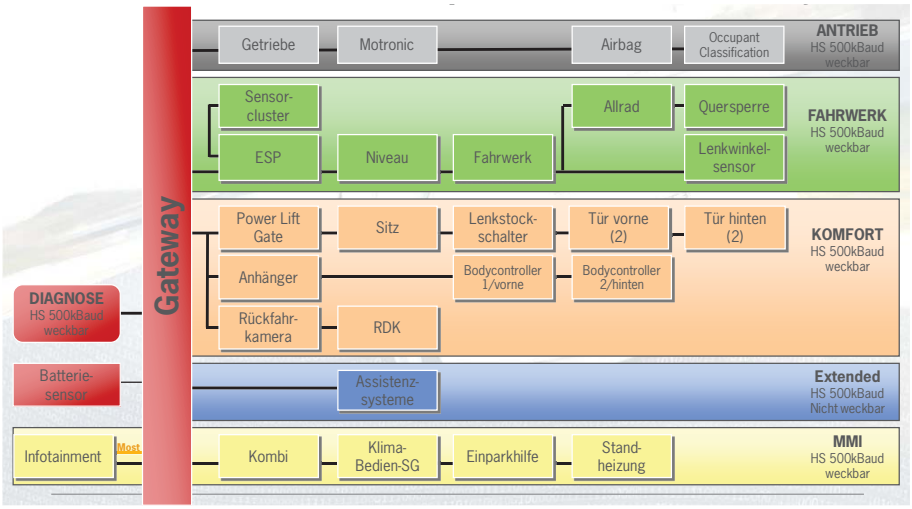


Abbildung 2.40.: Die E/E-Architektur segmentiert Funktionen über Domänen und besitzt als Kernkomponente ein zentrales Gateway-Steuergerät /[Mic07]/

on zwischen den Knoten gewährleistet. Durch den künstlich erzeugten Flankenwechsel im Bitmuster einer Botschaft werden längere gleichwertige Bitsequenzen unterdrückt, um die Synchronisation auf den Signalflanken zeitlich limitiert auszusetzen. Ergänzend dient die Anwendung von CRC-Prüfungen (*Cyclic Redundancy Check*), welche dem Empfänger gestatten die vom Sender gesondert addierten Prüfbits in einer Botschaft zu vergleichen, um die Datenintegrität übertragener Botschaften zu gewährleisten. Per Bestätigungsdienst (*Acknowledge*) quittiert der Empfänger zusätzlich eine erhaltene Botschaft.

Übertragungsgeschwindigkeit: Dieses Feldbusprotokoll liefert bei einer Baudrate von bis zu 1Mbit/s verhältnismäßig gute Latenzzeiten im Millisekundenbereich. Dadurch lässt sich die grundlegende Anforderung nach Echtzeitfähigkeit für performante Kontroll-, Steuer- und Regelsysteme im Fahrzeug für ein breites Anwendungsspektrum erfüllen.

Prinzipiell bietet sich zur Steigerung der verfügbaren Bandbreite eine Übertragungsgeschwindigkeit von 1Mbit/s an, was allerdings zu Lasten der im Fahrzeugserienbereich wichtigen Übertragungsrobustheit geht. Da mit erhöhter Übertragungsgeschwindigkeit eine höhere Busfrequenz einhergeht, werden Einschränkungen in der Flexibilität der verbauten Leitungslängen des Kabelbaums notwendig. In Abbildung 2.41 werden Leitungslängen in Abhängigkeit zu angelegter Baudrate dargestellt. Neben den eingeschränkten Abmessungen der physikalischen Verkabelung erschweren auch schlechte Eigenschaften in der elektromagnetischen Verträglichkeit den Einsatz der maximalen Übertragungsgeschwindigkeit von 1Mbit/s in Fahrzeugserienprojekten. In buslastigen Segmenten des Fahrzeugs werden daher bevorzugt Baudraten um 500 kbit/s implementiert.

Datendurchsatz: Mit mehreren parallelgeschalteten CAN-Bussen lassen sich heute mehr als 70 vernetzte Steuergeräte im Fahrzeug umsetzen. Durch spezielle Betriebsmodi (*Silent Mode*) lassen sich einzelne Steuergeräteprogrammiervorgänge durch hohe Auslastung des Busses performant umsetzen.

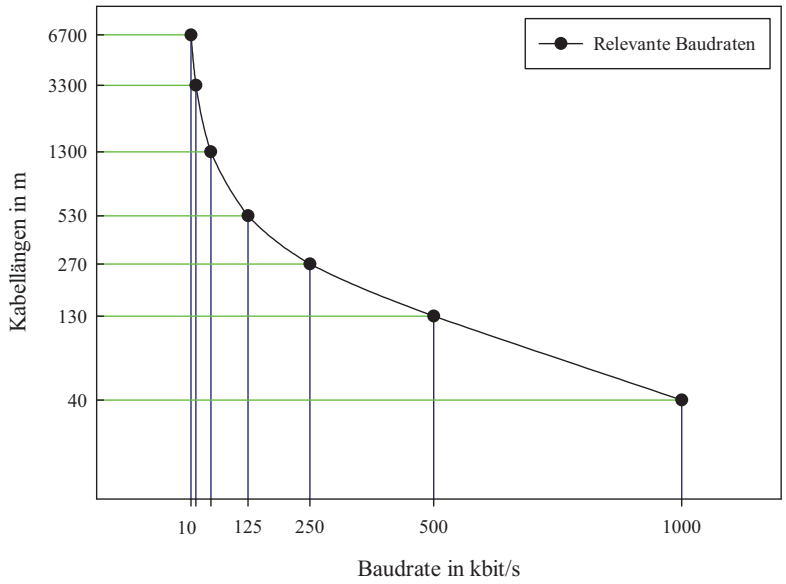


Abbildung 2.41.: Logarithmische Darstellung von maximalen Leitungslängen in Metern pro angelegte Baudrate in kbit/s

Fehlerdetektion und -beschränkungsmöglichkeiten: Durch die Fehlerbitmarkierung und eine gezielte Deaktivierung fehlerhafter Teilnehmer lassen sich Fehler im Übertragungsverhalten identifizieren und dessen Auswirkungen unterdrücken.

Fehlertoleranz: Die physikalische Bitübertragungsschicht zeigt bei den Betriebsfrequenzen annehmbare Eigenschaften für die Variabilität in der Auslegung des Bordnetzes. Dabei bietet der Low-Speed-CAN-Bus bei 125kbit/s die Möglichkeit einer potentiellen Eindrahtübertragung bei Ausfall einer der beiden Kabeladern in der Netzwerkverbindung, was den Grad an Fehlertoleranz erhöht.

Einfachheit: Ein weiteres Erfolgskriterium resultiert aus der Schlichtheit des Protokolls, welches zudem durch seine gute Integrationsfähigkeit für zusätzliche Steuergeräte und der Datenabsicherung auf physikalischer Ebene Vorteile im Automotivebereich bietet. Durch das asynchrone Datenübertragungsprinzip lassen sich Netzwerkteilnehmer ohne explizite zeitliche Planung des Buszugriffs hinzufügen. Lediglich die Festlegung der Botschaftspriorität führt zu externen Effekten im Bussystem durch potentielle zeitliche Verdrängungen von Botschaften, falls diese mit vergleichsweise niedrigeren Prioritäten festgelegt wurden. Derartige Einflüsse machen sich jedoch erst bei höheren Buslasten (>60%) bemerkbar und bleiben daher prinzipiell beherrschbar.

Reifegrad und Standardisierung: Trotz der auf den Fahrzeugeinsatz konzentrierten Entwicklung eignet sich der CAN-Bus auch für die Anwendung in der industriellen Automatisierungtechnik. Das dadurch erreichte breite Einsatzspektrum der Technologie wird durch die international anerkannte ISO-Standardisierung 11898 /[Rob91]/ verstärkt. Diese Historie begünstigt die Unterstützung des CAN-Bussystems auf Zuliefererseite und gilt als Treiber für ein breitgefächertes Angebot an Entwicklungs- und Analysewerkzeugen unterschiedlicher Hersteller auf dem Markt. Zusätzlich existieren neben

mehreren etablierten CAN-Transceivern auch zahlreiche Mikrocontroller verschiedener Halbleiterhersteller mit integrierter CAN-Zelle für dedizierte Einsatzzwecke (Antrieb, Fahrwerk, Gateway, ...) im Fahrzeug. Der Wettbewerb führt zu einem attraktiven Preisgefüge für unterschiedlichste Produktentwicklungen.

Zur Vertiefung der theoretischen Konzepte und technischen Umsetzungen der CAN-Technologie wird auf die offizielle Spezifikation /[Rob91]/ sowie einschlägige Fachliteratur /[Law07], [Ets09]/ weiterverwiesen.

Analyse der CAN-Entwicklung in Serienfahrzeugen

Um die zeitliche Entwicklung des CAN-Busses in PKWs nachzuvollziehen, werden die E/E-Architekturen eines Fahrzeugherstellers aus den letzten Jahren untersucht. Anhand ausgewählter Eigenschaften kann dabei ein Ausblick auf die Weiterentwicklung zukünftiger Netzwerkarchitekturen gegeben werden. Zur Analyse werden dabei folgende Aspekte der CAN-Netzwerkarchitektur betrachtet:

- Anzahl der vorhandenen CAN-Bussegmente, die über eine Gateway-Schnittstelle miteinander verknüpft werden
- Anzahl der insgesamt im Fahrzeug verbauten CAN-Steuergeräte
- Höhe der in der E/E-Architektur verfügbaren Bruttobandbreite
- Höhe der prozentualen Buslast unter der Worst-Case-Annahme pro CAN-Bussegment
- Höhe der durchschnittlichen Buslast in der E/E-Architektur unter der Worst-Case-Annahme pro CAN-Bussegment

Aus den erhobenen empirischen Daten gilt es Aussagen zu treffen, um folgende zukünftige Trends abzuschätzen:

- Anzahl der zu erwartenden CAN-Segmente und CAN-Steuergeräte
- Limits der aktuell definierten CAN-Segmente im Fahrzeug
- Effektivität einer weiteren CAN-Segmentierung hinsichtlich replizierter Gatewaydaten
- Faktoren im Wachstum des Kommunikationsbedarfs zwischen den Fahrzeuggenerationen

Da es bei der Abschätzung der fiktiven Netzwerkkennzahlen auf statistische Werte aus vergangenen Entwicklungen zurückgegriffen wird, soll zur Dokumentation die Wahrscheinlichkeit entsprechender Aussagen hinzugenommen werden. In der Untersuchung werden die verteilten Stichproben per Regressionsverfahren auf eine kubische Polynomdarstellung überführt, deren Genauigkeit und Korrektheit wahrscheinlichkeitsbehaftet berücksichtigt bleiben muss. Zur Orientierung werden daher in den anfolgenden Graphen (2.43, 2.44) Konfidenz- und Vorhersageintervalle für die Regressionskurven hinzugenommen.

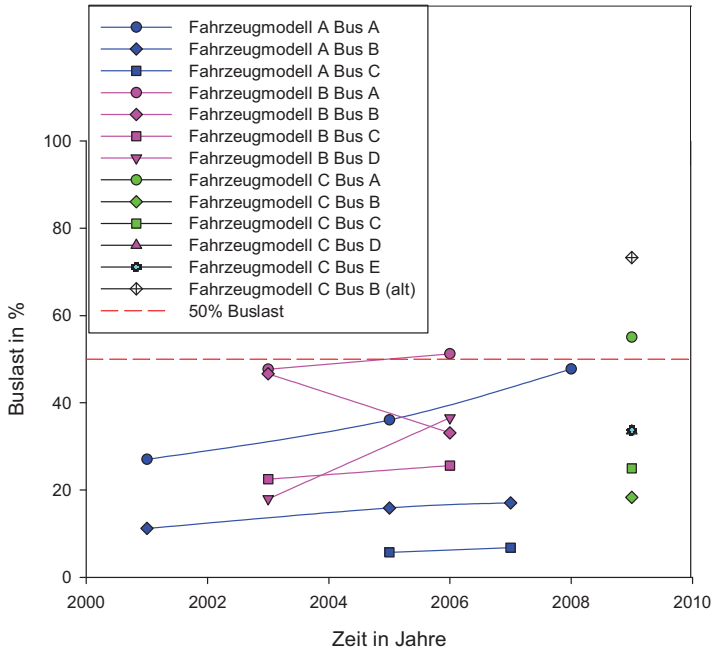


Abbildung 2.42.: Allgemeine Buslastentwicklung von drei verschiedenen Fahrzeugmodellen mit bis zu fünf Bussegmenten im Zeitraum 2001-2009

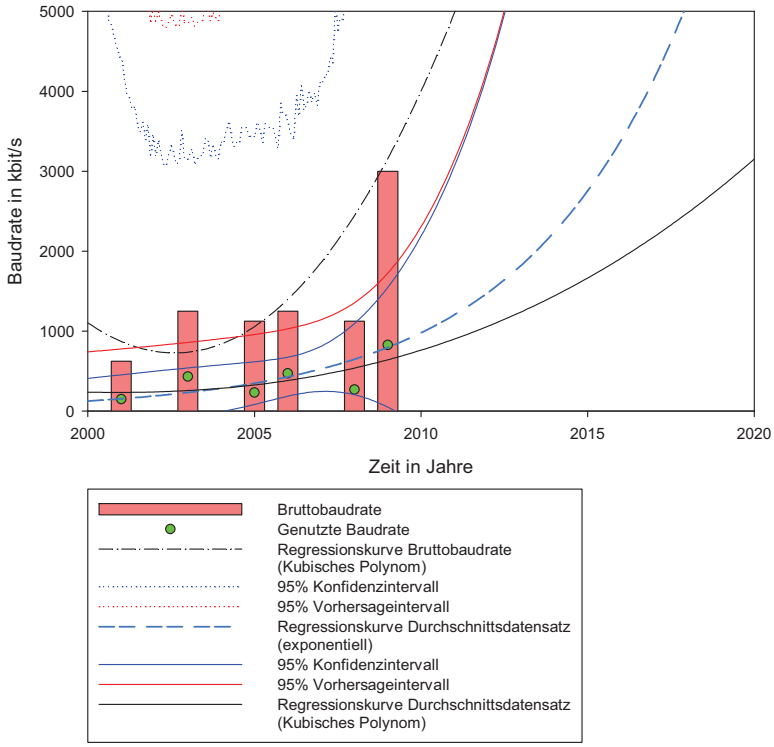


Abbildung 2.43.: Vergleich der Regressionsanalysen von Bruttobaudrate und genutzter Baudrate über mehrere Fahrzeugbaureihen

Das *Konfidenzintervall* stellt dabei wahrscheinlichkeitsabhängig den Bereich für mögliche Varianten der berechneten Regressionskurve auf Basis der erhobenen Stichproben dar[8]. Das *Vorhersageintervall* spezifiziert den Bereich zukünftiger Stichproben, in dem die gesuchten Werte mit einer vorgegeben statistischen Sicherheit zu finden sein werden.

Steuergeräte

Die Wachstumsrate CAN-basierter Steuergeräte innerhalb der E/E-Architektur kann mit einer gebrochen rationalen sowie einer linearen Funktion approximiert werden, da deren Parameter in einer Regressionsanalyse erfolgreich konvergieren. Die beiden Schätzungen unterscheiden sich erheblich bei der Prognose des zukünftigen Steuergerätewachstums. Diese Unklarheiten lassen sich durch komplementäre Einflüsse in der Entwicklung der E/E-Architekturen erklären.

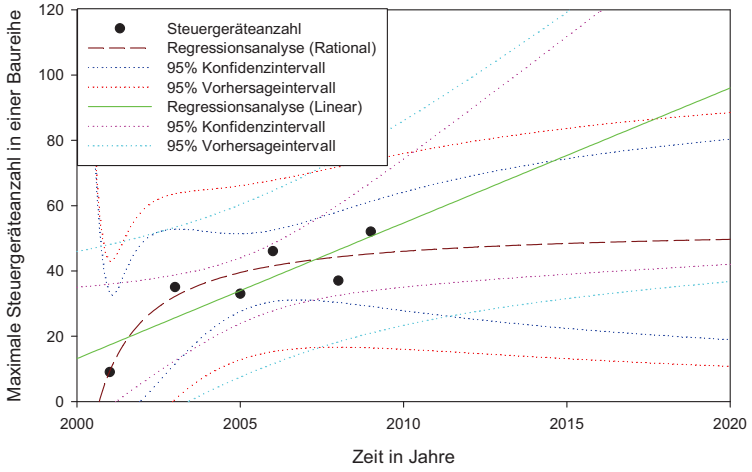


Abbildung 2.44.: Regressionsanalyseergebnisse auf Basis linearer und gebrochen rationaler Funktionen

Einerseits lässt sich eine weitere Optimierung aktueller Steuergerätelösungen hinsichtlich benötigter physikalischer Abmessungen der verbauten Komponenten und deren Kosten erzielen. In Ergänzung mit der verstärkten Elektrifizierung existierender Bauteile, beispielsweise zur Reduzierung des Verbrauchs, wird der lineare Anstieg in Regressionsabschätzung plausibel. Andererseits führt der Einsatz dritter Technologien wie zum Beispiel der LIN-Bus zur Abnahme CAN-basierter Steuergeräte. Der begrenzte Bauraum im Fahrzeug und das Bestreben Funktionen zu bündeln und integriert auf Steuergeräte zu implementieren rechtfertigt die Annahme einer moderaten Zunahme der Anzahl an CAN-Steuergeräten. Dabei darf nicht vernachlässigt werden, dass der Kostenaufwand für die Entwicklung und Organisation begrenzt bleiben muss, was sich in einer limitierten Steuergeräteanzahl ausdrückt. Konsequenterweise lässt sich das Ergebnis der zweiten Regressionsabschätzung mit einem abnehmenden Trend des Steuergerätezuwachses begründen.

[8]Bei einem Wert für ein 95% Konfidenzintervall sind die Endpunkte des Intervalls gegeben durch $\overline{x} \pm t(v,z) * \frac{s}{\sqrt{n}}$, wobei $\overline{x}$ den Erwartungswert, s die Stichprobenstandardabweichung und $t(v,z)$ die t-Statistik für $v = n - 1$ Freiheitsgrade definieren sowie $z = 1.96$ als p-Quantil der Standardnormalverteilung spezifiziert wird

	2001	2005	2009	2015
Max. CAN-Segmente (pro Baureihe)	2	4	6	9,5
Anstiegsfaktor (CAN-Busse/Jahr)	-	0,5	0,5	0,58
Max. CAN-Steuergeräte (p.B.)	9	37	52	75
Anstiegsfaktor	-	4,11	1,41	1,44
Bruttobaudrate (kbit/s)	625	1250	3000	7000
Anstiegsfaktor	-	2,0	2,4	2,3
Genutzte Datenrate (kbit/s)	149,09	431,25	1055,55	2500,00
Anstiegsfaktor	-	2,77	2,45	2,37
Durchschnittliche Auslastung (in %)	23,9	34,5	35,2	35,7
Bereinigtes Datenaufkommen (kbit/s)	-	-	621,01	1444,45
Anstiegsfaktor	-	-	-	2,33
Datenanstieg (innerhalb einer Baureihe)	-	26,29	35,53	-
Anstiegsfaktor	-	1,09	1,11	-
Datenanstieg (zwischen den Baureihen) (kbit/s)	-	188,99	623,83	1444,45
Anstiegsfaktor	-	1,78	2,45	2,32

Tabelle 2.3.: Übersicht und Prognose zur Entwicklung der CAN-Technologie in Serienfahrzeugen

Insgesamt kann somit geschlossen werden, dass zum aktuellen Zeitpunkt keine zuverlässige Aussage über das Wachstum der Steuergeräte im Fahrzeug auf Basis existierender Daten gegeben werden kann.

2.4.2. Local Area Interconnect

LIN spezifiziert ein einfaches kostenoptimiertes Bussystem mit einer Maximalbaudrate von 20kBit/s, welches vorzugsweise für Anwendungen mit begrenzter Komplexität implementiert wird, beispielsweise bei einfacher Sensorik/Aktorik im Fahrzeug (etwa zur Ansteuerung eines intelligenten Schalters). Der Vorteil dieses nach dem Master/Slave-Kommunikationsprinzip arbeitenden Busprotokolls liegt in dem niedrigen Preis, da die Kommunikation physikalisch im Eindrahtbetrieb umgesetzt werden kann. Gleichzeitig wird auf einen dedizierten Kommunikationscontroller in der Regel verzichtet, da sich das Protokoll bereits mit standardisierten digitalen seriellen Schnittstellen implementieren lässt.

2.4.3. Media Oriented Systems Transport

MOST findet seine Anwendungen im Multimediabereich. Die Spezifikation definiert im Gegensatz zu anderen Kommunikationssystemen die Schichten 1-7 des ISO/OSI-Modells und bietet damit dem Endanwender eine einheitliche definierte API (*Application Programming Interface*). Die Kommunikationsfunktionalität stützt sich auf die *Low-Level-* und *Basic-Layer-System-Services*, die einerseits direkt im MOST-Transceiver abgebildet (Schicht 2) und als MOST *NetServices* (Schichten 3-5) und *Application Socket* (Schicht 6) implementiert werden. Die MOST-Technologie basiert auf einer Ringtopologie, die per Tokenmechanismus den Buszugriff reguliert. Der Physical Layer hat sich in Form eines optischen Mediums etabliert. Bei einer Bandbreite von ca. 23Mbit/s sowie einer Systemfrequenz von 44,1 oder 48 kHz eignet sich der MOST ideal zur Übertragung von *Streaming*-Daten (synchrone Datenübertragung). Dadurch können sehr gut Audio- und mit eingeschränkter Qualität Videodaten über den Ring verschickt werden. Wei-

terhin existiert ein asynchroner Datenkanal zur Paketübertragung sowie ein separater Kontrollkanal zur Konfiguration der MOST-Teilnehmer und der (a-)synchronen Datenübertragung. Zur weiteren Information bezüglich MOST und LIN wird auf /[ZS06]/ weiterverwiesen.

2.4.4. FlexRay

Das in dieser Arbeit im Mittelpunkt stehende Kommunikationssystem FlexRay wird seit dem Jahr 2000 analog zu LIN in einem herstellerübergreifenden Konsortium entwickelt, welches das Ziel verfolgt ein neues hochperformantes, fehlertolerantes und echtzeitfähiges Kommunikationssystem zu definieren /[Fle05a]/. Als Vorlage dient das stark verwandte Protokoll TTP/C /[KG94], [TTT03]/, dem aus patentschutzrechtlichen Gründen der Einzug in die Fahrzeugindustrie verwehrt geblieben ist. Trotzdem bilden die technischen Eigenschaften von TTP/C die Grundlage für das FlexRay-Protokoll, welches um die Konzepte des proprietären optischen Bussystems für sicherheitskritische Anwendungen *Byteflight* /[BPG04]/ erweitert worden ist.

FlexRay spezifiziert ein Multi-Masternetzwerk, welches in Bus-, Stern- oder Hybridtopologieform über ein elektrisches Medium auf der Physical Layer-Ebene (ISO/OSI-Schicht 1) kommuniziert. Besondere Eigenschaften entstehen durch die dezentrale, fehlertolerante Uhrensynchronisation /[WL88]/ der Netzwerkteilnehmer, wodurch ein Zeitscheibenkommunikationsprinzip TDMA (*Time Division Multiple Access*) ermöglicht und damit ein deterministisches Zeitverhalten geschaffen wird. Bei der hohen Bandbreite von 10Mbit/s werden die durch das Kommunikationssystem verursachten Genauigkeitsschwankungen (*Jitter*) konstant im Mikrosekundenbereich gehalten, was eine Voraussetzung für schnelle und über die Steuergerätegrenzen hinweg verteilte Regelungskreise (*Closed Loop Control*) darstellt. Es wird ein zweiter physikalischer Kanal zur redundanten Datenkommunikation vorgehalten, um eine Voraussetzung für fehlertolerante Kommunikation, die räumliche Redundanz (*Spatial Redundancy*), zu ermöglichen. Dadurch können permanente oder transiente physikalische Fehler ausgeglichen werden /[Tha00]/. Im Falle von byzantinischen Fehlern werden allerdings noch zusätzliche Mechanismen wie zum Beispiel die zeitlich redundante Verarbeitung von Daten (*Time Redundancy*) erforderlich /[Kop97]/. Der erstmalige Einsatz von FlexRay in einer Fahrzeugserienentwicklung erfolgte 2005 im BMW X5 in Form einer elektronischen Dämpferregelung /[NSF07]/.

Tab. 2.4 stellt die drei verwandten zeitgesteuerten Bustechnologien TTP, TTCAN und FlexRay zum Vergleich gegenüber. Für eine ausführliche technische Beschreibung von FlexRay-basierten Anwendungen wird auf /[Bro05]/ weiterverwiesen. Sämtliche für diese Arbeit relevanten Konzepte werden in Abs. 3.3 kurz zusammengefasst.

2.5. Grundlagen der modellbasierten E/E-Architekturentwicklung

Kernziele im E/E-Architekturdesign basieren auf dem Bestreben das Gesamtsystem *Fahrzeug* in Bezug etwa auf Hardwaretopologien, Mechanik, Regelung, Funktionalität oder Software /[LEk04]/ integriert zu entwickeln. Im Zuge der Etablierung von modellgetriebener Entwicklung eingebetteter Software im Automobil, eröffnen sich weitere Fragen hinsichtlich einer Ausbreitung in die Domänen der E/E-Architekturentwicklung

Protokoll	TTP	TTCAN	FlexRay
Übertragungs-geschwindigkeit	25 Mbit/s; höhere Baudraten möglich	1 Mbit/s; höhere Baudraten nicht möglich wegen CSMA/CD Zugriff	10 Mbit/s; höhere Baudraten möglich, aber aktuell nicht EPL unterstützt
Maximale Botschaftslänge für Applikationsdaten	Variabel für jeden Knoten, bis zu 240 Bytes	Variabel für jede Übertragung, bis zu 8 Bytes	Bis zu 256 Bytes; Im TDMA-Segment einheitliche Slotgröße
Uhrensynchronisation	Single fehlertolerant, verteilte Offset-Korrigierende Synchronisation; formal verifiziert (mit vier oder mehr Knoten), optionale Frequenz-Korrektur zu einer externen Referenzzeit	Master-slave Synchronisation, Netzwerk intern oder durch externe Ereignisse getrieben	Fehlertolerante verteilte Offset- und frequenzkorrigierende Synchronisation (formale Verifikation unbekannt)
Fehlerhypothese / Fehlertoleranz	Toleriert alle singulären Hardwarefehler, zwei redundante Kommunikationskanäle	Keine systematische Fehlertoleranz, keine redundante Kanäle	Toleriert einige singulären Hardware-fehler; keine Fehlerhypothese
Fehlerdetektion	Bitfehler auf dem Bus, Sende- und Empfangsfehler	Bitfehler auf dem Bus, Sende- und Empfangsfehler	Bitfehler auf dem Bus
Konsistenz-unterstützung	Bestätigung, Mitgliedschaft, Cliquenvermeidung	Bestätigung	-
Event-Handling-Strategie	Ereigniskanal über dem TDMA-Schema (CAN-Emulation)	TDMA und zusätzliches Arbitierungsfenster für CSMA/CA	TDMA und zusätzliches dynamisches Segment auf Minislot-Basis
Zusammensetzbarkeit von ereignisgesteuerten Nachrichten	Zeitlich zusammensetzbar	Nicht zeitlich zusammensetzbar	Nicht zeitlich zusammensetzbar
Verfügbarkeit	Chips seit 1998	Chips als Entwicklungs-muster	Chips konform zur Specification 2.1 Rev. A (2005)

Tabelle 2.4.: Gegenüberstellung der drei unterschiedlichen zeitgesteuerten Bussysteme TTP, TT-CAN und FlexRay

/[RK07]/. Motiviert werden diese Überlegungen durch

- Integrierte Datenhaltung sämtlicher im Produktlebenszyklus entstehender E/E-Artefakte (Funktions- und Produktdatenmanagement (FDM,PDM))
- Verknüpfung und Revision von Entwicklungsergebnissen in Form formalisierter Modelle (Applikationsfunktionen, Kabelbaum, physikalisches Netzwerke, etc.) und informeller Artefakte (Anforderungsspezifikationen, Ausstattungslisten, Entwicklerdokumentationen, etc.)
- Analyse und Optimierung ganzheitlicher E/E-Architekturen (globale Optimierung) nach arbiträren Problemstellungen (Kosten, Gewicht, Zeitverhalten, Zuverlässigkeit)
- Variantenmanagement und Austausch komplexer Entwicklungsergebnisse an den Schnittstellen zu beteiligten Zulieferern im Produktentstehungsprozess

2.5.1. Methoden und Werkzeuge

Die Vorteile einer modellgetriebenen Entwicklung lassen sich erhöhen, wenn die Modellierungssprache spezifisch an die vorliegende Entwicklungsbereiche adaptiert wird. Mit der *domänenspezifischer Modellierung* minimiert sich der Entwicklungsaufwand sowie die Anzahl an potentiellen Fehlerquellen, indem domänenspezifisches Wissen und dessen Designkonzepte sowie die zugehörige Quellcodeerstellung als Dreierverknüpfung integriert entwickelt werden /[PK02]/.

Bisher hat sich noch keine allgemein standardisierte domänenspezifische E/E-Modellierungssprache für den Fahrzeugbereich durchgesetzt. Diese Situation lässt sich durch verschiedene Aspekte begründen, die den Entwicklungsfortschritt in dem Bereich signifikant bestimmen. Beispielsweise fehlt einigen allgemeinen Standards der notwendige Zuspruch, da die *Gesamtkomplexität*, der *Abstraktionsgrad*, die *Vollständigkeit*, der *Reifegrad* oder eine *technische Werkzeugumsetzung* nicht ausreichend sind. So fokussiert die Architekturbeschreibungssprache EAST-ADL /[CCG+07]/ vorrangig auf einen integrativen Ansatz von der Featuremodellierung bis hin zur Implementierung, wird aber durch breitere Initiativen zur Softwarearchitekturentwicklung (AUTOSAR /[AUT07a]/), der eingebetteten Systementwicklung (MARTE /[FBSG07]/) und des System Engineerings (SysML /[Sys06]/) überlagert. Daher werden diese Entwicklungsstandards in ATESST /[Ini07]/ mit den EAST-ADL-Konzepten verschmolzen, was durch den starken Entwicklungsfluss in den jeweiligen Initiativen zu einer erschwerten allgemeinen Akzeptanz auf dem Markt führt.

Ein unabhängiges, standardisiertes Werkzeugangebot zum E/E-Architekturentwurf hat sich bislang nur eingeschränkt gezeigt. Diese Entwicklung bezieht sich auf das Bestreben der Fahrzeughersteller ihren Wissensstand zu schützen und konsequent für sich zu behalten. Dadurch wird eine unabhängige standardisierte Werkzeugentwicklung weitestgehend verhindert. Proprietäre Ansätze der Hersteller konzentrieren sich dabei einerseits auf die integrierte Entwicklung von logischen E/E-Architekturen und dem nachgelagerten Software-Generierungsprozess (ORPHEUS /[Sal06]/) oder auf das integrierte Funktions- und Produktdatenmanagement im Produktentstehungs- und -lebenszyklus (DEEP-C /[Sto07]/, [BZ08], [Teu08]/). Dabei wird in /[BZ08], [Teu08]/ eine standardisierte Werkzeugplattform /[Kle08]/ eingesetzt, um in der Funktion als *Daten-Backbone* die technische Grundlage für das integrierte Daten-, Varianten- und Versionsmanagement zu legen, und die zur Abbildung anwenderspezifischer Datenmodelle dient. Bei der Entwicklung AUTOSAR-basierter logischer Architekturen muss im Bereich der unabhängigen, standardisierten Werkzeugentwicklung ergänzend *Systemdesk* /[dSp08]/ aufgezählt werden, welches, ähnlich dem ORPHEUS-Ansatz, verstärkt der integrierten modellbasierten Entwicklung von logischen E/E-Systemarchitekturen dient. Im Bereich der reinen Planung, Analyse und Simulation des Bordnetzes und dessen Signalphysik existieren im Rahmen der technischen Architekturentwicklung eigenständige modellbasierte Werkzeuge (/[Men08], [Syn08]/).

Die Idee der gesamtheitlichen modellbasierten E/E-Architekturmodellierung stellt aktuell eine aus konzeptioneller Sicht vielfältig beleuchtete Domäne in der E/E-Entwicklung im Fahrzeug /[Wal08], [LEk04], [Cor08]/ dar. Mit PREEvision /[Aqu07]/ zeigt sich mittlerweile am Markt eine erste kommerzielle und integrierte Alternative zum heutigen proprietären oder verteilten E/E-Architekturdesign im Fahrzeugserienbereich. PREEvision konzentriert sich als Werkzeug dabei insbesondere auf den frühzeitigen modellbasierten Entwurf von E/E-Architekturen im Fahrzeugentwicklungsprozess.

Aufgrund des intensiven Einsatzes von PREEvision zur praktischen Validierung eines Teils dieser Arbeit wird das zugrunde gelegte Werkzeugkonzept anfolgend kurz erläutert.

2.5.2. E/E-Architekturentwicklung mit PREEvision

Das Werkzeug PREEvision fundiert seit 2005 als kommerziell erwerbliches Werkzeug auf dem Grundgerüst des Forschungsergebnisses *E/E-Architekturbewertungswerkzeug* aus

einer Kooperation zwischen DaimlerChrysler (Forschung und Entwicklung) und dem Forschungszentrum Informatik (FZI) der Universität Karlsruhe (TH). Der Grundgedanke bei der Entwicklung bezieht sich auf das Bestreben E/E-Architekturen schnell und kosteneffizient zu entwickeln unter Einhaltung restriktiver Zeit- und Kostenvorgaben. Neben dem eigentlichen Zweck der Modellierung und dem Aufbau von mehrschichtigen Architekturmodellen zur Systemdokumentation zeigen sich Vorteile bei der Erstellung von Berechnungsmetriken zur Modellabfrage sowie Konsistenzprüfungen und Propagationsregeln bei der Erstellung und Entwicklung des Modells. Basisfragen bei der Metrikanalyse ergeben sich aus dem Vorhaben der globalen Optimierung allgemeiner Systemeigenschaften (Kosten, Gewicht, Zuverlässigkeit, etc.).

2.5.2.1. Modellierungsebenen

Wegen der hohen Komplexität einer heutigen E/E-Architektur ist eine themenorientierte Gliederung des Gesamtmodells erforderlich, welche sich in PREEvision auf insgesamt fünf Ebenen erstreckt. Prinzipiell deckt eine vollständige Modellierung den Entwurfsprozess von der Ausstattungsbestimmung eines Fahrzeugs über deren Anforderungsspezifikation, der Erstellung der logischen und technischen Systemarchitektur bis hin zur Platzierung der Bauteile im Fahrzeug (*Packaging*) bei der mechatronischen Systementwicklung komplett ab. Gewöhnlicherweise führt der Designfluss dementsprechend über den sequentiellen Aufbau von Modellen in folgenden Entwicklungsebenen (s. Abb. 2.45):

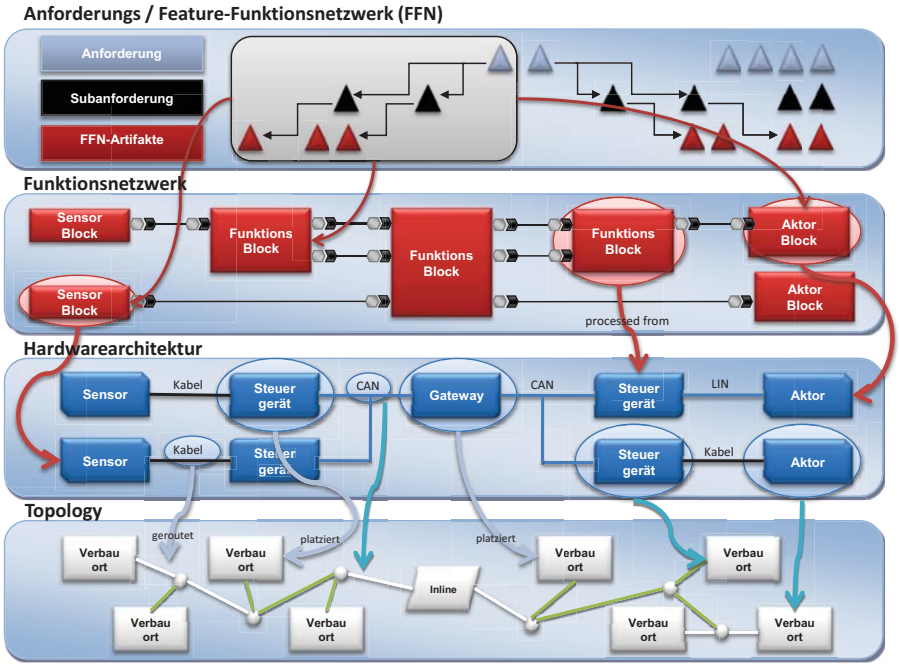


Abbildung 2.45.: Übersicht zu E/E-Modellierungsebenen nach dem PREEvision-Ansatz

1. Kundenanforderungen-Ebene (*Features*)
2. Funktionalitäten-Ebene (*Functional Feature Network*)

3. Funktionen-Ebene (*Funktionsnetzwerk*)

4. Komponentennetzwerk-Ebene (*Komponenten*)

5. Physikalische-Ebene (*Bauräume*)

Jede Ebene verfügt über ein eigenständig zu entwickelndes Modell. Der Anwender bestimmt in dem Zusammenhang die sequentielle Abfolge der Bedatung der Modellierungsdaten im Entwicklungsprozess. Intuitiv wird dieser Vorgang im *Top-Down*-Verfahren durchlaufen. Stehen allerdings die notwendigen Daten zur Modellierung nicht permanent zur Verfügung, lässt sich der Anwendungsprozess auch sprunghaft über mehrere Ebenen durchführen. Sollen die logische und die technische Seite der E/E-Architektur strikt getrennt entwickelt werden so eignet sich das *Meet-In-The-Middle*-Vorgehen. Dabei wird der jeweiligen Sicht entsprechend in disjunkten Bereichen der Modellierungsebenen simultan entwickelt und die produzierten Artefakte anschließend bei der Partitionierung miteinander verknüpft (*Mapping*) (s. Abs. 4.2.7).

Kundenanforderungen-Ebene (FL): In der Kundenanforderungen-Ebene sind alle Anforderungen spezifiziert, die in dem Kraftfahrzeug umgesetzt werden müssen (zum Beispiel ein CD-Player oder ein elektronisches Stabilitätsprogramm (ESP)). Diese Anforderungen können beispielsweise aus dem Anforderungsmanagementwerkzeug DOORS /[Tel06]/ importiert und synchronisiert werden. Dabei erfolgt eine Strukturierung in Gruppen. DOORS bietet dabei jedoch lediglich die Möglichkeit Anforderungen zu sammeln und zu gruppieren. Es existiert keine automatisierte Konsistenzprüfung der Anforderungen oder die Möglichkeit redundante Anforderungen zu finden.

Feature-Funktionalitäten-Netzwerk-Ebene (FFN): In der FFN Ebene werden die Anforderungen weiter verfeinert, indem sie in Wirkketten abgebildet werden. Eine Wirkkette besteht dabei aus einem Ereignis (*Requestor*), der die Anforderung auslöst, einem Ausstattungsmerkmal (*Functionality*), welche bestimmt wie die Reaktion auf das Ereignis aussieht und einer Komponente, die die ausgelöste Anforderung erfüllt (*Fullfiller*) und somit die eigentliche Funktion durchführt. Das Wirkkettenprinzip reduziert die Gefahr des Entwurfs inkonsistenter oder redundanter Systemteile.

Funktionsnetzwerk-Ebene (FN): In dieser Ebene werden gemäß den erhobenen Anforderungen logische Funktionen entwickelt. Die Definition dieser Funktionen erfolgt zu Beginn innerhalb einer Beschreibung von Funktionsbibliotheken. Durch einen Mapping-Vorgang werden die *Requestoren* der FFN-Ebene als Sensorblocktyp, die *Fulfiller* als Aktorblocktyp und alle Ausstattungsmerkmale als Funktionsblocktyp definiert. In der FN-Ebene wird die Nutzungshäufigkeit eines jeden Blocktyps ersichtlich und die Kommunikationsbeziehungen zwischen den verschiedenen Funktionen, Aktoren und Sensoren festgelegt. Die Semantik dieser logischen E/E-Architekturebene entspricht dem AUTOSAR-Standard /[AUT07a]/.

Komponentennetzwerk-Ebene (KMP): Das Komponentennetzwerk beschreibt und visualisiert die verschiedenen technischen E/E-Komponenten in der Systemarchitektur (Steuergeräte, Aktoren, Sensoren) und deren Verbindungen zur Datenübertragung. Die Verbindungen werden durch Bussysteme oder konventionelle Verbindungen realisiert.

Die inneren Komponentenarchitekturen (Hardware- und Softwarebausteine) der Steuergeräte lassen sich dabei optional in expliziter Form modellieren. In einer eigenen Darstellungsebene wird der Aufbau der angelegten Verbindungen zwischen den Komponenten genauer spezifiziert (Verkabelung). Dazu zählen arbiträre elektrische Verbindungskomponenten (Leitungen, Kabel, Adern, etc.)[9].

Physikalische-Ebene (TOP): Auf der physikalischen Ebene werden die für die technischen Komponenten vorgesehenen Einbauorte räumlich in verschiedenen Bauräumen platziert. Durch die Verbindung der verschiedenen Einbauorte fügen sich die Topologiesegmente zu einer Gesamttopologie zusammen. Durch die Abmessung der Größe von und der Abstände zwischen den Bauräumen ergeben sich die Werte für die Dimensionierung der Geometrie des zu entwickelnden Kabelbaum.

Neben dem heterogenen Ebenenkonzept tragen folgende Zusatztechniken zum Gesamtentwicklungskonzept PREEvision bei.

Mappings: *Mappings* verkörpern die Beziehungen zwischen den verschiedenen Ebenen in Form explizit zu definierender Objekte. Dadurch lassen sich die Anforderungen vertikal von der obersten Ebene bis in die vorgesehenen Einbauorte auf der untersten Ebene umsetzen. Die Mappings verknüpfen dabei folgende Artefakte (s. Tab. 2.5).

Anforderung
Mapping
Wirkkettenelemente (*Requestor*, textitFunctionality,*Fulfiller*)
Mapping
Funktionsnetzelemente (Sensor-, Funktions-, Aktorblock)
Mapping
E/E-Komponenten (*Sensor*, *Aktor*, *Steuergerät*)
Mapping
Bordnetzeelemente (Leitungen,Kabel)
Mapping
Fahrzeugtopologien (Einbauort, Topologiesegmente)
Mapping

Tabelle 2.5.: Vertikale semantische Verknüpfung zwischen den Modellierungsebenen durch explizite Mappingstrukturen

Variantenmanagement: Zur differenzierten Entwicklung einer Vielzahl an Fahrzeugderivaten und -baureihen zeigt sich ein feingranulares Variantenmanagement für die entwickelten Modelle als Schlüsselkriterium einer nachhaltigen E/E-Architekturplattformentwicklung. In einem Gesamtmodell werden dabei sämtliche Architekturalternativen berücksichtigt. Das führt zur vollständigen Modellierung sämtlicher Elemente aller Fahrzeugbaureihen und -derivate. Das entstehende Modell ist dadurch insgesamt inhaltlich überladen (150%-Modell) und semantisch fehlerhaft. Durch Filterung dieses Gesamtmodells lassen sich alle Architekturvarianten simultan entwickeln und pflegen. Die Zuordnung der Artefakte zu den Architekturvarianten ist dabei nicht strikt definiert

[9]Zusätzlich lassen sich zu den Standardkomponenten der Sensorik, den Steuergeräten und der Aktorik auch Komponenten der Leistungsversorgung hinzufügen, beispielsweise Batterie-, Generator-, und Massekomponenten.

und wird daher anwenderspezifisch gelöst.

Metrikberechnung: Die bedateten Artefakte lassen sich mithilfe arbiträr definierbarer Metriken über das Gesamtmodell berechnen. Dadurch lassen sich wichtige Eckdaten der E/E-Architektur wie Gewicht, Kosten und Zuverlässigkeit erfassen. In dieser Arbeit bildet die Metrikberechnung ein partielles Konzept der Parameterberechnung für FlexRay-basierte E/E-Architekturen.

Regelbasiertes Prüfen: Durch das Anlegen und Pflegen von Regelwerken lassen sich unterschiedliche Prüfungen des Modells durchführen. Dazu zählen einerseits Konsistenzregeln zur Vermeidung von Semantikfehlern und andererseits Modellabfrageregeln, um allgemeine Eigenschaften von Modellartefakten zu erfassen. Ergänzend gestatten Propagationsregeln die Umsetzung konditioneller Modellaspekte aufgrund kausaler Abhängigkeiten in der E/E-Architektur[10].

2.5.2.2. Modellierung

Das Kernprinzip des PREEvision-Ansatzes fundiert auf dem getrennten Umgang mit der Datenstrukturierung (*Datenrepository*), der Datenmodellierung sowie deren Konfiguration und Bedatung in einer eigenen Sicht (*Eigenschaftssicht*) (s. Abb. 2.46).

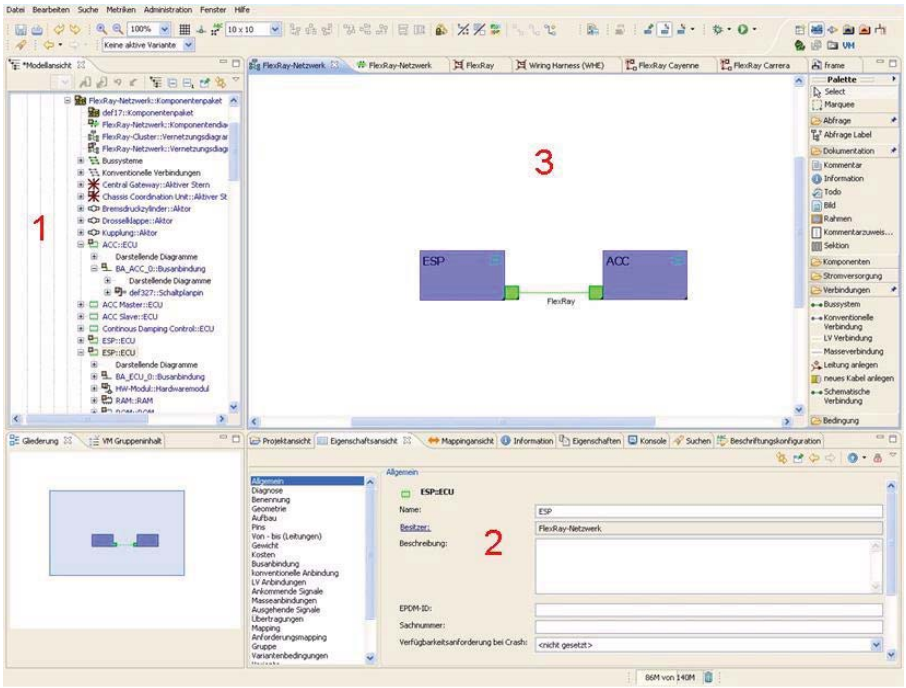


Abbildung 2.46.: PREEvision-Layout mit Datenrepository, Eigenschaftssicht und Datenmodellierung

Jede Modellierungsebene beinhaltet eigenständige Notationsformen zur Generierung eines jeweils entsprechenden Diagramms. Im Kontext der Abbildung und Analyse von

[10]Beispielsweise soll in der Regel beim Hinzufügen eines Steuergeräts und eines Busses zu einer Modellgruppierung auch der verbindende Bustransceiver in die Modellgruppierung aufgenommen werden.

Feldbussystemen sind die FN-, NET-, und TOP-Ebenen am bedeutendsten. Ein FN-Modell wird grundlegend in einer getrennten Abstraktion als *FN-Typenmodell* gepflegt, welches beliebig oft in der eigentlichen FN-Modellierung instanziert werden kann. Dadurch lassen sich mehrfach auftretende identische Strukturen des Gesamtmodells (*Klone*) besser beherrschen und fehlerträchtige Inkonsistenzen vermeiden. In Abb. 2.47 wird unten rechts die Kommunikation zwischen den verschiedenen Blocktypen (Aktor-, Funktions- oder Sensorblocktyp) explizit an den Schnittstellen und deren Signalinhalte als Attribute spezifiziert. Oben links wird in der Abbildung zum Vergleich die instanzierte Form dieses Ausschnitts aus einem Funktionstypenmodell dargestellt.

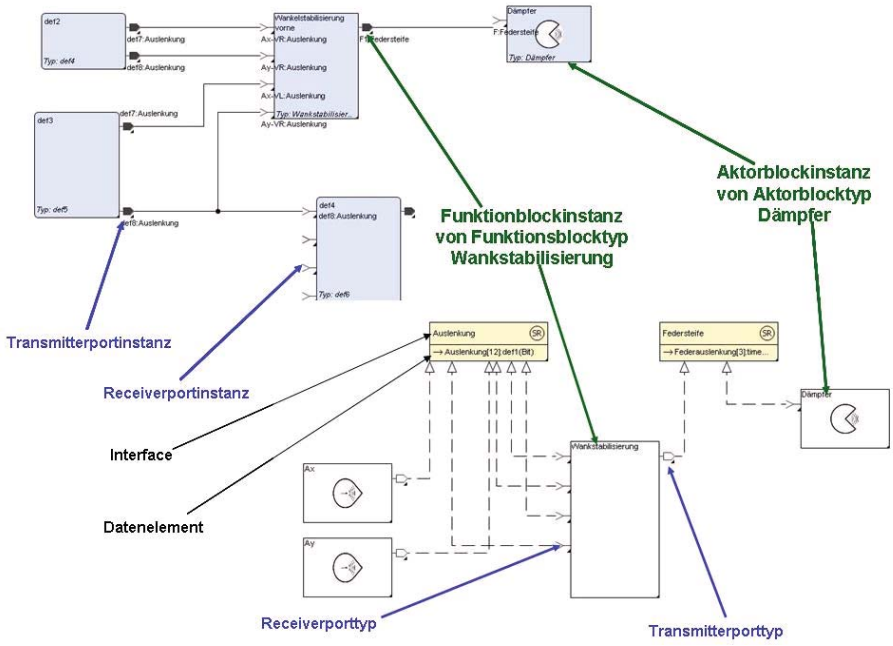


Abbildung 2.47.: Typenmodell und Instanzbildung in der FN-Ebene

2.6. Analyseverfahren

Die Umsetzung ganzheitlicher E/E-Architekturanalysen ist zum aktuellen Zeitpunkt kein standardisiertes Vorgehen der Fahrzeughersteller. Dies lässt sich teilweise mit der noch stark dezentralen Fahrzeugentwicklung begründen. Andererseits hat sich bis heute keine eindeutige Methodik zur Architekturanalyse durchsetzen können. Ein Grund dafür liegt an den unzureichend verstandenen oder der fehlenden

- Vorgehensweisen (Abgrenzung von statischen/dynamischen Analysen),
- geeigneten Werkzeugen,
- Vollständigkeit bisher entwickelter Methoden,
- Kapazität an Architekturspezialisten mit integriertem Wissen über E/E-Systeme.

Die Notwendigkeit der Strukturierung komplexer Systeme ist in der Softwareentwicklung bereits vor vielen Jahren erkannt worden /[Jac75]/. In /[Boo04]/ wird im Zusammenhang der Systemanalyse Folgendes konstatiert: „*Entwicklungsprodukte, inklusive*

Datenflussdiagramme, sind nicht in sich abgeschlossen. Sie sollten als Werkzeuge betrachtet werden für den Zweck das Verständnis des Entwicklers in Bezug auf ein Problem und dessen Implementierung zu verbessern.“ Ein Meilenstein wird allgemein mit der Entwicklung der strukturierten Analyse (SA) verbunden. Ziel dieser Methode ist die Formalisierung der Systemspezifikation in Bezug auf die Entwicklung von Software. Die konkrete Umsetzung der Ergebnisse aus der SA erfolgt während der Entwicklung eines strukturierten Designs der zu erstellenden Software. Speziell die Vorteile der technischen Spezifikation von Softwaresystemen mit den Hilfsmitteln der graphischen Modellierung, der hierarchischen Gliederung und eines *Top-Down*-Entwicklungsstils mit Funktions- und Datenflussmodellierung lassen sich in gleicher Weise auf den E/E-Architekturentwurf übertragen. Der Ursprung der Entwicklung der strukturierten Analyse liegt mitunter in der Entwicklung von SADT (*Structured Analysis and Design Technique*) /[DMR77]/, die sich partiell in den Grundideen der späteren Methoden der strukturieren Analyse /[DeM79]/, dem strukturierten Design /[YC75]/ und der strukturierten Programmierung /[Dij72]/ wiederfinden.

Die Architekturanalyse konzentriert sich auf die rein virtuelle Validierung der Systemanforderungen ohne Verwendung von realer Hardware /[Sch06]/. Primärziele sind statische Aussagen über zu erwartende Auslastungen auf den Steuergeräten (Prozessorlast, Verbrauch an (nicht-)flüchtigen Speicher) und den verbindenden Bussystemen (Durchschnittsbuslast, Spitzenbuslast). Dabei ist von Interesse welche Entwicklung für die Auslastungen bei Systemerweiterungen zu erwarten ist und wann kritische Werte im Zeitbereich bei der Ende-zu-Endekommunikation im System (*Deadlines*) erreicht werden. Im Bezug auf sicherheitsrelevante Systeme stellt sich die Frage mit welchem Systemverhalten bei (Teil-)Ausfall von Systemkomponenten zu rechnen ist.

Prinzipiell kann die Architektur nach zwei Arten analysiert werden. Die Unterschiede dieser beiden Hauptklassen werden im Folgenden erörtert.

2.6.1. Statische Architekturanalyse

Die statische Architekturanalyse bezeichnet eine Analysemethodik, deren Fokus sich auf die Ableitung neuer Erkenntnisse auf Basis einer formal beschriebenen Architektur richtet. Konkret werden auf Basis der initial vorliegenden Architekturdaten strukturelle Eigenschaften und Prädikate geprüft /[Sch06]/.

Eine klassische statische Analyse in der Fahrzeugvernetzung ist die Berechnung der Systembuslasten aller verbauten Feldbussysteme. Diese ergeben sich aus den Datenbasen, die den Signalaustausch zwischen den Steuergeräten in jedem Feldbussystem in Form von Kommunikationsmatrizen spezifizieren. Nachfolgend soll die Idee und das Vorgehen der Buslastanalyse in Bezug auf ereignis- und zeitgesteuerte Technologien kurz erläutert werden.

Buslastanalyse

Während der Entwicklung und Pflege von CAN-Systemarchitekturen bildet die Buslastanalyse einen etablierten Arbeitsschritt bei der Weiterentwicklung und Freigabe von konfigurierten Bussystemen für die Serienproduktion. Bei zeitgesteuerten Architekturen lassen sich die Buslasten im statischen Segment im Gegensatz dazu ohne probabilistische Annahmen feststellen. Zum Vergleich mit den in 5.4 angewendeten Analysemethoden wird das Vorgehen bei der klassischen Buslastberechung eines CAN-Bussystems

zusammengefasst.

Ereignisgesteuerte Systeme: Bei CAN-Systemen ergibt sich die Notwendigkeit der Buslastanalyse aufgrund der getrennten Entwicklung des Funktions- und des Netzwerkdesigns. Folgende Schritte werden dabei durchlaufen:

1. *Applikation:* Festlegung von Restriktionen für Signallatenzen bei der Signalübertragung zwischen zwei Funktionskomponenten im Funktionsdesign,
2. *Partitionierung:* Zuordnung von Funktionskomponenten zu Hardwarekomponenten,
3. *Kommunikation:* Zuordnung von Signalen zu Netzwerkbotschaften und Bestimmung des Sendetyps für jede Nachricht,
4. *Verifikation:* Analyse der Anforderungen an das Zeitverhalten bei der Netzwerkkommunikation zur Einhaltung der Restriktionen aus dem Funktionsdesign bei gleichzeitiger Einhaltung der Buslastgrenzen des zugrunde gelegten Netzwerks.

Technisch lässt sich der Signaltransfer auf einem CAN-Netzwerk zwischen Sensor und Aktor in folgende Teile zerlegen /[Rai02]/:

1. Entprellung der Sensorwertabtastung und Übergabe an die Applikation,
2. Bearbeitung des Sensorwerts in der Basissoftware und Befüllung des Sendepuffers im CAN-Treiber,
3. Verzögerung des Botschaftsversand durch Warteschlange- und Busarbitrierungszeiten,
4. Verzögerung im Treiber des Empfängers und der Übergabe an die Applikation mittels der Basissoftware,
5. Berechnung und Senden des Stellwerts an den Aktor.

Die Summe der einzelnen Teilverzögerungen wird in Abb. 2.48 zur Übersicht dargestellt. Bei der Analyse der Buslast muss grundsätzlich beachtet werden, dass die Untersuchung der WC-Buslast (*Worst-Case-Busload*) nicht zwangsläufig in der Realität erreicht wird. Das liegt zum Teil an den Protokolleigenschaften, die je nach Nutzung zu Unterschieden in der Botschaftslänge (Normale und erweiterte Adressierung, Bitstopfen) führen. Zusätzlich muss logischerweise der Sendetyp (zyklisch, ereignisabhängig, hybrid) für jede definierte Botschaft betrachtet werden.

/[TB94]/ etabliert für sicherheitskritische Anwendungen eine Buslastberechnungsformel, die eine garantierte Nachrichtenverzögerung im CAN-Netzwerk GML (*Guaranteed Message Latencies*) berechnet.

$$w_m = B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{w_m + J_j + \tau_{bit}}{T_j} \right\rceil * C_j$$

$$R_m = J_m + w_m + C_m$$

$$C_m = \left(\left\lfloor \frac{34 + 8s_m}{5} \right\rfloor + 47 + 8s_m \right) * \tau_{bit}$$

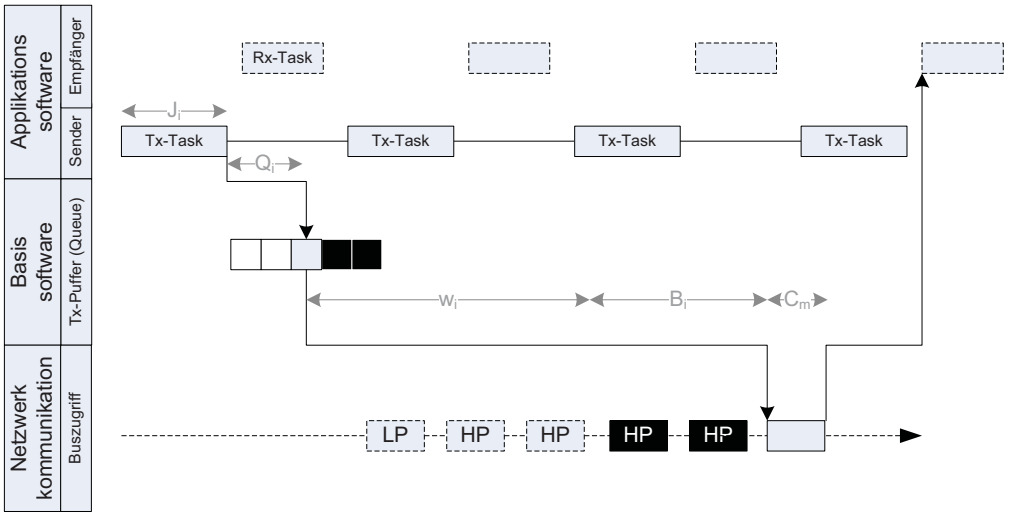


Abbildung 2.48.: Beiträge zum Zeitverhalten des Systems innerhalb einer Punkt-Zu-Punkt-Verbindung in einem CAN-Bussystem

Die Berechnung basiert auf der WC-Verzögerung w_m für eine zum Senden verstaute Nachricht m im CAN-Controller eines Steuergeräts. Mit B_m wird die längste Zeit erfasst, durch welche die gegebene Nachricht m aufgrund Botschaften mit niedrigerer Priorität verzögert werden kann. J_m bezeichnet den potentiellen beim Verstauungsvorgang auftretenden *Jitter*. C_m repräsentiert das längste Zeitintervall für die physikalischen Übertragung der Nachricht m auf dem Bus. Der Term τ_{bit} basiert auf der im Netzwerk vorliegenden Bitzeit und s_m beschreibt die endliche Länge der Nachricht m in Byte. Die in der Berechnungsformel eingesetzte Nachricht j bezieht sich auf die Elemente der Nachrichtengruppe *hp(m)*, welche sämtliche Nachrichten mit einer höheren Priorität als m beinhaltet.

Aus der Betrachtung ergeben sich dabei folgende WC-Annahmen und Abstraktionen im Hard- und Softwarebereich /[Rai02]/:

- Zyklisches Verhalten von ereignisgesteuerten Botschaften → statt des Sendetyps *spontan* wird mit einem zyklischen Sendetyp skaliert)
- *Burst*-Zustände → alle Nachrichten n sind simultan zum Zeitpunkt $t = 0$ sendebereit[11]
- WC-Bitstuffing → maximale Vergrößerung einer Botschaft durch Bitstopfen,
- Ideale CAN-Hard- und Software → Sendelatenzen ohne Verzögerungen oder Unterbrechungen, prioritätenorientierte Warteschlangen beim Senden und Abbruch von wartenden Botschaften im CAN-Controller

Diese offenen Punkte lassen sich durch die Integration von Netzwerkmanagementbotschaften, einkalkulierte Hochlaufverzögerungen der Steuergeräte und mit der Berücksichtigung verschiedener Betriebszustände des Gesamtsystems (Service-/Fahrbetrieb) adressieren.

[11] Dieser Fall ist mit einem entsprechend konfigurierten OSEK-Netzwerkmanagement oder bei der Diagnose nicht abbildbar.

Nach Abb. 2.48 setzt sich die Gesamtlatenz bei den CAN-Bussystemen daher wie folgt zusammen:

$$\text{Latenzzeit} = \text{Warteschlangenlatenz} + (\text{Sendelatenz} + \text{Übertragungslatenz}) + \text{Empfangslatenz}$$

Als Ergänzung lassen sich zwei Prämissen anführen, die bei der Buslastanalyse für CAN-Bussysteme beachtet werden müssen. Dazu zählt, dass die Gesamtsendelatenz einer Botschaft m die maximale Sendetoleranz (*Deadline*) D_m stets unterschreiten muss. Wenn D_m dabei kleiner als die Zykluszeit T_m ausgelegt worden ist, lassen sich Botschaftsverluste im Netzwerk verlässlich ausschließen. Zusätzlich müssen die Effekte der Prioritäteninversion beim Senden am Bus berücksichtigt werden[12].

Als weiteres Beispiel für den Anwendungszweck der statischen Systemanalyse lässt sich der in der Softwareentwicklung bereits seit Jahren erforschte Bereich der WCET-Analyse nennen (vgl. Abs. 3.1.5.1). Das Ziel dabei ist die Analyse und Berechnung von maximalen Ausführungszeiten WCETs (*Worst Execution Times*) von Softwareprogrammen unter Berücksichtigung unterschiedlich leistungsfähiger Hardwarekonzepte /[PF99], [KLFP02], [WEE+08]/ (s. Abs. 3.1.5.1). Interessant wird die statische Analyse bei der Verknüpfung und Interpretation der Ergebnisse von Teilanalysen. So erweist es sich als sinnvoll WCET-Analysen von Anwendungssoftware der Steuergeräte mit den Ergebnissen der Analyse von Übertragungslatenzen beim Signalaustausch über die Netzwerke zu verbinden. Die Möglichkeit der Verkettung der Ergebnisse trägt dazu bei, Verletzungen im Zeitbereich bei zeitkritischen Systemen frühzeitig zu erkennen.

2.6.2. Dynamische Architekturanalyse

Die signifikante Differenzierung der dynamischen von der statischen Architekturanalyse entsteht durch die aktive Untersuchung vorliegender Datenmodelle in Form einer Systemstimulierung mit konkreten Eingangsdaten /[Sch06]/. Dieses Szenario entspricht einer Systemsimulation zur Überprüfung des Systemverhaltens und zur Generierung von Ausgangsdaten, die einer Analyse unterzogen werden. Allgemein lässt sich speziell die Simulation zur Umsetzung der dynamischen Architekturanalyse als das „Experimentieren mit Modellen“ auffassen /[AJ78]/. Die Analyse innerhalb des experimentellen Vorgehens basiert auf der Interpretation von Systemreaktionen, die durch bestimmte Systemstimuli erzielt werden.

Grundsätzlich lässt sich die dynamische Architekturanalyse in den Bereichen anwenden, bei denen eine theoretisch formale Untersuchung nicht mehr vollständig oder nur mit hohem Aufwand durchgeführt werden kann. Dazu zählen bei der allgemeinen Fahrzeugentwicklung vorrangig Bereiche wie Betriebsfestigkeit, Crash, Fahrdynamik, Strömung und Verbrennung. Für den Bereich der E/E-Architektur bietet sich die funktionale Simulation der verteilten Anwendungen auf den Steuergeräten unter Berücksichtigung temporaler Aspekte und vorhandener Systemressourcen an /[Sch06]/. Der im Rahmen dieser Arbeit relevante Teil der dynamischen Architekturanalyse konzentriert sich auf die Simulation des Weck- und Startverhaltens eines FlexRay-Clusters im Fahrzeug.

[12] Die Buslastanalyse ist für die zeitgesteuerten Architekturen trivial, da das Kommunikationsverhalten bereits durch den Busschedule vorgegeben wird. Daher kommt der Parametrierung des Bussystems umso mehr Bedeutung zu (s. Abs. 4.6).

KAPITEL 3

Zeitgesteuerte Architekturen

Das Prinzip der Zeitsteuerung steht im Kontrast zu den klassischen ereignisgesteuerten Bussystemen. Es werden die signifikanten Unterschiede zwischen diesen beiden Paradigmen herausgearbeitet und prägnante Eigenschaften der zeitgesteuerten Architekturen dargestellt. Der Stand der Technik existierender Entwicklungsmethoden im Kontext flexibler zeitgesteuerter Architekturen wird erläutert. Abschließend werden die relevanten Spezifika des flexiblen zeitgesteuerten Bussystems FlexRay zusammengefasst.

Der Ursprung zeitgesteuerter Systeme korreliert mit der Entwicklung des Paradigmas der synchronen Sprachen. Die Programmiersprachen ESTEREL und LUSTRE unterscheiden beispielsweise strikt zwischen logischer und zeitlicher Steuerung /[Ber00], [HCRP94]/. Bei diesen Ansätzen erfolgt die Ausgabe berechneter Daten synchron mit der Dateneingabe. Dabei darf die maximale Berechnungsdauer der Ausgabedaten die minimalen Zykluszeiten für externe Ereignisse niemals überschreiten. Dadurch lässt sich das System mithilfe der Zeitsteuerung durch gezeitete externe Ereignisse betreiben. Die Grundidee der Zeitsteuerung unter Berücksichtigung maximaler Ausführungszeiten (WCET) findet sich in der an der TU Wien entwickelten zeitgesteuerten Architektur TTA (*time-triggered architecture*) /[Kop98]/ wieder. Mit TTP/C (*time-triggered protocol class C*) /[TTT03]/ ist ein leistungsfähiges zeitgesteuertes fehlertolerantes Bussystem zur TTA hinzugefügt worden, welches sich als Entwicklungsvorlage in großen Teilen mit dem flexiblen zeitgesteuerten Bussystem FlexRay überdeckt. Die technischen Aspekte (flexibler) zeitgesteuerter Architekturen werden im Folgenden vorgestellt.

3.1. Aspekte der Zeitsteuerung

Im Unterschied zu den ereignisgesteuerten Architekturen unterliegen die zeitgesteuerten Architekturen strikten Richtlinien, Vorgaben und damit Einschränkungen (*Cons-*

traints) innerhalb ihrer Gesamtspezifikation /[Kop98]/. So gilt die Grundregel, dass exakte Zeit- und Wertbereiche für sauber zerlegte und gekapselte Subsysteme vorliegen müssen. Die Subsysteme können zu einem Gesamtsystem komponiert werden (*Cluster*), wodurch zeitgesteuerte Architekturen als zusammensetzbar (*composable*) bezeichnet werden können.

Eine herausragende Eigenschaft einer zeitgesteuerten Architektur ist die Fähigkeit Information in hoher temporaler Präzision zu behandeln. Im Fahrzeugbereich tritt diese Anforderung überall dort auf, wo genaue Zustandsaussagen zu exakt bestimmten Zeitpunkten erwartet werden. Speziell hochauflösende und sich schnell ändernde Variablen, in /[Kop98]/ als *Realtime Entity* bezeichnet, erfordern ein besonderes Maß an Exaktheit. Im Fahrzeug sind Schaltvorgänge im hohen Drehzahlbereich ein Beispiel für eine Funktion mit hoher temporaler Präzision. Um ein Echtzeitsystem innerhalb einer zeitgesteuerten Architektur abzubilden, müssen dessen *harte* und *weiche Echtzeitbedingungen* erfüllt werden. Diese Bedingungen sind in der Regel mit der Beendigung eines Prozesses und der Rückgabe der entsprechend zu einem Zeitpunkt erforderlichen Informationen verknüpft.

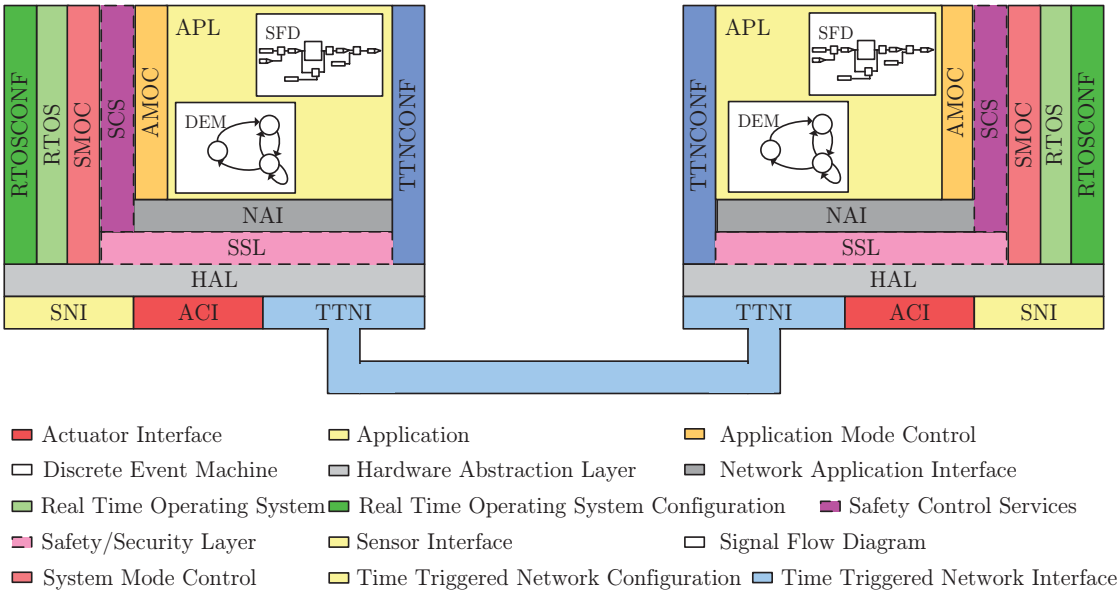


Abbildung 3.1.: Beispielhafte Softwareschichtenarchitektur für ein zeitgesteuertes verteiltes System

Eine verteilte zeitgesteuerte Architektur in Form eines *Clusters* besteht somit aus zeitgesteuerten Subsystemen, welche an wohldefinierten Schnittstellen über ein zeitgesteuertes Netzwerkprotokoll Information austauschen (s. Abb. 3.1). Folgende Abstraktionsebenen, Dienste, Steuerungen und Konfigurationen sind in diesem Zusammenhang von Bedeutung:

- Schnittstellen zu Sensorik, Aktorik und dem zeitgesteuerten Bussystem,
- Allgemeine Hardwareabstraktionsschichten für Hardware-Komponenten,
- Optionale Sicherheitsebenen und -dienste,
- Schnittstellen zwischen Applikations- und Netzwerksoftware,

- Steuerung der Betriebsmodi auf Applikations- und Systemebene,
- Echtzeitbetriebssystem für die Prozesssteuerung,
- Konfigurationen für Netzwerk und Betriebssystem,
- Funktionen auf Applikationsebene auf Basis formaler Notationen (Datenflussdiagramme, Zustandsautomaten).

3.1.1. Abgrenzung von Ereignis- und Zeitsteuerung

Das Prinzip der Datenkommunikation innerhalb reaktiver Systeme bezieht sich in Abstraktion auf die Implikation des Prinzips „*Ursache → Wirkung*“. Von ereignisgesteuerten Systemen wird gesprochen, wenn eine Kausalität die Kommunikation einer Information zwischen einem Sender und dem zugeordneten Empfänger auslöst, beispielsweise verkapselt in einer Botschaft. Dabei wird das Zeitverhalten der Botschaftsübertragung in erster Linie von kausalen Einflüssen, welche eine Kommunikation erfordern, bestimmt.

Im Gegensatz dazu stehen die zeitgesteuerten Systeme, deren Verhalten ausschließlich durch das Fortschreiten einer allgemeinen vorgegebenen logischen Zeit bestimmt wird. Im Kontext der zeitgesteuerten Kommunikation wird dieses Paradigma statisch durch vordefinierte und bestimmten Sendern zugeordnete Zeitpunkte für die Botschaftsübertragung umgesetzt.

Der Bereich der vernetzten eingebetteten Systeme im Fahrzeug wird in diesem Zusammenhang durch heterogene Systeme bestimmt, die einerseits zeitgesteuert eine zyklische Kommunikation zur Realisierung regelungstechnischer Algorithmen umsetzen und andererseits ereignisgesteuerte Eingriffe, etwa bei der Steuerung durch den Fahrer, verarbeiten müssen. Je nach Medienzugriffskontrolle müssen bei zeit- oder ereignisgesteuerten Bussystemen jeweils Kompromisse zugestanden werden. Bei ereignisgesteuerten Bussystemen wird die zyklische Kommunikation nur durch künstliche Steuerung in der Steuergerätesoftware erreicht während sich rein zeitgesteuerte Bussysteme problematisch im Umgang mit schnellen Reaktionen auf spontane Ereignisse zeigen. Mit den flexiblen zeitgesteuerten Bussystemen wird versucht beiden Anforderungen innerhalb einer Technologie bestmöglich gerecht zu werden.

3.1.2. Eigenschaften physikalischer Zeit

Die Grundlage für die Qualität eines zeitgesteuerten Systems wird durch die Eigenschaften der dem Netzwerk zugrunde gelegten physikalischen Zeit gelegt. Der folgende Abschnitt definiert die wichtigsten Begriffe im Umfeld der Zeitsteuerung. Für Details wird auf /[Sch96], [Kop97]/ weiterverwiesen.

Die Basis der physikalischen Zeit steht in Relation zu den existierenden *physikalischen Uhren*. Diese sind in einem verteilten zeitgesteuerten Bussystem auf jedem Knoten notwendig, um den Verlauf physikalischer Zeit messen zu können. Eine Zeitmessung erfolgt im Zusammenhang mit einem *Zähler* und einem *physikalischen Oszillator*, dessen Schwingungen den Zähler in gleichmäßigen diskreten Zeitschritten, den *Mikroticks*, inkrementieren. Die Schrittweite $\frac{1}{f^z}$ des Zählers wird allgemein als *Granularität* der gewählten Uhr g^k bezeichnet. Die Qualität einer physikalischen Uhr orientiert sich an der Qualität einer Referenzuhr.

Referenzuhr: Aus Sicht eines abstrakten Beobachters wird eine einzigartige Referenzuhr definiert, die im perfekten Einklang mit dem internationalen Standard der Zeit steht. Dadurch schreiten Zählschritte einer Referenzuhr synchron mit Zählschritten des internationalen Zeitstandards fort. Die Referenzuhr basiert auf der Idee einer feinauflösenden Uhr ohne Berücksichtigung relativistischer Effekte mit einer hohen Mikrotickfrequenz im Bereich von $10^{15}\mu T/s$. Durch diese feine Granularität wird es möglich Digitalisierungsfehler bei der Vergabe von Zeitstempeln für auftretende Ereignisse gering zu halten. Durch diese absoluten Zeitstempel lassen sich einerseits die Zeitintervalle zwischen dem Auftritt zweier Ereignisse bestimmen und anderseits deren logische Ordnung bezogen auf den Zeitverlauf (temporale Ordnung) erfassen. Die Eigenschaften sämtlicher Uhren des verteilten Systems (Frequenz/Granularität) werden auf Basis der μT-Anzahl der Referenzuhr gemessen. Die temporale Ordnung der Ereignisse, welche die Granularität der Referenzuhr g^z unterschreiten, bleibt dabei unberücksichtigt.

Uhrendrift: Der Drift einer Uhr zwischen zwei Zeitpunkten i und $i+1$ in μT beziffert das Frequenzverhältnis zwischen einer Uhr k und der Referenzuhr z zum Zeitpunkt i. Der Drift wird bestimmt durch die Messung eines diskreten Zeitschritts der Uhr k mit der Referenzuhr z und anschließender Division durch die Anzahl an nominalen μTs der Referenzuhr in diesem Zeitschritt ($i \rightarrow i+1$). Eine perfekte Uhr unterscheidet sich dabei nicht von der Referenzuhr. Ein gemessener Unterschied $\Delta\left[z(\mu T_{i+1}^k) - z(\mu T_i^k)\right]/n(k)$ quantifiziert die Driftrate.

Jede in einem Systemverbund verbaute freilaufende Uhr wird trotz Synchronisation mit der Referenzuhr zu einem arbiträren Zeitpunkt in endlicher Zeit jegliche begrenzte relative zeitliche Abweichung überschreiten. Perfekte Uhren lassen sich aufgrund von *Temperatureinflüssen*, *Spannungsschwankungen* und *Alterungseffekten* nicht auf Schwingquarze abbilden.

Physikalische Uhren: Zur Abbildung logischer Uhren auf elektronische Bauteile werden *Oszillatoren* benötigt, um ungedämpfte elektrische Schwingungen zu erzeugen. Diese Komponenten können aus selbstschwingenden Bauteilen oder aus mehreren, zu einer Oszillatorschaltung zusammengefügten, Bauteilen bestehen.

Geschaltete Taktnetzwerke werden in einer Baumstruktur angeordnet (*clock tree*), um Laufzeitunterschiede bei der Erzeugung und Verteilung der Taktsignale in einer Baugruppe (*skew*) minimal zu halten. Durch den Einsatz der PLL-Technik (*Phase-Locked-Loop*) lassen sich durch phasengekoppelte Regelschleifen sehr stabile Taktgeneratoren bauen, die auf einem Referenztakt basieren und nahezu jede beliebige Ausgangsfrequenz erzeugen können. Dadurch können beispielsweise zentrale Takte auf Quarzbasis aus einem größeren System zur Taktgeneration herangezogen werden. Entsprechende PLL-basierende Bausteine wandeln diesen zentralen Systemtakt auf den im Cluster zu applizierenden Takt um (*Translation*).

Globale Zeit: Gegeben sei ein Uhrenensemble einer Knotenclusters mit jeweils einer Uhr pro Knoten. Diese Uhren laufen intern synchronisiert, da die absolute zeitliche Differenz zweier arbiträrer Uhren durch die maximale Präzision des Clusters zwischen potentiellen Synchronisationsphasen bestimmt ist. Durch die Bildung einer Submenge an Mikroticks für jede Uhr wird ein nominaler Zeitschritt erzeugt, der clusterweit eine identisch Länge besitzt. Diese Zeitschritte (*Makroticks*) beschreiben somit ein auf Clusterebene einheitliches Zeitkonzept, welches als *globale Zeit* bezeichnet wird.

3.1.3. Synchronisation

Der Aspekt der Zeitsteuerung in einem Bussystem erfordert das Verständnis hinsichtlich spezieller Eigenschaften zur Gewährleistung einer korrekten Ausführbarkeit der Netzwerkkommunikation. Mit entsprechender Priorität sind dabei die nachfolgenden Attribute zu behandeln, um eine optimierte Systemausprägung zu entwickeln:

Genauigkeit: Der zeitliche Versatz einer Uhr j gegenüber einer Referenzuhr z zum Zeitpunkt i wird als $Genauigkeit_i^j$ bezeichnet. Unter $Genauigkeit^j$ versteht sich die maximale zeitliche Abweichung der Uhr j zur Referenzuhr über eine arbiträre Periode, etwa der Buszykluslänge. Die periodische Synchronisation der Uhr zur Referenzuhr wird als *externe Synchronisation* bezeichnet.

Präzision: In einer Menge an Uhren $C = 1, 2, ..., n$ wird die maximale zeitliche Verschiebung zweier arbiträrer Uhren (*offset*)

$$\Pi_i = \max_{\forall j,k \in C} (offset_i^{j,k})$$

als Präzision von C Π_i zum diskreten Zeitpunkt i bezeichnet. Bei der Betrachtung der Präzision über ein arbiträres Zeitintervall, etwa der Buszykluslänge, bezeichnet $max(\Pi_i)$ die allgemeine Präzision des Uhrenverbunds pro Buszyklus. Die diskreten Zeitpunkte und die Präzision als absolute Abweichung einer Uhr von der Referenzuhr werden bei FlexRay in Mikroticks gemessen. Durch die endliche Driftrate einer physikalischen Uhr ist eine periodische *Resynchronisation* der einzelnen Elemente in der Uhrenmenge C erforderlich. Findet die Synchronisation wechselseitig ohne externe Beeinflussung statt, so wird von *interner Synchronisation* gesprochen. Bei FlexRay definiert sich die *Clusterpräzision* durch den zeitlichen Versatz der diskreten Absatzzeitpunkte eines Makroticks zwischen zwei Kommunikationscontroller (CC) im Netzwerk. Die *Ratenpräzision* bleibt dabei durch die lokalen Uhrenabweichungen in einem Buszyklus begrenzt, wobei sich die Clusterpräzision prinzipiell über unendliche Zyklen aufschaukeln kann (*Drift*).

Mikrotick: Der kleinste diskrete Zeitschritt auf Hardwarebasis wird im Zusammenhang mit zeitgesteuerten Bussystemen als *Mikrotick* bezeichnet. Bei einem FlexRay-System sind drei unterschiedliche Mikroticklängen für die Uhrenbasis zur Konfiguration eines CC zulässig $\mu T = (12,5ns; 25ns; 50ns)$. Die Mikrotickanzahl weicht in den Buszyklen leicht voneinander ab, da unterschiedliche Mikroticklängen pro Knoten konfiguriert werden können und sich damit der nominale Makrotick nicht zwangsläufig einheitlich durch eine ganzzahlige Menge an Mikroticks in jedem CC umsetzen lässt.

Worst-Case und Best-Case Präzision: Aufgrund der unterstützten skalierbaren Fehlertoleranz lässt sich die Präzision in zwei verschiedene Ansätze untergliedern. Für den Fall der Toleranz von byzantinischen Fehlern, einer Fehlerklasse bei ein Teilnehmer im Cluster zu einem Zeitpunkt unterschiedliche Ergebnisse für ein Ereignis an verschiedene Empfänger kommuniziert, beispielsweise für einen Zeitstempel, wird die Worst-Case-(WC)-Präzision herangezogen. Entscheidend zeigt sich eine Knotenanzahl $n > 3$, da nach der Regel $n = 3k + 1$ k byzantinische Fehler in einem n-Knotennetzwerk abgefangen werden können. Ausnahmesituationen bei mehr als k byzantinischen Fehlern

oder fehlenden Synchronisationsbotschaften von einem oder mehreren Steuergeräten in einem FlexRay-Doppelzyklus bleiben dabei unberücksichtigt. Die Best-Case-(BC)-Präzision geht von der Annahme aus, dass keine byzantinischen Fehler im Netzwerk toleriert werden müssen. In diesem Fall lassen sich höhere Nettodatenraten auf dem Bussystem umsetzen. Bei der Resynchronisation werden die Abweichungen der Uhren innerhalb ihrer Präzision durch eine Absatzkorrektur (*offset correction*) zurückgesetzt und Abweichungen zwischen den Taktraten jeder Uhr auf Makrotickebene korrigiert (*rate correction*).

3.1.4. Zeitgesteuerte Kommunikationssysteme

Als Teilgebiet der zeitgesteuerten Architekturen haben sich im Laufe der Zeit die zeitgesteuerten Busprotokolle entwickelt. Die *SAFEbus*-Technologie (1992) /[HD92]/ der Firma Honeywell gilt als eine der frühen Implementierungen und kommt in der Boeing 777 zur Flugsteuerung zum Einsatz. Die mitunter bekannteste und am intensivsten erforschte zeitgesteuerte Bustechnologie ist das *Time-Triggered Protocol* (TTP) /[KG94]/. Speziell das Derivat TTP/C ist vorrangig für die Anwendung in sicherheitskritischen Systemen innerhalb der Avionik und dem Fahrzeugbereich konzipiert worden. Parallel zu den neuentwickelten zeitgesteuerten Protokollen hat es Versuche gegeben, den CAN-Bus als etablierte ereignisgesteuerte Technologie im Fahrzeugserienbereich um ein zeitgesteuertes Paradigma zu erweitern (*Time-Triggered CAN* (TTCAN)) /[Rob02]/. Aktuell wird dem flexiblen zeitgesteuerten Bussystem FlexRay im PKW-Bereich das größte Potential zugeschrieben /[NSF07]/.

Ein wesentlicher Unterschied zwischen ereignis- und zeitgesteuerten Kommunikationssystemen liegt in der konkreten Konfiguration des jeweiligen Netzwerkprotokolls. Während sich der CAN-Bus im Wesentlichen durch seine Baudrate und dem physikalischen Abschlusswiderstand konfigurieren lässt, so erfordert ein zeitgesteuertes System eine deutlich komplexere Konfiguration. Die im Fall von FlexRay verankerten 74 Parameter spannen einen theoretischen Lösungsraum von 10^{48} Varianten auf /[ASH06]/. Ein zusätzlicher Aufwand besteht in der Erfüllung zahlreicher Einschränkungen für diesen gültigen Lösungsraum mithilfe von 43 mathematischen Bedingungen (*Constraints*) sowie 19 Annahmen/Gleichungen, die der Parameterverbund erfüllen muss /[Fle05a]/. Die Parameter beziehen sich allgemein auf folgende Eigenschaften des Systems:

Temporale/Dynamische Ausprägung: Der zeitliche Ablauf zur Etablierung einer Buskommunikation (*StartUp*) muss zwischen den Teilnehmern koordiniert verlaufen, um eine synchrone zeitgesteuerte verteilte Interaktion zu erzielen. Es werden die logische Startprozedur und die StartUp-autorisierten Netzwerkknoten festgelegt. Während der Kommunikation muss garantiert bleiben, dass statische vorab definierte Kommunikationsbereiche (*slots*) zuverlässig einem Teilnehmer exklusiv zum Senden zur Verfügung stehen. Die notwendige Größe dieser Slots muss je nach gewünschter Botschaftslänge errechnet werden. Die aneinandergereihten Slots ergeben ein Kommunikationsschema (*Schedule*), der gleichermaßen vorab auf eine fixe Länge eingestellt wird. In zyklischen Phasen müssen in dem Zusammenhang die Teilnehmer resynchronisiert werden, um ein arbiträres Auseinanderdriften der lokalen Steuergerätezeiten zu vermeiden. Konsequenterweise werden fixe Bereiche der Bandbreite zyklisch zur Synchronisation des Bussystems erforderlich. Je nach Systemspezifikation fallen diese Bereiche zeitlich unterschiedlich lang aus

und müssen daher busspezifisch berechnet werden.

Physikalische Ausprägung: Bei einer elektrischen Bitübertragungsschicht trägt die Busphysik aufgrund hoher Übertragungsfrequenzen einen signifikanten Beitrag zur Gesamtkomplexität des Systems bei. Einflüsse auf die Signalform müssen an jeder Stelle der Netzwerktopologie berücksichtigt werden. Dazu zählen Veränderungen in der Bitamplitude (*Dämpfung*), der Bitbreite (*asymmetrische Verzögerung*) und bei einer Signalreflexion im System. Im Systemdesign wirkt sich dieser Sachverhalt bei der Definition der Parameter zur Ausbreitungsverzögerung in den Transceivern, im Netzwerk sowie bei der Festlegung der umzusetzenden Makroticklänge aus.

Sicherheitsspezifische Ausprägung: Zu diesem Bereich zählen einerseits triviale Themen wie die Anzahl der aktiven Sternkoppler oder die Verwendung eines redundanten Übertragungskanals im System. Andererseits gibt es auch abstrakte Parameter zur Spezifikation der Weck- und Startversuche für jeden StartUp-Knoten oder zur Konfiguration des abgestuften Fehlerpropagationsmodells beim Vorliegen von Kommunikationsfehlern und dem daraus resultierenden Synchronisationsverlust im Knoten.

Der Kern des zeitgesteuerten Bussystems fundiert auf einem einfachen oder mehrstufigen Zeitkonzept. Bei FlexRay wird auf fünf Zeitebenen abstrahiert, um beliebig genau verschiedene Konfigurationsbereiche festlegen zu können (s. Abb. 3.2[1]).

Zyklus: Die gröbste Abstraktionsebene beschreibt die Zyklusebene. Dazu zählt neben der eigentlichen Zykluslänge eines Parametersatzes auch die Verwendung potentieller Zyklusiterationen in denen Dateninhalte variiert werden können(→ *Slot*).

Segment: Der FlexRay-Buszyklus lässt sich in einen Bereich für die statische Kommunikation (fixe Zeitscheibenzuordnung) und für die dynamische Kommunikation (flexible Zeitscheibenzuordnung) untergliedern. Das Symbolfenster erlangt Relevanz im Falle des Einsatzes eines Buswächters (*bus guardian*)[2] /[Fle04a]/. Über das Intervall der Netzwerkruhephase propagiert sich der Prozess der Absatzkorrektur zur Resynchronisation des FlexRay-Clusters.

Slot: Die beiden Segmente zur Standardkommunikation (statisch/dynamisch) werden in Zeitschlitze (*slots*) zerlegt. Die statische Slotlängen sind einheitlich für eine zu determinierende Nutzdatenlänge (<254Byte) definiert. Die dynamische Slotlänge der kurzen Slots des dynamischen Segments (*minislots*) ist für jeden Knoten in jedem Zyklus unabhängig definierbar. Dabei wird über einen speziellen Parameter (*pLatestTx*) der letztmögliche Sendezeitpunkt im dynamischen Segment, abhängig von der Nutzdatenlänge, festgelegt. Dadurch wird ein Sendevorgang über die dynamische Segmentlänge hinaus verhindert.

Makro: Zur Generierung einer einheitlich präzise skalierbaren Abbildung des kontinuierlichen Zeitverlaufs auf ein digitales System wird die Idee eines clusterweit identischen

[1] fokussiert auf den statischen Bereich.

[2] Der Buswächter gibt den Buszugriff eines Netzwerkknotens nur an den vorkonfigurierten Zeitintervallen im Zyklus frei und erhöht somit die Fehlertoleranz des Clusters. Da aktuell der Einsatz des Buswächters im Fahrzeugserienbereich keine Rolle spielt, bleibt dieser technische Aspekt in der Arbeit unbehandelt.

Zeitschritts (*macrotick*) angewendet. Die Länge ist dabei flexibel in einem Bereich zwischen 1 und 6 μs wählbar[3].

Mikro: Die unterste Ebene bildet die Schnittstelle zu den hardwarespezifischen kleinsten digitalen Zeitschritten (*microticks*) eines FlexRay-Knoten. Sei k ein Knoten aus einem verbundenem Cluster mit der Knotenmenge K, so gilt[4]

$$\forall k \in K : gdMacrotick_k = \frac{\sum_{i=1}^{n=pMicroPerCycle_k} * pdMicrotick_k}{gdMacrotick_K}$$
$$mit gdMacrotick_k \geq gdMacrotick_K, gdMacrotick_k \in \mathbb{R}_0^+, gdMacrotick_K \in \mathbb{N}_0^+$$

Das bedeutet, dass der nominale Makrotick eines Clusters nicht zwangsläufig mit der zeitlichen Länge des durchschnittlichen Makroticks übereinstimmt.

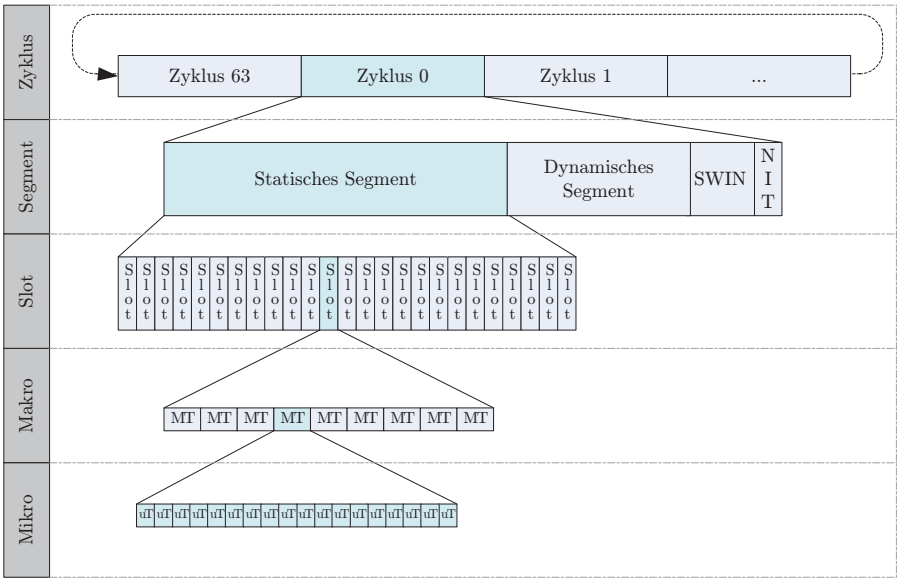


Abbildung 3.2.: Fünfstufiges Zeitkonzept der FlexRay-Technologie

Signalausbreitungsverzögerung: Der Datenversand in einem physikalischen Netzwerk führt zu einer statischen zeitlichen Verzögerung, die im Netzwerk je nach Länge des verbindenden Kommunikationsmediums zweier beliebiger Teilnehmer im Gesamtsystem variiert.

Uhrenpräzision: Durch Quarzungenauigkeiten in den verbauten Oszillatoren eines FlexRay-Steuergeräts kommt es zu dynamischen zeitlichen Schwankungen, welche die Gesamtsystempräzision verschlechtern.

[3]Der komplette Bereich erschließt sich aus den spezifischen Definitionsbereichen für verschiedene Baudraten in Abhängigkeit von der vorliegenden Taktfrequenz der verwendeten Kommunikationscontroller und ist daher nicht vollständig für jeden Bustakt valide!

[4]Die Präfixe der Notation werden konform zur FlexRay-Spezifikation [Fle05a] gehalten (p $\hat{=}$ lokaler FlexRay-Knotenparameter, g $\hat{=}$ globaler FlexRay-Parameter, d $\hat{=}$ Timing-Parameter.)

3.1.5. Grundmethoden im Entwicklungsprozess

Bei der Analyse des Zeitverhaltens softwarebasierter Echtzeitsysteme liegt das Hauptaugenmerk darauf, das konzipierte System auf temporale Korrektheit zu verifizieren /[PV97a]/. Bei der Entwicklung und Umsetzung eines zeitgesteuerten Systems beziehen sich die Echtzeitanforderungen auf die Themen *Busscheduling* zur Ablaufplanung der Netzwerkkommunikation und auf das *Knotenscheduling*. Bei dem Letzteren stehen die drei Disziplinen WCET-Analyse (*worst-case-execution-time*), Analyse der Hardwareaktivitäten und der *Schedulability-Test* als grundlegende Entwicklungsmethoden im Fokus (s. Abb. 3.3).

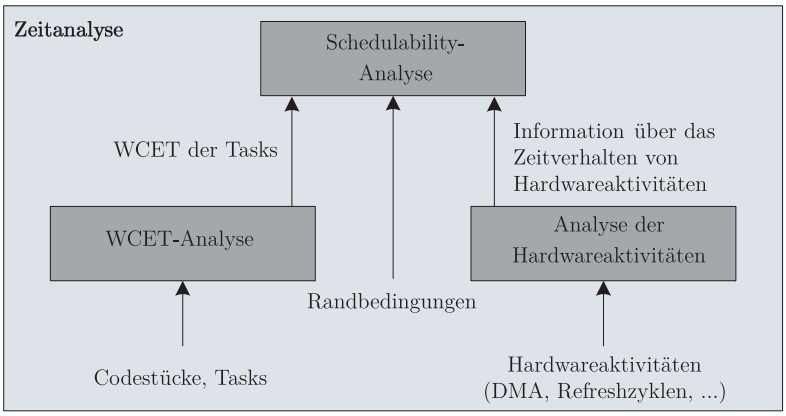


Abbildung 3.3.: Zusammenhang zwischen den drei Grunddisziplinen bei der Entwicklung zeitgesteuerter Architekturen /[Rin02]/

3.1.5.1. WCET-Analyse

Das Ziel der WCET-Analyse liegt in der Bestimmung der maximalen Ausführungszeit von Codestücken in einem Softwaresystem. Dabei müssen gewisse Randbedingungen beachtet werden, etwa dass ein Codeblock frei von Unterbrechungen ist, um die eigentliche Grundausführungszeit zu ermitteln. Die errechneten Ergebnisse lassen sich bei der Planung der Ausführungszeiten von Betriebssystemtasks eines Echtzeitbetriebssystem weiterverwenden.

Die stringenten Kostenanforderungen bei der Fahrzeugentwicklung implizieren die Notwendigkeit der exakten Berechnung der WCET für eine exakte Übertragung des Softwaresystems auf eine möglichst günstige, dass heißt ressourcenminimierte, Zielplattform. Allgemein wird in der Terminologie zwischen berechneter und aktueller BCET (*best-case-execution-time*) und WCET unterschieden /[Pus93]/. Das Intervall der realen Ausführungszeit des Codestücks liegt dabei eingebettet im Zeitintervall der realen BCET und WCET. Dieses Zeitfenster für die tatsächliche Ausführungszeit lässt sich dabei selbst in das Intervall einbetten, welches zwischen der berechneten BCET und WCET aufgespannt wird. Der Zusammenhang des Kontrollflusses zwischen einer Vielzahl an unterschiedlichen Anweisungen mit spezifischen Ausführungszeiten lässt sich dabei graphisch als Kontrollflussgraph darstellen, indem die Knoten die Anweisungen und die gerichteten Kanten den Kontrollfluss beschreiben. Die entstehenden Pfade zwischen zwei Knoten im Kontrollflussgraph führen im erweiterten Kontext zur Ausführungszeit der dort verankerten Anweisungssequenzen.

Das Konzept der WCET-Analyse lässt sich allgemein in die zwei Klassen der dynamischen und der statischen Analyse einteilen. Die dynamische Analyse beschreibt eine nichtdeterministische Methode, die mit Heuristiken gezielte Modifikationen der Eingangsparameter durchführt und die zugehörige Ausführungszeit des Programms auf der Zielplattform misst. Dadurch lässt sich, wenn auch mit stochastischen Effekten behaftet, die Ausführungsdauer des WCET-Pfades „von unten“ angenähert ermitteln /[KP96], [AHP99]/. Aufgrund dieser Näherung ist die dynamische WCET-Analyse kein zuverlässiges Mittel zur Bestimmung der WCET in sicherheitskritischen Systemen.

Komplementär zum dynamischen Ansatz wird bei der statischen WCET-Analyse die maximale Ausführungszeit unabhängig von der Zielplattform beim oder nach dem Kompilierungsvorgang theoretisch berechnet. Nach der Erfassung des WCET-Pfades lässt sich die WCET durch die Modellierung der ausführbaren Pfade, welche die Ausführungszeiten der durchschrittenen Instruktionen des Programms auf Basis der modellierten Zielhardware repräsentieren, messen /[LME98]/. Die Herausforderung bei dieser Methode liegt in der Ermittlung des WCET-Pfades und der deterministischen statischen Ausführungszeiten der jeweiligen Codestrukturen /[SAHP97], [PV97b]/. Das Vorgehen bei der Pfadmodellierung unter Berücksichtigung der Hardwareaktivitäten im Aufbau der Mikroarchitekturen der Zielplattform (*Caches*, *Pipelines*) bildet einen eigenen Forschungsbereich und wird innerhalb dieser Arbeit nicht adressiert.

Für einen detaillierten Einblick in die Konzepte der WCET-Analyse, insbesondere der Pfad- und Mikroarchitekturmodellierung wird auf /[Sha89], [PK89], [KL91], [Par92], [LME98], [FW99], /[Rin02]/ weiterverwiesen.

3.1.5.2. Schedulability-Test

Auf Basis einer WCET-Analyse folgt im nächsten Schritt eine Untersuchung, ob und in welcher Form sich die analysierten und zu Betriebssystemtasks zugeordneten Programmteile auf die zugrunde gelegte Hardwareplattform abbilden lassen. Das lokale Komponentenscheduling innerhalb eines Schedulingprozesses bezeichnet die zielgeführte Suche zur Erzeugung eines Ablaufplans für Betriebssystemkomponenten. Ein wichtiger Aspekt bezieht sich auf die Einhaltung sämtlicher Ablaufzeiten der in einem lokalen System ablaufenden Prozesse. Dafür müssen die *deadlines*, also die spätest tolerierbaren Zeitpunkte für den Abschluss der Ausführung der Tasks, in allen Fällen eingehalten werden. Gestaltungsspielraum bei der Umsetzung resultiert aus der Variabilität in der Festlegung von Anordnung, Zykluszeit und Unterbrechbarkeit einer jeden Task im Betriebssystemschedule. Der Vorgang der Suche, ob ein gültiger Schedule für eine bestimmte Konstellation an Prozessen, die sich mindestens eine gemeinsame Ressource teilen, gefunden werden kann, wird allgemein als *Schedulability-Test* /[Kop97]/ bezeichnet.

Beim *Schedulability-Test* stellt sich zuerst die Frage nach einem Algorithmus, welcher zuverlässig bestimmt, ob für eine Menge an Tasks ein gültiger Schedule existiert. Auch der komplementäre Fall ist von Relevanz, um zuverlässig bestimmen zu können, ob ein derartiger Schedule nicht generierbar ist. In /[GJ75]/ wurde nachgewiesen, dass die Komplexität für diesen Algorithmus zu der Klasse der NP-vollständigen Problemen zählt und sich somit nicht in polynomieller Zeit berechnen lässt. Daher reduziert sich die Suche auf zwei alternative Testmethoden /[Kop97]/:

Hinreichender Schedulability-Test: Falls in einem Test für bestimmte Task-Konfiguratio-

nen kein Schedule gefunden wird, obwohl theoretisch eine gültige Konfiguration möglich ist, so wird der Schedulability-Test als *hinreichend* eingestuft.

Notwendiger Schedulability-Test: Für den komplementären Fall, indem ein Test eine ungültige Task-Konfiguration fälschlicherweise als korrekt deklariert, wird der Schedulability-Test als *notwendig* bezeichnet.

Für die in einem Schedulability-Test berücksichtigten Tasks werden jeweils deren WCET in der Analyse eingesetzt. Die Ausführungsdauer e eines Tasks $t(i)$ wird $e(i)$ und die Zykluszeit eines Prozessors als $p(i)$ bezeichnet. Eine gültige Echtzeitschedulingstrategie garantiert das Einhalten sämtlicher *deadlines* einer Taskgruppe. Für die Kalkulation der Prozessorverfügbarkeit lassen sich periodisch aktivierte Tasks m leicht zur Prozessorauslastung in Relation setzen.

$$\sum_{i=1}^{m} \frac{e(i)}{p(i)} \leq Ul$$

Dieser Satz bezieht sich auf eine arbiträre Anzahl an Tasks e, die auf p Prozessoren mit einheitlichen Grundzyklen verteilt werden. Ul bezeichnet das obere Limit (*upper limit*), welches abhängig von der zugrunde gelegten Schedulingstrategie variiert. Beispielsweise lässt sich die notwendige Bedingung für die Ableitung eines gültigen Schedules aus der Summe sämtlicher zu verteilender Tasks m pro Prozessor mit $\sum_{i=1}^{m} e(i)/p(i) \leq 1$ prüfen, falls als EDF (*Earliest-Deadline-First*) Schedulingstrategie festgelegt wird.

Eine Herausforderung beim Schedulability-Test ergibt sich aus den vielschichtigen Derivaten an Scheduling-Strategien, die jeweils geprüft werden müssen. Beispielsweise basiert die RM-Strategie (*rate-monotonic*) mit einer statischen Prioritätenvergabe auf einem anderen Wert für Ul. In /[LL73]/ wird bewiesen, dass die Gleichung $Ul = m * (2^{1/m} - 1)$ ($\approx 0,69$) für den RM-Fall eine hinreichende Größe bildet, um zuverlässig zu prüfen, ob eine Menge an Tasks m in einen gültigen Schedule überführt werden kann[5].

Ähnlich zu dem Bereich der WCET-Analyse bildet die Schedulinganalyse eine eigenständigen Forschungsbereich, der sich mit der Vielzahl an Konzepteinflüssen, etwa durch statische, dynamische oder gemischte Prioritätenvergabe oder beim Umgang mit gemischt hart/weichen Echtzeitanforderungen, zu befassen hat. Für eine Vertiefung der Thematik zählen die Publikationen /[ABR+93], [TC94]/ als grundlegende Referenzen für diesen Bereich.

3.1.5.3. Busscheduling

Aus Sicht des vernetzten zeitgesteuerten Systems sind die exakten Positionen der Kommunikationsslots als Nebenbedingungen für das Design des verteilten Echtzeitsystems zu berücksichtigen, falls die partitionierten Funktionen über das Bussystem synchron miteinander interagieren sollen /[ABR+93]/. Daher wird es für den Zweck eines holistischen Schedulingansatzes erforderlich den beim Schedulability-Test gefundenen Be-

[5] Das schließt nicht aus, dass bei einem größeren Wert (>0.69) niemals ein gültiger Schedule gefunden werden kann.

triebssystemschedule der Tasks eines jeden bussynchronen Steuergeräts mit dem Busschedule des zeitgesteuerten Bussystems abzustimmen.

3.2. Stand der Technik beim Systementwurf

Wie bereits einleitend in diesem Kapitel erläutert, liegt der Ursprung des zeitgesteuerten Paradigmas schon einige Jahre zurück, was zu einer breitflächigen Forschung und Entwicklung in diesem Kontext geführt hat. Speziell der Begriff der „Synchronität" in einem System findet sich dabei stetig in den Ansätzen wieder. Ein wichtiger Meilenstein liegt in dem Ansatz CSP (*communicating sequential processes*), einer Prozessalgebra, die als Schwerpunkt auf die Interaktion kommunizierender Prozesse fokussiert. Ein Leitgedanke bezieht sich auf den nachrichtensynchronen Datenaustausch. Dabei entsteht die Nachrichtenverteilung auf Basis eines korrespondierenden Austauschs von Kommunikationsanweisungen zwischen den Prozessen. Dieses Szenario findet sich in der Form nicht im Fahrzeug wieder, da hier die Ausführung eines Prozesses und dessen Kommunikationsverhalten im Netzwerk nichtdeterministisch und unabhängig von etwaigen korrespondierenden Prozessen des verteilten Systems erfolgt.

Ein weiterer Meilenstein in der Umsetzung des zeitgesteuerten Paradigmas ist mit der Entwicklung von *Esterel* verbunden /[Ber00]/. *Esterel* definiert eine synchrone Programmiersprache, die im Kontext der Entwicklung verteilter reaktiver Systeme ihre Anwendung findet. Im Vordergrund steht der Umgang mit der Ausführung parallel ablaufender Systeme und den Aspekten präemptiver Programmierung. Eine Umsetzung der Programmiersprache liegt heute modellbasiert in einer durchgehenden Werkzeugkette vor /[Tec08]/, die hauptsächlich bei der Entwicklung von sicherheitskritischer Software im Avionikbereich eingesetzt wird.

Ein interessantes Programmiermodell für harte Realzeitsysteme wird durch die Semantik von *Giotto* /[HHK01]/ definiert. Dabei stehen die periodische Interaktionen von verteilten Tasks und deren Zustände im Vordergrund. Eine Funktion in einer Task wird periodisch ausgeführt. Es werden neben den Eingabewerten auch Taskzustände berücksichtigt und nach der Ausführung die ermittelten Ausgabewerte und neue gültige Taskzustände zurückliefert. Dabei wird analog zu einer berechneten WCET eine logische Ausführungszeit LET (*logical execution time*) zugrunde gelegt, die in ihrer zeitlichen Ausprägung die tatsächliche Ausführungszeit einer Tasks in jedem Fall übersteigt. Dadurch werden *deadline*-Verletzungen vermieden[6]. Obwohl *Giotto* als eigenständiges Programmiermodell sehr attraktiv für die Überführung in ein Entwicklungskonzept für verteilte zeitgesteuerte Echtzeitsysteme erscheint, erfordert ein Serieneinsatz im Fahrzeug ein breites ausgereiftes Portfolio für Softwarearchitekturen, Prozesse, Standards und Werkzeuge. Mit dem auf *Giotto* basierenden TDL-Konzept (s. Abs. 3.2.1.2) ist mittlerweile eine erste spezifische Implementierung unter anderem für FlexRay entwickelt worden.

Zusammenfassend lässt sich analog zu /[MG07]/ folgern, dass aktuell noch eine Lücke zwischen den meisten theoretischen Entwicklungsmethoden und deren Umsetzung in der praktischen Anwendung existiert. Dementsprechend spielen viele Ansätze aktuell nur eine eingeschränkte oder keine Rolle bei der Entwicklung verteilter zeitge-

[6]Das Problem der Ermittlung der Länge einer jeden LET einer Tasks entspricht dem analogen Fall zu den Problemstellungen bei der WCET-Analyse.

steuerter Realzeitsysteme im Fahrzeugserienbereich[7]. Eine zukünftige Herausforderung bleibt der Nachweis der Umsetzbarkeit im industriellen Kontext unter Berücksichtigung gegebener Randbedingungen (z.B. Abstraktionsgrade, Integration von Entwicklungsartefakten, Kosten, Produktlebenszyklen, Standards, Werkzeuge, etc.). Daher werden die Inhalte der FlexZOOMED-Methode in der vorliegenden Arbeit weitestgehend im engen Kontext zu den Anforderungen einer Entwicklung im Fahrzeugserienbereich umgesetzt.

3.2.1. Integrierte Entwicklungsmethoden

Der Bereich der modellbasierten Entwicklung zeitgesteuerter Architekturen wird in verschiedenen Forschungsarbeiten aus methodischer Sicht unterschiedlich behandelt. Anfolgend werden mit *DECOS*, *TDL* und *VIETTA* drei signifikante integrierte Entwicklungsmethoden aus den letzten Jahren als Referenzbeispiele dargestellt. Alternativ lässt sich in dem Bereich auch das Projekt *SETTA* /[SBV+02]/ anführen.

3.2.1.1. DECOS

Um eine möglichst flexibel einsetzbare und komponierbare Fahrzeugelektronik zur Generierung und Pflege von Fahrzeugvarianten mit verschiedenen Ausstattungsstufen zu garantieren, muss eine geeignete *Plattform* definiert werden. DECOS differenziert dabei in einer Trennung zwischen der Applikationslogik und den zugrunde liegenden Plattformtechnologien, den Hardwarecharakteristika /[HOP05]/. Die in der Hardwareplattform steckenden Systemressourcen müssen dabei hinreichend detailliert gekapselt werden, um eine Integration der Applikationsfunktionalität auf der Zielplattform zu ermöglichen. Dabei ist das Ziel eine maximale Flexibilität, Wiederverwendbarkeit und Erweiterbarkeit im Sinne einer Pflege der Applikationsmodelle zu erreichen, unabhängig von Technologieänderungen in der zugrunde liegenden Plattform. *DECOS* propagiert so genannte DAS (*Distributed Application Services*), welche gemäß des modellgetriebenen Architekturkonzepts in plattformunabhängigen (*Platform Independent Model*) und plattformabhängigen (*Platform Specific Model*) Modellen repräsentiert werden /[OMG01]/.

Die Herausforderung des verteilten System-Designs liegt in der notwendigen präzisen Spezifikationsform für den Systemdesigner, dessen Rolle in der Regel durch den Systemintegrator verkörpert wird. Da die Implementierung und die entsprechende Systemintelligenz der einzelnen Komponenten durch die jeweiligen Zulieferer bestimmt werden, muss die Spezifikation aus Sicht des Systemintegrators durch ein *Framework* unterstützt werden, welches eine Reihe an Anforderungen mit sich bringt.

Die Verknüpfung zwischen den unabhängigen und den spezifischen Modellen erfolgt per Vorwärtstransformation, die als *virtuelle Integration* beschrieben wird. Die Bezeichnung entstammt von der Idee eines *Ressource Templates*, welches die Hardware abstrakt beschreibt und dabei eine virtuelle Plattform darstellt. Das plattformunabhängige Modell bschreibt eine Dekomposition der Systemfunktionalität in *Jobs*, die über wohldefinierte Zugänge (*Ports*) an einer verbindenden Schnittstelle (*Linking Interface*) kommunizieren können. Durch die Zerlegung lassen sich auch kritische Systemteile komponieren. Deren Subsysteme können je nach Bedarf zertifiziert und validiert werden. Das plattformspezifische Modell beschreibt die Zuordnung der *Jobs* zu Partitionen und die Zuordnung des virtuellen Netzwerks zwischen den Partitionen zu einem

[7] Als weitere Beispiele lassen sich beispielsweise Ansätze mit FSM, Hybride Automaten, LSC, MSC, Petrinetze, Prozessalgebren, Statecharts, Temporallogik, gezeitete Automaten und Z nennen (vgl. /[MG07]/).

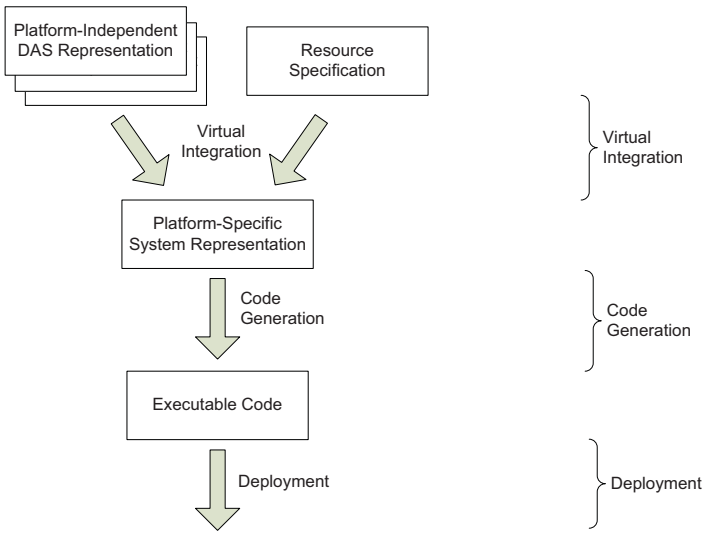


Abbildung 3.4.: DECOS System Designfluss

zeitgesteuerten Feldbussystem wie beispielsweise FlexRay /[Fle05a], [ZS06]/. Höhere Architekturdienste, die an den Schnittstellen der *Jobs* bereitgestellt werden wie z.B. *Gateways* oder Diagnoseprüfungen werden ebenfalls in dem plattformspezifischen Modell parametriert. *DECOS* beschreibt die verschiedenen Hardwarekomponenten im Rahmen von Ressourcenprimitiven (*Resource Primitives*), die im Metamodell der plattformspezifischen Lösung definiert sind.

3.2.1.2. Timing Definition Language

Die *Timing Definition Language* (TDL) /[FFPT05]/definiert einen formalen Ansatz zur abstrakten Spezifikation verteilter eingebetteter Systeme. Ziel in diesem Konzept ist die hardwareunabhängige Entwicklung eines verteilten Systems auf Basis eines logischen Modells. Dieses Modell fundiert auf der *Giotto*-Programmsemantik (s. Abs. 3.2) und richtet sich an die Entwicklung regelungstechnischer eingebetteter Systeme mit harten Echtzeitanforderungen.

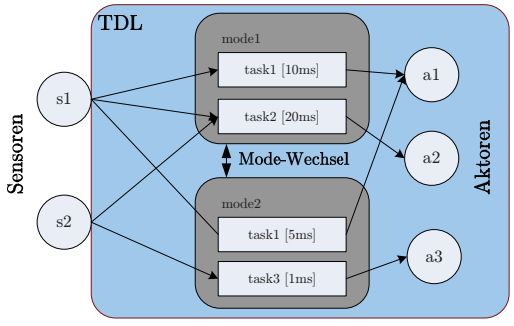


Abbildung 3.5.: Beispielhafter Aufbau einer Komponente in einem TDL-Modell /[zei08]/

Ein TDL-Modell basiert grundsätzlich auf der Idee, dass sich ein verteiltes System aus der Interaktion zwischen Sensorik, Funktionalität und Aktorik zusammensetzt, wo-

bei die Funktionalität in Abhängigkeit des Ausführungsmodus abgearbeitet wird. Ein Modus spezifiziert periodisch ausgeführte Aktivitäten, welche sich aus Taskaktivierungen, Aktualisierungen der Aktorik-Daten und Moduswechsel in den Komponenten zusammensetzen. Die Ausführung der Aktivitäten, in Form von Tasks, werden je nach Systemkondition modusabhängig ausgeführt. Gleichzeitig nehmen die Modi Einfluss auf die Taskausführungszeiten (s. Abb. 3.5) und werden hardwareunabhängig spezifiziert. Dieser Ansatz basiert auf der Idee einer logischen Ausführungszeit LET (*logical execution time*). Dabei wird analog zu einer berechneten WCET eine logische Zeit zugrunde gelegt, die in ihrer zeitlichen Ausprägung die tatsächliche Ausführungszeit einer Task in jedem Fall übersteigt. Dadurch lassen sich *deadline*-Verletzungen vermeiden.

Der TDL-Ansatz ist mittlerweile als *Plug-In*-Lösung in die etablierten Werkzeuge Matlab/Simulink /[The07a]/ integriert worden, um die Regelsystementwicklung direkt unterstützen zu können. Auf Basis generierter Artefakte (Code, Konfigurationen) lassen sich hardware- oder feldbusspezifische Parameter des Systems in einer nachgelagerten Werkzeugkette konfigurieren.

3.2.1.3. VIETTA

/[Rin02]/ beschäftigt sich anschaulich mit technisch notwendigen Adaptionen des Entwicklungsprozesses zur Unterstützung zeitgesteuerter Systeme im Bereich automotiver Anwendungen. Speziell die Analyse der maximalen Ausführungszeiten von Softwareprozessen, deren Verteilung und zeitliche Ablaufplanung innerhalb der beteiligten Steuergeräte werden intensiv untersucht. Parallel werden die Schnittstellen zur Anwendungsentwicklung mit klassischen modellbasierten Softwareentwicklungswerkzeugen /[Ber00], [The07a]/ und eine Einbettung des Gesamtkonzepts in den Entwicklungsprozess beschrieben. /[Rin02]/ abstrahiert in seiner Arbeit von der konkreten Auslegung einer flexiblen zeitgesteuerten Bustechnologie und deren Parametrierung. Der Fokus könnte ergänzend von der exakten Spezifikation des Zeitverhaltens der Softwareprozesse auf das Zeitkonzept der Signalkommunikation auf dem Bus erweitert werden. Weiterhin bleiben firmenspezifische Abläufe innerhalb der Fahrzeugserienentwicklung und deren nicht-/funktionale Anforderungen, beispielsweise der Entwurf der technischen Bordnetzarchitektur mit Plattform- und Variantenanforderungen, unberücksichtigt.

Im Vergleich dazu sind Teile der in /[Bro05]/ betrachteten Umfänge auf Basis einer kommerziellen Werkzeugkette entstanden, die konzeptuell in Analogie zu den Ansätzen von /[Rin02]/ stehen. Aus heutiger Sicht lässt sich bestätigen, dass sich dieser Ansatz nicht durchgesetzt hat, da eine Abbildung auf die dezentrale Organisationsform des Fahrzeugherstellers in Zusammenarbeit mit mehreren Zulieferern bisher nicht definiert worden ist.

3.2.2. Verifikation zeitgesteuerter Systeme

Neben den entwurfsorientierten Ansätzen existiert mit /[Pik06]/ eine verwandte Arbeit im Bereich der formalen Verifikation zeitgesteuerter Systeme. Dabei liegt der Schwerpunkt gesamtheitlichen auf der Klasse der synchronisierten fehlertoleranten Steuerungs- und Kommunikationsarchitekturen. Auf Basis der partiell synchronen Systeme, einer Erweiterung der abstrakten ungezeiteten synchronen Systeme um zeitgesteuerte Paradigmen, werden Annahmen mithilfe des mechanischen Theorembeweisens verifiziert. Dabei werden die realisierten Buskommunikationsabläufe mit der Kombination von

begrenzten Modellprüfungen (*model-checking*) und dem automatisierten Lösen auf notwendige zeitgesteuerte Annahmen geprüft. Grundlage der Arbeit ist die bei der NASA entwickelte *SPIDER Fly-by-wire*-Busarchitektur /[NAS07]/.

3.2.3. Parametrierungsstrategien

Aktuell fehlen weiterführende Veröffentlichungen zur methodischen Entwicklung von FlexRay-Systemkonfigurationen. Daher werden als Referenz zwei verfügbare unterschiedliche Beschreibungen zur Entwicklung eines FlexRay-Systemdesigns betrachtet, die einerseits in Form einer Werkzeugdokumentation (*Variante 1*) und andererseits als Abschnitt in dem einzigen verfügbaren FlexRay-Grundlagenbuch /[Rau07b]/ (*Variante 2*) vorliegen.

Variante 1: In /[TTA07]/, einem dedizierten FlexRay-Designwerkzeug, erfolgt die Systemkonfiguration in einem Zwei-Level-Ansatz, was einer Trennung von Netzwerk- und Knotendesign entspricht. Dabei konzentriert sich die Definition des FlexRay-Parametersatzes weitestgehend auf die Definition der Buszykluslänge, der Anzahl an statischen Slots, der statischen Slotlänge und der Nutzdatenlänge eines statischen Slots:

1. gdCycle
2. gNumberOfStaticSlots
3. gdStaticSlot
4. gStaticSlotPayload

In diesem Zusammenhang wird eine Erweiterung des Konzepts hinsichtlich einer dritten mittleren Abstraktionsebene zur Gruppierung von Signalen in PDUs auf Schedulingebene unterstützt. Dies entspricht den Anforderungen des *AUTOSAR*-Standards /[AUT07a]/. Dadurch wird von der Signalebene im Netzwerkdesign abstrahiert.

Variante 2: In folgender Tab. 3.1 wird der Ansatz von /[Rau07b]/ dargestellt, der zu einer pragmatischen Konfiguration eines FlexRay-Systems führt. Allerdings setzt dieses Verfahren einige Annahmen wie die Buszykluslänge, die Nutzdatenlänge oder die Dämpfungswerte in der Uhrensynchronisation voraus, deren Herkunft nicht plausibilisiert wird.

3.2.4. Methodenvergleich

Während des Erstellungszeitraums dieser Arbeit sind drei kommerzielle Designentwicklungswerkzeuge für die FlexRay-Systementwicklung am Markt verfügbar. Nachfolgend wird der Funktionsumfang dieser Werkzeuge im Abgleich mit den Ansätzen wissenschaftlicher Entwicklungsmethoden für zeitgesteuerte Bussysteme im Kontext einer systematischen Architekturentwicklung analysiert.

Im Bereich der logischen Architekturentwicklung interessiert die Möglichkeit der funktionalen Spezifikation einer Fahrzeugarchitektur über Funktionsmodule mit Schnittstellen zum Datenaustausch. Dabei ist es von Relevanz, ob den Modellen ein geeignetes

Parameter	Herkunft	Ableitung
Baudrate	Keine Vorgabe	gdBit, gdSampletick, gdCASRxLowMax, gdWUPSymbolRxIdle, gdWUPSymbolRxLow, gdWUPSymbolRxWindow, gdWUPSymbolTxIdle, gdWUPSymbolTxLow
Controllerfrequenz	Hardware	pdMicrotick, gdMacrotick, adprecision
gdCycle	Keine Vorgabe	pMicroPerCycle, gMacroPerCycle
gdMacrotick	Keine Vorgabe	pMicroPerCycle, gMacroPerCycle
cClockDeviationMax	Keine Vorgabe	pMicroPerCycle, gMacroPerCycle
gNumberOfStaticSlots	Keine Vorgabe	Keine Ableitung
gPayloadLengthStatic	Keine Vorgabe	Keine Ableitung
gPayloadLengthDynamicMax	Keine Vorgabe	Keine Ableitung
gSymbolWindow	je nach Einsatz	Keine Ableitung
gdMinPropagationDelay	Netzwerktopologie	adprecision, pDelayCompensation
gdMaxPropagationDelay	dCableDelay, dBDRx10, dBDTx10, dStarDelay	adprecision, pDelayCompensation
pdMaxDriftDamping	Keine Vorgabe	adprecision
gdMaxDriftDamping	Keine Vorgabe	adprecision, gdActionPointOffset, gdMinislotActionPointOffset, SafetyMargin
dBDRxia	Keine Vorgabe	gdTSSTransmitter
dStarTruncation	Hardware, cClockDeviationMax	gdTSSTransmitter
gdTSSTransmitter	Keine Vorgabe	pDecodingCorrection
gdStaticSlot	gPayloadLengthStatic, gdActionPointOffset, gdTSSTransmitter	Keine Ableitung
gdMiniSlot	gdMinislotActionPointOffset, adprecision, gdMaxPropagationDelay, gdTSSTransmitter	Keine Ableitung
gdDynamicSlotIdlePhase	gdMiniSlot, adprecision, gdMaxPropagationDelay	Keine Ableitung
pOffsetCorrectionOut	adPrecision	Keine Ableitung
pRateCorrectionOut	gdCycle, cclockdeviationmax	Keine Ableitung
pExternOffsetCorrection	Keine Vorgabe	Keine Ableitung
pExternRateCorrection	Keine Vorgabe	Keine Ableitung
gdNIT	pOffsetCorrectionOut, gNumberOfDynamicSlots, gdCycle, gNumberOfStaticSlots	gOffsetCorrectionStart
gMacroInitialOffset[A/B]	gdTSSTransmitter	Keine Ableitung
gMicroInitialOffset[A/B]		Keine Ableitung
pdListenTimeout	pdMaxDrift, pdAcceptedStartupRange	Keine Ableitung
gListenNoise	Keine Vorgabe	Keine Ableitung
gColdStartAttempts	Keine Vorgabe	Keine Ableitung
gNetworkManagementVectorLength	Keine Vorgabe	Keine Ableitung
gMaxWithoutClockCorrectionPassive	Keine Vorgabe	Keine Ableitung
gMaxWithoutClockCorrectionFatal	Keine Vorgabe	Keine Ableitung
pAllowActiveToPassive	Keine Vorgabe	Keine Ableitung

Tabelle 3.1.: Tabellarische Auflistung für den Prozess der Herleitung einer FlexRay-Konfiguration

Zeitmodell zugrunde liegt, welches für weiterverarbeitende Prozessschritte, beispielsweise zum Scheduling, herangezogen wird. Unter der technischen Architekturspezifikation zählt die konkrete Erfassung der Netzwerktopologie sowie der Steuergerätekomponenten in bedateten Modellen, um integrierte Aussagen über Verkabelungsgrößen, Verkabelungsart oder Hardwareeigenschaften abzuleiten. Speziell beim FlexRay-Bussystem stellt der Parametrierungsprozess eine eigenständige komplexe Entwicklungsdisziplin dar. Dabei stellt sich die Frage nach einer vollständigen Abbildbarkeit des Parametersatzes im Entwicklungskonzept und ob zusätzliche Mittel zur Strukturierung des Parametermodells vorhanden sind. Um einen ausgereiften Parametrierungsprozess durchzuführen, spielt die Einbettung der logischen und technischen Architektur und deren Varianten eine wesentliche Rolle.

Zu den Analysemöglichkeiten zählt bei der statischen Analyse die Modellverifikation mithilfe statischer Prüfmethoden (Metriken, Konsistenzprüfungen, etc.) und bei der dynamischen Analyse die Simulation. Die Unterstützung hervorgehobener Methoden (WCET-Analyse, Schedulability-Test) bei der Entwicklung verteilter zeitgesteuerter Systeme steht dabei im Vordergrund. Zusätzlich stellt sich die Frage nach einem erhöhten Automatisierungsgrad während der Entwicklung. Dazu zählen Optimierungstechniken und Möglichkeiten zur Generierung von Netzwerkschedules.

In Tab. 3.2 werden fünf verschiedene relevante Ansätze aus der Forschung und aus dem kommerziellen Bereich in Bezug auf die genannten Leistungsmerkmale direkt miteinander verglichen. Die Inhalte der Entwicklungsansätze sind in Abs. 3.2 aufgeführt.

	Methode					
Eigenschaften	DECOS	VIETTA	TDL	DaVinci	DesignerPro	TTX
	Forschung		Kommerziell			
Logische Architektur	+	+	+	-	-	-
Zeitmodell	+	+	+	-	-	-
Funktionsnetze	+	+	+	-	-	-
Technische Architektur	o	o	o	-	o	-
Komponenten	+	-	+	-	+	-
Topologien	-	o	-	-	-	-
Parametrierung	-	-	o	o	o	+
Vollständigkeit	-	-	+	+	+	+
Strukturiert	-	-	-	-	-	+
Varianten	-	-	-	-	-	-
Analyse	o	+	+	+	o	o
dynamisch	o	+	+	-	-	-
statisch	+	+	+	+	+	+
Schedulability	o	+	-	-	-	+
WCET-Analyse	-	+	-	-	-	-
Automatisierung	o	o	o	-	-	o
Optimierung	o	-	o	-	-	-
Scheduling	+	o	+	-	-	+

Tabelle 3.2.: Übersicht zum Umfang der Entwicklungsmethodiken von etablierten entwickelten Lösungen im Bereich der zeitgesteuerten Architekturen

3.3. Konzepte des flexiblen zeitgesteuerten Bussystems FlexRay

Das Hochgeschwindigkeitskommunikationssystem FlexRay stellt in dieser Arbeit die Referenztechnologie für das zentral behandelte Thema der flexiblen zeitgesteuerten Architekturen im Fahrzeugserienbereich. Die für das Verständnis der nachfolgenden Kapitel notwendigen Aspekte von FlexRay werden daher kurz beschrieben.

3.3.1. Übersicht

Im Jahre 1999 hat sich eine herstellerübergreifende Initiative mit dem Ziel der Spezifikation einer neuen Feldbustechnologie gebildet, um den zukünftig erwarteten gesteigerten Anforderungen an E/E-Systeme in Serienfahrzeugen gerecht zu werden. Folgende Eigenschaften stehen dabei im Vordergrund:

Performanz: Für die Verknüpfung kommunikationsintensiver verteilter Fahrzeugfunktionen soll eine Bandbreitenerhöhung des Netzwerks um Faktor 10 den Bedarf für die Umsetzung eines gesteigerten Datenaufkommens im Fahrzeug decken.

Erweiterbarkeit: Systemintegrationen innerhalb des Netzwerks sollen funktional unabhängig komponierbar durchgeführt werden. Speziell eine Interferenz der Buszugriffszeiten bei mehreren Netzwerkteilnehmern, die verstärkt bei hoher Busauslastung auftreten, muss ausgeschlossen werden.

Verlässlichkeit: Die funktionale Sicherheit der im Netzwerk zu applizierenden Funktionen muss je nach Anforderung gewährleistet bleiben. Ein skalierbar einsetzbares Konzept geeigneter Fehlertoleranzmaßnahmen spielt in diesem Zusammenhang eine entscheidende Rolle.

Flexibilität: Die drei vorangegangenen Eigenschaften dürfen je nach Einsatzszenario anwendungsspezifisch konfektioniert werden, um die individuellen Anforderungen des Fahrzeugherstellers erfüllen zu können.

Präzision: Durch ein deterministisches Paradigma können verteilte Systeme präzise entwickelt und aufeinander abgestimmt werden. Die Momentaufnahme eines Systemzustands (*snapshot*), die sich in verteilten Komponenten synchron verarbeiten lässt, unterstützt die Entwicklung hochperformanter verteilter Regelungen unter Verwendung eines zwischengeschalteten Bussystems.

3.3.2. Physikalische Bitübertragungsschicht

FlexRay stellt prinzipiell ein von der physikalischen Übertragungsart unabhängiges Feldbussystem dar. Aufgrund der Anforderungen einer maximaler Bandbreitenbereitstellung und kosteneffizienter Produktion des Fahrzeugherstellers basiert die erste Generation der FlexRay-Technologie auf einem elektrischen Übertragungsmedium, das vorzugsweise mit 10Mbit/s betrieben wird.

Aus Sicht eines Fahrzeugherstellers bringen Netzwerktechnologien mit elektrischen Übertragungsmedien eine Reihe von Anforderungen mit, die es im Fahrzeug zu beherrschen gilt. Grund dafür sind eine Reihe von Faktoren, die zu einer stabilen Serienentwicklung beitragen:

- Flexibilität in der Anordnung und dem vorgesehenen Verbauort für Steuergeräte im Fahrzeug,
- Geringe Anzahl an Bauteilvarianten (Hardwarebestückoptionen eines Steuergeräts),
- Zuverlässiger Signalaustausch innerhalb der geforderten maximalen Systemlebensdauer (Robustheit, Stabilität, Fehlertoleranz),
- Geringe störende Wechselwirkungen mit anderen Systemen (Elektromagnetische Störein- und -abstrahlung),
- Austauschbarkeit von Hardwarekomponenten (Veränderungen des Hardwarelayouts),
- Energieeffiziente Betriebsmodi (Weckbarkeit, Schlafmodus),
- Leichte Integrationsmöglichkeit weiterer Steuergeräte.

In der E/E-Entwicklung fliesst ein entsprechend großer Aufwand in die Analyse der Komponenten und deren Wechselwirkung mit umliegenden Bauteilen. Um eine korrekte Funktionsweise des Netzwerks zu gewährleisten, müssen die Einflüsse sämtlicher Bauteile auf die physikalische Qualität der Signaldarstellung des Busprotokolls beherrscht werden. FlexRay erfordert aufgrund seiner hochfrequenten Übertragungstechnik besondere Beachtung im Bezug auf die Abmessungen der für das Fahrzeug vorgesehenen Netzwerktopologien. Die Prämisse liegt auf einer verlässlichen Übertragung eines Bits zwischen allen möglichen Kommunikationsverbindungen im Netzwerk. Der Umgang mit den physikalischen Parametern wird als Ergänzung separat von der Spezifikation der physikalischen Bitübertragungsschicht behandelt /[Fle06b]/.

3.3.2.1. Topologien

Die Anordnung der physikalischen Netzwerkstruktur ermöglicht eine anwendungsorientierte Konfiguration des Feldbussystems. Dies erfolgt über den Einsatz unterschiedlicher Topologievarianten (s. Abb. 3.6):

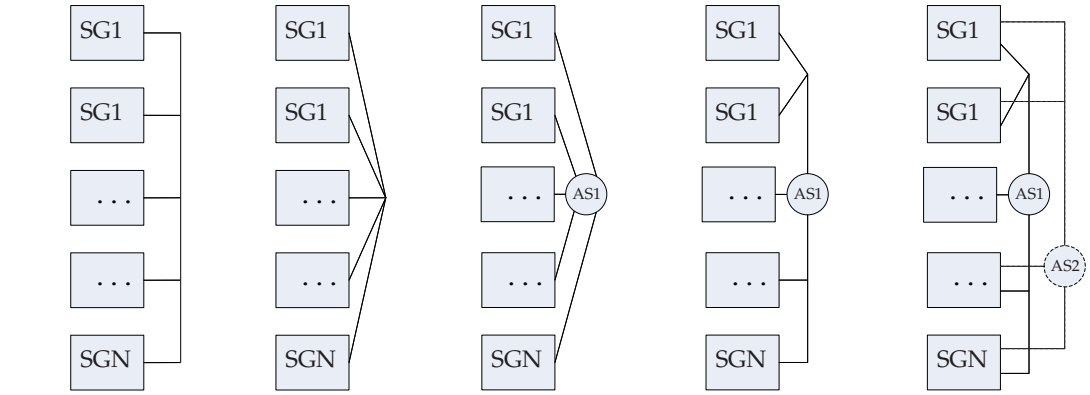


Abbildung 3.6.: Netzwerktopologievarianten für FlexRay-Architekturen

Passiver linearer Bus: Die Basisbusstruktur entsteht durch die Anbindung der einzelnen Netzwerkteilnehmer per Stichleitungen (*Stubs*) an einen längeren Übertragungskanal, der als gemeinsamer Bus die Knoten verbindet. Die Längen der Stichleitungen, deren Abstände auf dem Bus und die Maximallänge des Busses sind zu beachten.

Passiver Stern:

Bei der passiven Sterntopologie werden die Busteilnehmer per Spleißverfahren in einem Punkt des Netzwerks physikalisch verknüpft. Dabei werden die Übertragungskabel im Bordnetz ohne zwischengeschaltete physikalische Terminierung in einer Komponente zusammengeführt. Als Konsequenz erhöht sich das Potential für Signalreflexionen im Übertragungsmedium, was je nach Ausprägung zu einem Risiko für die fehlerfreie Signaldarstellung auf dem Bus werden kann.

Aktiver Stern:

Um die kostenrelevanten Anforderungen an den Kabeltyp im Netzwerk zu reduzieren eignen sich aktive Komponenten zur Verknüpfung mehrerer Netzwerkabschnitte. Dazu zählen die aktiven Sternkoppler (s. Abb. 3.7).
Diese Komponenten können ersatzweise zur passiven Sterntopologie im Netzwerk verbaut werden, um die Netzwerkteilnehmer zu verknüpfen. Dabei werden die empfangenen Signale jedes Sternzweigs auf alle restlichen angeschlossenen Sternzweige gespiegelt. Ein Vorteil besteht in dem internen Logikbaustein des aktiven Sternkopplers, der das auf dem Bus anliegende Differenzbussignal in ein im Wertebereich diskretes (digitales) Datensignal wandelt. Dadurch ergibt sich die Möglichkeit zum gezielten Ausgleich

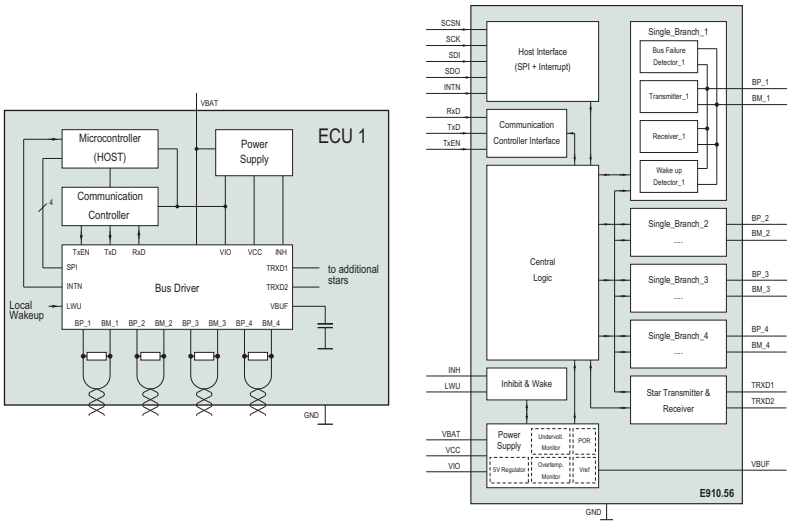


Abbildung 3.7.: Blockdiagramm zum internen Aufbau eines Sternkopplers und Beispiel für dessen Applikation auf einem Steuergerät /[ELM07]/

der verzerrenden Effekte bei der Signalausbreitung im Netzwerk wie Signaldämpfungen oder -rauschen. Aktive Sterne werden daher je nach Systemkonfiguration bei bestimmten Verkabelungsabständen ($d > 24m$) der Netzwerkteilnehmer erforderlich /[Fle04b]/. Je nach Aufbau wird technisch zwischen einem monolithischen (integrierter Baustein) und einem diskreten (Verknüpfung von Einzelbausteinen) aktiven Sternkoppler unterschieden. Dieser Unterschied beeinflusst entsprechende Effekte im Zeitbereich der Signalweiterleitung.

Der Einsatz eines aktiven Sternkopplers erfordert eine frühzeitige Beachtung im Systementwurf bei der Systemparametrierung. Durch die aktiven Eigenschaften entsteht beim initialen Aufbau der Kommunikationsverbindung eine Verkürzung des eingehenden Bussignals an den Ein- und Ausgängen im Sternkoppler aufgrund von internen Reaktionszeiten. Weiterhin führt die Überbrückung des Bussignals zu einer kontinuierlichen Verzögerung. Signalverkürzung und -verzögerung bilden elementare Bestandteile einer FlexRay-Systemparametrierung.

Einkanaliges Hybrid:

Grundsätzlich lassen sich die drei Grundtopologien linearer passiver Bus, aktiver Stern und passiver Stern miteinander kombinieren, wobei grundsätzliche Regeln über Anzahl und Anordnung der Grundtopologieformen berücksichtigt bleiben muss /[Rau07b], [Fle04b]/. Starken Einfluss haben passive Sterntopologien auf die Signalausprägung im Zeitbereich. Bisherige Analysen schließen daher passive Sterne in einkanaligen Hybridtopologien aus /[Fle06a]/.

Zweikanaliges Hybrid:

Zweikanalige Hybridtopologien entstammen dem Wunsch nach skalierbarer Fehlertoleranz im Sinne der Erhöhung der Netzwerkzuverlässigkeit. Alternativ könnte anstatt einer redundanten Datenübertragung eine beliebige Netzwerkbedatung auf dem zwei-

ten Kanal im Hybrid übertragen werden. Aufgrund technischer Einschränkungen, beispielsweise der limitierten Hardwarepufferanzahl in jedem Kommunikationscontroller, die für beide Kanäle aufzuteilen sind, verliert die unterschiedliche Bedatung der Netzwerkkanäle an Attraktivität[8].

3.3.2.2. Signaldefinition

Die Datenübertragung zwischen zwei Bustreibern basiert auf einem zeit- und wertkontinuierlichen Signal. Dieses wird als Differenzsignal aus den beiden Spannungswerten der zwei FlexRay-Kabeladern pro Kanal U_{BP} und U_{BM} gewonnen. Signifikant ist Varianz der Signalausprägung an unterschiedlichen Stellen des Netzwerks. Wesentlich dafür sind die Werte direkt am Sendetransceiver (*TP1*) und am Empfangstransceiver (*TP4*). Dazwischen lassen sich die Werte zusätzlich an den Sende- und Empfangpins der Stecker der zwei in dieser Übertragung beteiligten Steuergeräte messen (*TP2* und *TP3*). Eine gültige Signalübertragung zur Darstellung eines logischen Bitwerts liegt absolut betrachtet an der Stelle *TP1* stets im Bereich zwischen $|600mV - 2000mV|$ und bei *TP4* im Bereich zwischen $|300mV - 2000mV|$.

Signalkollisionen treten protokollbedingt im asynchronen Zustand bei der Initialisierung eines FlexRay-Systems auf. Zur Verhinderung von Datenverlusten werden in diesen kritischen Phasen drei Bussymbole[9] (*WakeUp*, *MediaTest* und *CollisionAvoidance*) eingesetzt und ohne Nutzdaten übertragen werden. Da diese Signale alle durch den dominanten Datenwert *Data_0* ausgedrückt werden, führt eine Kollision in diesem Bereich zu zeitlichen Verlängerungen der angenommenen Übertragungszeiten von *Data_0* des jeweiligen Symbols. Diese Effekte müssen bei der Konfiguration berücksichtigt werden.

3.3.2.3. Signalübertragung

Bei der Signalpropagation im Netzwerk wirken physikalische Effekte, die im Sinne einer zuverlässigen Funktionstüchtigkeit zu berücksichtigen sind. Daraus lassen sich Annahmen ableiten, die vorab eines Systemeinsatzes im Design eingehalten bleiben müssen.

Asymmetrische Ausbreitungsverzögerung: Eine FlexRay-Transceiverarchitektur für ein elektrisches Übertragungsmedium erzeugt ein leicht asymmetrisches Signal auf der Senderseite (s. Abb. 3.8). Daher existieren unterschiedliche Anstiegs- und Abfallzeiten für die Signalflanken beim Wechsel von "´*0*“ auf "´*1*“ und umgekehrt.

$$dBusTx01 \neq dBusTx10$$

Zusätzlich erzeugen zwei verschiedene Treibermechanismen in der Signalerzeugung eine Unterbrechung in den Signalflanken, die in die Berechnung der Anstiegs- und Fallzeiten einfliesst. Auf der Empfängerseite ergeben sich unterschiedliche Ausprägungen der Anstiegs- und Fallzeiten des Bussignals und des digitalen Signals am Decoderpin

[8] Aktuell spielen zweikanalige Topologievarianten, aufgrund teurer Verkabelungsaufwände bei derzeitig eingeschränkten technischen Anwendungsmöglichkeiten, keine Rolle in der Fahrzeugserienentwicklung. Daher werden diese Topologievarianten in der vorliegenden Arbeit nicht weiter berücksichtigt.

[9] Die Bussymbole bilden Steuersignale und bestehen aus clusterweit einheitlich definierten Mustern an *Data_0*- und *Data_1*-Sequenzen.

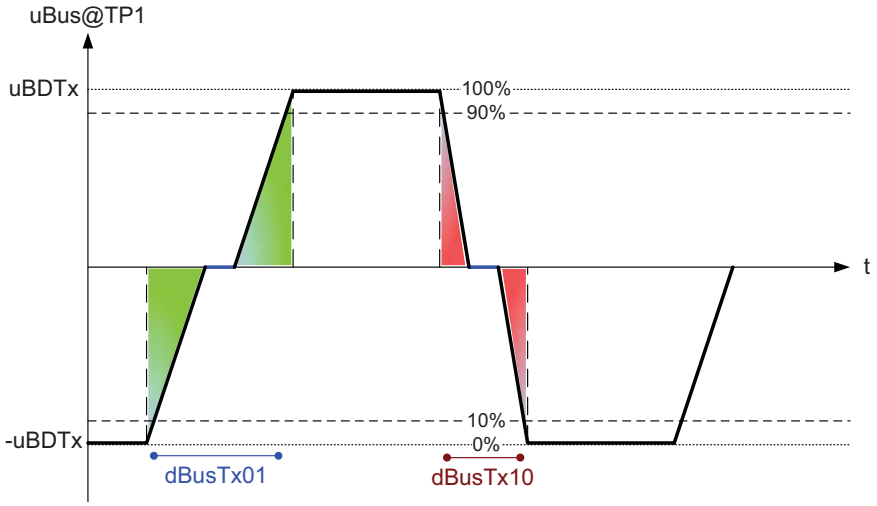


Abbildung 3.8.: Asymmetrische Verzögerung am sendenden FlexRay-Bustreiber

RxD eines FlexRay-Kommunikationscontrollers (s. Abb. 3.9). Verstärkt wird dieser Effekt durch Hysteresetoleranzen auf dem *RxD*-Ausgang des FlexRay-Transceivers.

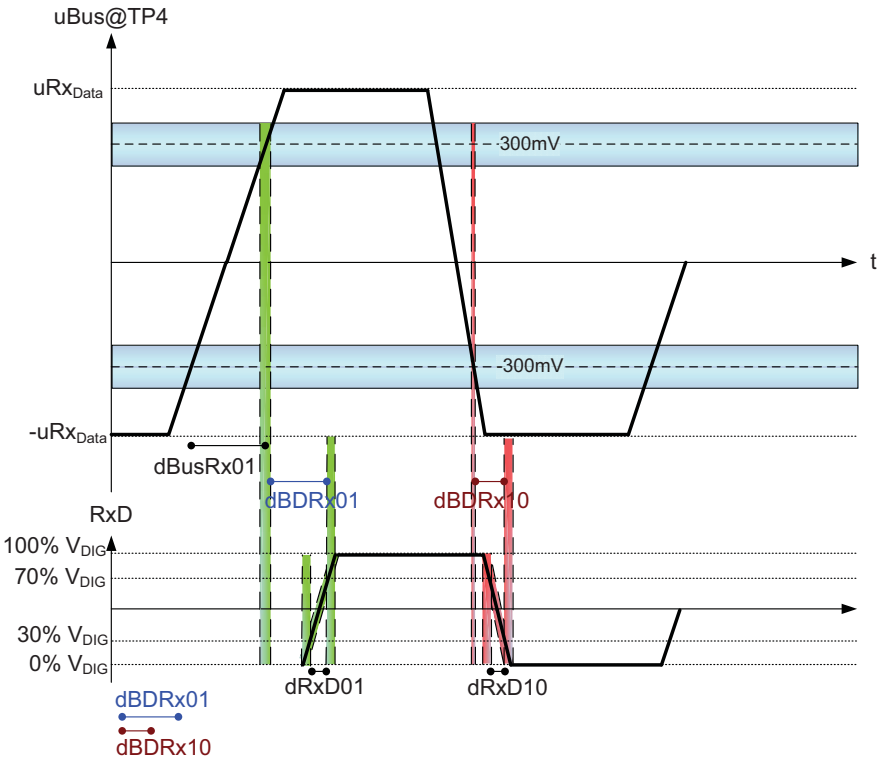


Abbildung 3.9.: Asymmetrische Verzögerung am empfangenden FlexRay-Bustreiber

$$dBusRx01 \neq dBusRx10 \neq dRxD10 \neq dRx01$$

Sendeverkürzung: Das Setzen von Eingabe- und Ausgabeverbindungen innerhalb eines aktiven Sterns und die Aktivitätserkennung des Busses am FlexRay-Receiver benötigen physikalische Zeit während der Datenübertragung. Dadurch ergibt sich eine Verkürzung des übertragenen Bussignals um mehrere Bitlängen am Signalanfang (s. Abb. 3.10). Dieser Effekt wird durch eine spezifisch addierte Startsequenz *TSS* (*Transmission Start Sequence*) kompensiert. Die zum Übertragungsstart zeitliche Ausprägung der *TSS* d_{TSS} abzüglich des kummulativen Effekts der Signalverkürzung entspricht:

$$1 \leq d_{TSS} \leq (gdTSSTransmitter + 1)\,[Bit]$$

EMV: Elektromagnetische Einflüsse auf der elektrischen Bitübertragungsschicht des Bussystems verstärken je nach Applikation einer Topologie die verschiedenen physikalischen Effekte bei der Signalübertragung im Fahrzeug.

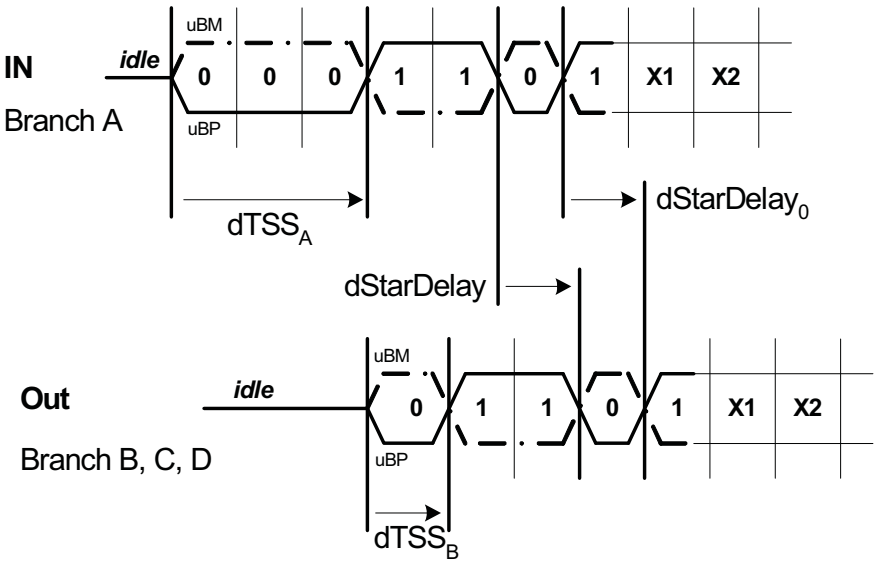


Abbildung 3.10.: Sendesignalverkürzung im FlexRay-Sternkoppler /[Fle04b]/

Glitches: Im Bereich der Signalübertragung können durch kurzeitige physikalische Störungen (bspw. EMV-Einstrahlungen) temporäre Verfälschungen des Signalwerts entstehen. Bei elektronischen Komponenten werden *Glitches* auch durch abweichende Signallaufzeiten in Schaltungen verursacht, welche boolesche Berechnungsformeln verfälschen (*race conditions*). Lokal begrenzte *Glitches* lassen sich in FlexRay mithilfe des Mehrheitsvotierungsverfahren bei der Dekodierung eines Bits ausgleichen.

Signalflanken/-pegel: Bei der Signalausprägung interessieren vorrangig die Latenzen beim Wechsel des Signalpegels. Dies ist entscheidend, da durch die asymmetrische Signalverzögerung eine Verkürzung der Signalflanken erzeugt wird. Die Auswirkungen müssen durch schnelle Wechselzeiten des Signalpegels bei der physikalischen Bitdarstellung beschränkt werden.

Signaldämpfung: Die Spezifikation des physikalischen Bordnetzes führt je nach Topologie- und Leitungseigenschaften zu Signaldämpfungen. Während des Systemdesigns muss dabei beachtet werden, dass kritische Schwellwerte des Spannungspegels für die Darstellung des logischen Signalwerts nicht unter- oder überschritten werden dürfen.

3.3.2.4. Hardwarekomponenten

Allgemein lassen sich die Komponenten eines FlexRay-Clusters in die Bereiche der Ausführungseinheiten, der Busanbindung und des Übertragungskanals untergliedern. Die Ausführungseinheit verkörpert die Komponente, welche die Protokollmaschine implementiert. Dabei wird zwischen integrierten Lösungen, etwa dedizierte Mikrocontroller mit eingebetteten FlexRay-Logikbausteinen, und eigenständigen Protokoll-Controllern (*Stand-Alone*-CC) unterschieden. Während die eingebetteten Chips vorrangig als ASICs implementiert sind, bieten sich bei den *Stand-Alone*-CCs auch FPGA-Varianten an. Bei der Systemanbindung interessiert die Art und der Aufbau des Transceivers (Standard/-Stern, diskret/monolithisch) sowie sämtliche Bauteile, welche Einfluss auf die Signalausprägung nehmen (Terminierungen, Drosseln, Kondensatoren). Der Übertragungskanal basiert auf der Spezifikation des Bordnetzes (Topologien, Kabelsegmente, Dämpfungseigenschaften, Verzögerungseigenschaften).

3.3.3. Protokollschicht

Die Protokollschicht beinhaltet die grundlegenden Operationen, die zur Ausführung der stabilen zeitgesteuerten Kommunikation notwendig sind. Dazu zählt die Systemkonfiguration, das Weck- und Aufstartverhalten des Clusters, die Kodierung und Dekodierung des Netzwerksignals, die zyklische Uhrensynchronisation und das Verhalten im Fehlerfall. Der Kern zur Koordination sämtlicher Operationen erfolgt in der Protokollablaufkontrolle, deren Konzept zusammen mit den wichtigsten Teilfunktionen kurz erläutert wird.

3.3.3.1. Protokollausführungskontrolle

Die Protokollausführungskontrolle POC (s. Abb. 3.11) lässt sich in Form eines endlichen Automaten darstellen. Die Protokollalgorithmen werden dabei aus der Perspektive des prozeduralen Ablaufs in vier verschiedene Verhaltensklassen eingeteilt:

1. Verhalten zur Überführung der POC in einen startfähigen Zustand (*READY*). Nach dem Anliegen der Mindestspannung wird der Zustand *DEFAULT CONFIG* betreten und die POC initialisiert. Im Zustand *CONFIG* wird diese Ausgangskonfiguration verifiziert. Mit einer verifizierten Konfiguration werden die Kernprozesse initialisiert und der Zustand *READY* erreicht.

2. Verhalten, das von dem *READY*-Zustand nach *NORMAL ACTIVE* führt. In diesem Schritt wird der *StartUp* eines Clusters oder die Integration in ein bereits aktives Cluster ausgeführt. Für den Fall eines weckfähigen Clusters kann auch eine entsprechende *WakeUp*-Prozedur angestossen werden.

3. Verhalten, das bei Erreichen des normalen Operationszustandes gegeben ist (*NORMAL ACTIVE*). Befehle und Statusinformationen werden zwischen dem Kom-

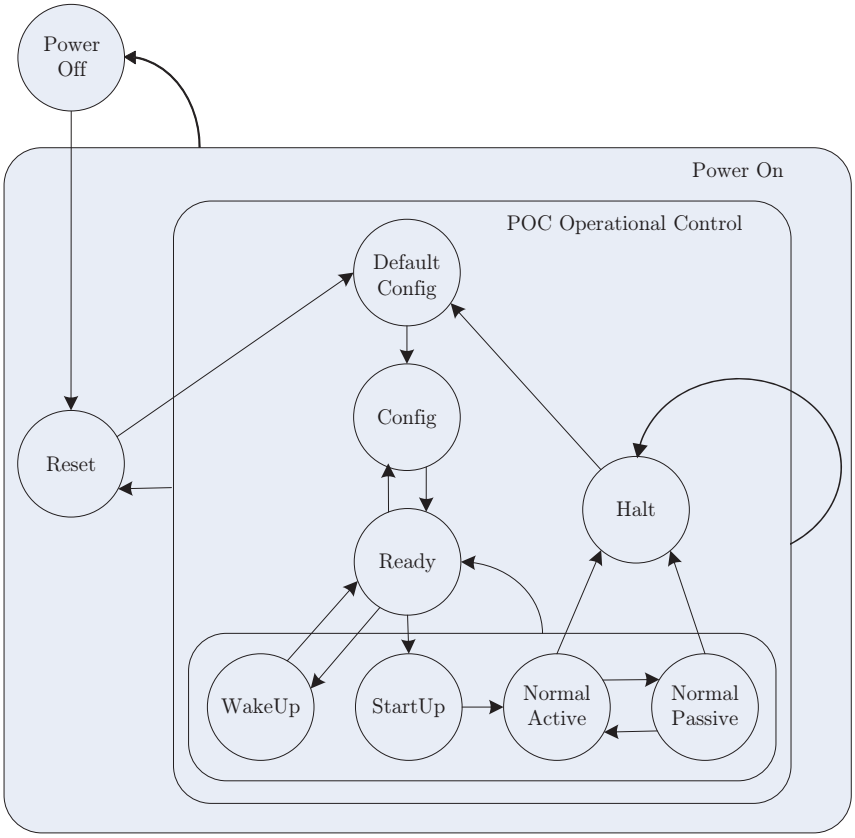


Abbildung 3.11.: Protokollablaufkontrolle POC auf einem FlexRay-Kommunikationscontroller

munikationscontroller und dem restlichen Steuergerät (*Host*) ausgetauscht. Während des normalen Operationszustands wird eine permanente Fehlerüberprüfung durchgeführt (Synchronisationsfehler). Bei detektierten Fehlern lässt sich ein Steuergerät optional erst in einen passiven Betriebszustand (*NORMAL PASSIVE*) überführen, um eine direkte Terminierung der Integration im Cluster zu vermeiden.

4. *Host*-Befehle, die den regulären Verhaltensfluss unterbrechen. Zu diesen entziehenden Befehlen zählt beispielsweise ein spezieller *Freeze*-Befehl, der die Protokollmaschine beim Eintreten eines schwerwiegenden Fehlers abrupt in einen Zustand *HALT* überführt.

3.3.3.2. Weck- und Aufstartverhalten

Aus technischer Sicht ist der Startvorgang (*StartUp*) zur Erreichung einer stabilen synchronen Buskommunikation in einem zeitgesteuerten Bussystem als kritische Phase zu betrachten /[SK06]/.

In Abhängigkeit von der Konfigurationseinstellung können die Netzwerkknoten direkt nach Erreichen der Betriebsspannung den Startvorgang einleiten. Falls mindestens ein Teilnehmer den Schlafmodus (*Sleep*) unterstützt, muss vorab des Startversuchs ein Weckvorgang (*WakeUp*) eingeleitet werden. Beim eigentlichen Startvorgang reagieren die FlexRay-Kommunikationscontroller auf spezielle StartUp-Botschaften im Übertragungskanal. Ein gesetztes StartUp-Bit in den Steuerdaten der Botschaft identifiziert se-

mantisch eine StartUp-Botschaft. Auf Basis der detektierten Ankunftszeit lassen sich somit einzelne Zeitstempel für die Netzwerkteilnehmer, die eine StartUp-Botschaft versendet haben, ausrechnen. Da sich die Knoten die globale Zeit des Clusters aus den einzelnen StartUp-Botschaften errechnen, müssen bei einem Startvorgang mindestens zwei StartUp-fähige Netzwerkteilnehmer aktiv den Clusterstart initiieren.

Ein Clusterstart, der nicht nach dem Verlassen eines Schlafmodus dem Weckvorgang folgt, sondern direkt nach der Initialisierung der Spannungsversorgung stattfindet, wird allgemein als Kaltstart (*Coldstart*) bezeichnet. Abb. 3.12 zeigt die Ablaufsequenz eines potentiellen WakeUp- und StartUp-Vorgangs. Folgendes Szenario wird hierbei exemplarisch durchgespielt:

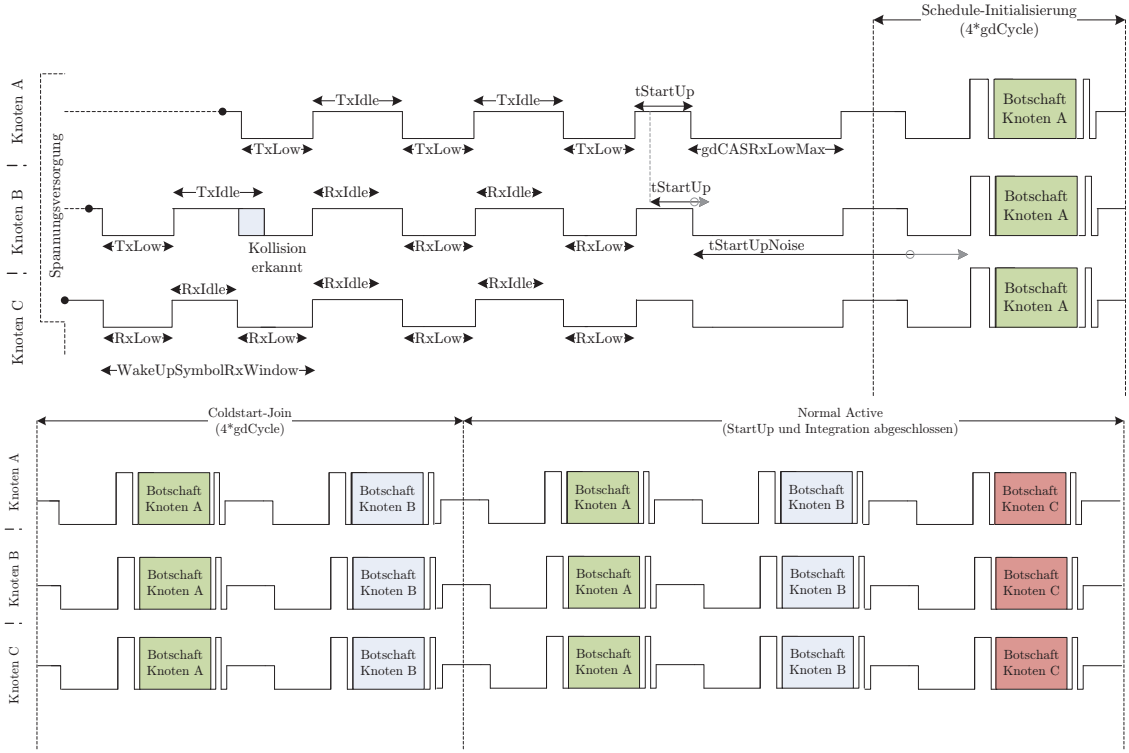


Abbildung 3.12.: Szenario einer Ablaufsequenz des WakeUps mit anschließendem Clusterkaltstart

1. In dem Cluster sind drei FlexRay-Steuergeräte verbaut, darunter zwei Coldstart-Knoten (*A*,*B*) und ein Non-Sync-Knoten (*C*), der während des StartUp-Vorgangs eine passive Rolle einnimmt.

2. Bei anliegender Spannungsversorgung sind alle Steuergeräte auf Anhieb im Schlafmodus, wobei der interne Steuergerätehochlauf unterschiedliche Zeitintervalle beansprucht[10].

3. Der von Knoten B initiierte Weckvorgang wird von dem Weckvorgang des später initialisierten Knoten A überlagert. Knoten B bricht nach der Detektion der WakeUp-Kollision den Weckvorgang ab.

[10]Das Anwendungsbeispiel abstrahiert von der Tatsache, ob ein unmittelbarer Schlafzustand nach dem Anliegen der Spannungsversorgung sinnvoll/möglich ist.

Spannungsversorgung	50	210	ms
Weckvorgang	60	1896	μs (10Mbit/s)
Prestartphase	66	10030	μs (10Mbit/s, gdCycle$\leq$5ms)
Synchronisationsphase	262	40000	μs (10Mbit/s, gdCycle$\leq$5ms)
Summe	50,4	261,9	ms (10Mbit/s, gdCycle$\leq$5ms)
	100,1	261,9	ms (10Mbit/s, gdCycle=5ms)
-Spannungsversorgung	50,1	51,9	ms (10Mbit/s, gdCycle=5ms)

Tabelle 3.3.: Einflussfaktoren zum Zeitverhalten beim Weck- und Startvorgang eines Clusters

4. Knoten A versucht im Anschluss des Weckvorgangs das Cluster kaltzustarten. Durch den Ablauf des Timers *tStartUp* kann Knoten A durch die anhaltende Busruhe einen Kaltstartversuch beginnen. Knoten B initialisiert seinen Timer *tStartUp* später als Knoten A. Über das CAS-Symbol (*collision avoidance symbol*) detektiert Knoten B, dass keine Busruhe auf dem Übertragungskanal herrscht. Während des Ablauf des neuinitialisierten Timers *tStartUpNoise* versucht Knoten B anschließend die am Bus anliegenden Signale einem StartUp-Vorgang zuzuordnen[11].

5. Bei der Schedule-Initialisierung wird die StartUp-Botschaft des Knoten A vier Buszyklen lang an der richtigen Stelle im Schedule übertragen.

6. Knoten B schließt sich dem StartUp-Vorgang nach der korrekten Dekodierung und der Validierung der Zeitstempel der vier StartUp-Botschaften an. Vier Buszyklen lang wird dabei zusätzlich die StartUp-Botschaft des Knoten B gesendet, auf den sich Knoten A gleichermaßen synchronisiert.

7. Nach acht Buszyklen ist eine stabile Bussynchronisation erreicht und sämtliche konventionelle Knoten beginnen ihre Botschaften gemäß des geplanten Busschedules zu übertragen.

Aus dem Ablauf lässt sich entnehmen, dass mehrere Faktoren die benötigte Zeit bis zur stabilen Bussynchronisation beeinflussen:

Wie in Tab. 3.3 ersichtlich, wird die zeitliche Ausprägung des Aufstartverhaltens der Knoten signifikant durch die Latenz beim Anlegen der Spannungsversorgung und dem Initialisieren der einzelnen Knoten beeinflusst. Zusätzlich spielt die Buszykluslänge für die Berechnung des Startvorgangs eine wichtige Rolle. Durch eine Reduzierung dieser beiden Einflüsse lässt sich während des gesamten Weck-/Startvorgangs in einem 5ms-Zyklus bei einer Busgeschwindigkeit von 10Mbit/s eine Varianz im Bereich von 3,6% erzielen. Wird die Tatsache miteinbezogen, dass der Kaltstartknoten bereits nach dem sechsten Buszyklus (respektive siebten Buszyklus beim anschließenden Kaltstartknoten) kommunikationsbereit ist, so ergibt sich ein weiterer Unterschied beim Kommunikationsbeginn von zwei Buszyklen.

Zeitliche Einflüsse

Folgende Zeitannahmen wirken sich verzögernd bei der Etablierung der Buskommunikation innerhalb der StartUp-Prozedur aus:

- Die Verzögerungsausbreitung eines gesendeten Signals im Netzwerk,

[11] Falls kein StartUp innerhalb des Zeitintervalls *tStartUpNoise* am Bus durchgeführt wird hat der Knoten B erneut die Möglichkeit selbst die Rolle des führenden Kaltstarters einzunehmen. Dadurch wird vermieden, dass ein potentiell korrupter Knoten A einen StartUp über Knoten B kontinuierlich verhindert.

- Der potentiell mögliche Drift innerhalb der knotenlokalen Uhren,
- Die Verarbeitungsdauer bei der Ausführung von Prozessschritten im Knoten,
- Die Bereitstellungsdauer einer ausreichenden Spannungsversorgung auf dem Netzwerkkanal/Knoten.

3.3.3.3. Uhrensynchronisation

Bei der Uhrensynchronisation /[Ung07]/ werden die Abweichungen der Uhren innerhalb ihrer Präzision zurückgesetzt (*Offset-Korrektur*) Zusätzlich wird der Startzeitpunkt des Zyklus ($T_i(z)$) für jeden Knoten korrigiert. Gleichzeitig wird die Taktrate τ zur Generierung der Makroticks für die Uhr eines jeden Clusterknotens neu justiert, um den Präzisionsfehler zu korrigieren (*Rate-Korrektur*). Entsprechend lässt sich die lokale Zeit T_i eines nicht korrigierten Knotens i zum Zeitpunkt t durch

$$T_i(t) = \tau * t + T_i(0)$$

ausdrücken. Der *Offset*-Fehler entsteht durch die Akkumulation maximaler Fehler im Bereich der *Quantisierung*, der *Mikrotickverteilung*, bei *Rundungsfehlern* und durch die *Topologie*. Der *Rate*-Fehler, der nur alle zwei Zyklen neu berechnet wird, resultiert durch zweimalige *Quantisierung*, *Rundungsfehler* und die *Ratendämpfung*.

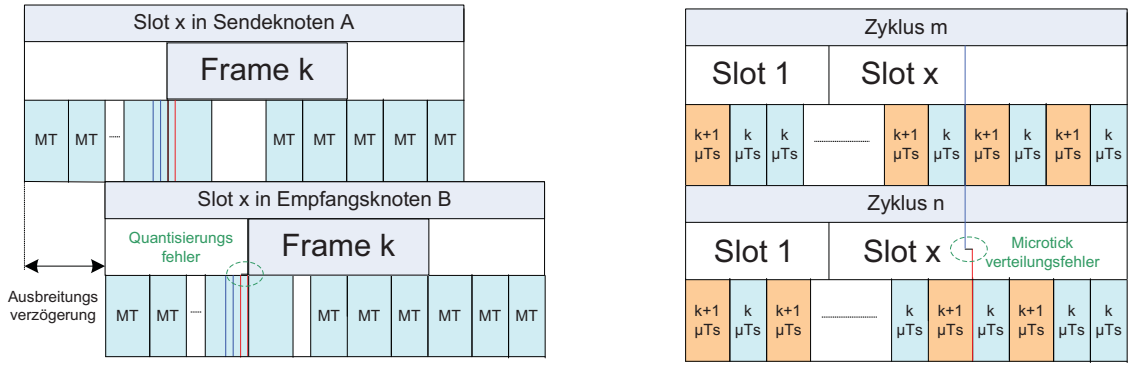
$$\epsilon_{Offset} = \Delta\epsilon_{Quant} + \Delta\epsilon_{Dist} + \Delta\epsilon_{Div} + \Delta\epsilon_{Top}\epsilon_{Rate} = 2 * \Delta\epsilon_{Quant} + \Delta\epsilon_{Div} + \Delta\epsilon_{Damp}$$

Genauigkeitsgrenzen

In einem FlexRay-Cluster finden sich konzeptbedingt Ungenauigkeiten, die als Fehler im Systementwurf zu berücksichtigen sind. Diese Einflüsse resultieren aus Effekten der Busphysik und der implementierten Berechnungsterme zur Synchronisation des Netzwerks. Demzufolge korrelieren die anfolgend aufgeführten Aspekte mit der Bestimmung der im Cluster vorhandenden Präzision /[Ung07]/.

Quantisierungsfehler: Das Versenden einer Botschaft im FlexRay-Netzwerk basiert zeitlich auf der Mikrotickkante des sendenden Kommunikationscontrollers. Auf der Empfängerseite wird dabei die erste Kante des Mikrotickintervalls als Empfangszeitpunkt akzeptiert, in dem die Botschaft detektiert wurde. Im ungünstigsten Fall fällt der Botschaftsempfang exakt auf eine Mikrotickkante, wodurch der zu dieser Zeit inkrementierte Mikrotickzählerwert als gültig angenommen wird. Dadurch kann es zu einem Diskretisierungsfehler von bis zu einem Mikrotick pro Zyklus kommen ($\text{err}_{Quant} < 1\mu T/Cycle$) (s. Abb. 3.13).

Mikrotickverteilungsfehler: Der Makrotickgenerierungsprozess (MTG) des Kommunikationscontrollers basiert auf einem Algorithmus, der die Anzahl der Mikroticks pro Zyklus auf die Anzahl der Makroticks verteilt. Besonders im Fall der Ratenkorrektur ist es jedoch nicht immer möglich, eine einheitliche diskrete Anzahl an Mikroticks auf die fixe Anzahl an Makroticks im Zyklus gleichmäßig zu verteilen. Dadurch entstehen in jedem FlexRay-Kommunikationscontroller eine eigene Anordnung von geringfügig unterschiedlich langen Makroticks im Vergleich



(a) Diskretisierungsfehler gegenüber dem Sender bei der Detektion einer empfangenen Botschaft auf Mikrotickebene.

(b) Mikrotickverteilungsfehler auf Zyklusebene

Abbildung 3.13.: Quantisierungsfehler und Mikrotickverteilungsfehler in einem Cluster

zur nominellen uniformen Makrotickgröße. Durch die Abstraktion von diesem Verteilungsprozess in den Berechnungstermen der FlexRay-Uhrensynchronisation kommt es systemweit zu einer zusätzlichen Uhrenabweichung in den Knoten. Eine Ungenauigkeit zu den Berechnungsformeln kann mit $\text{err}_{Dist} < 1\mu T/Cycle$ abgeschätzt werden (s. Abb. 3.13).

Topologiefehler: Die Verteilung von Botschaften im Netzwerk führt zu einer zeitlichen Verzögerung, die je nach physikalischem Abstand zweier Kommunikationsknoten variiert. Je präziser die minimale Verzögerung zwischen zwei Steuergeräten spezifiziert wird, desto besser kann diese Ungenauigkeit über einen Kompensationswert in den Berechnungstermen ausgeglichen werden. (*pdDelayCompensation gdPropagationDelayMin*).

Rundungsfehler: Durch die Ganzzahlarithmetik bei der Berechnung der Ratenkorrektur im Kommunikationscontroller werden Ganzzahldivisionen mit einem Rundungsfehler behaftet, der einheitlich mit $\text{err}_{Div} < |0.5\mu T|$ taxiert wird.

Driftdämpfungsfehler: Während wiederkehrende Offsetverschiebungen im Netzwerk über den Kompensationswert *pdDelayCompensation* ausgeglichen werden, stellen wiederkehrende Ratenverschiebungen ein Risiko dar. Durch eine permanente Korrektur der Zykluslänge in die gleiche Richtung kann ein beliebiger Clusterdrift (gegen Null oder gegen Unendlich) entstehen. Diesem akkumulierenden Effekt wird entsprechend mit einem Dämpfungswert entgegengewirkt, der die Ratenkorrektur bei geringer Abweichung im Knoten unterdrückt. Diese Dämpfung führt allerdings zu einem zusätzlichen Fehler in der Ratenkorrektur, der direkt Einfluss auf die Gesamtpräzision des Clusters nimmt ($\text{err}_{Damp} < |pDelayCompensation * \mu TLengthMax|$).

3.3.4. Systemparametrierung

Bei der Systemparametrierung interessieren die allgemeinen Aspekte der Konfiguration der physikalischen Bitübertragungsschicht sowie des Netzwerkprotokolls. Diese lassen sich bis auf detaillierte Justierungen im Bereich jedes einzelnen Netzwerkteilnehmers

in Form von knotenlokalen Parametern verfeineren. Die nachfolgende Auflistung spiegelt dabei die wesentlichen Bereiche der Konfiguration zeitgesteuerter Kommunikation wieder.

3.3.4.1. Konfiguration der elektrischen Bitübertragungsschicht

Grundlegende Parameter des elektrischen Physical Layers gilt es vorab zu ermitteln.

Baudrate: Die Baudrate beschreibt die Schrittgeschwindigkeit bei der Datenübertragung /[Trö05]/. Durch die Festlegung der Baudrate wird die auf dem Netzwerk verfügbare Bandbreite bestimmt, welche die Performanz des Bussystems beeinflusst.

Controllerfrequenz: Ein auf einem Knoten verbauter FlexRay-Kommunikationscontroller wird mit einer hardwareabhängigen Taktfrequenz betrieben. Taktfrequenzen im Bereich von 20-80Mhz decken das aktuell spezifizierte Einsatzspektrum ab.

Maximale Ausbreitungsverzögerungen: Die Identifikation der im Netzwerk entstehenden physikalischen Ausbreitungsverzögerung bei der Signalübertragung stellt einen wichtigen Faktor zur Ableitung potentieller Topologievarianten dar.

Minimale Ausbreitungsverzögerungen: Die Ermittlung minimaler Ausbreitungsverzögerungen zeigt sich relevant um den Wertebereich der Ausbreitungsverzögerung und damit unnötige Verluste in der Protokollperformanz zu vermindern.

Bustreiber: Der Signalwechsel am elektrischen Bustreiber zwischen Busruhe und Busaktivität und vice versa generiert ein zusätzlichen Zeitbedarf bei der Signalübertragung (*dBDRxia*).

Aktive Sterne: Die Signalspiegelung zur Verteilung innerhalb einer sternförmigen Topologie führt zu einer Signalverkürzung an den Ein- und Ausgängen beim Übertragungsaufbau im Netzwerk zwischen zwei Teilnehmern (*dStarTruncation*, *nStarPathM,N*). Dabei muss der physikalische Aufbau (diskret, monolithisch) des aktiven Sterns berücksichtigt bleiben.

Uhrengenauigkeit: Obwohl nach /[Fle05a]/ die Qualität der Uhrengenauigkeit mit einem konstanten Wert definiert worden ist, kann der Einsatz hochwertigerer Quarze zu einer besseren Protokollperformanz führen. Daher sollte die Substitution dieses konstanten Faktors berücksichtigt bleiben (*cClockDeviationMax*, *gdTSSTransmitter*).

Netzwerkteilnehmer: Da die Anzahl der im Netzwerk verbauten Knoten einen Effekt auf die Qualität des vorliegenden Netzwerksignals haben, darf der Einfluss eines zugefügten Knoten nicht unberücksichtigt bleiben.

3.3.4.2. Konfiguration des Netzwerkprotokolls

Die Parameter auf Netzwerkebene hängen bereits indirekt mit den Anforderungen der auf dem Bussystem abzubildenden Anwendungsfunktionen ab.

Anzahl der Sendeknoten: Durch die Festlegung der Menge an sendeberechtigten Netzwerkknoten ergeben sich Grenzwerte für die Anzahl einzuplanender Sendeslots im Busschedule.

Anteil statischer/dynamischer Kommunikation: In Abhängigkeit zur Quantität der Signale eines vorhandenen Kommunikationstyps wird die Aufteilung des statischen und dynamischen Segments im Busszyklus bestimmt.

Aufstartverhalten: Die Anzahl der authorisierten Netzwerkknoten zur Ausführung von Weck- und Startversuchen sowie die möglichen Wiederholversuche im Fehlerfall werden clusterweit festgelegt.

3.3.4.3. Knotenparametrierung

Ergänzend zur Konfiguration der physikalischen Bitübertragungs- und der Protokollschicht ergibt sich die Festlegung einiger knotenlokaler Parameter, die anfolgend aufgelistet werden.

Kanalzuweisung: Jeder Knoten wird individuell einem oder beiden Kanälen auf dem FlexRay-Segment zugeordnet.

Startvorgang/Synchronisation: Die Lokalisation der Start-/Synchronisationsbotschaft eines Knotens im Schedule ist festzulegen[12].

Weckeigenschaften: Die Zuordnung einer Weckfähigkeit eines Knotens in Bezug auf den FlexRay-Kanal und die zeitliche Ausprägung des WUP lässt sich spezifizieren.

Sendeverhalten im dynamischen Segment: Die maximal gültigen Botschaftslängen und der dazu korrespondierende spätest tolerierbare Sendezeitpunkt im dynamischen Segment sind knotenabhängig festzulegen.

Uhrenkorrektur: Eingriffe in die Uhrenkorrekur sind durch Adaption des Driftdämpfungsfaktors oder extern über die Schnittstelle zum Host-Prozessors pro Knoten möglich.

Fehlertoleranz: Einstellung des Knotenverhaltens im Fall von Synchronisationsfehlern und Wechsel in einen passiven oder inaktiven Zustand.

Integrationsfähigkeit: Festlegung von tolerierbaren Abweichungen in der Genauigkeit bei der Uhrensynchronisation im Sinne einer Knotenintegration in ein synchrones oder sich synchronisierendes Cluster.

[12] In diesem Bereich könnte auch der *Singleslot*-Modus eingeordnet werden, um die Kommunikation auf dem Bus in kritischen Phasen, wie etwa beim Startvorgang, auf einen Slot pro Knoten zu begrenzen.

KAPITEL 4

Methodik

Einleitend werden in diesem Kapitel sämtliche FlexRay-spezifischen Anforderungen zusammengefasst, die bei der Definition der E/E-Entwicklungsmethodik zu berücksichtigen sind. Darauf aufbauend wird das Konzept der entwickelten Methode FlexZOOMED zum eindeutigen Verständnis formal spezifiziert. Weiterhin wird der Bezug zwischen der FlexRay-Parametrierung und der abstrakten E/E-Metamodellierung hergestellt. Anschließend wird die Umsetzung der E/E-Entwicklungsmethodik auf domänenspezifische Modelle erläutert und die umfassende Integration von Entwicklungsaspekten für flexible zeitgesteuerte Architekturen beschrieben. Als Ergänzung dient der letzte Abschnitt, der sich in dem Kontext der entwickelten Methodik mit der Formulierung und technischen Lösung von identifizierten Optimierungsproblemen auseinandersetzt.

Mit dem Wechsel von der CAN- /[Rob91]/ auf die FlexRay-Technologie /[Fle05b], [Rau07a]/ in Fahrzeugsegmenten mit hohem Kommunikationsbedarf zwischen den Steuergeräten, vollzieht sich ein Wandel in der Entwicklungssystematik eines Fahrzeugherstellers in Bezug auf die Weiterentwicklung seiner E/E-Architekturen. Um die leistungsfähigen Potentiale von FlexRay auszuschöpfen, müssen neuartige Entwurfsprinzipien beachtet werden, welche auf die Eigenschaften eines zeitgesteuerten Bussystems zurückgeführt werden können /[Kop97], [SBV+02]/. Speziell die starke Verlagerung von Aufgaben innerhalb des Entwicklungsprozesses in die Anforderungsanalyse und Designphase ist dabei zu berücksichtigen /[MB97]/. Der FlexZOOMED-Ansatz setzt diese Prämissen durch eine modellbasierte Beschreibung und Analyse der FlexRay-Systemspezifikation auf der E/E-Architekturebene um. Sämtliche Inhalte dieses Kapitels bilden eine Grundlage für die konzeptionelle Umsetzung der FlexZOOMED-Entwicklungsmethodik.

4.1. Übersicht

Bei der Definition einer gültigen FlexRay-Systemspezifikation eröffnet sich für den Fahrzeughersteller eine Reihe an technischen Fragestellungen, deren Antworten sich unter anderem durch gezielte E/E-Architekturentscheidungen ergeben. Der FlexZOOMED-Ansatz fokussiert auf diese Aspekte, indem der FlexRay-Systementwurf auf die Architekturebene verlagert wird. Zum Verständnis der Abhängigkeiten zwischen der Vernetzung und der Architekturentwicklung werden diese beiden Sichten in Bezug auf die FlexRay-Systementwicklung konkretisiert. Der folgende Abschnitt ist inhaltlich an /[BR07]/ angelehnt.

4.1.1. Anforderungen an die FlexRay-Systementwicklung

Die Entwicklung der vernetzten E/E-Systemarchitektur eines Fahrzeugs erfolgt in einem bilateralen Prozess zur Abdeckung aller funktionaler/nichtfunktionaler Anforderungen aus Architektur- sowie aus Vernetzungssicht /[BMG07]/.

Architektursicht

Zum Entwurf und Integration eines FlexRay-Busses innerhalb einer E/E-Systemarchitektur müssen allgemeine Anforderungen eingehalten werden, um die Gesamtqualität eines Fahrzeugs über seinen Lebenszyklus gewährleisten zu können. Für einen transparenten Integrationsprozess eines FlexRay-Systems aus Architektursicht gilt es daher die wechselseitigen Abhängigkeiten zwischen den allgemeinen architekturellen Anforderungen und dem konkreten Netzwerkdesign zu beherrschen (s. Abb. 4.1).

Abbildung 4.1.: Wechselseitige Abhängigkeiten der Architektur- und der Netzwerkentwicklung

Bei der Integration von Steuergeräten sind in einer existierenden flexiblen zeitgesteuerten Umgebung die Eigenschaften einer FlexRay-Systemkonfiguration in Bezug auf Architekturdesignentscheidungen zu plausibilisieren. Die folgenden Klassifikationskriterien beeinflussen die FlexRay-Systemkonfiguration:

Zeitverhalten (Performanz): Die Applikation des flexiblen dynamischen Segments im

FlexRay-Kommunikationszyklus bewirkt eine periodische Unterbrechung der deterministisch erfassbaren Datenkommunikation innerhalb des statischen Segments. Die für eine synchrone Sensordatenerfassung und -auswertung relevanten vernetzten Systeme bestimmen dabei die Größen für Gesamtzykluslänge und Zyklussegmentierung.

Systemerweiterung/-integration: Eine Reservierung von FlexRay-Slots wird notwendig, um das Netzwerk für eine spätere Systemmodifikation oder -integration flexibel zu halten. Multiplexing-Techniken ermöglichen dabei eine effizientere Vergabe der Bandbreite auf Kosten einer erhöhten Systemkomplexität.

Systemmanagement: Zur Gewährleistung der Netzwerkverfügbarkeit und Optimierung des Energieverbrauchs müssen geeignete Coldstart- und Sync-Steuergeräte (*nodes*) definiert werden, welche eine korrekte Funktionsweise eines FlexRay-Netzwerks (*cluster*) auch in unterschiedlichen Varianten je nach Ausstattungswunsch ermöglichen. Eine geeignete Anzahl an Weck- und Startversuchen zur Behandlung von Fehlerszenarien ist zu identifizieren.

Plattform- und Variantenbildung: Die Slots eines Kommunikationszyklus können je nach Anwendung als plattform- oder modellspezifisch ausgelegt werden. Dies beeinflusst die Freiheiten für das Scheduling von Applikationsprozessen auf den verschiedenen Netzwerkteilnehmern (Knoten).

Quality of service: Für eine performante, effiziente Steuergeräteprogrammierung können große Botschaften mit einem hohen Nutzdatenanteil implementiert werden, welches zu einer starken Belastung der verfügbaren Netzwerkbandbreite führen kann. Um das Netzwerk nicht statisch mit Botschaften zu belegen, welche nur in speziellen Situationen gebraucht werden, eignet sich das dynamische FlexRay-Segment. Analog zu CAN-basierten Netzwerken können dabei allerdings Lastzustände (*burst effects*) auftreten, die es zu beherrschen gilt.

Topologien: Da sich die physikalische Netzwerktopologie entscheidend auf die Systempräzision durch Verzögerungseffekte bei der Signalausbreitung auswirkt, gilt es die Einführung neuer Netzwerkknoten bereits zur Systemdesignphase zu analysieren.

Komponenten: Jede Hard-/Software-Komponente eines FlexRay-Systems, beispielsweise der Kommunikationscontroller bedingt zusätzliche individuelle Restriktionen bezüglich der Verwendung in einem Cluster. Beispiele sind unterschiedliche Puffereigenschaften oder Speicherschnittstellen.

Systemverlässlichkeit: Schließlich sind Alterungseffekte innerhalb eines zeitgesteuerten Systems mit verteilter Echtzeit zu beachten, um eine sichere Systemoperation bei langer Einsatzdauer zu garantieren. Leichtes Abweichen der Oszillatorgenauigkeit kann hierbei kritisch werden.

Netzwerksicht

In der Analyse- und Designphase eines FlexRay-Systems müssen folgende Hauptaufgaben gelöst werden, um ein vollständig und funktional korrekt spezifiziertes Netzwerk

zu erhalten (s. Abb. 4.2, /[Bro07]/):

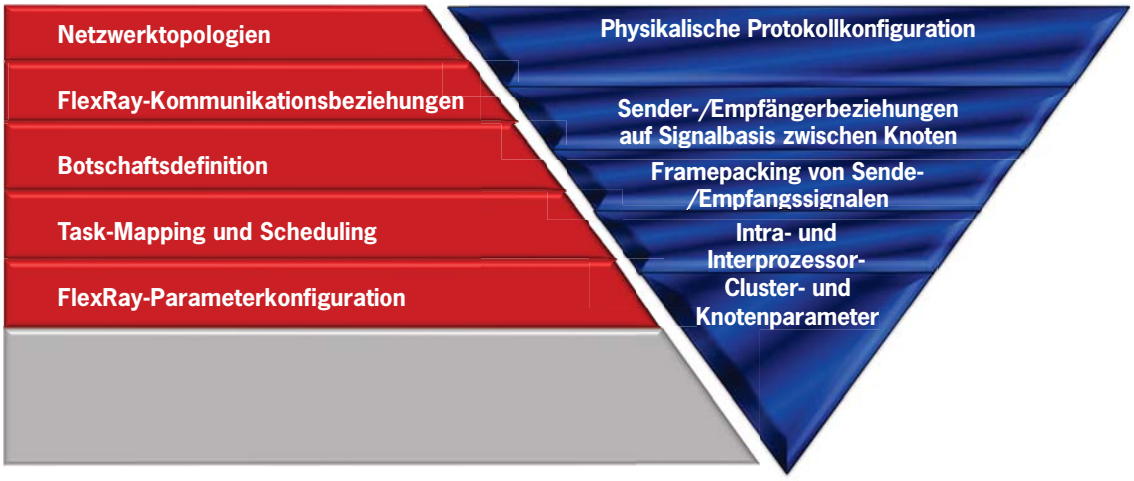


Abbildung 4.2.: Netzwerkintegrationsvorgehen zur Definition einer FlexRay-Architektur

Netzwerktopologien: Die Wahl der FlexRay-Systemtopologie trägt entscheidend zu einer späteren Protokollkonfiguration bei, da sich die Distanz der Punkt-zu-Punkt-Kommunikation auf die Präzision eines Netzwerks auswirkt.

FlexRay-Kommunikationsbeziehungen: Um einen FlexRay-Schedule entwerfen zu können, müssen vorab die Sender-/Empfängerbeziehungen auf Basis von Signalkommunikation zwischen den vernetzenden Knoten definiert werden.

Botschaftsdefinition: Anhand der identifizierten Sende-/Empfangssignale werden Botschaften definiert. Ziel ist eine möglichst geringe Gesamtanzahl an Botschaften. Diese Anforderung wird in der Regel über Heuristiken gelöst /[SN03]/.

Task-Mapping und Scheduling: Für eine korrekte Intra- und Interprozessorkommunikation wird eine holistische Scheduling-Analyse zur zeitlichen Planung von Prozessen (*tasks*) benötigt /[PEPP06]/. Dabei gilt es zu beachten, dass für den Botschaftsversand/-empfang Sende- und Empfangsprozesse definiert werden müssen und als Stützstellen entlang des Kommunikationszyklus angeordnet werden. Dabei kann die Struktur des Busschedules abgeleitet werden.

FlexRay-Parameterkonfiguration: Die Herausforderung im FlexRay-Designprozess korreliert mit einer komplexen Konfiguration einer großen Anzahl an Netzwerkparameter (>80). Diese definieren die Eigenschaften des Protokolls hinsichtlich seiner kompletten Ausprägung. Dazu zählen zusammengefasst folgende Eigenschaften:

- Physikalische Signaldarstellung auf dem Kommunikationsmedium,
- Zeitabhängiger Kommunikationsplan des Busses,
- Weck-, Start- und Aktivitätsverhalten,

- Fehlerkontrolle und -behebung,
- Verteilte globale Synchronisation,

In Abb. 4.3 werden sämtliche relevante Parameter (keine Konstanten/Hilfsparameter) des FlexRay-Kommunikationssystems abgebildet.

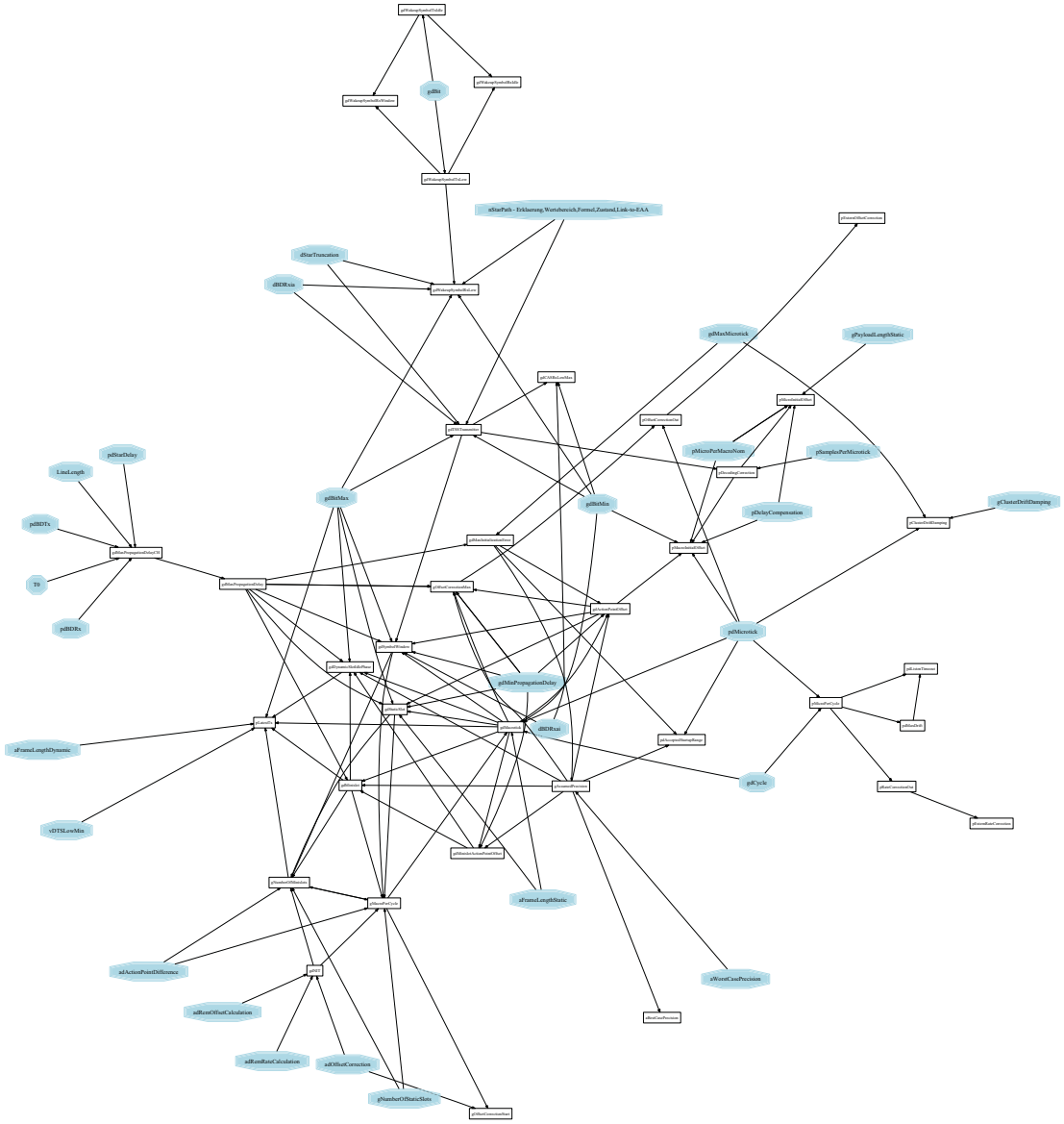


Abbildung 4.3.: FlexRay-Parameterbaum mit farblicher Abgrenzung unidirektionaler Parameter (*Startparameter*)

Die Ergebnisse der vorhergehenden Schritte werden in einen gültigen FlexRay-Parametersatz überführt. Dabei wird zwischen den globalen, auf Clusterebene gültigen,

und den lokalen Node-spezifischen Parametern unterschieden. Allgemeine Parameter, welche die grundlegenden Merkmale der Protokoll- und Physical Layer-Konfiguration charakterisieren, werden als High-Level-Parameter zusammengefasst betrachtet (bspw. Baudrate, Datenlänge, Präzision, Verzögerungswerte, Signalverkürzung, Sternanzahl, Makro-/Mikrotick, ...). Aus den allgemeinen Parametern werden Protokoll- und Physical-Layer-spezifische Parameter abgeleitet, um die Funktionsweise der grundlegenden Protokollmechanismen gegenüber einer konkreten Vernetzungstopologie zu verifizieren.

4.1.2. Systemintegration von FlexRay-Architekturen

Um mit einer flexiblen zeitgesteuerten Technologie wie FlexRay eine langfristige Produktqualität zu gewährleisten, müssen die Eigenschaften einer Systemkonfiguration interpretiert und die Auswirkungen notwendiger Modifikationen verstanden werden.

FlexRay-Designparadigmen

Die Komplexität der Transformation eines FlexRay-Konzepts A in ein alternatives Konzept B wird anhand des Beispiels zweier Kommunikationszyklen mit unterschiedlicher Zykluslänge dargestellt (s. Abb. 4.4).

Protokoll:

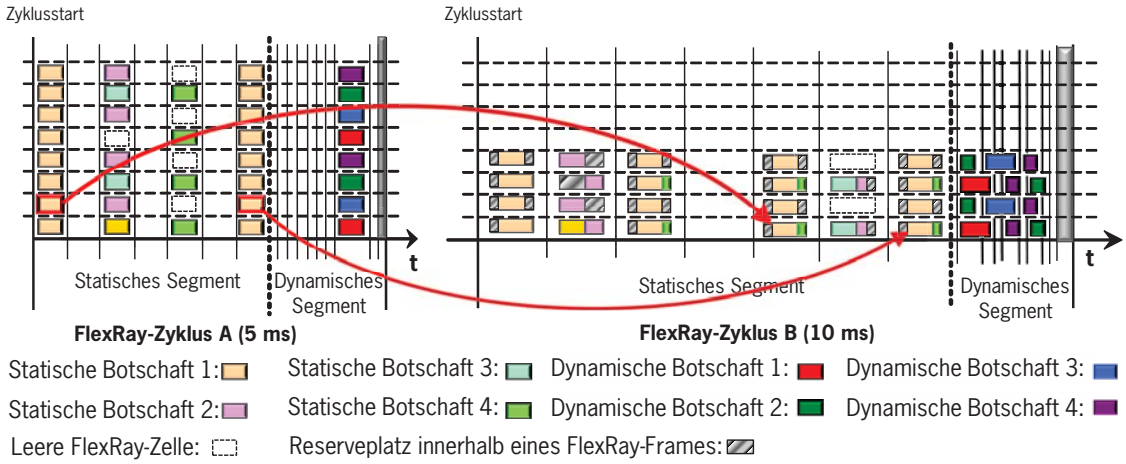


Abbildung 4.4.: Gegenüberstellung unterschiedlicher FlexRay-Paradigmen anhand des Buszugriffsverfahren

Obwohl beide Ansätze optisch sehr unterschiedlich wirken, erzeugen sie ein nahezu identisches Kommunikationsverhalten.

Performanz: Um in Zyklus B die gleichen Abtastraten wie in Zyklus A zu ermöglichen, muss die statische Botschaft 1 konsequenterweise innerhalb des Zyklus B mehrfach wiederholt verschickt werden, was zu einer zusätzlicher Auslastung statischer Bandbreite

führt. Um die statische Botschaft 2 in den richtigen Zyklen zu senden, muss die Botschaft dekomponiert und einzelne Signale an den richtigen Lücken der restlichen Slots platziert werden, was eventuell eine Anpassung der Sende-/Empfangsbeziehungen in Zyklus B benötigt.

Systemerweiterung/-integration: Weitere Systeme werden in beiden Ansätzen unterschiedlich integriert. Während in Zyklus A auf leere Zellen oder komplett reservierte Slots zurückgegriffen wird, so ergibt sich in Zyklus B die Möglichkeit die noch leeren Stellen in den einzelnen Botschaften zu belegen. Im Gegensatz zu den leeren Slots in Zyklus A müssen allerdings in Zyklus B beim Auffüllen von Botschaften die identischen Sender-/Empfängerbeziehungen übernommen werden.

Plattform- und Variantenbildung: Für eine Plattform- und Variantenbildung eignet sich eher die Vergabe einzelner FlexRay-Zellen/-Slots wie in Zyklus A, um ein kompliziertes Aufsplitten von Botschaften (statische Botschaft 2) zu vermeiden.

Quality of service: Wie im dynamischen Segment für jeden Bus-Schedule ersichtlich, liegt es an dem Netzwerkdesigner, die zeitlichen Limits für die parallele Steuergeräteprogrammierung einzuhalten. Durch den längeren Zyklus B wird der Einsatz paralleler Transportkanäle innerhalb eines Buszyklus notwendig, um den gleichen Datendurchsatz wie in Zyklus A zu erzielen. Je nach Leistungsfähigkeit des Empfangssteuergeräts kann die Nutzdatenlänge einer dynamischen Botschaft und deren Zykluswiederholrate beliebig skaliert werden.

Systemverlässlichkeit: Da die doppelte Zykluslänge B auch einen größeren Spielraum für Clusterdrifts zulässt, müssen alle Aspekte, die der Uhrensynchronisation zugerechnet werden können (*rate-*, *offset-correction*), entsprechend der Systempräzision und damit an verfügbare Oszillatorqualitäten im Netzwerk angepasst werden.

Physical Layer:

Für eine FlexRay-Systemintegration/-erweiterung muss offensichtlich das Layout der Netzwerktopologie adaptiert werden. Im Folgenden wird eine Erweiterung der Netzwerktopologie A mit 4 Knoten und einem Sternkoppler auf 6 Knoten und zwei Sternkopplern dargestellt.

Topologien: Durch die Einführung eines Node E verlängert sich der Maximalabstand zwischen Knoten und Stern auf dem Sternast sowie die maximale Distanz einer Punkt-zu-Punkt-Verbindung im Cluster (Node E zu Node F). Die zusätzliche Verzögerung über einen zweiten eingeführten aktiven Sternkoppler muss im Wertebereich der FlexRay-Parametrierung abgesichert werden.

Komponenten: Eventuelle Vorgaben des FlexRay-Kommunikationscontrollers, beispielsweise in der Puffer-Verwendung, müssen berücksichtigt bleiben. Falls die Knoten abweichende FlexRay-Hard-/Softwaremodule oder -varianten /[AUT07a]/ besitzen, werden Interoperabilitätsuntersuchungen erforderlich.

Systemmanagement: Durch eine Funktionsanalyse werden kritische Situationen identi-

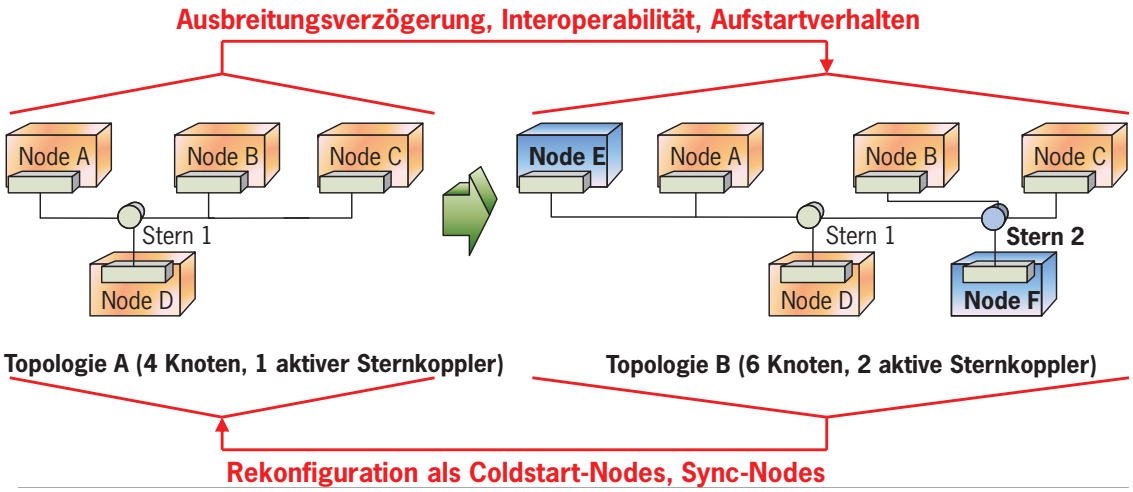


Abbildung 4.5.: Beispiel für eine mögliche Systemerweiterung eines 4 Knoten-FlexRay-Systems auf ein 6 Knoten-FlexRay-System mit zusätzlichem Sternkoppler

fiziert, in denen die integrierten Nodes E und F ein aktives Cluster zur Kommunikation benötigen. Dadurch kann entschieden werden, ob die hinzukommenden Knoten kaltstartfähig ausgelegt werden müssen. Reduziert man die Topologie B auf Topologie A, so wird eine Überprüfung und eventuelle Hinzunahme von Sync-Knoten zur Sicherung eines funktionsfähigen FlexRay-Systems notwendig.

4.1.3. Integrationsvorgehen und Optimierungspotentiale

Um die beispielhaft vorgestellten Aufgaben der Modifikation eines FlexRay-Systems aus Sicht der FlexRay-Konfiguration zu beherrschen, muss der Eingriff in einen FlexRay-Parametersatz verstanden sein. Dies wird am Beispiel der drei FlexRay-Parameter mit den höchsten Abhängigkeitsgraden im FlexRay-Parametergraph (vgl. Abb. 4.3) exemplarisch erläutert:

Makrotick: Eine Analyse der Parameterverknüpfungen über einen Abhängigkeitsgraphen zeigt, dass der Makrotick als die nominale Einheit für einen clusterweit einheitlichen zeitlichen Schritt mit bis zu dreizehn weiteren Parametern korreliert, die wesentlich zu der Struktur des FlexRay-Schedules beitragen. Die Signifikanz des Makroticks lässt sich auch daraus ableiten, dass die verfügbare Slotanzahl in den jeweiligen Segmenten auf Basis der pro Zyklus existierenden Anzahl von Makroticks errechnet wird.

Präzision: Eine Veränderung der Vernetzungstopologie führt in der Regel zu veränderten Verzögerungswerte des Kabelbaums. Dabei wird auch die Gesamtpräzision des Clusters tangiert, da sich der Initialisierungsfehler eines einzelnen Knotens bei der Integration in ein aktives FlexRay-System verstärken kann, was die Wahrscheinlichkeit für instabile Netzwerkhochläufe erhöht.

Statische Slotlänge: Eine Umstellung der Nutzdatenlänge für statische FlexRay-Bot-

schaften in langfristigen Projekten sollte vermieden werden, da eine Modifikation in diesem Bereich zu einer Neuberechnung der statischen Slotlänge führen würde. Ein statischer Slot hängt mit acht weiteren Parametern zusammen und seine Größe bestimmt logischerweise die Slotanzahl im statischen Segment. Folglich läuft ein Eingriff an dieser Stelle auf ein komplettes Redesign hinaus.

Konsequenterweise stellt das FlexRay-Systemdesign aus Konfigurationssicht eine klassische Mehrzieloptimierungsaufgabe dar. Folgende Ziele gilt es dabei zu beachten:

- Nutzdateneffizienz (möglicher Datendurchsatz pro Bandbreite),
- Systempräzision hinsichtlich serienrelevanter Topologien,
- Segmentgrößen (Systemperformanz für jeweilige Applikationen),
- Slotgrößen (Bestmögliche Skalierbarkeit),
- Slotvergabe (Zielabtastraten und Verlust von statischer Bandbreite).

4.1.4. Folgerung

Wie in den vorhergehenden Abschnitten dargestellt, ist es aus Anwendersicht essentiell die FlexRay-Systemintegration hinsichtlich allgemeiner Fragestellungen im Architekturdesign zu beherrschen. Dabei zeigt es sich entscheidend die Anforderungen aus der allgemeinen E/E-Systemarchitekturentwicklung zu präzisieren und ihren Zusammenhang zu einer FlexRay-Systementwicklung herauszuarbeiten. So implizieren einfache Modifikationen des Protokollzugriffs der Sicherungsschicht oder Adaptionen der Vernetzungstopologie auf der Bitübertragungsschicht bereits komplexe Migrationsentwicklungsschritte. Daher sollte der Designansatz langfristig als Meet-In-The-Middle-Strategie erfolgen, um das komplexe Bottom-Up-Design der FlexRay-Parameter zu vereinfachen.

Ein Wechsel zu einer standardisierten Parameterkonfiguration vereinfacht die Systemintegration signifikant, da die Anzahl potentiell notwendiger Anpassungen reduziert wird. Eine weitere Vereinfachung versprechen standardisierte Netzwerktopologien und Verkabelungsgrößen, die den Lösungsraum für FlexRay-Systeme einschränken. Allgemein führen proprietäre Modifikationen in der Regel zu aufwändigen und komplexen Entwicklungsaufwänden, welche den Spielraum zur Erweiterung eines konzipierten FlexRay-Systems beschränken.

Wie gezeigt, zeichnet sich der FlexRay-Standard durch die präzise Umsetzung herstellerspezifischer Designvorgaben aus. Diese Eigenschaft fordert im Gegenzug aber ein wesentlich tieferes Systemverständnis im Vergleich zu den Erfahrungen mit der CAN-Technologie. Um die Akzeptanz des Standards weiter zu verbessern ist eine Vereinfachung der Applikation der Technologie erforderlich, um den Aufwand für komplexe und damit test- und kostenintensive Aufgaben im Zusammenhang mit Parametrierungseinstellungen zu reduzieren. Grundsätzlich gilt es die Methoden der Anforderungs- und Systementwurfsphase in Hinblick auf die spezifischen Eigenschaften der neuen FlexRay-Technologie weiterzuentwickeln, um deren Beherrschbarkeit und umfassende Nutzung zu garantieren. Entsprechende Konzepte werden im folgenden Abschnitt vorgestellt.

4.2. Formale Systemmodellierung und -spezifikation

„System analysis is frustrating, full of complex interpersonal relationships, indefinite and difficult. In a word, it is fascinating. Once your hooked, the old easy pleasures of system building are never again enough to satisfy you.“ - Tom DeMarco

Die Idee der modellgetriebenen Entwicklung /[Sta73]/ hat sich allgemein als rechnergestützte Entwurfsmethodik für komplexe verteilte Systeme etabliert. Die Sinnhaftigkeit der modellbasierten Spezifikation ergibt sich dabei aus der Möglichkeit der Formalisierung grundlegender Aspekte im Systementwurf /[Boo91]/, dem damit entwickelten Verständnis für die Systemzusammenhänge und der weitergehenden Automatisierung von Entwurfsschritten.

In der Domäne der E/E-Architekturentwicklung haben sich bisher die Bereiche *Anforderungsmodellierung*, *logische Systemarchitektur*, *technische Systemarchitektur*, *Partitionierung* und *Softwarearchitektur* als signifikant herausgestellt (s. Abb. 4.15). Diese Dekomposition ist industrieweit allgemein akzeptiert. Deren konkrete Spezifikation, Umsetzung und Anwendung gilt allerdings noch als Forschungsbereich.

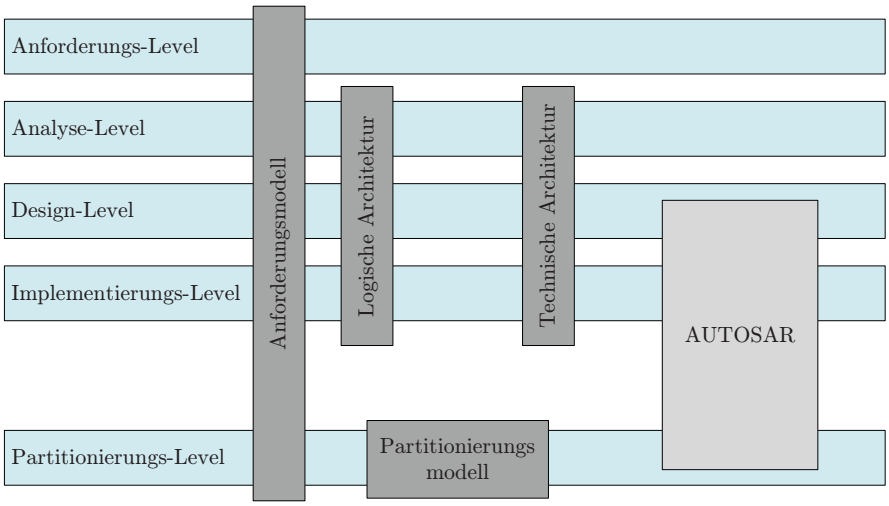


Abbildung 4.6.: Abstraktionsstufen der modellbasierten Entwurfsmethode MOSES /[Kle06]/

Der in dieser Arbeit entwickelte modellgetriebene Ansatz (s. Abb. 4.7) orientiert sich an den Bereichen der logischen und technischen Systemarchitektur, wobei der Aspekt der Partitionierung als inhärentes Teilgebiet betrachtet wird, welches keine explizite Modellierung erfordert. Softwarearchitektonische Einflüsse werden gleichermaßen nicht auf Basis einer formalen Modellierung berücksichtigt.

Die logische Systemarchitektur spaltet sich in dem Ansatz in die *statischen Funktionsnetz-* und die *dynamischen Verhaltensmodelle* auf. Ergänzend zu den dynamischen Verhaltensmodellen lassen sich die Interaktionsmodelle hinzufügen, die allerdings keinen Schwerpunkt in dieser Arbeit darstellen.

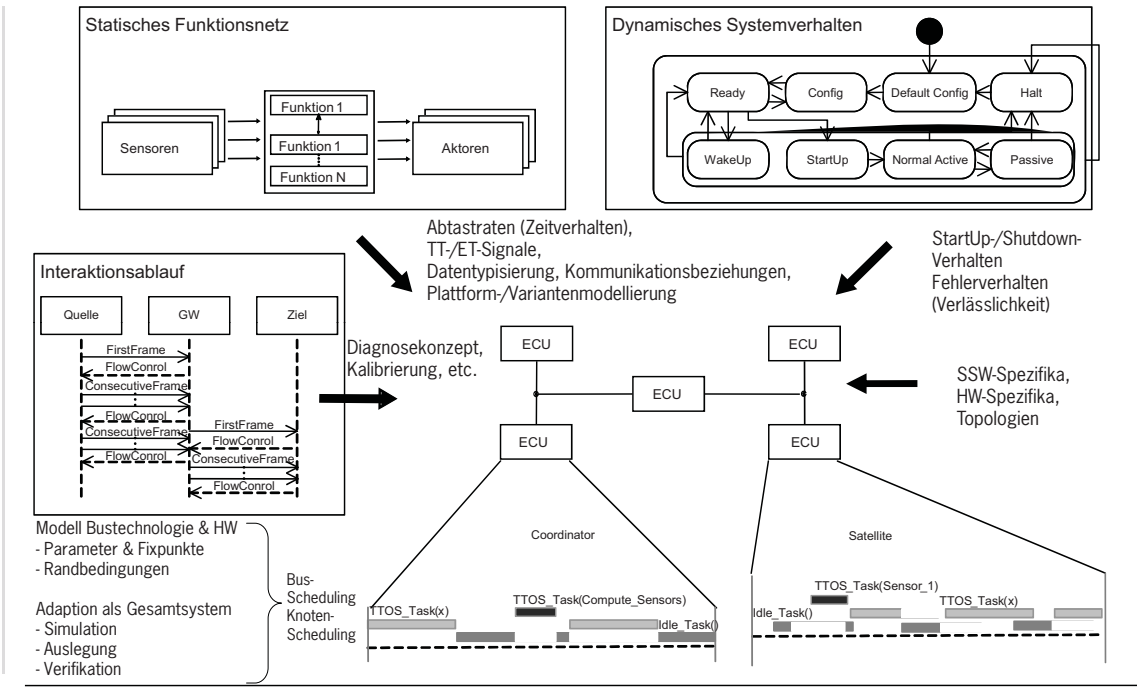


Abbildung 4.7.: Idee der integrierten modellbasierten Entwicklung zur statischen und dynamischen Analyse eines zeitgesteuerten Systemdesigns

4.2.1. Modellierungsgrundlagen und Notation

Unter der Modellbildung versteht sich in der Informatik *„... die Darstellung der unter dem Gesichtspunkt einer gegebenen Aufgabenstellung wesentlichen Strukturen, Zusammenhänge und Vorgänge eines Anwendungsgebiets.“* /[Bro98], [BS03]/. Eine Modellbeschreibung basiert dabei auf formalen Mitteln in Form von Datenstrukturen, Programmiersprachen, Graphiken oder logischen Formeln. Je nach Anwendungsgebiet ergibt sich eine Differenzierung zwischen Domänenmodellen, um etwa die Struktur einer Software zu erfassen (*Architekturmodell*), die Programmablauflogik eines Codes zu beschreiben (*Datenflussmodell*) oder den Aufbau von Daten, beispielsweise in einer Datenbank, zu spezifizieren (*Datenmodell*). In dieser Arbeit liegt der Fokus vorrangig auf der Architekturmodellierung. Für die Entwicklung eines Systemmodells innerhalb einer arbiträr verbundenen Anzahl an Modellierungsebenen gelten grundlegende Eigenschaften, die es zu beachten gilt:

Zeitkontinuität: Bei der Spezifikation zeitsensitiver Artefakte im Modell ist zu beachten, dass die physikalische Zeit als uniformer und essentieller Bestandteil einer jeden Spezifikation eines reaktiven Systems berücksichtigt bleiben muss /[Rom06]/. Eine zeitintegrierende Betrachtung innerhalb einer strukturierten Modellierung gilt als komplex und ist speziell bei der Anwendungssoftwareentwicklung signifikant. Bei der Systemmodellierung auf Architektursebene liegt der Fokus auf den zu definierenden feingranularen Diskretisierungsstufen der Zeit.

Modellabstraktion: Aus Entwicklersicht ist eine vollständige Systemmodellierung über

sämtliche Bereiche der Architekturentwicklung aufwandsbezogen unattraktiv und aus Rechnersicht aufgrund beschränkter Ressourcen technisch limitiert. Daher sind geeignete Abstraktionsebenen des Modellierungsgrads innerhalb und zwischen den Architekturdomänen zu identifizieren.

Notationssyntax

Tab. 4.1 gibt eine Übersicht zu den syntaktischen Elementen, die später bei der Beschreibung des modellgetriebenen Spezifikationsansatzes verwendet werden.

$\mathrm{R} = \{S, C, FC, ...\}$	Mengensymbole
$\mathrm{V} = \{x, y, z, ...\}$	Individuenvariablen
$\mathrm{F} = \{Ge, \pi, p, ...\}$	Funktionen
$\wedge, \vee, \neg, ...$	Aussagenlogische Junktoren
$\sim, \sqsubseteq, ...$	Relationen
$\cup, \cap, ...$	Mengenoperationen
$\subseteq, \subset, ...$	Mengenrelationen
$\exists, \forall$	Quantoren
$\{...\}, (...), :, ...$	Hilfssymbole

Tabelle 4.1.: Symbol- und Zeichenerklärung

4.2.2. Konzept der statischen Modellanalyse

Innerhalb der statischen Modellanalyse spielen dynamische Aspekte eines Systems, wie dessen Verhalten oder die Interaktion von Systemkomponenten, eine untergeordnete Rolle. Falls dynamische Eigenschaften des Systems zur Komplementierung der statischen Modellanalyse essentiell hinzugerechnet werden müssen, so empfiehlt sich eine Kombination der beiden Analysemethoden. Die Simulation physikalischer Netzwerkeigenschaften erfordert beispielsweise Spezialwerkzeuge, deren Simulationsergebnisse jedoch als Eingabevektoren in der statischen Modellanalyse einen Mehrwert generieren. Die Einbettung der dynamischen Modelloptimierung wird für diesen Zweck exemplarisch in der Arbeit behandelt (s. Kap. 5).

4.2.2.1. Entwurfsprinzipien

Zur Verbesserung der Leistungsfähigkeit eines Modellierungskonzepts ist es zweckmäßig dessen Methodik auf allgemeine Anforderungskriterien der Softwareentwicklung /[EEE98], [EE84]/ anzupassen.

Modellstrukturierung: Um die hohe Komplexität der integrierten Architekturentwicklung zu beherrschen ist es notwendig eigenständige Strukturierungskonventionen zu definieren. Dazu zählen auch die Methoden der Generalisierung und Hierarchisierung, der Modularisierung und Modellverknüpfung (*Partitionierung*).

Modellvollständigkeit: Unvollständige Modellfragmente bergen Risiken in der Konzeption und Analyse von FlexRay-Architekturen. Daraus lässt sich die Notwendigkeit von abgestuften Spezifikationsarbeitsständen ableiten, um iterativ ein Gesamtmodell entwickeln zu können.

Modellbedatung: Die Modellbedatung erfolgt in einem mehrteiligen Prozess, indem Modellattribute einer Abstraktionsstufe horizontal über das gesamte Architekturmodell addiert werden. Dieser Prozessschritt lässt sich iterieren, um das gesamte Modell sukzessiv auf die gewünschte Komplexitätsstufe zu verfeinern. Abstrakt lässt sich dieses Vorgehen analog zu /[Bor94]/ formalisieren.

Das *Prozessmodell* zur iterativen Anwendung des entwickelten Konzepts stützt sich auf die Idee ein System SYS unter gegebenen (nicht-)funktionalen Randbedingungen C (*Konditionen*) zu beschreiben. Das Prozessmodell wird dabei über die Summe aller Teilprozesse inklusive deren vorliegender Randbedingungen definiert. Bei der Weiterentwicklung mit mehreren Iterationsschritten verändert sich das Prozessmodell $P_i = (SYS_i, C_i)$ mit i Entwicklungsschritten. Dadurch lässt sich ein Transformationsschritt m auf einem Prozessmodell auch als Transformationsschritt auf dessen zugrunde gelegte Systemspezifikationen und Randbedingungen auffassen ($m(P) = P'$ mit $P' = (SYS', C')$.

Beispiel:

Ein Fahrwerk wechselt in den Modus „*Sport*“ bei Fahrzeuggeschwindigkeit $v > 80km/h$. Der Wechsel zwischen den Fahrwerksmodi erfolgt bei intakter Bereitstellung des Fahrzeugniveaus in einem vorgegebenen Zeitfenster. Das vorliegende Fahrzeugniveau wird dabei aus der Menge Z aller eingehenden Höhenstandssensorwerte errechnet.

$$P = (k_{Daempfer} = k_{Sky} * \frac{v_{Aufbau}}{v_{Relativ}}, \{v > 80, pos(Z), \forall z \in Z(\Delta t(z) < 100ms)\})$$

4.2.2.2. Der Funktionsbegriff und Systemeigenschaften

Zum Verständnis der nachfolgenden formalen Darstellung des Modellierungskonzepts für verteilte flexible zeitgesteuerte Architekturen wird als Ausgangsbasis der Begriff *Funktion* definiert. Zusätzlich erfolgt die Einordnung der Eigenschaften eines Systems in Bezug auf transformierende Funktionen.

Definition 4.2.1 (Funktion) *Eine Funktion beschreibt eine linkstotale und rechtseindeutige Relation f zwischen zwei Mengen D und Z. Dabei ist f eine Teilmenge des kartesischen Produkts $D \times Z$.*

Im Kontext der Systembeschreibung verteilter eingebetteter Systeme bildet eine Funktion einen transformierenden Schritt auf Eingabedaten und erzeugt zur Funktionsbeschreibung konforme Ausgabedaten. Falls die Eingabedaten permanent und zeitlich unbeschränkt digital (in diskreten Abständen) auf Ausgabedaten abgebildet werden, so wird von einer Transformation auf unendlichen Datenströmen gesprochen. Dieser transformierende Schritt lässt sich system- und umgebungsspezifisch als Algorithmus umsetzen.

□

Offensichtlich stehen Funkionen stets in enger Beziehung zu einem System. Allgemein lässt sich der Begriff *System* und dessen Eigenschaften im Zusammenhang mit verteilten Systemen wie folgt beschreiben /[Saw]/:

Definition 4.2.2 (System) *Ein System wird durch die Systemgrenze, den (meist) zeitabhängigen in arbiträrer Anzahl vorliegenden Ein- und Ausgangsgrößen $u(t)$, $y(t)$ sowie durch das mathematische Modell $D(y) = u$ bzw. $y = G(u)$ definiert. Die Abbildungsvorschriften (Operatoren) D und G beschreiben das Ein-/ Ausgangsverhalten in impliziter bzw. expliziter Form und enthalten die Parameter p des Systems*[1].

$$SYS : U \rightarrow YmitU = \{u_1, u_2, ..., u_m\}, Y = \{u_1, u_2, ..., u_n\}$$

□

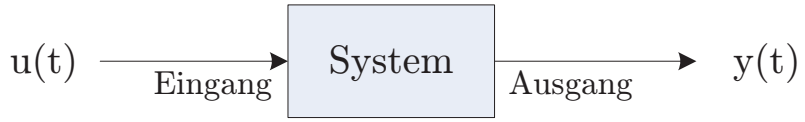


Abbildung 4.8.: Grunddarstellung eines Systems aus funktionaler Sicht mit arbiträrer Anzahl an Ein- und Ausgängen

Nach folgenden Kriterien lassen sich Systeme generell klassifizieren:

Dynamisch/statisch: Dynamische Systeme speichern den Verlauf einer Anregung $u(t)$ und schwingen sich auf diese ein. Das Verhalten bezieht über den Verlauf der Zeit und lässt sich daher in Form von Differentialgleichungen abbilden. Statische Systeme agieren ohne das Abspeichern des Verlaufs der Anregung $u(t)$. Eine Abbildung erfolgt auf Basis algebraischer Gleichungen.

Zeitkontinuierlich/zeitdiskret: Ein System heißt zeitkontinuierlich, wenn die Signalwerte zu jedem beliebigem Zeitpunkt gegeben sind, etwa bei einem Feder-Masse-System. Ein System heißt zeitdiskret, wenn die Signalwerte nur zu diskreten Zeitpunkten vorliegen.

Wertekontinuierlich/wertediskret: Wertekontinuierliche Systeme unterliegen keinen Einschränkungen innerhalb des Wertebereichs zur Abbildung einer Funktion über bedatete Systemvariablen. Physikalische Prozesse lassen sich daher exakt auf zugeordnete physikalische Werte abbilden. Werte- oder ereignisdiskrete Systeme reduzieren den Wertebereich auf fixe quantisierte Werte. Der Zeitpunkt einer Werteänderung wird als Ereignis bezeichnet.

Linear/nichtlinear: Lineare Systeme basieren auf Systemgleichungen, die die Verstärkungseigenschaft (*Homogenität*) und die Additions-Eigenschaft (*Superposition*) erfüllen. Diese Eigenschaften sind dabei für arbiträre Systemstimulationen $u(t)$ zu allen Zeiten t erfüllt.

Homogenität:
$Gcu(t) = cGu(t)$ mit c - beliebige Konstante,

[1] Die Operatoren D und G werden je nach Systemeigenschaft als Differentialterme in Abhängigkeit der Zeit oder als algebraische Gleichungen spezifiziert.

Superposition:
$Gu_1(t) + u_2(t) = Gu_1(t) + Gu_2(t)$ mit $u_{1,2}(t)$ - beliebige Signale.

Nichtlineare Systeme ergeben sich dementsprechend etwa bei Potenz-, Multiplikations- sowie Divisionsoperationen von mindestens zwei Signalen $u_{1,2}(t)$.

Zeitinvariant/zeitvariant: Ein System wird als zeitinvariant bezeichnet, wenn die zugrunde liegende Systemgleichungen ein identisches Verhalten bei identischen Eingangssignalen unabhängig von einem Zeitpunkt (*Verschiebungseigenschaft*) aufweisen. Falls Systemparameter bzw. Koeffizienten in den Gleichungen ihre Werte abhängig mit dem Verlauf der Zeit verändern, wird der Bereich der zeitvarianten Systeme erfasst.

Kausal/nicht kausal: Ein System heißt kausal, wenn die Ausgangsgrößen ausschließlich vom Verlauf der anliegenden Eingangsgrößen bis zu einem Zeitpunkt t abhängen, wobei gilt

$\exists y \in Y : y(t) = Gu(t') \rightarrow t \geq t'$.

Nichtkausale Systemgleichungen sind im Gegenzug als realitätsfern einzustufen und spielen im Zusammenhang mit reaktiven Systemen im Kontext physikalischer Prozesse keine Rolle.

Finit/infinit: Finite Modelle eines System sind limitiert in der maximal möglichen Ausprägung der Systemteile/-strukturen. Finite Systeme sind im Bereich rechnergestützter Systeme bedeutend, da deren Ressourcen begrenzt vorhanden sind. Infinite Modelle unterliegen im Gegenzug keiner Beschränkung. Beispielhaft lassen sich die Automaten aus dem zeit- und wertdiskreten Bereich anführen, die in endlicher oder unendlicher Form konstruierbar sind.

Übertragungsfunktion eines zeitkontinuierlichen linearen Systems:

Durch eine Laplace-Transformation ist es beispielsweise möglich ein System aus dem Zeit- in den Bildbereich abzubilden. Die Übertragungsfunktion hat die Eigenschaft, dass kein lineares Differentialgleichungssystem (LGS) im Übertragungsglied zu lösen ist.

Zustandsgleichungen eines zeitkontinuierlichen linearen Systems:

Eine Möglichkeit ein *lineares System* zu beschreiben basiert auf linearen Gleichungen /[Föl05]/[2]:

$$\dot{x}(t) = A * x(t) + B * u(t),$$
$$y(t) = C * x(t) + D * u(t).$$

Dabei beschreibt A eine Systemmatrix, B eine Steuerungs- oder Eingangsmatrix, C eine Beobachtungs- oder Ausgangsmatrix und D eine Durchgangsmatrix. Sind diese Matrizen *konstant*, so ergibt sich ein *lineares und zeitinvariantes* System. Zur Lösung muss $\dot{x}(t)$ integriert werden, um $x(t)$ berechnen zu können.

[2]Hinweis: x und y liegen in Vektorform vor!

$$x(t) = \int_{t_0}^{t} \dot{x}(\tau) d\tau + x(t_0).$$

Der Zustandsraum für ein lineares System lässt sich in einem Signalflussdiagramm darstellen (s. Abb. 4.9).

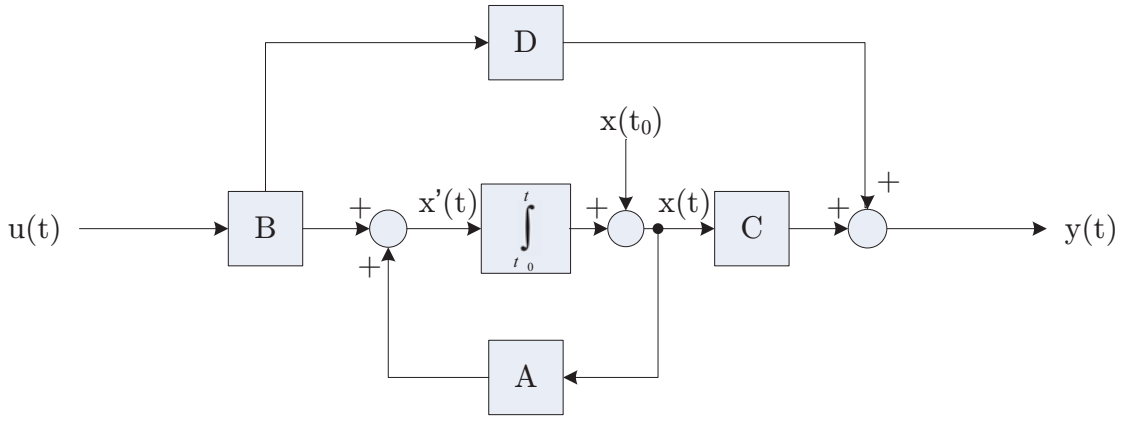


Abbildung 4.9.: Signalflussdiagramm zur Darstellung des Zustandsraums eines linearen Systems

Zustandsgleichungen eines zeitdiskreten linearen Systems:

Falls die Darstellung der Zeit statt als vektorwertige Funktion mit einer Folge von Vektoren beschrieben wird, so lässt sich das dynamische lineare System für den zeitdiskreten Bereich erfassen. Dieser Fall erlangt seine Relevanz im Kontext der Abtastung innerhalb von Digitalrechensystemen, die im Fahrzeugumfeld vorzufinden sind. Als Konsequenz werden die Zustandsdifferentialgleichungen für den kontinuierlichen Fall durch Differenzengleichungen für den diskreten Fall substituiert ($x(t) \rightarrow x(k), k \in Z$).

Während sich zeitkontinuierliche Systeme im Bereich von Anwendungssimulationen eignen, zeigen sich Vorteile der zeitdiskreten Systeme im Bereich des Einsatzes in Echtzeitsystemen.

4.2.2.3. Funktionsnetzmodellierung

Zur Spezifikation logischer E/E-Systemarchitekturen eignet sich der Funktionsnetzansatz. Der Kerns dieser Methode bezieht sich auf die im Fahrzeug vorliegende Anwendungsdomäne der *reaktiven Systeme*. In diesem Umfeld ist der Umgang mit automatisierten Regel- und Steuerungsapplikationen und elektronischen Schaltkreisen etabliert. Dabei sind Operationen zur Transformation von Datenflüssen und auf höherer Ebene der Einsatz von booleschen Funktionen und Transferfunktionen in Form von Blockdiagrammstrukturen von Bedeutung. Mithilfe dynamischer Gleichungen lassen sich dabei diese Strukturen kapseln. Im Bereich der Softwareentwicklung korreliert dieser Ansatz

mit dem Konzept der Modellierung von Datenflussdiagrammen /[Kah74], [McG82]/.

Das Hauptaugenmerk bei der Funktionsnetzbildung liegt auf der Strukturierung und Modularisierung integrierter E/E-Funktionen in Komponenten. Dabei wird von der Sicht der konkreten Funktionsalgorithmik, dass heißt von der konkreten Ausprägung der Funktion, beispielsweise einem Regelschleifenmodell zur Fahrwerksdämpfung, abstrahiert.

Definition 4.2.3 (Funktionsnetz) *Das Funktionsnetz fn aus der Menge FN (**function net**) aller Funktionsnetze wird durch eine Menge an Funktionsblöcken FC (**functional component**) beschrieben, welche über eine Menge an Sende- und Empfangsschnittstellen PO (**ports**) Daten senden und empfangen können. Das Funktionsnetz fn ist dabei aus einer arbiträren Submenge aus FC zusammengesetzt und bildet einen gerichteten Graphen. Dadurch lässt sich die logische Systemarchitektur SYS_{logic} wie folgt definieren:*

$$SYS_{logic} : FN \rightarrow \mathcal{P}(FC) \text{ mit } \mathcal{P}(FC) \text{ als Potenzmenge über } FC,$$

Das bedeutet für ein Funktionsnetz fn aus FN:

$$SYS_{logic}(fn) = FC_{fn} \text{ mit } fn \in FN, FC_{fn} \subseteq FC, |FC_{fn}| \geq 2$$

*Die Ports PO unterscheiden sich in Sende- und Empfangsports SP (**sender port**) und RP (**receiver port**), wobei SP und RP in PO strikt disjunkt auftreten.*

$$PO = SP \cup RP \text{ mit } SP \cap RP = \emptyset$$

Jeder Funktionsblock fc besitzt eine arbiträre Anzahl an Sende- und Empfangsports, die sich mit folgenden Relationen

$$s : FC \rightarrow \mathcal{P}(SP) \text{ mit } s(fc) \subseteq PO, |SP| \geq 0,$$
$$r : FC \rightarrow \mathcal{P}(RP) \text{ mit } r(fc) \subseteq PO, |RP| \geq 1$$

zuweisen lassen. Zwischen den Sende- und Empfangsports können injektive unidirektionale Kommunikationsbeziehungen für ein Funktionsnetz definiert werden:

$$\pi : SP_{fn} \rightarrow RP_{fn} \text{ mit } SP_{fn} = \bigcup_{fc \in FC_{fn}} s(fc) \subseteq PO,$$
$$RP_{fn} \subseteq \bigcup_{fc \in FC_{fn}} r(fc) \subseteq PO$$

*Die durch die Funktion π definierten Relationen zwischen den Sende- und Empfangsports werden als Konnektoren aus der Menge SRC (**send-receive-connector**) zum Datenaustausch aufgefasst.*

$$SRC_{fn} = \bigcup_{fc_{send} \in FC_{fn}} \bigcup_{sp \in s(fc_{send})} \{((fc_{send}, sp), (fc_{receive}, \pi(sp))) \; mit \; \pi(sp) \in r(fc_{receive})\}$$
$$, |SRC_{fn} \geq 1|$$

Ein Element src aus SRC_{fn} des Funktionsnetzes FN wird durch ein Zweitupel mit jeweils einem Funktionsblock aus FC als Sender und Empfänger sowie zweier Ports eindeutig definiert.

$$src = ((fc_{send}, sp), (fc_{receive}, rp))$$
$$: src \in SRC_{fn}, fc_{send}, fc_{receive} \in FC_{fn}, sp \in SP_{fn}, rp \in RP_{fn}$$

Da fc_{send} und $fc_{receive}$ nicht zwangsläufig unterschiedliche Funktionsblöcke darstellen müssen, wird eine Rückkopplung im Funktionsnetz nicht explizit ausgeschlossen.
In einer integrierten Betrachtung erzeugen Funktionsnetze die logische Architektur eines Systems mit

- *syntaktischen Schnittstellen (RP → SP),*
- *abstrahiertem Verhalten (=Funktionen) (FC),*
- *der Möglichkeit zur Komposition von Systemteilen.*

□

4.2.2.4. Abstraktion und Verfeinerung

Ein entscheidender Vorteil der Funktionsnetzmodellierung ergibt sich aus den Konzepten der Abstraktion und Verfeinerung bereits entwickelter Funktionsnetze. Die Grundidee dabei basiert auf der Interpretation eines Funktionsnetzes als hierarchischen Funktionsblock hfc, wobei Abhängigkeiten zwischen der Menge der hierarchischen Funktionsblöcke und der Menge der allgemein zugrundeliegenden Funktionsblöcke existieren (vgl. Abb. 4.10).

Unter HFC (***hierarchical functional component***) versteht sich die Menge aller hierarchisch komponierten Funktionsblöcke. Bei der Abstraktion eines Funktionsnetzes werden Substrukturen, die sich als Teilfunktionsnetz auffassen lassen, durch einen hierarchischen Funktionsblock hfc substituiert. Dadurch gilt für die neue gültige Menge aller Funktionsblöcke FC':

$$FC' = HFC \cup FC'' : FC'' \subset FC, HFC \neq \emptyset$$

Ein hierarchischer Funktionsblock enthält die in der Menge FC' abstrahierten Elemente aus der ursprünglichen Funktionsblockmenge FC.

$$SYS_{logic}(hfn) = FC_{hfn} \text{ mit } FC_{hfn} \subseteq FC', FN' = FN \cup \{hfn\}$$

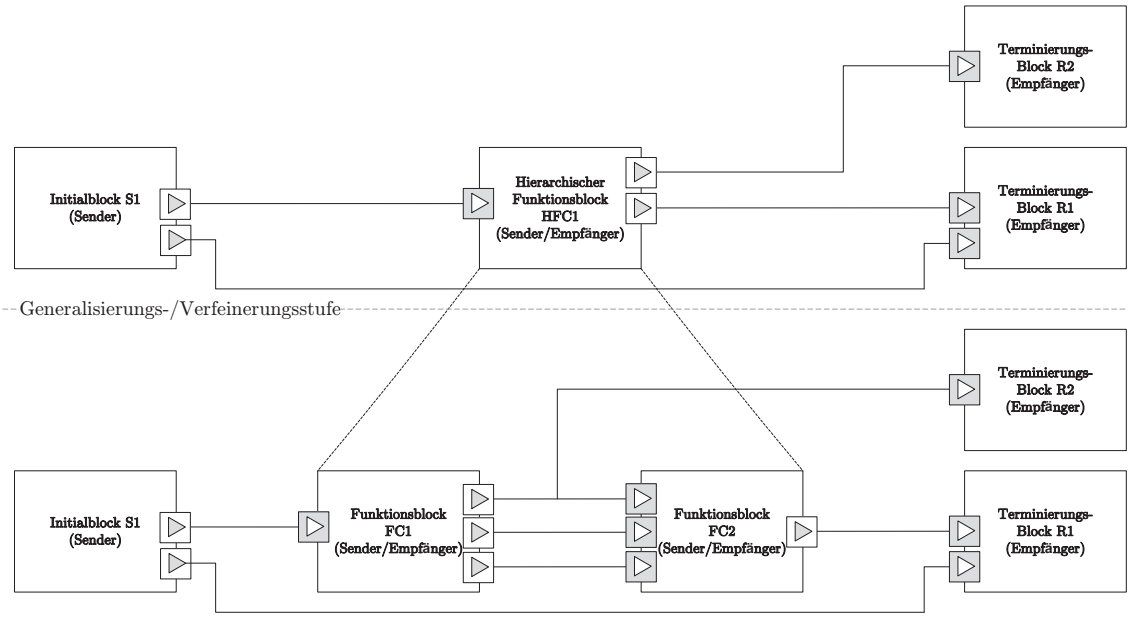


Abbildung 4.10.: Generalisierungs-/Verfeinerungsrelation zwischen HFCs und FCs

Die Zusammenhänge zwischen einem generalisierten und einem verfeinerten Funktionsnetz werden durch die Verwendung der Empfangsports RP'_{fn} eines hierarchischen Funktionsnetzes hfn innerhalb eines verfeinerten Funktionsnetzes fn offensichtlich. Die Ports RP'_{fn} werden dabei durch eine partielle Menge der Sendeports aus der Menge SP_{hfn} des generalisierten Funktionsnetzes hfn bedatet:

$$RP'_{fn} = \bigcup_{fc \in FC_{fn}} r(fc) \backslash RP_{fn} \text{ mit } RP'_{fn} \subseteq \bigcup_{fc \in FC_{hfn}} r(fc) \subseteq RP_{hfn},$$
$$\pi : SP'_{hfn} \rightarrow RP'_{fn} \text{ mit } SP'_{hfn} \subseteq \bigcup_{fc \in FC_{hfn}} s(fc)$$

Allgemein sind Funktionsnetze mit Vierfach-Tupeln über die Mengen der Funktionsblöcke, der Senderelationen zwischen den zugeordneten Sende-/Empfangsports und der referenzierten Sende-/Empfangsports (FC, π, SP, RP) vollständig beschrieben.

Für den Zweck einer Verfeinerung eines Funktionsnetzes lässt sich ein Funktionsblock $fb \in FC$ durch eine eigene Funktionsnetzstruktur fn substituieren. In der verfeinerten Sicht wird ein allgemeines Funktionsnetz um Funktionsblöcke, Senderelationen und Sende-/Empfangsports erweitert. Dabei steht ein Funktionsnetz aus der Menge FN in Relation ($\sim$) zu einem Vierfachtupel mit Teilmengen aus FC, SP und RP sowie der Sendefunktion π:

Funktionsnetz Verfeinerungsstufe i:

$$fn_i \sim (FC, \pi, SP, RP)$$

Verfeinerungsschritt auf fc:

$$fc \sim (FC_{fc}, \pi_{fc}, SP_{fc}, RP_{fc}) \text{ mit } fc \in FC$$

Funktionsnetz Verfeinerungsstufe $i+1$:

$$fn_{i+1} \sim ((FC \backslash fc) \cup FC_{fc}, \pi \cup \pi_{fc}, SP \cup SP_{fc}, RP \cup RP_{fc})$$

Über eine Kapselung Ge_i (***generalization***) werden die Mengen eines flachen Funktionsnetzes fn durch die Mengen eines hierarchischen Funktionsnetzes hfn ersetzt. In einem Verfeinerungsschritt $\sqsubseteq_i$ (***refinement***) lassen sich einzelne Funktionsblöcke durch Subfunktionsnetzstrukturen substituieren. In beiden Fällen ergibt sich eine wachsende hierarchische Struktur, die sich in verschiedene Ebenen untergliedern lässt. Je nach Weiterentwicklung des Funktionsnetzes entsteht dadurch, unabhängig von der Anzahl an Kapselungen/Verfeinerungen, ein zyklenfreier Graph, wobei Rückkopplungen im System bei Bedarf umgesetzt werden können. Gleichzeitig werden redundante Kommunikationspfade im Netzwerk vermieden und als Konsequenz die Konstruktion unsauberer Funktionsschnittstellen unterbunden. Zusätzlich werden bei der Graphentraversierung zur Analyse der Kommunikationspfade inkonsistente oder unvollständige Funktionsketten identifiziert (Fehlen eines initialen Senderblocks oder eines finalen Empfängerblocks).

Abstraktionsfunktion Ge_i:

$$Ge_i : FC \rightarrow HFC : \exists fc_i, fc_j \in FC(Ge_i(fc_i) = Ge_i(fc_j) \rightarrow fc_i \sim fc_j)$$

Dabei gilt es folgende Forderung für Ge_i zu garantieren:

Surjektivität:

$$\forall hfc \in HFC \exists fc \in FC : Ge_i(fc) = hfc$$

Dadurch bleibt gewährleistet, dass für jeden abstrakten Funktionsblock stets ein konkreter Funktionsblock im Modell existieren muss.

Einzigartigkeit:

$$\forall fc \in FC : |[fc]| = 1 \rightarrow Ge_i(fc) = fc \text{ mit}$$
$$[fc] = \left\{ fc' \in FC, Ge_i(fc) = Ge_i(fc') \right\}$$

Jedem abstrakten Funktionsblock liegt die Verschmelzung von mindestens zwei konkreten Funktionsblöcken zugrunde. Singuläre Funktionsblöcke einer Äquivalenzklasse ($[fc]$) lassen sich in einer hierarchischen Sicht ausschließlich auf sich selbst abbilden.

$$\forall x \in FC : (Ge_{i+1}(x) = x) \rightarrow x \in FC'$$
$$: fn_i \sim (FC, \pi, SP, RP), fn_{i+1} \sim (FC', \pi', SP', RP'), HFC \subset FC', FC' \cap FC \neq \emptyset$$

In diesem Zusammenhang bleiben alle nicht-transformierten Funktionsblöcke bei einem Abstraktionsschritt $i+1$ erhalten.

Aus Sicht einer übergeordneten Funktionsblockmenge

$$V = \{fc_1, fc_2, fc_3, ...\} \text{ mit } fc_i \in FC \cup HFC, i = \mathbb{N}_0^+$$

mit sämtlichen konkreten und abstrahierten Funktionsblöcken lässt sich eine Verfeinerungsrelation $\sqsubseteq_i \subseteq V \times V$ als Gegenstück zur Abstraktionsfunktion Ge_i definieren. Dabei gilt:

$$\forall fc, fc' \in V : fc \sqsubseteq_i fc' \Leftrightarrow Ge_i(fc) = fc'$$

Dadurch werden eindeutige Zuordnungen zwischen Funktionsblöcke unterschiedlicher Abstraktionsstufen garantiert.

Über mehrere Ebenen $i, i+1, ...$ hinweg betrachtet existiert durch $\sqsubseteq_i$ eine partielle Ordnung sowie eine Abhängigkeitsstruktur der Funktionsblöcke in Form eines Baums. Dabei existiert für jeden Subfunktionsblock stets nur jeweils ein zugeordneter abstrakter Funktionsblock:

$$\forall fc, fc', fc'' \in V : fc \sqsubseteq_i fc' \wedge fc \sqsubseteq_i fc'' \rightarrow fc' \sqsubseteq_i fc'' \vee fc'' \sqsubseteq_i fc'$$

4.2.2.5. Zeitkettenmodell

Ein signifikanter Vorteil der Funktionsnetzmodellierung zur Beschreibung logischer E/E-Architekturen entsteht bei der Definition eines Zeitmodells als Teil der Funktionsspezifikation. Diese Ideen finden sich in ähnlicher Form in den folgenden Ansätzen /[RZE+02], [AUT07b]/ wieder und werden partiell in der Konzeption der Zeitmodellierung in dieser Arbeit berücksichtigt. Das Sendeverhalten in der Zeitmodellierung verwendet periodische und ereignisgesteuerte Übertragungskonzepte. Dabei gilt es im ersten Fall Ungenauigkeiten in der Übertragung (*Jitter*) sowie mögliche Übertragungsperioden zu spezifizieren. Die zweite Variante erfordert Angaben über Minimalwerte (Sperrzeiten) und Maximalwerte (Zeitüberschreitungen) zwischen dem Auftreten von Ereignissen.

Einbettung in die funktionale Modellierung:

1. Kommunikation zwischen sendende SW-Komponente A und empfangende SW-Komponente B,

2. Konnektor spezifiziert Zeitfluss zwischen Senderport S und Empfängerport R,

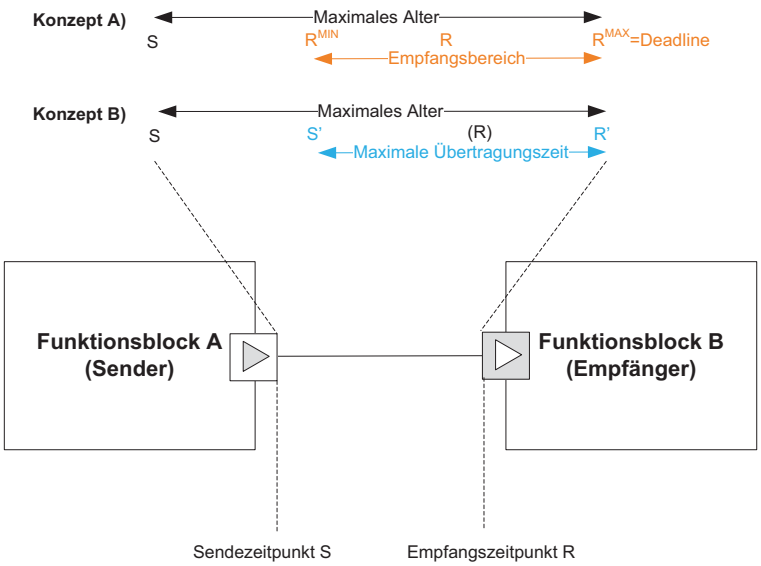


Abbildung 4.11.: Darstellung zweier Modelle zur Spezifikation des Zeitverhaltens an den Kommunikationsschnittstellen zweier Softwarekomponenten in der Funktionsnetzebene

3. Min-Max-Modell: $\overline{SR^{Min}} < \overline{SR} < \overline{SR^{Max}} = MaxAge$,
4. Max-Age-Modell: $\overline{SR} = \overline{S'R'} < \overline{SS'} + \overline{S'R'} = \overline{SR'} < MaxAge$.

Im Gegensatz zur Abstraktion/Verfeinerung steht bei der Zeitanalyse die sequentielle Verkettung der Funktionskomponenten im Vordergrund.

Definition 4.2.4 (Zeitkette) *Eine Zeitkette TC (**time chain**) beschreibt den zeitlichen Verlauf einer Signalkommunikation auf Basis der Spezifikation eines verteilten Systems, etwa einem Funktionsnetz FN.*

Die Struktur entsteht durch die Relation zwischen einer sendenden und einer empfangenden Komponente. Diese Komponenten liegen als Funktionsblöcke FC vor. Dabei wird die Basisstruktur eines Funktionsblocks um Zeitinformationen erweitert:

Kommunikationspfad:

$$P = (fc_1, fc_2, ..., fc_n) \text{ mit } fc_i, fc_{i+1} \text{ sind adjazent, } i \in [0; n-1],$$

Zeitabbildung:

$$d : C \rightarrow \mathbb{R}_0^+ \text{ mit } C = FC \cup SRC,$$

Zeitkette:

$$TC = (d(fc_1), d(src_{12}), d(fc_2), d(src_{23}), ..., d(fc_n)),$$
$$\text{mit } d(fc_i) \text{ als „deadline“, } d(src_{ij}) \text{ als „sending latency“}$$

*Die in den Funktionsblöcken eingebettete Funktion, etwa eine Regelschleife oder eine Steuerung, toleriert einen zeitlichen Verzug bei der Auswertung bis zu einer bestimmten Frist (**deadline**). Durch diese Restriktion lässt sich eine genauere Spezifikation der Signalübertragung an der*

Sende-/Empfangsbeziehung SRC ableiten. Dabei muss die Ausprägung der SRC die deadline des sendenden und des empfangenden Funktionsblocks unter allen Betriebszuständen einhalten. Die SRC unterliegt den Eigenschaften der technischen Übertragungsform zwischen den FCs (bspw. dem Übertragungskonzept (ereignis-/zeitgesteuert), des Sendetyps (spontan/zyklisch), etc.).

Durch die Addition der im FN spezifizierten Zeitinformationen ergeben sich TCs, die entsprechend der Gesamtsystemanforderung ausgelegt werden können.

□

Prinzipiell gleicht die Zeitkettenmodellierung dem Konzept der Transformation von Datenflussgraphen (DFG) in Problemgraphen bei der Partitionierung im Entwicklungsfeld des Hardware-/Softwarecodesigns /[PT07]/ (s. Abs. 4.2.2.8/s. Abb. 4.12).

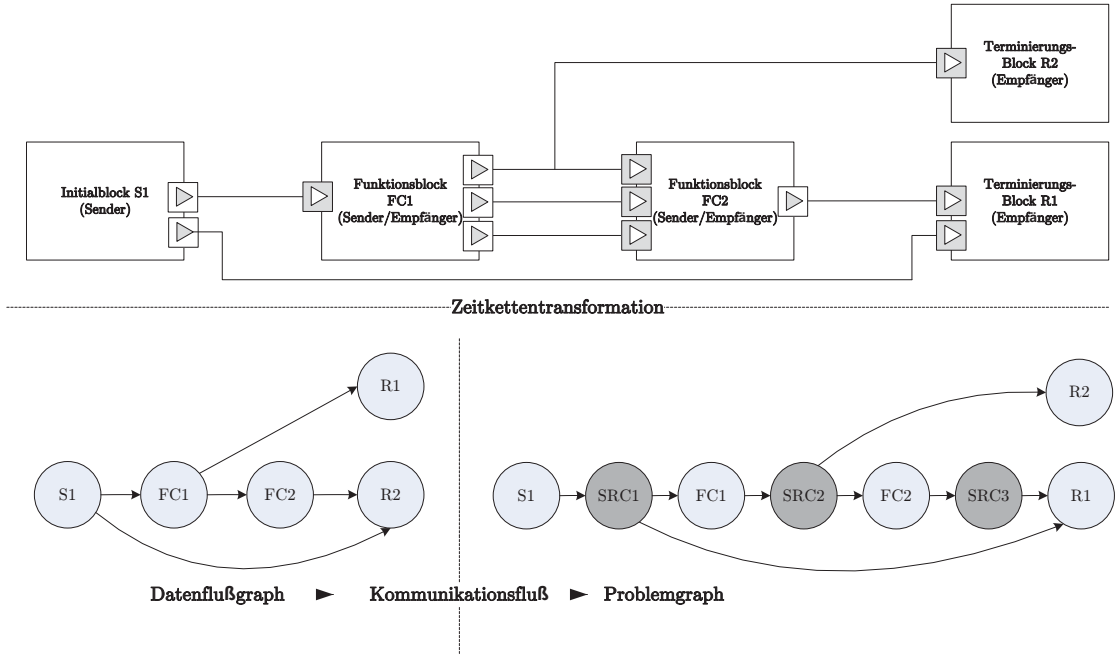


Abbildung 4.12.: Transformation eines Datenflussgraphen zu einem Problemgraphen auf Basis des Zeitkettenansatzes

Die signaltransformierenden Knoten lassen sich durch die Hinzunahme softwaretechnischer Eigenschaften zeitlich verfeinern, etwa im Bereich eines Betriebssystems (Unterbrechungskonzept, Prozessblockierung, präemptives Multitasking, Prozessperioden, Jitter) oder der Softwareinitialisierung (Offsets, Anlaufverzögerung). Eine feingranulare Analyse heterogener Einflüsse auf eine erweiterte Zeitkettenmodellierung ist eine umfassende Aufgabe, welche nicht im Rahmen dieser Arbeit behandelt wird. Das EU-Projekt TIMMO beschäftigt sich mit dieser Thematik im Kontext automotiver Regelungssysteme zur Ergänzung des AUTOSAR-Standards um Timing-Aspekte /[ERG08], [Ric08]/. Dieses Konsortium untersucht dabei auch die Anwendbarkeit existierender Lösungen aus Sicht der Modellierung für eingebettete Echtzeitsysteme, beispielsweise das UML-Profil MARTE /[FBSG07]/. Aktuell liegen noch keine offiziellen Timing-Spezifikationen des TIMMO-Konsortiums vor.

4.2.2.6. Schnittstellenspezifikation

Ein wesentlicher Aspekt des Architekturdesigns bezieht sich auf das saubere, in diesem Sinn korrekte und vollständige, Design der internen und externen Architekturschnittstellen. Dabei lässt sich der allgemeine Schnittstellenbegriff in Unterbegriffe gliedern /[Kop97]/:

1. *Kontrolleigenschaften* beziehen sich auf die Handhabung kausaler Aspekte im Umgang mit Steuerungssignalen an der Schnittstelle. Beispielsweise bezieht sich die Aktivierung einzelner Tasks innerhalb eines Steuergeräts oder der Wechsel zwischen Systemmodi allgemein auf diese zu spezifizierenden Kontrolleigenschaften.

2. *Dateneigenschaften* spezifizieren die konkrete Struktur und Semantik der an der Schnittstellen übergebenen Datenelemente.

3. *Temporale Eigenschaften* definieren Restriktionen für Daten und Kontrollsignale im Zeitbereich an der betrachteten Systemschnittstelle.

4. Das *funktionale Vorhaben* dient der Beschreibung notwendig zu erfüllender Eigenschaften in der Anwendungsdomäne. Als Beispiel lassen sich Vorgaben aus Sicht der funktionalen Sicherheit /[IEC05], [ISO07], [Che08]/ innerhalb einer Anwendungsdomäne anführen. Entwickelte Funktionen garantieren diese Eigenschaften gegenüber den an der Schnittstelle interagierenden Partnern.

Anhand der dreiteiligen Klassifikation wird das Schnittstellenverhalten für das vorliegende Konzept erläutert:

Kontrolleigenschaften: Die Signifikanz der Kontrolleigenschaften eines betrachteten Systems steigt, falls dessen Grundkonzept einen stark interaktiven Charakter hat oder unter Berücksichtigung hoher Sicherheitsanforderungen /[IEC05], [ISO07], [Che08]/ zu entwickeln ist. Im ersten Fall bezieht sich der Ansatz auf Client-Server-Systeme, bei denen gezielt auf Anfragesignale/-botschaften (*request*) reagiert wird (*response*). Bezogen auf ein FN wird der Ausgangsport FC_Out_i mit $i \in \mathbb{N}_0^+$ ausschließlich in Abhängigkeit einer *request*-Anfrage am Eingangsport $FC\_In_i = req$ mit $i \in \mathbb{N}_0^+$ bedatet (vgl. /[BS01]/).

Dateneigenschaften: Zu den Dateneigenschaften zählen die aus der Informationstheorie bekannten Typen (*Integer, Real, Boolean, etc.*), die Signalauflösung, Signalversätze (*offsets*) und optional die Signalminimal- und maximalwerte[3].

Temporale Eigenschaften: An den Schnittstellen der FC findet der eigentliche Signalaustausch statt. Darunter versteht sich das Senden und das Empfangen von arbiträr definierbaren Daten über Sende- und Empfangsports. Dabei findet eine Transformation der temporalen Eigenschaften des eigentlichen Signals konform zu den Anforderungen der zugeordneten Übertragungsform (CAN, FlexRay, etc.) statt.

Die temporalen Eigenschaften auf Signalbasis lassen sich formal spezifizieren.

[3] Im Bereich der automotiven Systeme gibt es die Unterscheidung zwischen Rohwert- und Filtersignalen. Anforderungen hinsichtlich der Einhaltung von Abtastgrenzen (*Nyquist-Shannon-Abtasttheorem*) oder der Verwendung von Tiefpassfiltern bleiben in diesem Ansatz unberücksichtigt /[WP07], [Föl05]/.

Definition 4.2.5 (Signalzeitverhalten (signal timing)) *Das Signalzeitverhalten t_i wird auf einem Architektursignal $i \in S$ aus der Menge S aller Signale einer E/E-Architektur definiert. t_i bildet dabei ein 4-Tupel, bestehend aus dem Signaltyp ST, dem Subtyp SST, der Signallänge SL und der Signalphase SP.*

$$t : S \rightarrow ST \times SST \times SL \times SP,$$
$$\forall i \in S : t(i) = (st_i, sst_i, sl_i, sp_i) \wedge st_i = spontan \rightarrow sp_i = \perp$$
$$mit\ st_i, sl_i \neq \perp$$

□

Der Signaltyp lässt sich untergliedern in spontane oder zyklische Kommunikation. Der Subtyp spezifiziert eine optionale Ergänzung des Signaltyps für hybride Signaltypen als Binärrelation auf den Mengen $ST \times SST$ mit $ST = \{zyklisch, spontan\}$ und $SST = \{zyklisch, spontan, aktiv, \perp\}$ (s. Tab. 4.2)[4].

Tx-Rx-Beziehung			Temporaleigenschaften			
Name	Sender	Empfänger	ST	SST	SL (Bit)	SP (ms)
Signal *a*	ECU1	ECU2,3	zyklisch	$\perp$	16	20
Signal *b*	ECU1	ECU4	zyklisch	spontan	64	10
Signal *c*	ECU1	∞	zyklisch	zyklisch	8	20/10
Signal *d*	ECU2	ECU1	zyklisch	spontan	8	40
Signal *e*	ECU2	ECU1,4	spontan	$\perp$	32	$\perp$
Signal *f*	ECU2	∞	zyklisch	$\perp$	24	5
Signal *g*	ECU3	ECU2	zyklisch	zyklisch	80	80/5
Signal *h*	ECU4	ECU1,2	zyklisch	spontan	16	40
Signal *i*	ECU4	ECU2,3	zyklisch	$\perp$	32	10
Signal *j*	ECU4	∞	spontan	$\perp$	24	$\perp$

Tabelle 4.2.: Beispiel zur Übersicht der Kriterien zur Attributierung eines Funktionsnetzes

Auf Basis der vollständig spezifizierten temporalen Eigenschaften sämtlicher Signale erfolgt eine Transformation dieser Darstellung in eine buskonforme Abbildung. FlexRay-Systeme erfordern dabei die Zuordnung auf die einzelnen Übertragungssegmente (*statisch*, *dynamisch*) und die Platzierung an einer passenden Stelle im Gesamtschedule (*scheduling*). Es existiert kein eindeutiger, optimaler Algorithmus für diese Transformation unter Berücksichtigung von Synchronitätsanforderungen bei der Signalübertragung. Die Berechnungskomplexität des Schedulingprozesses für die Abbildung der Signale zu Busbotschaften zählt zu den NP-vollständigen Problemen /[NG97], [GJ79]/. Die Entwicklung proprietärer Algorithmen auf Basis unterschiedlicher Heuristiken zur Verteilung von Botschaften auf dem Schedule, etwa mit genetischen Algorithmen /[SP96]/ bildet dabei einen eigenen Forschungszweig, der kein zentrales Thema innerhalb dieser Arbeit stellt. An dieser Stelle wird auf Forschungsergebnisse weiterverwiesen /[Nos96], [NG97]/.

[4]Die Signaltypen werden als Abstraktion von den etablierten Sendetypen aus /[Que06], [Vec07]/ abgeleitet. Von einer feingranularen Differenzierung durch die Konkretisierung der *active*-Bedingung, etwa bei der Interpretation des Signalwerts, der Reaktion bei Signaländerung oder der Bedatung durch Softwaremodule, wird abstrahiert.

Die Abhängigkeit zwischen Busschedule und Signalzeitverhalten wird nachfolgend auf Grundlage des entwickelten formalen Ansatzes beschrieben, auf dessen Basis Algorithmen zum Scheduling aufsetzen können.

Wie oben definiert bildet das FN eine Systemstruktur, bestehend aus FC und SRC. Die Sende-/Empfangsbeziehungen lassen sich für sämtliche FC innerhalb der Konnektivitätsmatrix abbilden. Die Zeilenvektoren beschreiben alle n Funktionsblöcke, die eine Information von einem bestimmten FC empfangen. Ergänzend beschreiben die Spaltenvektoren sämtliche Informationen, die eine einzelne Komponente von m Funkionsblöcken empfängt.

Algorithmus zur Transformation eines FN in den statischen FlexRay-Schedule:

1.) Anlegen der Konnektivitätsmatrix für das FN:

$$\begin{pmatrix} SRC_{FC11} & SRC_{FC12} & ... & SRC_{FC1n} \\ SRC_{FC21} & SRC_{FC22} & ... & SRC_{FC2n} \\ & & ... & \\ SRC_{FCm1} & SRC_{FCm2} & ... & SRC_{FCmn} \end{pmatrix}$$

2.) Eliminierung aller leeren Zeilenvektoren (keine Sender):

$$\begin{pmatrix} SRC_{FC11} & SRC_{FC12} & ... & SRC_{FC1n} \\ SRC_{FC21} & SRC_{FC22} & ... & SRC_{FC2n} \\ & & ... & \\ SRC_{FCm'1} & SRC_{FCm'2} & ... & SRC_{FCm'n} \end{pmatrix}$$

3.) Partitionierung der FC des FN und Adaption der Konnektivitätsmatrix:

$$\begin{pmatrix} SRC_{PC11} & SRC_{PC12} & ... & SRC_{PC1q} \\ SRC_{PC21} & SRC_{PC22} & ... & SRC_{PC2q} \\ & & ... & \\ SRC_{PCp1} & SRC_{PCp2} & ... & SRC_{PCpq} \end{pmatrix}$$

4.) Integration der Temporaleigenschaften (Matrixverfeinerung) auf Signalebene (x):
($\hookrightarrow$ Matrixreplikation : $\forall i \in FC : |i\_Out_j| > 0 \rightarrow \exists A_F C_i$)

$$\begin{pmatrix} SRC_{PC1_11} & SRC_{PC1_12} & ... & SRC_{PC1_1q} \\ SRC_{PC1_21} & SRC_{PC1_22} & ... & SRC_{PC1_2q} \\ & & ... & \\ SRC_{PC1_x1} & SRC_{PC1_x2} & ... & SRC_{PC1_xq} \end{pmatrix} \begin{pmatrix} SRC_{PCp_11} & SRC_{PCp_12} & ... & SRC_{PCp_1q} \\ SRC_{PCp_21} & SRC_{PCp_22} & ... & SRC_{PCp_2q} \\ & & ... & \\ SRC_{PCp_x1} & SRC_{PCp_x2} & ... & SRC_{PCp_xq} \end{pmatrix}$$

5.) Paketierung (max(Zeilenvektorsumme) $\leq$ 1) $\rightarrow$ Slotbelegung:
($\hookrightarrow$ Harmonisierung bei mehreren Empfängern.)

$\forall i \in SRC_{PCi} : max(SRC_{PCi_jk'}, SRC_{PCi_jk'+1}, ..., SRC_{PCi_jk''}) +$
$max(SRC_{PCi'_jk'}, SRC_{PCi'_jk'+1}, ..., SRC_{PCi'_jk''}) \leq 1 \rightarrow$

$SRC_{PCi_j} = ((SRC_{PCi_jk'} + SRC_{PCi'_jk'}), (SRC_{PCi_jk'+1} + SRC_{PCi'_jk'+1}), ...,$
$(SRC_{PCi_jk''} + SRC_{PCi'_jk''})) \wedge SRC_{PCi_j} = ()$ mit $1 < k', k'' < q, 1 < j < x$

Abbildung der Zeilen auf einzelne Sende-Slots im FlexRay-Schedule:

$\forall i \in SRC_{PCi} : SRC_{PCi_j} \rightarrow Slot_l$ mit $1 < i < p, 1 < j < x, 1 < l < p * x$

Beispiel:

Gegeben sei ein FN mit den vier Funktionsblöcken FC_{1-4} und den folgenden $SRC = \{SRC_1, SRC_2\}$. Die SRC sind über Ein- und Ausgabeports mit den FC verbunden:

$SRC_1 = (FC_1\_Out_1, FC_3\_In_1)$ $SRC_2 = (FC_2\_Out_1, FC_3\_In_2, FC_4\_In_1)$

Aus den SRC lässt sich im Schritt 1 die Konnektivitätsmatrix A_{FN} erstellen:

$$A_{FN} = \begin{pmatrix} \perp & 0 & 1 & 0 \\ 0 & \perp & 1 & 1 \\ 0 & 0 & \perp & 0 \\ 0 & 0 & 0 & \perp \end{pmatrix}$$

In Schritt 2 werden FC_3, FC_4 rausgefiltert, da diese reine Empfänger darstellen.

$$\begin{pmatrix} \perp & 0 & 1 & 0 \\ 0 & \perp & 1 & 1 \\ 0 & 0 & \perp & 0 \\ 0 & 0 & 0 & \perp \end{pmatrix} \rightarrow \begin{pmatrix} \perp & 0 & 1 & 0 \\ 0 & \perp & 1 & 1 \end{pmatrix}$$

In Schritt 3 wird die Konnektivitätsmatrix partitioniert. In diesem Beispiel werden die sendenden FC auf verschiedene Steuergeräte implementiert. Dabei ergibt sich keine Änderung an A_{FN}.

In Schritt 4 wird A_{FN} gemäß der temporalen Signaleigenschaften verfeinert. Dadurch wird die Bedeutung der Matrixzeilen modifiziert. Sie spezifizieren alle Sendesignale der beiden sendenden FC. Die Matrixwerte entsprechen dem Füllgrad eines einzelnen Slots des FlexRay-Schedules auf Basis einer einheitlichen Buszykluslänge[5].

$$(5ms, 10ms, 20ms, 40ms) \rightarrow (1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8})$$

[5]Signalperioden, die nicht dem Wert $basecycle^i, i \in \mathbb{N}^+$ entsprechen, müssen im dynamischen Segment übertragen werden

$$\left(\begin{array}{cccc}
\bot & 0 & 1 & 0 \\
\bot & 0 & \frac{1}{2} & 0 \\
\bot & 0 & \frac{1}{4} & 0 \\
\bot & 0 & \frac{1}{4} & 0 \\
\bot & 0 & \frac{1}{4} & 0 \\
\hline
0 & \bot & 1 & 1 \\
0 & \bot & 0 & \frac{1}{2} \\
0 & \bot & 0 & \frac{1}{2} \\
0 & \bot & \frac{1}{4} & 0 \\
0 & \bot & \frac{1}{8} & \frac{1}{8}
\end{array}\right)$$

Dabei werden durch die Ausgangsports für jede sendenden FC eine jeweils eigene Sendematrix erzeugt ($A_{FN'} \rightarrow (A_{FC1}, A_{FC2})$).

$$A_FC_1 = \left(\begin{array}{cccc|c}
\bot & 0 & 1 & 0 & 24 \\
\bot & 0 & \frac{1}{2} & 0 & 42 \\
\bot & 0 & \frac{1}{4} & 0 & 16 \\
\bot & 0 & \frac{1}{4} & 0 & 32 \\
\bot & 0 & \frac{1}{4} & 0 & 16
\end{array}\right) \quad A_FC_2 = \left(\begin{array}{cccc|c}
0 & \bot & 1 & 1 & 32 \\
0 & \bot & 0 & \frac{1}{2} & 16 \\
0 & \bot & 0 & \frac{1}{2} & 24 \\
0 & \bot & \frac{1}{4} & 0 & 8 \\
0 & \bot & \frac{1}{8} & \frac{1}{8} & 42
\end{array}\right)$$

Durch Hinzunahme der Größenangaben der versendeten Signale für jeden Zeilenvektor lassen sich die Sendevorgänge mit identischen Zykluszeiten auf Signalebene paketieren ($\left[(\frac{1}{4},16),(\frac{1}{4},32),(\frac{1}{4},16)\right] \rightarrow \left[(\frac{1}{4},32),(\frac{1}{4},32)\right]$ und $\left[(\frac{1}{2},16),(\frac{1}{2},24)\right] \rightarrow \left[(\frac{1}{2},48)\right]$: max(*signal package size*) = 48). Die Größe der Signalpakete wird anschließend wieder entfernt.

$$A_FC_1 = \left(\begin{array}{cccc|c}
\bot & 0 & 1 & 0 & 24 \\
\bot & 0 & \frac{1}{2} & 0 & 42 \\
\bot & 0 & \frac{1}{4} & 0 & 32 \\
\bot & 0 & \frac{1}{4} & 0 & 32
\end{array}\right) \quad A_FC_2 = \left(\begin{array}{cccc|c}
0 & \bot & 1 & 1 & 32 \\
0 & \bot & 0 & \frac{1}{2} & 40 \\
0 & \bot & \frac{1}{4} & 0 & 8 \\
0 & \bot & \frac{1}{8} & \frac{1}{8} & 42
\end{array}\right)$$

$$A_FC_1 = \left(\begin{array}{cccc}
\bot & 0 & 1 & 0 \\
\bot & 0 & \frac{1}{2} & 0 \\
\bot & 0 & \frac{1}{4} & 0 \\
\bot & 0 & \frac{1}{4} & 0
\end{array}\right) \quad A_FC_2 = \left(\begin{array}{cccc}
0 & \bot & 1 & 1 \\
0 & \bot & 0 & \frac{1}{2} \\
0 & \bot & \frac{1}{4} & 0 \\
0 & \bot & \frac{1}{8} & \frac{1}{8}
\end{array}\right)$$

Im 5. Schritt wird eine verfeinerte Paketierung auf Slotebene durchgeführt.

$$A_FC_1 = \left(\begin{array}{cccc|c}
\bot & 0 & 1 & 0 & 1 \\
\bot & 0 & 1 & 0 & 1
\end{array}\right) \quad A_FC_2 = \left(\begin{array}{cccc|c}
0 & \bot & 1 & 1 & 1 \\
0 & \bot & \frac{3}{8} & \frac{5}{8} & \frac{7}{8}
\end{array}\right)$$

Daraus folgt, dass sich das FN im statischen Segment innerhalb zweier statischer Slots mit jeweils einer 5ms-Botschaft für FC_1 und innerhalb zweier statischer Slots mit einer 5ms-Botschaft sowie einer Multiplexbotschaft für FC_2 im FlexRay-Schedule abbilden lässt. In der neu eingefügten rechten Spalte lässt sich dabei der Füllgrad eines jeden

Slots angeben[6].

4.2.2.7. Physikalisches Netz

In der vorgestellten Modellierungssprache erfolgt die Realisierung der technischen Systemarchitektur auf Basis einer physikalischen Netzsicht.

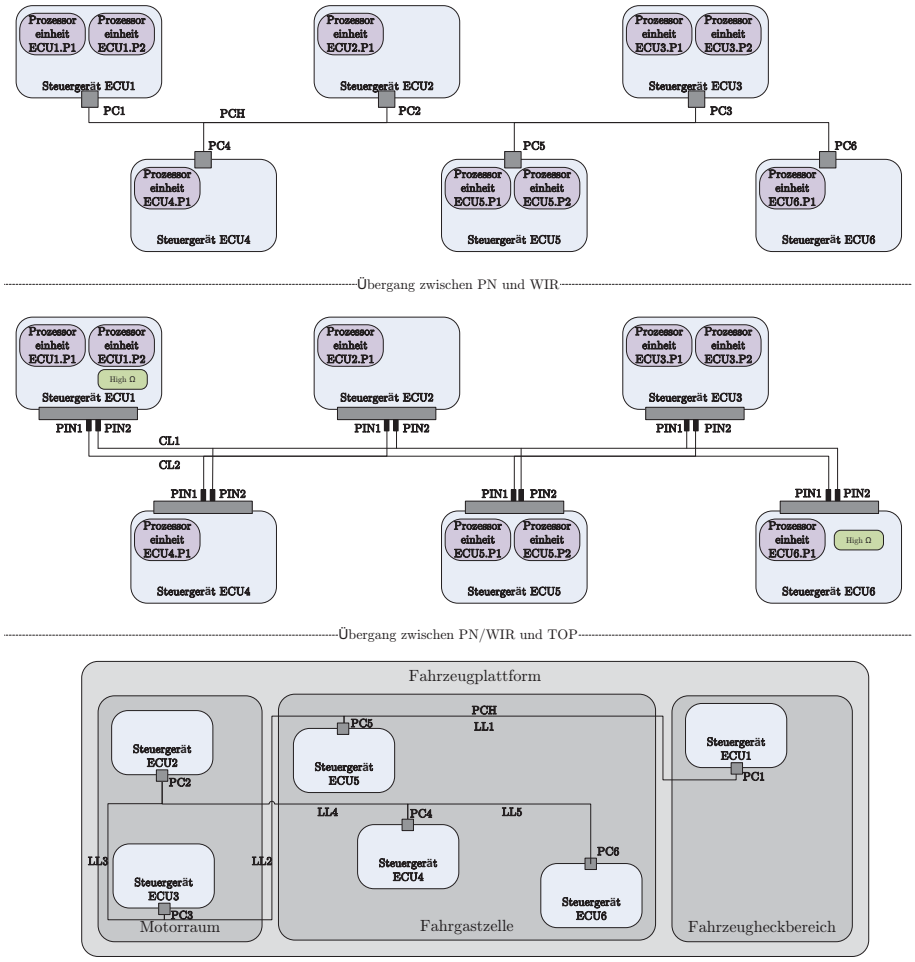


Abbildung 4.13.: Spezifikationsbereiche des physikalischen Netzwerks auf PN, WIR und TOP-Ebene

Definition 4.2.6 (Physikalisches Netz) *Das physikalische Netz PN komprimiert die Aspekte der physikalischen Bordnetzkonfiguration. Dazu zählen die Steuergeräteanordnung (**topology**) TOP, die Systemverkabelung (**wiring**) WIR und die Steuergeräteplatzierung im Fahrzeug (**package**) PCK. Konkret lässt sich die Instanz eines Funktionsnetzes FN auf Submengen der Steuergeräte (ECU) und der physikalischen Übertragungskanäle (PCH) abbilden. Die Verknüpfung zwischen den Steuergeräten und den Übertragungskanälen erfolgt über die Potenzmengenbildung der steuergeräte- und netzwerkseitigen Stecker PC und NC. Die Stecker werden mit der bijektiven Funktion v aufeinander abgebildet:*

[6] *Achtung:* Der Füllgrad für jeden Slot entspricht **nicht** der Summe des Zeilenvektors, da Signale mehr als einen Empfänger besitzen können.

$$PN_{fn} = (ECU_{fn}, PC_{fn}, NC_{fn}, PCH_{fn}, v) \text{ mit } ECU_{fn} \subseteq ECU,\ PCH_{fn} \subseteq PCH,$$

$$PC_{fn} = \left[\bigcup_{e \in ECU_{fn}} c(e) \right] \text{ mit Relation } c : ECU \rightarrow \mathcal{P}(PC)$$

ordnet jedem Steuergerät eine Menge von steuergeräteseitigen Steckern zu,

$$NC_{fn} = \left[\bigcup_{p \in PCH_{fn}} n(p) \right] \text{ mit Relation } n : PC \rightarrow \mathcal{P}(NC)$$

ordnet jedem physikalischen Übertragungskanal eine Menge von busseitigen Steckern zu.

$$v : PC_{fn} \rightarrow NC_{fn},$$

definiert eine bijektive Abbildung aus der Menge der steuergeräteseitigen Stecker auf die Menge der busseitigen Stecker.

Dabei gelten folgende Prämissen:

Eindeutigkeit:

$$\forall e, e' \in ECU_{fn} : e \neq e' \Rightarrow c(e) \cap c(e') = \varnothing$$

Jeder Stecker wird exklusiv einem Steuergerät zugeordnet, wobei ein Steuergerät mehrere Stecker besitzen kann.

Redundanz:

$$\forall p, p' \in PCH_{fn} : p \neq p' \Rightarrow n(p) \cap n(p') \subseteq NC_{fn},$$

Jeder Stecker kann eine arbiträre Anzahl an physikalischen Kanälen bedienen, wobei die Anzahl der netzwerkseitigen Stecker identisch zur Anzahl der steuergeräteseitigen Stecker ist.

Erweiterbarkeit:

Eine Erweiterung $(ECU'_{fn}, PC'_{fn}, NC'_{fn}, PCH'_{fn}, v')$ *einer physikalischen Netzinstanz* $(ECU_{fn}, PC_{fn}, NC_{fn}, PCH_{fn}, v)$ *lässt sich wie folgt definieren:*

$$ECU_{fn} \subseteq ECU'_{fn} \text{ und } PCH_{fn} \subseteq PCH'_{fn};$$
$$\forall e \in ECU_{fn}, e' \in ECU'_{fn} : e = e' \Rightarrow c(e) \subseteq c(e');$$
$$\forall p \in PCH_{fn}, p' \in PCH'_{fn} : p = p' \Rightarrow n(p) \subseteq n(p');$$
$$\forall s \in \bigcup_{e' \in ECU_{fn}} c(e') : v(s) = v'(s)$$

Durch diese drei Vorbedingungen erhält eine Erweiterung sämtliche Relationen c, n zwischen den Elementen in ECU_{fn} *und* PCH_{fn}*. Konsequenterweise bleibt die bijektive Abbildung zwischen den übernommenen Elementen in dem neuen physikalischen Netzwerk identisch zu den*

Elementen aus dem zugrunde gelegten physikalischen Netzwerk.

*TOP umfasst die Zuordnung der Steuergeräte zu verschiedenen Bustechnologien und deren Instanzen in der E/E-Architektur. Zusätzlich wird das Anbindungskonzept zum Systembus spezifiziert (**Standard-Transceiver**, **Stern-Transceiver**).*

WIR verfeinert das TOP-Modell um Artefakte der Bordnetzentwicklung (Stecker-, PIN-Belegung, Kabelattribute (Signalverzögerung, Signaldämpfung)). Dazu zählt die Untergliederung physikalischer Übertragungskanäle PCH in partielle Übertragungskanäle PCHP zur Spezifikation von Bussystemen mit zwei oder mehr Kabeladern und etwaige Klemmen zur Anbindung der Spannungsversorgung der Steuergeräte.

PCK verfeinert das TOP- und das WIR-Modell durch Addition vernetzungsrelevanter Aspekte, die sich durch den Verbau der Steuergeräte im Fahrzeug ergeben. Aus Sicht des Kommunikationssystems interessiert dabei ausschließlich die geometrischen Abmessungen der Verbindungen zwischen den Steuergeräten aus dem TOP-Modell.

Der Zusammenhang der drei Ebenen TOP ↔ WIR ↔ PCK, lässt sich in drei Verfeinerungsschritten darstellen:

Ebene 0:

Das Funktionsnetz wird mit einer Menge an Steuergeräten implementiert, die über einen zeitgesteuerten Bus kommunizieren. Dabei muss mindestens ein Steuergerät aus der Menge der existierenden Steuergeräte verwendet werden.

$$p : FN \rightarrow \mathcal{P}(ECU),$$
$$p(fn) = ECU_{fn} \text{ mit } fn \in FN, ECU_{fn} \subseteq ECU, ECU_{fn} \neq \emptyset$$

Ebene 1:

Jedem Steuergerät wird eine Teilmenge der Menge an Verkabelungselementen CL (cable element) zugewiesen, welche die Verbindung zu anderen Steuergeräten repräsentiert. Dadurch wird die physikalische Ausprägung des zeitgesteuerten Busses in der Struktur eines Baumes mit den Steuergeräten als Knoten des Graphen definiert.

$$q : ECU \rightarrow \mathcal{P}(CL),$$
$$fn \sim (ECU_{fn}, CL_{fn}, q)$$

Ebene 2:

*Im letzten Schritt gilt es den Baum aus Ebene 2 auf ein Fahrzeugbordnetz abzubilden, wobei je nach PCK-Anforderung eine unterschiedliche Teilmenge der Segmente des Bordnetzes WS (**wiring segments**) für die Platzierung der Verkabelungselemente CL benützt werden müssen.*

$$w : CL \rightarrow \mathcal{P}(WS),$$
$$fn \sim (ECU_{fn}, CL_{fn}, WS_{fn}, q, w)$$

Da für jedes Element aus WS ein Attribut zur Beschreibung seiner geometrischen Länge existiert,

lassen sich unter anderem die exakten Bordnetzabmessungen in Abhängigkeit zur Ausprägung eines Funktionsnetzes fn und dessen Partionierung p auf dem zeitgesteuerten Bussystem ermitteln.

□

4.2.2.8. Partitionierung

Der Partitionierungsbegriff spielt eine signifikante Rolle bei der festen Allokation der FC von Funktionsnetzen zu Hardwarekomponenten, etwa einem Steuergerät. Das grundlegende Problem einer Partitionierungsaufgabe ist im Bereich des Hardware-/Softwarecodesigns seit einiger Zeit Grundlage wissenschaftlicher Forschung /[KL02]/. Deren Ziele lassen sich dabei in verwandter Form auf die Problemstellung der E/E-Partitionierung im Fahrzeug übertragen. Anfolgend wird der Partitionierungsbegriff genauer definiert /[KL02], [PT07]/:

Definition 4.2.7 (Partitionierung) *Auf Systemlevelebene wird eine zu entwickelnde Anwendung als Prozessgraph (**task graph**) dargestellt, wobei die Prozesse Knoten des Graphen darstellen (vgl. FN). Die Knoten lassen sich in unterschiedlicher Art und Weise umsetzen mit Auswirkungen auf nicht-/funktionale Aspekte, etwa Hardwaregröße, Ausführungszeit und Kosten.*

*Die gemeinsame Bestimmung der Anzahl und Art der für eine Implementierung erforderlichen Komponenten (Register, Speicherbänke, funktionale Einheiten, Busse) wird als **Allokation** bezeichnet. Bei der **Bindung** (**Mapping**) werden diesen Komponenten Variablen, Operationen und Datenkommunikationen zugeordnet. Diese Zuordnung logischer Aspekte zu Hard- und Softwarekomponenten als Umsetzung zu einem (verteilten) System, wird allgemein als **Partitionierung** bezeichnet. Ergänzt wird der Partitionierungsschritt durch die **Ablaufplanung** zur Abbildung des spezifizierten Verhaltens des Systems auf diskrete Zeitschritte und -intervalle.*

*Das **erweiterterte Partitionierungsproblem** bezieht sich auf das Mehrzieloptimierungsproblem bei der Partitionierung, etwa zur Reduzierung von Hardwarefläche, Einhaltung von Zeitfristen (deadlines) im Prozess der Allokation, Bindung und Ablaufplanung.*

Formal lässt sich das Partitionierungsproblem folgendermaßen charakterisieren.

Gegeben ist eine Menge von Elementen $E = \{e_1, e_2, ..., e_n\}$. *Gesucht ist eine Partition* $P = \{p_1, p_2, ..., p_m\}$, *so dass*

$$p_1 \cup p_2 \cup ... \cup p_m = E,$$
$$p_i \cap p_j = \emptyset \text{ mit } \forall i, j : i \neq j, p_i \neq \emptyset,$$
$$c(P) = min \text{ mit } c(P) \text{ als Kosten für Partitionierung } P.$$

□

Bei der Betrachtung der Partitionierung im Bezug auf eine flexible zeitgesteuerte E/E-Architektur zeigt sich eine Beschränkung auf wenige Aspekte als ausreichend (vgl. Abb. 4.14). So interessiert vorrangig die Partitionierung eines Funktionsblocks FC aus dem Funktionsnetz FN auf die verfügbaren Steuergeräte eines Fahrzeugnetzwerks. Dabei steht die Analyse der für eine Partitionierung determinierten Sende-/Empfangsbeziehungen pro Steuergerät und deren Zeitanforderungen im Vordergrund. Aus Komponentensicht lässt sich die Sicht auf die Inter-/Intraprozessorkommunikation innerhalb

eines Steuergeräts verfeinern. Aus Netzwerksicht reduziert sich dieser Bereich auf eine Abstraktion auf die zeitlichen Mindest- und Maximalabstände bei der Bearbeitung sequentieller Signale (s. Abs. 4.2.2.5).

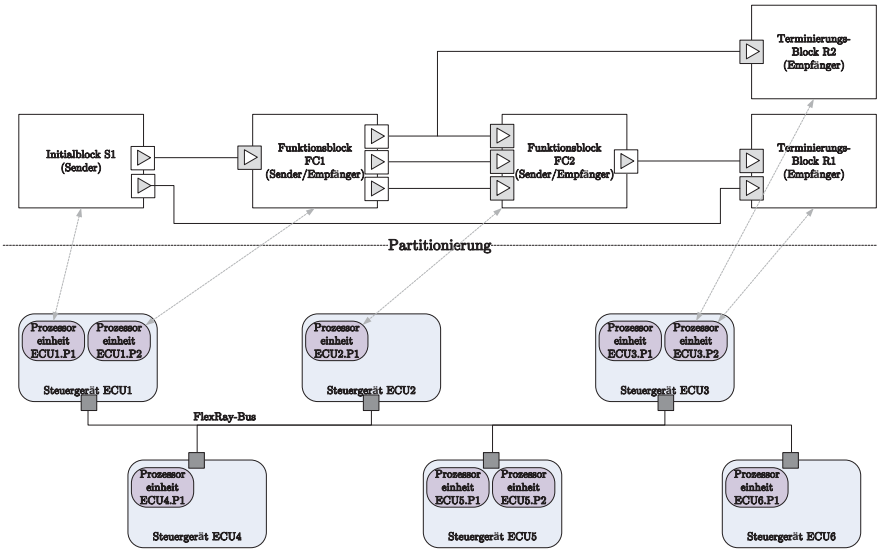


Abbildung 4.14.: Partitionierung eines FN auf eine physikalische Netzwerkarchitektur

Die Kostenfunktion $c(P)$ eines Partitionierungsschritts verkörpert ein mehrdimensionales Optimierungsproblem, dessen Eingangsgrößen sich nicht allgemeingültig gegeneinander abwägen lassen. Beispielsweise ist von Fall zu Fall zu unterscheiden, wie der Ressourcenverbrauch in einem Steuergerät im Vergleich zum Bandbreitenverbrauch eines Feldbussystems zu gewichten ist. Eine durchgehende Analyse unterschiedlicher E/E-Architekturpartitionierungsstrategien ist eine umfangreiche Aufgabe, die nicht weiter innerhalb dieser Arbeit verfolgt wird.

4.2.3. Konzept der dynamischen Modellanalyse

Bei zeitgesteuerten Systemen eignet sich für die dynamische Modellanalyse das Konzept der gezeiteten Automaten. Mit dieser Formalisierung lässt sich die Systemsimulation für synchrone verteilte Systeme umsetzen.

Gezeitete Automaten

Gezeitete Automaten bieten die Möglichkeit zur Simulation und Verifikation verteilter Echtzeitsysteme. Durch die Option der Simulation der virtuellen globalen Zeit eines zeitgesteuerten Bussystems eignet sich dieser Ansatz, um Aspekte der Nebenläufigkeit und Synchronisation von logischen Komponenten zu verifizieren. Nachfolgend werden Syntax und Semantik dieser formalen Notation nach /[BY04]/ erläutert:

Definition 4.2.8 (Gezeiteter Automat) *Ein gezeiteter Automat A besteht aus einem Tupel* $\langle N, l_0, E, I \rangle$ *wobei*

- *N eine endliche Menge an Zuständen,*

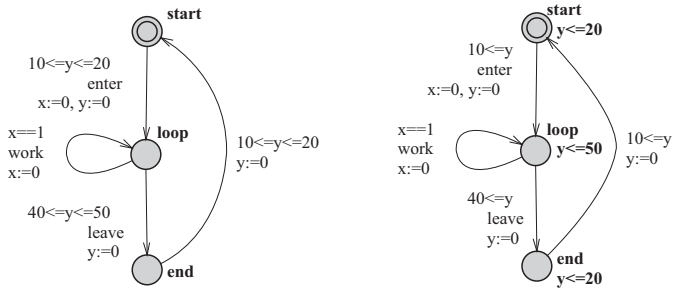


Abbildung 4.15.: Beispiel für einen gezeiteten Automaten mit lokalen Invarianten /[BY04]/

- l_0 *der Anfangszustand,*
- $E \subseteq N \times B(C) \times \sum \times \mathcal{P}(C) \times N$ *eine Menge an Zustandsübergängen und*
- $I : N \rightarrow B(C)$ *eine Zuweisung von Invarianten zu Zuständen ist.*

Der Ausdruck $l \overset{g,a,r}{\longrightarrow} l'$ *bezeichnet* $\langle l, g, a, r, l' \rangle \in E$.

Die Menge der Variablen C aus dem Wertebereich der reellen Zahlen mit den Bezeichnern $x, y, ...$ *bildet die Uhren eines Echtzeitsystems. Das endliche Alphabet* $\sum$ *mit den Bezeichnern* $a, b, ...$ *steht für ausführbare Aktionen.* $B(C)$ *bezeichnet die Menge an Clock Constraints.*

□

Clock Constraints: Für die Uhren existieren Vorbedingungen (*clock constraints*), welche die Ausführung von Zustandsübergängen im Automaten beinflussen. Transitionen lassen sich dabei nur ausführen, wenn der zum aktuellen Zeitpunkt anliegende Uhrenwert die jeweils zum Zustandsübergang zugehörige Vorbedingung erfüllt. Ein Clock Constraint ist eine konjunktive Formel von atomaren Vorbedingungen in der Form $x \sim n$ oder $x - y \sim n$ für $x, y \in C, \sim \in \{\leq, <, =, >, \geq\}$ und $n \in \mathbb{N}$. Clock Constraints dienen als Wächter (*guards*) für gezeitete Automaten.

Operationale Semantik: Aus Sicht der operationalen Semantik wird der gezeitete Automat aus der Kombination des gegenwärtigen Zustands mit dem gegenwärtigen diskreten Zeitpunkt als Transitionssystem definiert. Der Automat folgt zeitlichen Verzögerungen auf Basis von Verzögerungstransitionen oder als Folge von aktivierten Zustandsübergängen, den Aktionstransitionen.

Veränderungen von Zeitwerten der Uhren werden über Uhrenzuweisungen (*clock assignments*) in Form von Funktionen zum Verknüpfen von C mit nicht-negativen reellen Werten $\mathbb{R}^+$ realisiert. Falls u, v solche Funktionen beschreiben, dann bedeutet $u \in g$, dass die Uhrenwerte aus u den Wächter g erfüllen. Für $d \in \mathbb{R}^+$ sei $u + d$ die Bezeichnung für das Clock Assignment, das alle $x \in C$ zu $u(x) + d$ verknüpft und somit einen einheitlichen konsistenten Zeitschritt definiert. Für $r \subseteq C$ sei $[r \mapsto 0]\, u$ das Clock Assignment, welches alle Uhren in r auf den Wert 0 zurücksetzt und gleichzeitig mit u für alle restlichen Uhren in $C \backslash r$ übereinstimmt.

Das gezeitetes Transitionssystem beschreibt somit Zustände als Paare $\langle l, u \rangle$ mit Transitionen, die nach folgenden Regeln definiert sind:

- $\langle l, u\rangle \xrightarrow{d} \langle l, u+d\rangle$ wenn $u \in I(l)$ und $(u+d) \in I(l)$ für eine nicht-negative reelle Zahl $d \in \mathbb{R}^+$,
- $\langle l, u\rangle \xrightarrow{a} \langle l', u'\rangle$ wenn $l \overset{g,a,r}{\rightarrow} l'$, $u \in g$, $u' = [r \mapsto 0]u$ und $u' \in I(l')$.

Der in dieser Arbeit benutzte Dialekt der gezeiteten Automaten *Uppaal* /[BY04]/ beschränkt Zustandsinvarianten auf nach unten abgeschlossene Vorbedingungen in der Form $x \leq n$ oder $x < n$ mit $n \in \mathbb{N}$.

Die Aspekte der Nebenläufigkeit und Kommunikation lassen sich mittels paralleler Komposition von gezeiteten Automaten erreichen. Die Applikation eines parallelen Kompositionsoperators ermöglicht die Interaktion zwischen Automaten und bietet ein Instrument zur Nachbildung einer Hand-Shake-Synchronisation von Prozessen. Für weiterführende Details zum Konzept der gezeiteten Automaten wird an dieser Stelle auf /[BY04]/ weiterverwiesen.

4.2.4. Parameterkonfiguration

Der Zweck der statischen und der dynamischen Modellanalyse ergibt sich durch die Hinzunahme sämtlicher Systemparameter der flexiblen zeitgesteuerten Bustechnologie FlexRay. Zur Komplexitätsreduktion wird dabei eine Zerlegung der abhängigen Systemparameter innerhalb unterschiedlicher Klassen erforderlich.

4.2.4.1. Klassifikation von Systemmerkmalen

Durch den mehrdimensionalen Zusammenhang des Systemparametersatzes ist eine Klassifikation zur Komplexitätsreduzierung und zu einem besseren Verständnis des Gesamtsystems notwendig. Vier unterschiedliche Klassifikationsmöglichkeiten bieten sich an:

Funktionsebene	Modellebene
Elektrischer Physical Layer	Topologien
Kodierung/Dekodierung	Netzwerke
WakeUp/StartUp	Funktionsnetze
Fehlerkontrolle	Dynamisches Verhalten
Kommunikation	
Synchronisation	
Hierarchien	Qualitätsmerkmale
Gesamtarchitektur (Aktorik,Sensorik)	Performanz
Netzwerk (Cluster)	Erweiterbarkeit
Komponente (ECU)	Genauigkeit
Teilkomponente (CC, Transceiver)	Lebensdauer
Logische Module (SW)	Effizienz
	Zuverlässigkeit

Tabelle 4.3.: Funktionsorientierte, modellierungsspezifische, hierarchische und qualitätsorientierte Klassifikation

Während der Systemimplementierung zeigte sich, dass nur eine Verknüpfung diverser Teilklassen aus allen vier Bereichen zu gleichförmig partiellen Parametersätzen führt (s. Abs. 5).

4.2.4.2. Parameterstrukturierung

Eine Strukturierung des Parametersatzes eröffnet einen ersten Schritt zur Analyse einer FlexRay-Konfiguration und zur Identifikation von Optimierungspotentialen. FlexRay-Parametersätze lassen sich nach den Restriktionen aus /[Fle05a], [Fle04b]/ berechnen. Ein allgemeiner sequentieller Vorgang zur vollständigen Berechnung der Systemparameter ist dafür nicht explizit vorgegeben. Prinzipiell bauen die Systemparameter aufeinander auf und lassen sich linear ableiten. Allerdings lassen sich die für die Systemauslegung signifikanten Parameter, etwa die Zyklus-, Segment- oder Slotlängen, durch Relationen stufenweise auf die hardwarenahen Parameter, beispielsweise die Systempräzision, die Makroticklänge oder die maximale Uhrenabweichung überführen. Jedoch erschwert sich der Konfigurationsprozess, da unterschiedliche Basisparameter in der Regel zu uneinheitlichen Hardwareparametern führen. In Abb. 4.16 wird die zeitliche Zusammensetzung des FlexRay-Zyklus anhand der konfigurierbaren Parameter dargestellt.

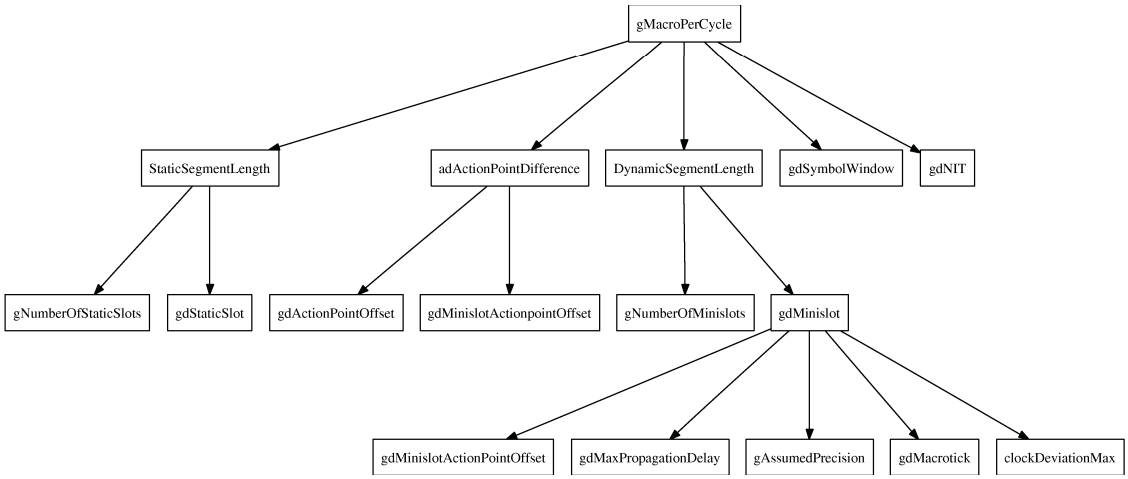


Abbildung 4.16.: Hierarchische Darstellung der Basisparameter zur Zusammensetzung eines FlexRay-Zyklus im Zeitbereich

Zur Optimierung der Nutzdateneffizienz (s. Abs. 4.6) wird der Fokus in der Parameterdefinition in Richtung des Bereichs „*Statisches Segment*“ verschoben.

4.2.4.3. Parameterberechnung

Um einen Parametersatz unter Zielkriterien vollständig berechnen zu können, ist ein Parametermodell erforderlich, welches die Optimierungsmethoden der Gütekriterien und extern vorgegebene Werte, beispielsweise Hardwaredaten, berücksichtigt sowie die ableitbaren Parameterwerte unter Verwendung der bedateten *SYS*-Modellen ermittelt.

Definition 4.2.9 (Parametermodell) *Ein Parametermodell beschreibt die Abhängigkeit metamodellbasierter Systemdaten unter Verwendung dedizierter Algorithmik zur vollständigen, durchgängigen Berechnung individueller Systemparameter.*

Bei der Klassifizierung wird zwischen ***Implementierungsparametern****, die bei der späteren Systementwicklung identisch übernommen werden und* ***Analyseparametern*** *zur Ermittlung*

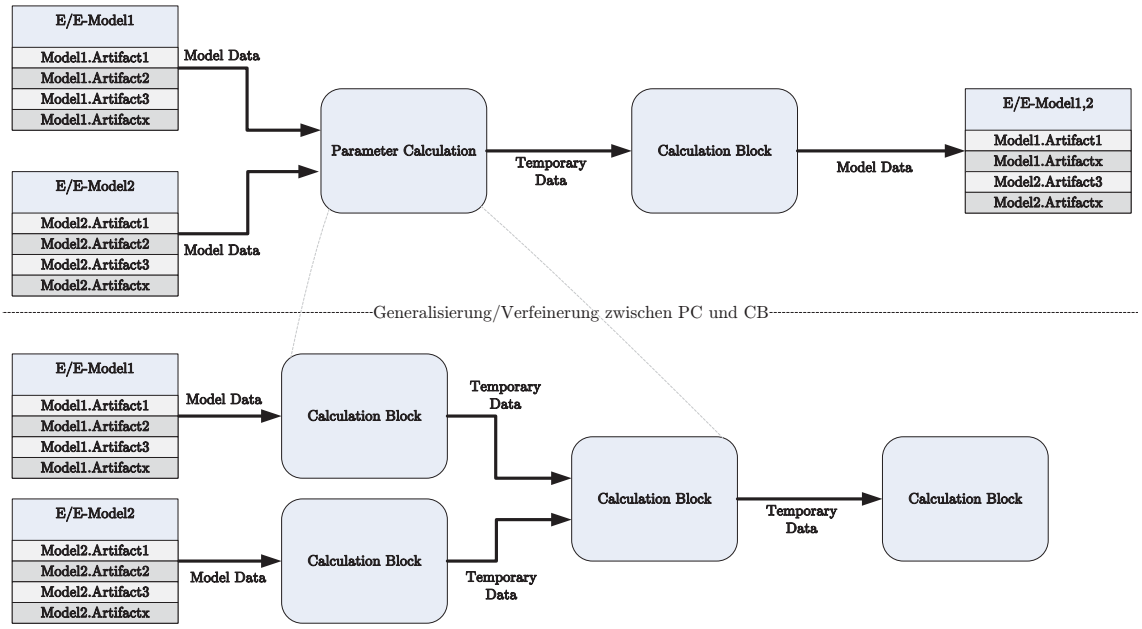


Abbildung 4.17.: Parametermodell zur Berechnung von Modelldaten/Parametern auf Basis existierender Modelldaten

abstrakter Systemqualitätseigenschaften unterschieden.

*Das **Parametermodell** Ein Parametermodell PM wird durch die Grundstruktur einer Parameterberechnung beschrieben. PM rechnet auf Basis eingehender **Modelldaten** MD innerhalb eines oder mehreren **Berechnungsblöcke** CB (**calculation blocks**) und schreibt die Ergebnisse in das Modell zurück. Eine **Parameterberechnung** (**parameter calculation**) PC wird autark in einem oder durch Verknüpfung mehrerer CB umgesetzt. Grundlage eines CB ist die Berechnung einer mathematischen Funktion, eines Algorithmus oder die Verfeinerung der vorgeschalteten CB, die temporär errechnete Daten TD (**temporary data**) bereitstellen. Diese Datenkanäle sind unidirektional ausgeführt, um Zyklen in der Parameterberechnung auszuschließen.*

$$PM : MD \times PC \rightarrow MD,$$
$$\textit{mit der Kardinalität } |MD| \geq 2 \textit{ und } |CB| \geq 1.$$
$$PC : CB \times TD \times CB \rightarrow CB,$$
$$\textit{mit der Kardinalität } |CB| \geq 1 \textit{ und } \exists d \in TD(|CB| \geq 2).$$

□

Wie in Abb. 4.17 dargestellt bietet die explizite Modellierung der Berechnungsvorschriften auf einem existierenden Datenmodell die Chance zur strukturierten, gegliederten Erfassung von Modelldaten, eine Entwicklung und Pflege von Berechnungsformeln sowie die Rückführung in das Originalmodell. Dadurch bietet sich die Möglichkeit eines idealen *Roundtrip-Engineerings* zur homogenen, simultanen Entwicklung der E/E-Architekturmodelle auf Basis einer flexiblen zeitgesteuerten Architektur mit ihren zugehörigen Parametersätzen.

4.3. Metamodellierung

Metamodelle bilden die Grundlage für die Entwicklung domänenspezifischer Modellierungskonzepte /[FHS02], [KT08]/. Nach /[Jer03]/ wird das Metamodell als explizites Modell der Konstrukte und Regeln, welche für den notwendigen Bau eines spezifischen Modells innerhalb der vorliegenden Anwendungsdomäne notwendig sind, beschrieben. Konsequenterweise verkörpert ein gültiges Metamodell eine Ontologie, wobei im Gegensatz eine Ontologie nicht zwangsläufig als Metamodell modelliert werden muss. Allgemein lässt sich ein Metamodell aus drei Perspektiven betrachten:

1. Als eine Menge an Bausteinen und Regeln zur Konstruktion von Modellen.

2. Als ein Modell aus der Domäne des spezifischen Anwendungsgebiets.

3. Als eine Instanz eines weiteren Modells.

4.3.1. E/E-Metamodellierung

Das Konzept des verwendeten Modellierungswerkzeugs PREEvision /[Aqu07]/ fundiert auf einem zugrunde gelegten Metamodell. Dabei handelt es sich um ein MOF-basiertes (*meta object facility*) /[OMG02]/ Modell, welches semantische Regeln für die Modellbildung definiert.

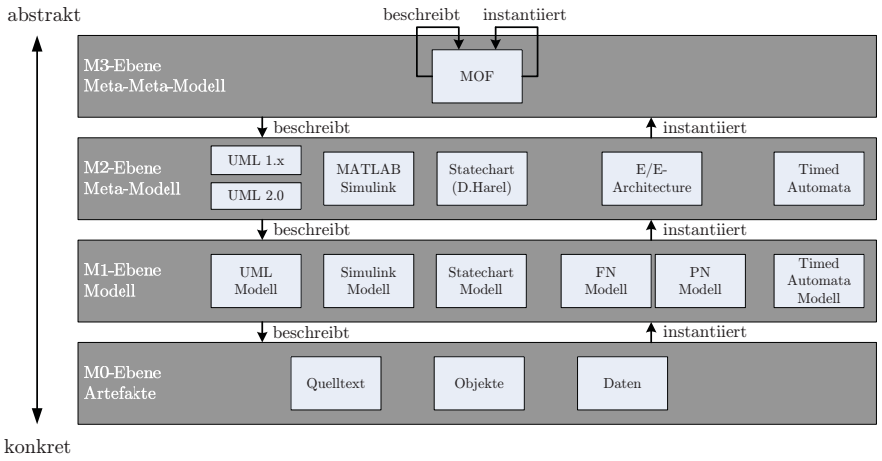


Abbildung 4.18.: Metamodellierungsebenen nach dem MOF-Standard

Abb. 4.18 visualisiert anschaulich die Abstraktionsstufen beim Übergang von konkreten Modellen zur abstrakten Metamodellierung. Domänenspezifische Sprachen siedeln sich beim Einsatz der MOF-Semantik auf der M2-Ebene an, um Aufbau, Abhängigkeiten und Strukturen für die Modellnotation zu definieren. Aus Benutzersicht interessiert vorrangig die M3-Ebene zur Umsetzung von erhobenen Datensätzen in zugeordnete Modellartefakte. Die folgende Abb. 4.19 zeigt einen kleinen Metamodellauschnitt, indem

die Modellierung der Komponenten *Aktor*, *Sensor* und *ECU* beschrieben wird. Die Hierarchie definiert, dass *Sensorik/Aktorik* und *ECU* jeweils von der Klasse *ElektrikElektronik* erben und über spezifische Attribute verfügen (Klemmen, Kühlung, Speicher, etc.).

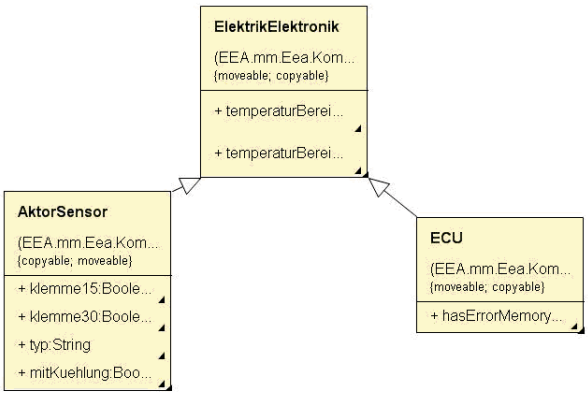


Abbildung 4.19.: Beispielhafter Ausschnitt aus einem Metamodell zur Beschreibung von E/E-Architekturen im Fahrzeug

4.3.2. Konzept der FlexRay-Metamodellierung

Die Spezifikation eines FlexRay-Systemdesigns differenziert sich signifikant von der bekannten Vorgehensweise im Bereich der ereignisgesteuerten Bussysteme im Bereich der CAN-Technologie. Im Detail sind präzise Einstellungen einer individuellen Ausprägung des Bussystems erforderlich, die in der Applizierung ein fundiertes Verständnis der Technologie voraussetzen (vgl. /[Rau07b]/). Beim Aufbau eines FlexRay-tauglichen E/E-Metamodells müssen die Teilentwicklungsschritte im Entwurf einer flexiblen zeitgesteuerten E/E-Architektur (s. Abb. 4.20) berücksichtigt werden.

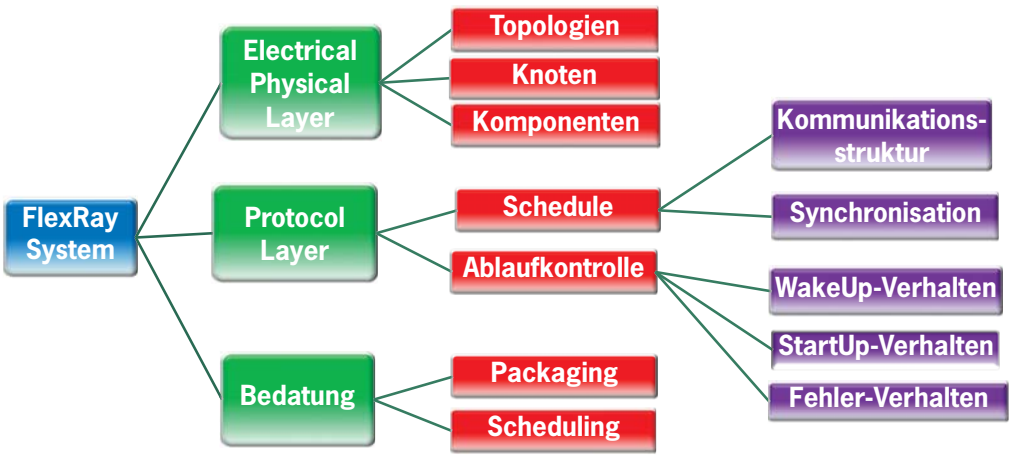


Abbildung 4.20.: Hierarchische Dekomposition der Entwicklungsbereiche zum Systementwurf

4.3.3. Bus- und Knotenparameter

Wie in Abb. 4.21 dargestellt lassen sich die Parameter in den Klassen *Physical Layer*, *Kodierung/Dekodierung*, *Fehlerkontrolle*, *WakeUp/StartUp* und *Kommunikation* gliedern. Dies entspricht der Klassifikation auf *Funktionsebene* (s. Abs. 4.2.4). In Tab. 4.4 werden die wichtigsten FlexRay-Parameter gemäß der Funktionsebenenuntergliederung zusammengefasst. Eine domänenspezifische Modellierung dieser Parametergruppen erfolgt nach Zuordnung zwischen Parametergruppe und Modellierungsebene.

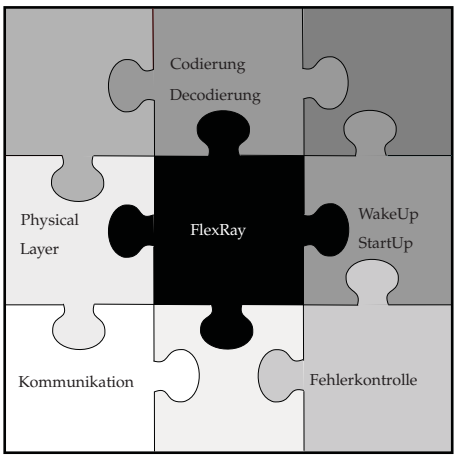


Abbildung 4.21.: Übersicht zu den Teilbereichen der FlexRay-Technologie

In dem Zusammenhang werden die einzelnen Funktionsebenen den Modellierungsebenen des FlexZOOMED-Ansatzes zugeordnet. Dabei zeigt sich speziell im Kontext der physikalischen Bitübertragung und der (De-)Codierung eine Aufsplittung über mehrere Ebenen hinweg. Während sich das WakeUp- und StartUp-Verhalten ausschließlich mit einer dynamischen Modellanalyse (*DV*) untersuchen lässt, wird die Fehlerkontrolle partiell sowohl in der statischen als auch in der dynamischen Modellanalyse abgebildet.

4.4. Domänenspezifische objektorientierte Modellierung

Wie in Abschnitt 4.1.1 beschrieben, stellt sich bei der FlexRay-Systementwicklung die Frage nach einem Lösungsansatz zur Abarbeitung mehrerer Aufgaben entlang eines sequentiellen Prozesses. Die Grundlage des FlexZOOMED-Konzepts basiert auf der Modellierung unterschiedlicher Strukturen zur Beschreibung der zu entwickelnden E/E-Architektur.

4.4.1. Konzept

Eine sinnvolle Gliederung eines FlexRay-Designs basiert auf der Dekomposition in den Ebenen der physikalischen und der architekturellen Netzwerkstruktur, dem statischen Funktionsnetz, und des allgemeinen dynamischen Verhaltens eines Systems. Folgende Ebenen und deren Bedatung im Kontext der E/E-Architekturentwicklung für Fahrzeuge werden im Folgenden erläutert.

Elektrische Bitübertragungsschicht	
gAssumedPrecision	gChannels
gdPropagationDelayMax	gdPropagationDelayMin
gdSampleClockPeriod	gSyncNodeMax
(De-)Kodierung	
gdBit	gdBitMax
gdBitMin	gdMacrotick
gdMaxMicrotick	gdTSSTransmitter
WakeUp/StartUp	
gdCASRxLowMax	gdMaxInitializationError
gdWakeupSymbolRxIdle	gdWakeupSymbolRxLow
gdWakeupSymbolRxWindow	gdWakeupSymbolTxIdle
gdWakeupSymbolTxLow	gColdstartAttempts
gListenNoise	gNetworkManagementVector
Kommunikation	
gdActionPointOffset	gdCycle
gdDynamicSLotIdlePhase	gdMinislot
gdMinislotActionPointOffset	gdNIT
gdStaticSlot	gdSymbolWindow
gMacroPerCycle	gNumberOfMinislots
gNumberOfStaticSlots	gPayloadLengthStatic
Fehlerkontrolle	
gClusterDriftDamping	gMaxWithouClockCorrectionFatal
gMaxWithoutClockCorrectionPassive	gOffsetCorrectionMax
gOffsetCorrectionStart	-
Parameterklasse	Modellierungsebene
Physical Layer	PN (TOP,WIR,PK)
Kodierung/Dekodierung	PN (TOP,WIR,PK)
WakeUp/StartUp	DV
Kommunikation	FN
Fehlerkontrolle	TOP/DV

Tabelle 4.4.: Parameterzuordnung zu den Modellierungsaspekten und -ebenen

4.4.2. Statische logische Systemarchitektur

Die logische Systemmodellierung lässt sich auf die Ebenen FN und DV abbilden.

Funktionsnetze

Zwischen den Fahrzeugfunktionen werden Daten ausgetauscht, die sich als gerichteter Graph darstellen lassen. Diese Notationsform wird als Funktionsnetz (FN) bezeichnet (s. Abs. 4.2.2.3). Folgende Attribute sind dem Funktionsnetz aus Sicht einer domänenspezifischen Modellierung zuzuordnen:

Kommunikationstypen: Darunter versteht sich das Verhalten des Informationsaustauschs an den Schnittstellen der kommunizierenden Komponenten. Eine gleichförmige intervallartige Kommunikation wird als *zyklisch* typisiert. Diese Art erfordert zusätzliche Informationen über deren *Zyklusrate*.

Ein weiterer Typ basiert auf spontaner, als *sporadisch* bezeichneter, Kommunikation. Auch eine hybride Form der beiden Grundtypen existiert, falls neben einem zyklischen Datenaustausch auch speziell vordefinierte Dateninhalte oder empfangene Signale eines

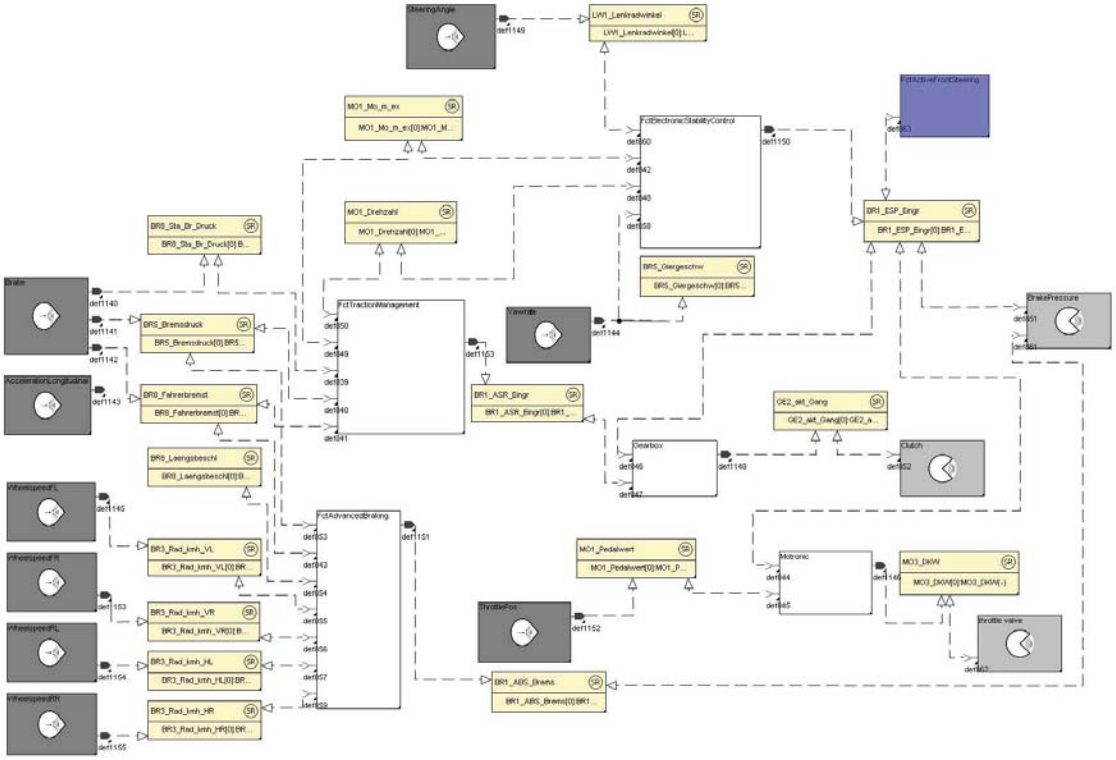


Abbildung 4.22.: Ausschnitt aus einem Funktionsnetz der Antriebs- und Fahrwerksdomäne

dritten Teilnehmers ein zusätzliches unmittelbares Versenden einer Information stimulieren. Diese Kombination kann *funktional* begründet sein oder in Abhängigkeit eines bestimmten Datenelements *signalbedingt* erfolgen.

Kommunikationsbeziehungen: Informationen können in verschiedenen Konstellationen zwischen Netzwerkkomponenten ausgetauscht werden. Eine Punkt-zu-Punkt-Kommunikation zwischen zwei einzelnen Knoten wird dabei als *p2p* bezeichnet. Eine andere Variante bildet die *Broadcast*-Kommunikation zwischen einer Sendekomponente und einer arbiträren Anzahl an Empfängern.

Kommunikationsgruppen: Eine Untergliederung der Menge an Kommunikationsdaten in einer Architektur ermöglicht eine entsprechende Paketierung sinnvoll komponierbarer Datengruppen. Eine klassische *funktionale* Untergliederung wäre bspw. eine Einteilung in Applikations-, Transport-, Diagnose-, Kalibrierungsbotschaften. Eine abstraktere Einteilung bringt die Dekomposition nach dem *Betriebsmodus*, beispielsweise Fahrzeugbetrieb und Service-/Werkstattbetrieb. Weiterhin kann es aus sicherheitsrelevanter Sicht auch sinnvoll sein in Fehler- oder Ausnahmesituationen je nach *Zustand* in abgestufte Modi zu gehen, um Informationen entweder zurückzuhalten oder im Netzwerk zu replizieren. Die Ausprägung eines Modus kann *temporal* für bestimmte Aktivitätszeiten festgelegt werden oder *lokal* auf spezielle Gebiete des Netzes bei der Signalpropagation beschränkt bleiben.

Das Funktionsnetz ermöglicht unter Hinzunahme der domänenspezifischen Modellie-

Funktionsnetzparameter		
Typen	Beziehungen	Gruppen
sporadisch	p2p	Funktion
zyklisch	broadcast	Modus temporal
hybrid funktional		Modus lokal
hybrid signalbedingt		Zustand temporal
		Zustand lokal
Relevante FlexRay-Zuordnungen		
Typen	→	Zykluslänge, Segmentierung
Beziehungen	→	Slot Offset, Erweiterung, Paketierung
Gruppen	→	Slot Offset, Erweiterung, Paketierung

Tabelle 4.5.: Übersicht der Kriterien zur Attributierung eines Funktionsnetzes und der Ableitung von grundlegendenen FlexRay-Eigenschaften

rung eine vollständige Abbildung der logischen Datenkommunikation der E/E-Architektur. Ein typisiertes Funktionsnetzwerk wird in Abb. 4.22 dargestellt. Durch die Ergänzung der Attribute in der Funktionsnetzmodellierung wird die Ableitung grundlegender Parameter zur Scheduledefinition eines FlexRay-Clusters ermöglicht.

Bei einer Reduzierung des Funktionsnetzes auf einen Basisumfang, bestehend aus einem 3-Tupel zwischen einem Sender *FCTractionManagement*, einem Signal *Motordrehzahl* und einer arbiträren Nummer an Empfängern, bspw. *ITT_ESP*, *FctElectronicStability-Control*, *Dämpfer_Niveau*, so ergibt sich im Werkzeug PREEvision die in Abb. 4.23 beispielhaft dargestellte Struktur. Eine instanziierte Form des typisierten Funktionsnetzwerks wird in Abb. 5.4 abgebildet.

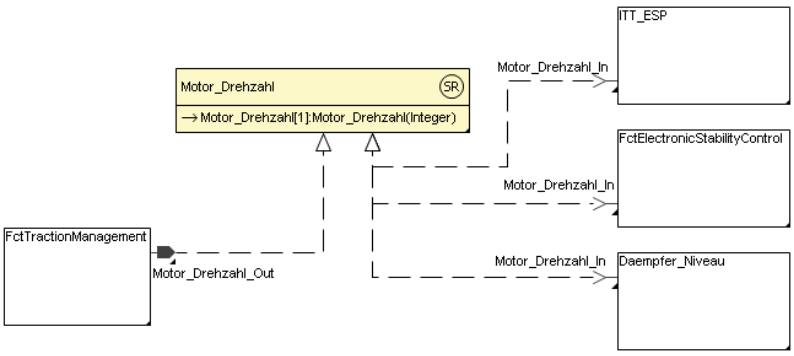


Abbildung 4.23.: Basisstruktur eines Funktionsnetzes mit (*Sender*, *Sendeport*, *Signal*, *Empfangsport* und *Empfänger*)

4.4.3. Statische technische Systemarchitektur

Bei der Zuordnung der in Abschnitt 3.3.4.1 vorgestellten relevanten Aspekte für die Definition der elektrisch physikalischen Bitübertragungsschicht ergeben sich eine Reihe von Zuordnungen zwischen Netzwerkkomponenten und dedizierten FlexRay-Parametern.

Physikalische architekturelle Netzwerkstruktur

Wie in 4.1 dargestellt, fokussiert der initiale Schritt der Netzwerkkonfiguration auf die

Modellierungskomponent	FlexRay-Parameter	Beschreibung	Beeinflusst
Bustreiber			
Netzwerkkonnektor	dBDTx	dBDTx01 + dBDTx10	gdMaxPropagationDelay
	dBDTxia	Sendeverkürzung (idle → active)	keine Formel
	dBDTxai	Sendeverlängerung (active → idle)	keine Formel
	dBDRxia	Empfangsverkürzung (idle → active)	gdTSSTransmitter gdWakeup-SymbolRxLow
	dBDRxai	Empfangsverlängerung (active → idle)	gdCASRxLowMax gdSymbolWindow
Netzwerkverbindungen			
Übertragungskanal	gdMinPropagationDelay	gdMinPropagationDelay	pDelayCompensation
	gdMaxPropagationDelay	gdMaxPropagationDelay	pDelayCompensation
	LineLength	LineLength	gdMaxPropagationDelay
	dBusTxia	Transitionsdauer (idle → active)	
	dBusTxai	Transitionsdauer (active → idle)	
Aktiver Sternkoppler			
Komponente	dStarTruncation	Signalverkürzung (Sternkoppler)	dStarAsym
	nStarPath	Anzahl der Sternpfade	
	dStarSetUpDelay	SetUp-Verzögerung	
	dStarGoToSleep	GoToSleep-Modus	
	dStarWakeUpReaction	WakeUp-Reaktionszeit	
	dBranchActive	Busaktivitätserkennung	
	dBranchFailSilentIdle	Timeout für Fehlerbehebung	
	dStarDelay	Ausbreitungsverzögerung (Negative Flanke)	
	dStarDelay0	Ausbreitungsverzögerung (Positive Flanke)	
	dStarTxia	Ausbreitungsverzögerung (idle → active)	
	dStarTxai	Ausbreitungsverzögerung (active → Idle)	
ECU			
Komponente	pKeySlotEnabled	Sync-/Coldstart-Fähigkeit	
	pSamplesPerMicrotick	Frequenz des FlexRay-CC	
	pChannels		
	pDecodingCorrection		

Netzwerkzustand		Beeinflusst
Komponenten	→	gMaxSyncNodes
Teilkomponenten	→	cclockdeviation, gdMacrotick
Netzwerkanbindungen	→	gdStarDelay Slewrates gdAsymmetricDelay gdTSSTruncation
Netzwerkverbindungen	→	Baudrate

Tabelle 4.6.: Zuordnung und Ableitung von Physical Layer relevanten FlexRay-Parametern zu Modellierungsartefakten einer technischen Systemarchitektur

physikalische Ausprägung der Netzwerkstruktur (s. Abb. 5.7). Der strikt sequentielle Entwurf und die Bedatung der jeweiligen technischen Modellierungsebene (TOP, WIR,

PCK) muss jedoch nicht strikt eingehalten werden. Folgende Attribute und Konfigurationsmöglichkeiten lassen sich der physikalischen Netzwerkstruktur zuordnen:

Leitungsspezifikation: Unter der Leitungsspezifikation versteht sich die Festlegung der exakten Daten eines Leitungstyps. Dabei werden dessen physikalische Eigenschaften berücksichtigt. Etwaige Größen basieren auf der Leitungslänge, Leitungsdämpfung und der spezifischen Ausbreitungsverzögerung.

Komponenten: In diesem Zusammenhang werden Steuergeräte als *physikalische Komponenten* eines Netzwerks aufgefasst. Der konkrete Aufbau des Steuergeräts bleibt dabei unberücksichtigt, da ausschließlich die Verteilung von Komponenten auf (Sub-)Netzwerke betrachtet wird.

Teilkomponenten: Zu den Teilkomponenten zählen dedizierte Hardware-Bausteine. Dazu gehören Host-Prozessoren, Kommunikationscontroller, Quarze (inklusive entsprechender Konfigurationen, bspw. der Frequenz für den Kommunikationscontroller des flexiblen zeitgesteuerten Bussystems).

Netzwerkanbindungen: Die Anbindung definiert den exakten Aufbau der Busschnittstelle einer Komponente. Dazu können Standardtransceiver und Sterntransceiver gerechnet werden.

Netzwerkverbindungen: Grundlegende globale Protokollparameter des Bussystems werden allgemein den Netzwerkverbindungen zugeschrieben. Dazu gehört vorrangig die anliegende Baudrate für die physikalische Bitübertragung im Netzwerk.

4.4.4. Dynamisches Verhalten (Protokollebene)

Wie in /[Fle05b]/ spezifiziert, inkludiert jeder FlexRay-Kommunikationscontroller eine einheitlich definierte Zustandsmaschine (POC) zur Steuerung des dynamischen Verhaltens innerhalb verschiedener Betriebszustände einer Netzwerkkomponente.

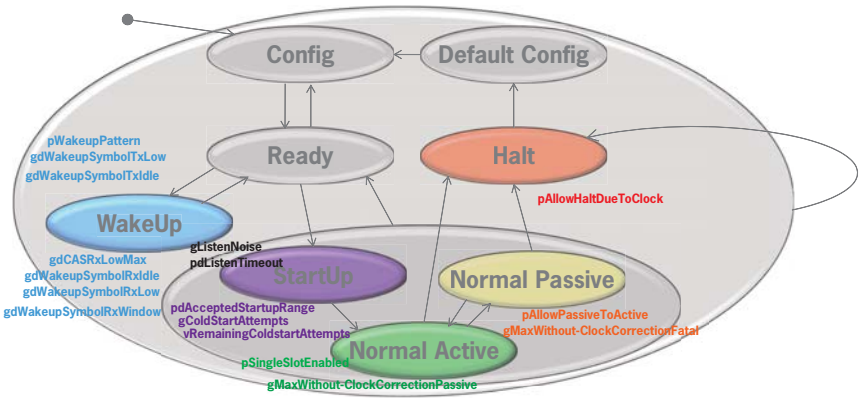


Abbildung 4.24.: Mit der Protokollzustandsmaschine eines FlexRay-Kommunikationscontrollers gemappte Parameter

Ein Teil des FlexRay-Parametersatzes bezieht sich dabei direkt auf das Verhalten der

POC (s. Abb. 4.24). Zum Verständnis werden die Zustandsübergänge und deren Zusammenhänge mit den FlexRay-Parametern erläutert.

Protokollzustand	Beeinflusst
WakeUp	gListenNoise pdListenTimeout pWakeUpPattern pWakeUpChannel gdWakeupSymbolRxIdle gdWakeupSymbolRxLow gdWakeupSymbolRxWindow gdWakeupSymbolTxIdle gdWakeupSymbolTxLow
StartUp	gListenNoise pdListenTimeout gColdstartAttempts pChannels gdCASRxLowMax pdAcceptedStartupRange pKeySlotUsedForStartup pKeySlotUsedForSync pKeySlotId pSingleSlotEnabled
Normal Active	pClusterDriftDamping pDecodingCorrection pMacroInitialOffset[AB]] pMicroInitialOffset[AB]] pdMaxDrift gOffsetCorrectionStart pExternRateCorrection pExternOffsetCorrection pRateCorrectionOut pOffsetCorrectionOut gMaxWithoutClockCorrectionPassive
Normal Passive	pAllowPassiveToActive
Halt	pAllowHaltDueToClock gMaxWithoutClockCorrectionFatal

Tabelle 4.7.: Ableitungen aus der dynamischen Verhaltenssicht

Ready → WakeUp: Die physikalischen Ausprägungen des WakeUp-Patterns (WUP) beeinflussen die Dauer eines Starthochlaufs im FlexRay-Steuergerät. Dazu werden die Längen für die Signalzustände *Idle* und *Low* für den Sender und den Empfänger sowie die maximal gültige WUP-Länge, bezogen auf die Anzahl der zu versendenden WakeUp-Symbole, berechnet. Auch die maximale Ausprägung des CAS-Symbols beim empfangenden Steuergerät kann in diesem Vorgang miteinbezogen werden. Während die konkrete Ausprägung des WUP für jede Baudrate statisch festgelegt ist, gibt es Spielräume bei der Definition der Wartezeiten im Falle detektierter Kommunikation auf dem FlexRay-Bus beim weckenden FlexRay-Steuergerät. Über *pWakeUpPattern* lässt sich die Anzahl der zu wiederholenden WakeUp-Sequenzen innerhalb eines WUP parametrieren.

Ready → StartUp → Normal Active: Beim initiierten StartUp-Vorgang müssen drei zum WakeUp-Vorgang vergleichbare Parameter bedatet werden. Die Wartezeiten vorab eines Startversuchs und bei detektierter Kommunikation auf dem Bus während des StartUp-Vorgangs sind dabei mit den Zeiten beim WakeUp identisch. Durch den Parameter *gColdstartAttempts* lässt sich die Anzahl der StartUp-Versuche für jedes Steuergerät clusterweit einheitlich skalieren. Während der Integration eines Steuergeräts im Cluster werden die Längen auftretender CAS-Symbole sowie die Abstände zwischen den detektierten StartUp-Botschaften gemessen und plausibilisiert (*gdCASRxLowMax*,*pdAcceptedStartupRange*). Ergänzend muss dem jeweiligen Steuergerät mitgeteilt werden, ob und in welchem Slot es StartUp- und Sync-Botschaften versendet und ob eine Einzelslotausführung oder der komplette Schedule intialisiert werden soll.

Normal Active → Normal Passive: Während der normalen aktiven Protokollausführung werden vorrangig Parameter benötigt, welche die stabile Protokollausführung und die Uhrensynchronisation beeinflussen. Zusätzlich wird der optionale Übergang in einen passiven Kommunikationsmodus explizit mit einem Parameter *gMaxWithoutClockCorrectionPassive* erfasst.

Normal Passive → Normal Active: Für den Zweck einer Steuergeräteerholung nach einem Fehlverhalten lässt sich die aktive Rückführung eines jeden Knotens in die Kommunikation im Nachgang eines passiven Zustandes optional definieren.

Normal Passive → Halt: Für den Fall, dass die Uhrensynchronisation mit einem latenten Fehler behaftet ist, stellt sich die Frage, ob ein Steuergerät aufgrund nicht plausibler Zeitwerte aus dem Cluster desintegriert werden soll. Dies lässt sich einerseits über einen Parameter festlegen und andererseits mit einem Zähler versehen, der die Anzahl fehlgeschlagener Synchronisationsversuche für eine Überführung in den *HALT*-Zustand spezifiziert.

4.5. Entwicklungsaspekte flexibler zeitgesteuerter Architekturen

Bei der Entwicklung eines zeitgesteuerten FlexRay-Systems ergeben sich im direkten Vergleich mit den CAN-basierten E/E-Architekturen einige Fragestellungen, die im Laufe des Systementwurfs adressiert werden müssen. Die im FlexZOOMED-Ansatz implizierten Aspekte verkörpern die entscheidenden Themen, die im Zusammenhang mit der FlexRay-Systemdefinition in einer Fahrzeugserienentwicklung bearbeitet werden müssen. Im Folgenden werden die entsprechenden Inhalte als Grundlage der später durchgeführten Systemanalysen vorgestellt.

4.5.1. Qualitätsparameter bei der Fahrzeugentwicklung

Zur Analyse der Qualität einer FlexRay-basierten E/E-Architektur ist es entscheidend sinnvolle Bewertungskriterien zu identifizieren. Dabei wird offensichtlich, dass sich die notwendigen Untersuchungsgebiete deutlich voneinander abgrenzen. Im Bereich der FlexRay-Parametrierung entstehen jedoch Überlappungen, die aufgrund der Systemrestriktionen zur Generierung des FlexRay-Parametersatzes akzeptiert und beherrscht werden müssen. Folgende Kriterien spezifizieren die Gesamtgüte einer festgelegten

FlexRay-Architektur, die während der Validierung der Arbeit auch als *Qualitätsparameter* bezeichnet werden[7].

Effizienz: In diesem Zusammenhang wird die verfügbare Nettodatenrate erfasst. Dabei muss zwischen dem statischen und dem dynamischen Segment differenziert werden. Daher lassen sich folgende Werte erfassen:

- Verfügbare Nutzdaten pro ms,
- Verfügbare Nutzdaten pro Zyklus bei maximaler Auslastung,
- Verfügbare Nutzdaten im statischen Segment,
- Prozentuale Einordnung vom *Best Case* entfernt (Statisches Segment),
- Verluste im Bereich der Einpassung des *Actionpoints* (Buszykluslänge ↔ Makroticklänge ↔ Systempräzision).

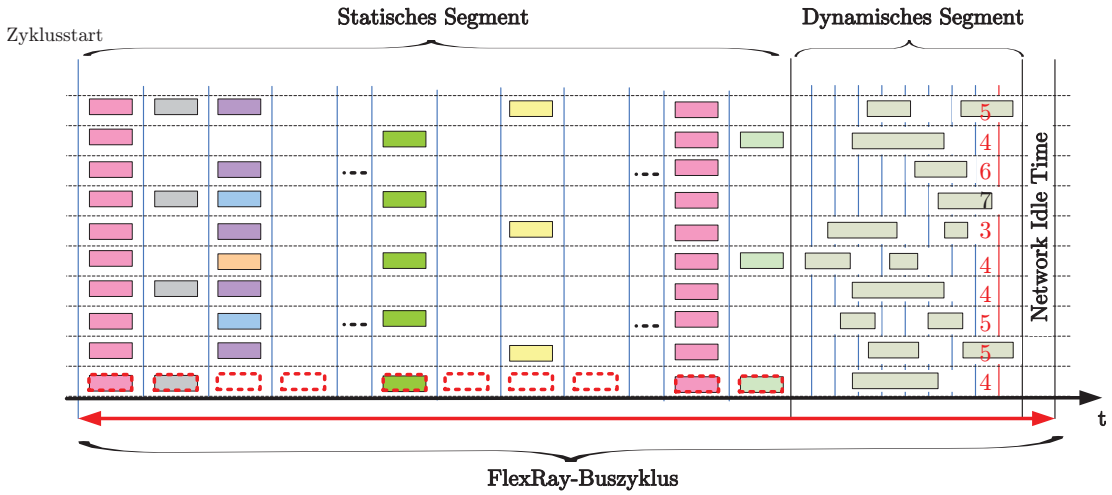


Abbildung 4.25.: Nutzdateneffizienz als Gütekriterium des FlexRay-Schedules

Aussagen über die Nutzdaten leiten sich aus dem logischen Modell auf der FN-Ebene ab. Dazu müssen alle Funktionsblöcke und deren Sendebeziehungen erfasst werden. Durch die vollständige Spezifikation des Signalzeitverhaltens lässt sich die restliche Nettobandbreite, bezogen auf den errechneten Parametersatz und der damit determinierten Bruttobandbreite, bestimmen. Je nach Fragestellung lässt sich die Bandbreite pro Zeitschritt, die Buszyklus- oder die Bussegmentlänge berechnen. Die Ergebnisse werden dem errechneten Optimum gegenübergestellt. [8] Einen weiteren Indikator für die Systemeffizienz bildet die relative zeitliche Differenz zwischen angenommener Systempräzision und dem *Actionpoint*.

[7] Die Qualitätsparameter dienen dem Nachweis der Tragfähigkeit des FlexZOOMED-Ansatzes, sind allerdings keinesfalls als vollständig im Bereich der FlexRay-Systementwicklung zu betrachten.

[8] Dabei muss berücksichtigt werden, welche minimale Anzahl statischer Slots nicht unterschritten werden darf, da diese Vorgabe mit der Bestimmung der optimierten Bandbreite im statischen Segment korreliert.

Robustheit: Ein robustes FlexRay-System erfordert einerseits eine stabile hohe Qualität des physikalischen Übertragungssignals im Netzwerk sowie andererseits Toleranzen bei der Konfiguration der Clustersynchronisation. Daraus ergeben sich folgende Qualitätsmerkmale:

- Toleranzen in der Bitlänge eines Signals (Bitlänge und Sicherheitspuffer),
- Toleranzen in der Alterung von Systemkomponenten (Quarzungenauigkeiten),
- Toleranzen bei der Skalierung des von der Präzision abhängigen *Actionpoints*,
- Toleranzen für die byzantische Fehlerkompensation,
- Toleranzen im Bereich der Uhrendämpfung bei der Uhrensynchronisation.

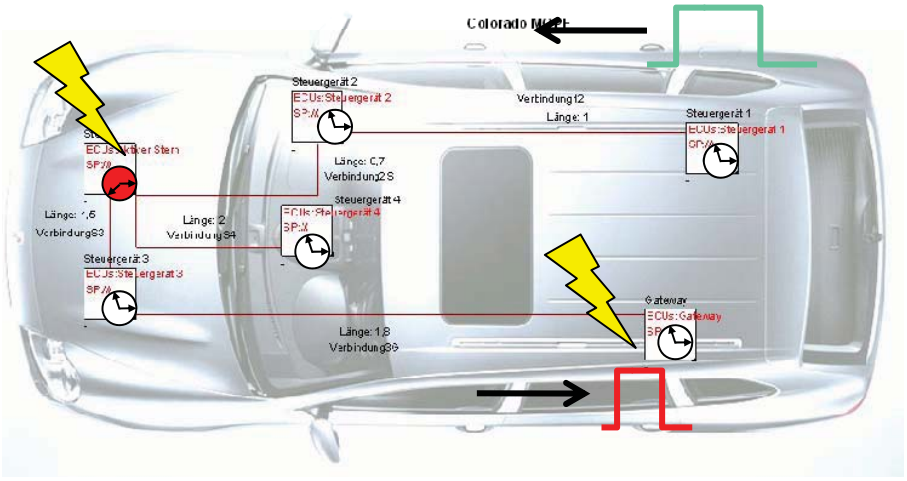


Abbildung 4.26.: Berücksichtigung physikalischer Effekte hinsichtlich der gesamten Netzwerkrobustheit

Die Signalausprägung im Netzwerk wird durch mehrere Einflüsse beeinträchtigt (Signaldämpfung, -verzögerung, -verzerrung), die auf die Eigenschaften der technischen Systemarchitektur zurückgeführt werden können (z.B. Kabelimpedanz, Reflexion, Schirmung). Durch Alterungseffekte verlieren die Quarze in den Steuergeräten an Genauigkeit über die Jahre. Diese Verschlechterung muss sich in der Auslegung des für das Gesamtsystem wichtigen Werts der *Systempräzision* bei der Architekturentwicklung einkalkuliert werden (*Actionpoint*-Definition). Ein interessanter Wert ergibt sich aus der Spezifikation der Uhrendämpfung, die ein Driften des Clusters verhindern soll, da ein hoher Wert die Gesamtpräzision des Uhrenensembles negativ beeinflusst.

EPL Erweiterbarkeit: Entscheidungskriterien in der EPL-Erweiterbarkeit beziehen sich auf eine Minimierung des Aufwandes bei notwendigen Architekturmodifikationen, etwa der Integration eines neuen Steuergeräts, zur Konsolidierung einer gültigen FlexRay-Systemkonfiguration. Ein Qualitätsmerkmal der technischen Systemarchitektur hängt in diesem Zusammenhang von dem erreichten Grad an Flexibilität für folgende Systemeingriffe ab:

- Umpositionierung von Steuergeräten als Zwischen- oder Endknoten (Busterminierungskonzept),

- Maximallängen der verbauten Leitungen im Bordnetz,
- Anzahl hinzufügbarer Leitungen (pro aktiven Sternkoppler/Hinzunahme von aktiven Sternkopplern),
- Anzahl hinzufügbarer Netzwerknoten in verschiedenen Fahrzeugsegmenten (Vorne/Mitte/Hinten).

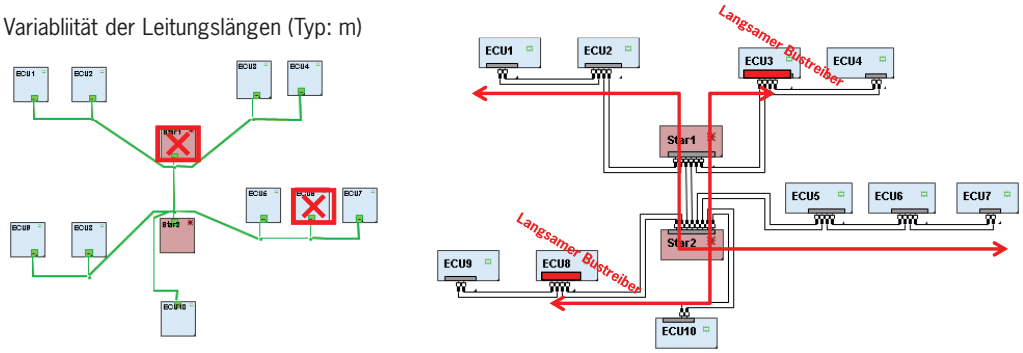


Abbildung 4.27.: Mögliche Beschränkungen im Erweiterungsbereich der physikalischen FlexRay-Netzwerktopologie

Je nach Systemausbaustufe und -erweiterung wird bei einem Rollentausch eines Netzwerkteilnehmers von einem Zwischen- zu einem Endknoten und umgekehrt eine Anpassung des Busterminierungskonzepts in den einzelnen Steuergeräten notwendig (Wechsel zwischen hoch- und niederohmiger Terminierung). Eine Reduzierung der Anzahl an Steuergerätevarianten mit unterschiedlicher Terminierung ist dabei signifikant. Zusätzlich interessiert die Variabilität im Bereich der Ergänzung von Busleitungen und die Beschränkung für deren Leitungslängen. Diese Fragestellungen werden dann entscheidend, wenn der Platz für die Positionierung im Fahrzeug in den einzelnen Verbaubereichen nur mehr eingeschränkt verfügbar ist.

Performanz: Dieser Bereich adressiert den in der E/E-Architektur maximal unterstützten Datendurchsatz pro diskreten Zeitschritt sowie die maximal zulässige Frequenz für die wiederholte Übertragung von Datenpaketen. Dieser Aspekt ist von Bedeutung bei der Auslegung der im System zu erwartenden Antwortzeiten für die Punkt-zu-Punkt-Kommunikation. Die Performanzwerte referenzieren auf beide Übertragungssegmente im Buszyklus, wobei dem statischen und dem dynamischen Segment jeweils leicht unterschiedliche Fragestellungen zukommen (vgl. Abs. 4.6). Zu diesem Bereich zählen folgende Themen:

- Antwortzeiten innerhalb des statischen Segments (Regelsystemfrequenz),
- Antwortzeiten innerhalb des dynamischen Segments (Jitter-Toleranzen bei der Übertragung),
- Durchsatzzeiten bei der Übertragung großer Datenpakete (Steuergeräteprogrammierung/Multimedia).

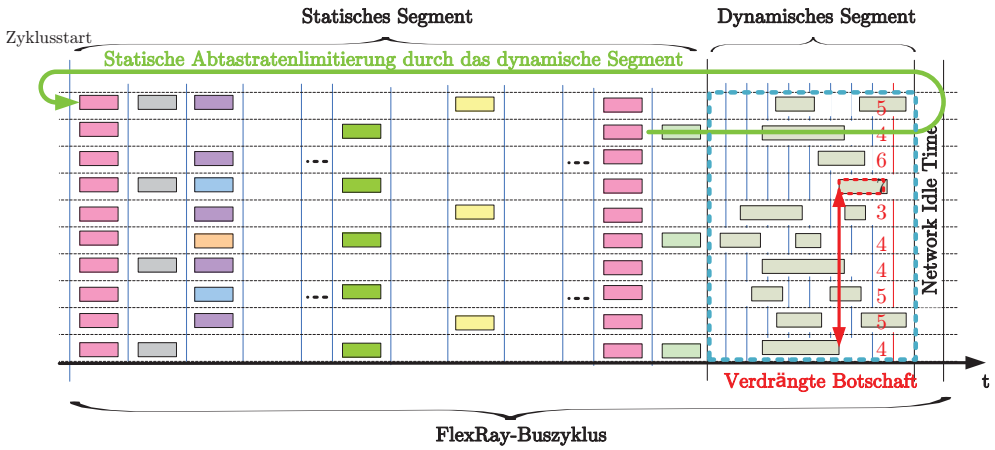


Abbildung 4.28.: Performanzaspekte innerhalb des FlexRay-Schedules

Aufgrund strikter Echtzeitanforderungen in regelungstechnischen Systemen sind niedrige Antwortzeiten und die damit verbundenen kurzen Busübertragungszyklen ein entscheidender Vorteil bei der Applikationsentwicklung von Fahrerassistenzsystemen. Während sich diese Anforderungen in der Konfiguration des statischen Bussegments niederschlagen, sind die Antwortzeiten und deren Jitter-Toleranzen ein wichtiges Kriterium für die Auslegung des dynamischen Segments. Buslastspitzen dürfen nicht zu arbiträren Verzögerungen von dynamischen Botschaften führen und im schlimmsten Fall die Übertragung eines Datenpakets komplett unterdrücken. Der verfügbare Datendurchsatz wird je nach Anwendungszweck für das jeweilige Übertragungssegment unterschiedlich bewertet, was unter anderem von den Anforderungen im Bereich des Steuergeräteprogrammierungsvorgangs oder den zu applizierenden Multimediafunktionen abhängt.

Auslastung: Die Systemauslastung bezieht sich im FlexZOOMED-Konzept auf die Eigenschaften der logischen Systemarchitektur und der damit verbundenen Ausprägung des FlexRay-Busschedules:

- Anteil leerer FlexRay-Zellen im statischen Segment,
- Anteil leerer FlexRay-Slots im statischen Segment,
- Erfassung existierender Sender-/Empfängerbeziehungen,
- Plattform- und Variantenauslastung.

Das Interesse für die Auslastung eines FlexRay-Busschedules ist nachvollziehbar, da diesem Wert eine vergleichbare Bedeutung zukommt wie der Buslast bei einem CAN-Bussystem. Dabei ist ein wichtiges Unterscheidungskriterium, ob die Auslastungen der FlexRay-Zellen oder der FlexRay-Slots im Buszyklus betrachtet werden[9]. Durch die Analyse der vorhandenen Sender-/Empfängerbeziehungen lassen sich Zentren in der Netzwerkkommunikation ermitteln, was unter dem Aspekt der plattform- und varianten-

[9] Eventuell sind trotz leerer FlexRay-Zellen bereits alle FlexRay-Slots partiell belegt, was eine Integration eines neuen Sendersteuergeräts verhindern würde.

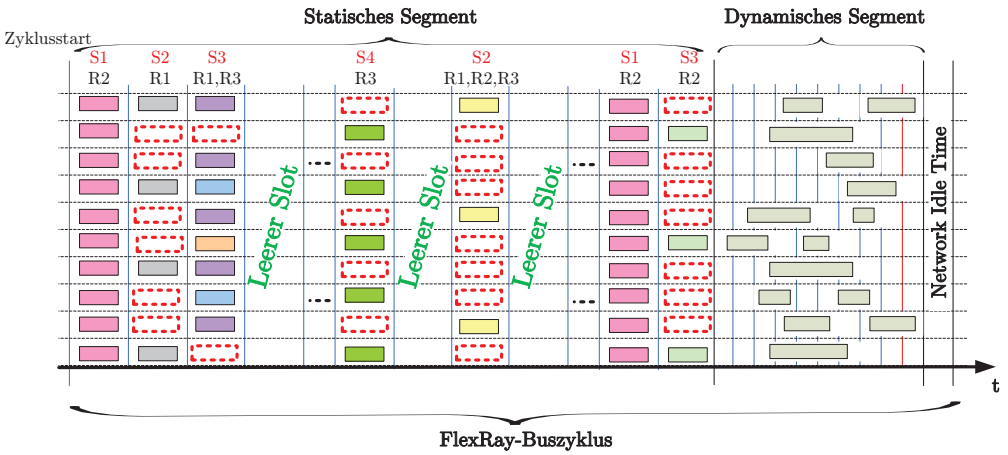


Abbildung 4.29.: Untersuchungsbereiche zur Analyse der FlexRay-Systemauslastung

spezifischen Auslastung des Bussystems eine Hilfestellung für eine sinnvolle Architekturweiterentwicklung bietet.

Prinzipiell lässt sich die Systemauslastung nicht nur auf Busebene, sondern auch auf Steuergeräteebene, betrachten. Durch die hohe Übertragungsfrequenz können durch den Bus induzierte Spitzenlasten in der Prozessorauslastung aufgrund einer hohen Anzahl an Unterbrechungen entstehen. Beispielsweise wird eine geplante statische Entzerrung der Kommunikation im dynamischen FlexRay-Segment beim Systemdesign nicht unterstützt. Mit einer unterbrechungsgesteuerten Detektierung eingehender Botschaften am Prozessor wären bei unmittelbar hintereinander gesendeten Botschaften und einer Nutzdatenlänge von 25Byte beispielsweise mit einem Unterbrechungszyklus am empfangenden Steuergerät von ca. 20μs zu rechnen.

Fehlertoleranz: Für die Betrachtung von Aspekten der funktionalen Sicherheit interessieren Maßnahmen, die den Grad an Fehlertoleranz auf Netzwerkebene erhöhen[10].

- Vorhandene FlexRay-Kanalredundanz,
- Anzahl eingesetzter FlexRay-Sternkoppler (kanalspezifisch),
- Zeitliche und strukturelle Botschaftsreplikation,
- Fehlerreduktionsmaßnahmen (*Node-Management-Vector*),
- Implementierung eines (de)zentralen Buswächterkonzepts.

Sämtliche Parameter in diesem Bereich basieren auf den technisch möglichen Umsetzungen rund um den Bereich der FlexRay-Spezifikation. Dabei muss berücksichtigt werden, dass eine erhöhte Fehlertoleranz aus Sicht aktueller FlexRay-Serienentwicklungen noch nicht explizit gefordert wird. Daher wird in den Fallstudien dieser Arbeit lediglich der aktive FlexRay-Sternkoppler in die Untersuchungen miteinbezogen.

[10]Dieser Bereich ist als einfache Ergänzung zu werten. Ein vollständiges E/E-Architekturbezogenes funktionales Sicherheitskonzept geht über den Kontext dieser Arbeit hinaus.

4.5.2. Weck- und Startvorgang

Wie in Abs. 4.4.4 beschrieben, fundiert jeder FlexRay-Knoten im Netzwerk auf dem Konzept einer Protokollsteuerung POC, welche sämtliche Betriebszustände des Steuergeräts am Bus bedient. Daher ergeben sich für die Entwicklung der E/E-Architektur folgende Aspekte für den Weck- und Startvorgang:

- Betriebszustände,
- Aktivitätszeiten (WakeUp-Verhalten, Teilnetze, Netzwerkmanagement),
- Systemaktivitätsmodi (WakeUp, Normal Passive),
- Systemmanagement.

Für jeden Knoten müssen die zulässigen Betriebszustände in Bezug auf die Spannungsversorgung (Kl.15/Kl.30) definiert werden. Dazu zählen die Konzepte zur Spezifikation des WakeUp-Verhaltens, die Zulässigkeit von partiellen Netzwerkbetrieben und das busübergreifende Netzwerkmanagement. Für den Ablauf der POC müssen die Zustandsübergänge und die Unterstützung optionaler Systemaktiviätsmodi, wie das aktive Wecken des Busses oder die passive Kommunikation, im Sinne eines Fehlerdegradierungsmodells bestimmt werden. POC-übergreifend gilt es ein abstraktes Systemmanagement zu konzipieren, um das Verhalten des Netzwerks clusterweit für bestimmte Systemaktivitätsmodi festlegen zu können.

4.5.3. Komponentenarchitektur

Die eigentliche Komponentenarchitektur eines FlexRay-Knotens lässt sich logisch in Form der zugrunde gelegten Softwarearchitektur und technisch im Sinne der Hardware-Architektur unterscheiden.

- Standardsoftware,
- Spannungsversorgung (Energiemanagement).

Bei der Standardsoftware lässt sich eine feingranulare Spezifikation auf Basis der treibernahen Softwaremodule durchführen, was aber prinzipiell über den Umfang des E/E-Architekturentwurfs hinausgeht. Zur Vereinfachung dient hier die Unterscheidung in Aspekte unterschiedlicher Softwarekonfigurationsvarianten mit spezifischen Ressourcenverbräuchen und Performanzeigenschaften. In Bezug auf die Hardware-Architektur sind neben der bereits definierten FlexRay-Kommunikationshardware (Transceiver, Kommunikationscontroller) die Spannungsversorgungskonzepte (Klemmenkonzepte, Systemstützung (DC/DC-Wandler)) für den Zweck eines erweiterten Energiemanagements festzulegen.

4.5.4. Verlässlichkeit

Neben den skalierbaren Fehlertoleranzkonzepten lässt sich die allgemeine Systemverfügbarkeit durch gezielte Festlegung der Anzahl an Steuergeräten, die zur Systemsynchronisation beitragen oder authorisiert sind das Netzwerk kaltzustarten, beeinflussen.

- Anzahl der *StartUp*-Knoten,
- Anzahl der *Sync*-Knoten.

4.5.5. Systemschnittstellen

Als zusätzlicher Komplexitätstreiber zählt die Herausforderung der Einbettung eines flexiblen zeitgesteuerten Bussystems in eine heterogene E/E-Architektur. Dazu zählen im Detail die Konfektion feldbusverknüpfender Gateway-Architekturen im Fahrzeug sowie die Eigenschaften der Feldbustechnologien, die mit der zeitgesteuerten Bustechnologie durch Datenaustausch interagieren.

Gateways: Ein Gateway spezifiziert in seiner Grundlage eine einfache Verknüpfung zwischen mehreren Feldbussystemen zur selektiven Weiterleitung von Botschaften in verschiedenen Subnetzwerken (*Routing*). Klassischerweise wird das Routing als statische Konfiguration a priori für dedizierte Botschaften definiert, allerdings existieren auch dynamische Routing-Verfahren (beispielsweise bei der Änderung eines Signalinhalts oder der Detektion spezieller aktiv geschalteter Funktionen. Im trivialen Fall werden die Informationen in identischer Ausprägung, dass heißt innerhalb einer Botschaft, von einem Quellbus zu n-Zielbussen weitergeleitet. Der komplexe Fall bezieht sich auf die Neupaketierung von Elementen aus mehreren Botschaften in einer neu generierten zusammengesetzten Botschaft oder auf die Neuberechnung von weiterzuleitenden Signalinhalten, etwa bei der Filterung von Rohwerten einer Sensormessung.

CAN-Bussysteme: Der CAN-Bus weicht durch sein ereignisgesteuertes Übertragungskonzept deutlich von den Eigenschaften eines FlexRay-Systems ab. Daher kommt es bei der Interaktion zwischen CAN- und FlexRay-Systemen zu Seiteneffekten. Speziell im Bereich der Sendelatenz muss diese Tatsache berücksichtigt bleiben. So bildet sich ein kritischer Bereich bei der Verknüpfung im Gateway, da eine zur FlexRay-Zeit asynchron eintreffende Botschaft im Extremfall einen FlexRay-Sendeslot verpasst und konsequenterweise mit der Verspätung von einem Sendezyklus auf dem FlexRay kommuniziert wird.

4.5.6. Globales Netzwerkscheduling

Die Herausforderung des globalen Netzwerkschedulings auf Basis eines TDMA-Netzwerkzugriffsverfahren basiert auf den Erkenntnissen des lokalen Komponentenscheduling mit einer Berechnungskomplexität aus der Klasse der NP-harten Problemen /[PPE+06], [NGH08]/.

Mit einer holistischen Schedulinganalyse wird die Propagation der lokalen Schedulinganalysen auf verteilte Systeme und deren Interferenzen untereinander analysiert. Als eine erforderliche Basisaufgabe muss der Schedule von Betriebssystemtasks auf den Schedule eines zeitgesteuerten Systems abgestimmt werden. In diesem Zusammenhang unterscheiden sich zwei maßgebliche Designphilosophien. Für den Zweck der Entwicklung eines vollständig deterministischen verteilten Systems müssen die in den Steuergeräten vorhandenen Applikationen synchron zur Buskommunikation ausgeführt werden (*applikationssynchrones Busscheduling*). Die zweite Variante verzichtet auf eine hart deterministische Umsetzung der verteilten Funktionen. In diesem Ansatz werden die Botschaften des zeitgesteuerten Feldbussystems arbiträr innerhalb des Busschedules verteilt. Dadurch lassen sich keine deterministische Latenzzeiten für die Zeitkettenmodellierung mehr garantieren (vgl. Abs. 4.2.2.5). Stattdessen können nur noch Maximalwerte für die Übertragungszeit angeben werden, wobei bei der Botschaftsübertragung mit Jit-

tern zu rechnen ist. Diese Variante ähnelt dem Vorgehen bei der CAN-Busentwicklung, bei der die Botschaften je nach Buslast mit geringen Auswirkungen in das Gesamtsystem eingefügt werden können.

Das Risiko einer rein bussynchronen Applikationsentwicklung besteht in den wechselseitigen Abhängigkeiten zwischen der Botschaftspositionierung im Schedule und dem funktionalen Softwaredesign im Steuergerät. Zur Wahrung der Flexibilität müssen Möglichkeiten zur zeitlichen Verschiebung von Botschaften künstlich geschaffen werden. Dies erfolgt durch gezielte Lückenbildung zwischen den Botschaften im FlexRay-Schedule. Dadurch entsteht eine flexible Erweiterbarkeit an verschiedenen lokalen Stellen des Busschedules. Alternativ wäre es prinzipiell möglich das Gesamtsystem mit einem verbreiterten Synchronisationsfenster im Busschedule zu versehen, welches gegebenenfalls später wieder verkürzt werden könnte. Dadurch wäre eine Überarbeitung der FlexRay-Parameter und damit eine zeitliche Verschiebung der Slots im Schedule umsetzbar, was Spielraum für die Abstimmung auf Applikationsebene bringen würde /[Ric07]/. Dieser Ansatz setzt jedoch einen massiven Eingriff in das Gesamtsystem voraus, der zu einem beträchtlichen Nachbearbeitungsaufwand in allen Steuergeräten führt.

Obwohl das Busscheduling den Fragestellungen des Applikationsschedulings nahe kommt, müssen ergänzende Aspekte im Bereich des FlexRay-Designs beachtet bleiben /[Ric07]/:

- Abwägung beim Verhältnis zwischen einer optimierten Systemauslastung und einer maximierten Systemflexibilität,
- Berücksichtigung der Aspekte einer verteilten Systemsynchronisation, der Über- und Unterabtastung im Zyklus (Multiplexstrategien) und des entstehenden Systemjitters bei unregelmäßigen Abständen zwischen den bedateten Sendeslots,
- Analyse der maximalen zeitlichen Schwankungen im Bezug auf applikationsasynchrones Busscheduling,
- Einsatz und Erweiterung des dynamischen FlexRay-Segments.

Konsequenterweise erfordert das globale Netzwerkscheduling eine ausgereifte Methodik für eine holistische Zeitanalyse. Dieses Thema geht im Detail über den Fokus der vorliegenden Arbeit hinaus und wird in diesem Zusammenhang initial für die E/E-Architekturentwicklung aus Sicht der FlexRay-Systemparametrierung behandelt. Speziell für den Bereich des Schedulings /[HHJ$^+$]/ etablieren sich aktuell eigenständige spezifische Werkzeugumsetzungen, die der formalen Analyse heterogener System-on-Chip-Lösungen sowie verteilter Systeme dienen. Die Grundlage dafür basiert auf der Kopplung von lokalen Schedulinganalyseproblemen unter Verwendung von Ereignisströmen, die das Ein- und Ausgabeverhalten von Tasks mit standardisierten Ereignismodellen beschreiben. Die Idee in diesem Konzept fundiert auf der Adaption des Zeitverhaltens der Ereignisse im Ereignisstrom und korreliert im Ansatz mit wissenschaftlich etablierten Methoden /[BS01], [BHS99]/.

Aus Sicht der FlexRay-Systemparametrierung bezieht sich das Schedulingproblem auf die Bestimmung geeigneter *Buszykluslängen*, *Botschaftsgrößen* und *Botschaftsverteilungen* auf das statische/dynamische Segment. Die bei der Bedatung des TDMA-basierten statischen Segments auftretenden Fragestellungen für FlexRay im Kontext des globalen

Netzwerkschedulings werden anfolgend zusammengefasst. In diesem Zusammenhang werden Tupel zur Abbildung der Zeitrestriktionen eingeführt (vgl. /[NGH08]/):

- N_i, die SG-ID, welche ein Signal sendet,
- T_i, die Generierungsperiode eines Signals aus den beteiligten Software-Tasks,
- O_i, der Signal-Offset einer ersten Signalinstanz relativ zum FlexRay-Zyklusstart $(O_i + k * T_i)$,
- DL_i, die Signallänge in Bits,
- A_i, das Signalalter beim Empfang der dafür bereitgestellten FlexRay-Botschaft am Empfängersteuergerät (Framepacking, Sendeprozess, Sendelatenz auf dem Bus, Empfangsprozess, Frameunpacking).

4.5.6.1. Applikationssynchrones Busscheduling

Die Idee des applikationssynchronen Busschedulings ergibt sich aus der zeitlich deterministischen Übertragungskette zwischen den auf unterschiedlichen Steuergeräten ausgeführten Applikationstasks. Dadurch eröffnet sich die Möglichkeit der Entwicklung zeitkritischer Regelsysteme, die über die Steuergerätegrenzen auf mehreren Komponenten verteilt ausgeführt werden (*Satelliten-Koordinator*-Konzept). In Abb. 4.30 wird das Zusammenspiel zwischen zwei Satellitensensoren, einem Koordinatorsteuergerät und einem Aktuator dargestellt.

4.5.6.2. Applikationsasynchrones Busscheduling

Beim applikationsasynchronen Busscheduling besteht das Risiko, dass sich der Sendezeitpunkt einer Sendertask durch das Driften einer Steuergeräteapplikation verschiebt, was im Extremfall zu Botschaftsverlusten führen kann (s. Abb. 4.31). Ein Angriffspunkt in diesem Bereich basiert auf der Spezifikation der zyklischen Ausführungszeiten einer Applikations- und Sendetask, die jedoch aufgrund restriktiver Vorgaben hinsichtlich Rechenperformanz maximiert auszulegen sind.

Die maximal zu applizierende periodische Übertragungsrate auf FlexRay orientiert sich an der Bedingung, dass die Alterung bei der Netzwerkübertragung A_i nicht die aus der Anwendung vorausgesetzte Signalaktualität D_i übersteigt:

$$T_i = max\left\{2^n \mid n \in \mathbb{N}, n \leq 6 \text{ und } \exists O_i^{FR}, \text{ so dass } A_i \leq D_i\right\}.$$

Die Auslastung eines einzelnen FlexRay-Slots h lässt sich über folgende Gleichung identifizieren:

$$\text{Wenn gilt: } \sum_{f_i \in G_h} \frac{1}{CR_i^{FR}} = 1, \text{dann werden alle 64 Zyklen eines FlexRay-Slots } h \text{ benutzt.}$$

Theorem: Ein unvollständig über seine Zyklen befüllter FlexRay-Slot lässt sich bei inkrementeller Anordnung der zu verteilenden Botschaften anhand deren Übertragungs-

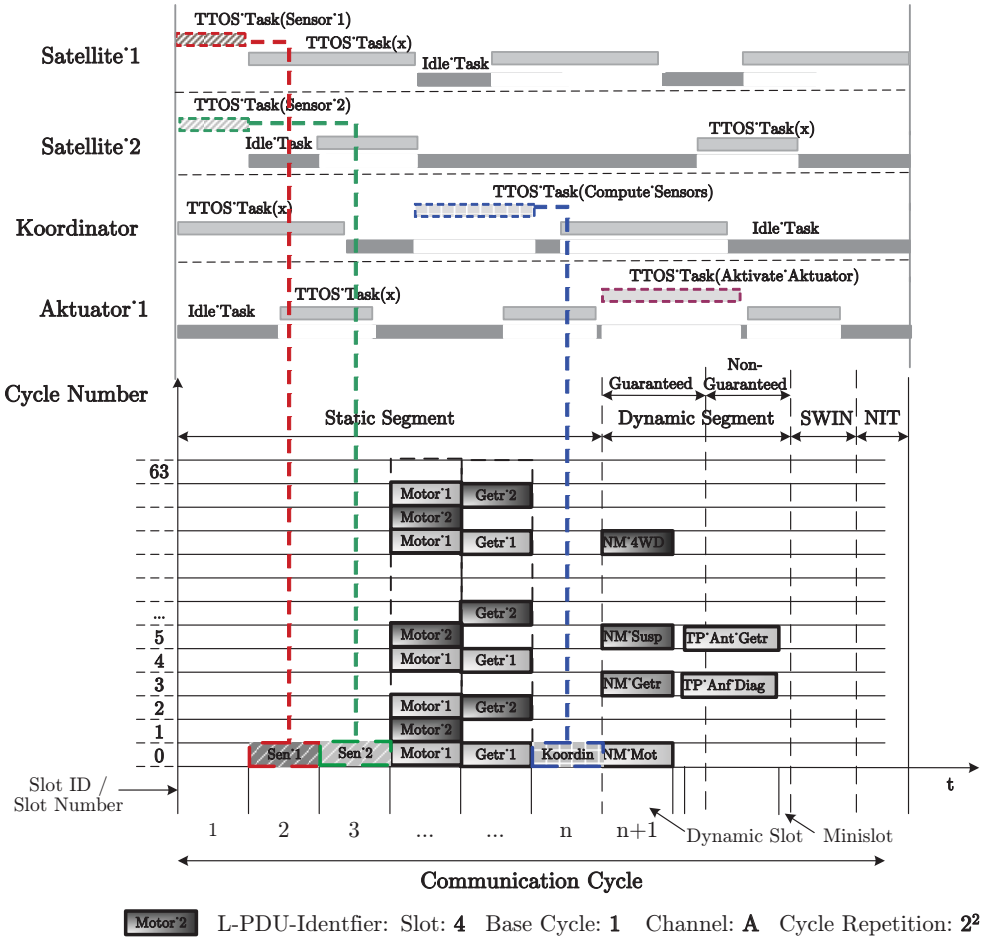


Abbildung 4.30.: Beispiel für synchrones Busscheduling über FlexRay (Satelliten-Koordinator-Konzept)

perioden stets vollständig befüllen.

Beweis: Falls Slot h mit einer Gruppe G an Botschaften partiell befüllt worden ist, so ist für die nächste zu platzierende Nachricht k in diesem Slot mit CR_k mindestens eine Lücke innerhalb des Basiszyklusintervalls $[0, CR_k - 1]$ vorzuhalten.

Dies führt zur folgenden Vorbedingung und deren Umformung:

$$\sum_{f_i \in G_h} \frac{1}{CR_{f_i}} < 1$$

$$CR_k - \sum_{f_i \in G_h} \frac{CR_k}{CR_{f_i}} > 0 \Rightarrow \sum_{f_i \in G_h} \frac{CR_k}{CR_{f_i}} \leq CR_k - 1$$

und $\sum_{f_i \in G_h} \frac{CR_k}{CR_{f_i}}$ entspricht die Auslastung des Slots h durch die platzierten Botschaften aus der Gruppe G_h im Slotzyklusintervall $[0, CR_k - 1]$.

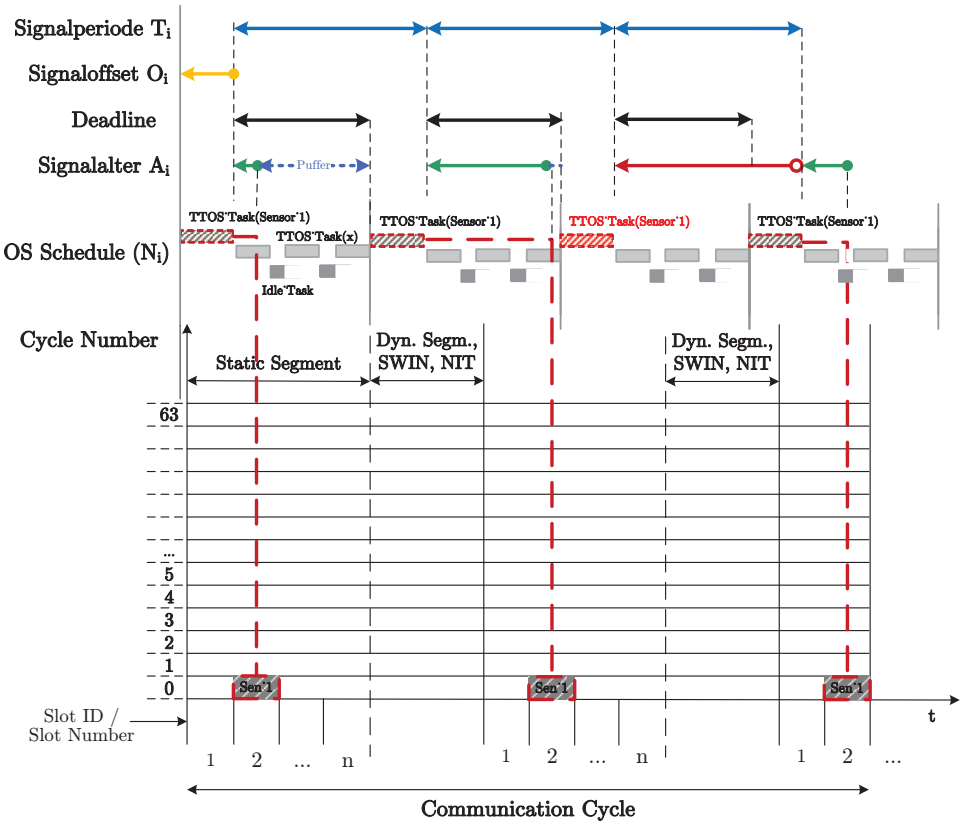


Abbildung 4.31.: Beispiel für asynchrones Busscheduling über FlexRay (Satelliten-Koordinator-Konzept)

4.5.6.3. Designziele

Beim Kommunikationsdesign stehen speziell bei der Implementierung des applikationsasynchronen Busschedulings buslast- und performanzoptimierende Metriken im Vordergrund, welche als Scheduling-Ziele angewendet werden:

1. Identifikation einer passenden Konfiguration hinsichtlich der notwendigen Signalaktualität aller Signale (bei applikativ asynchronem Buszugriff),

2. Platzierung von synchronisierten Kommunikationstasks in Ergänzung mit synchronisierten Applikationstasks (bei applikativ synchronem Buszugriff),

3. Bandbreitenoptimierte Erweiterbarkeit der Busschedules,

4. Plattform- und Variantenscheduling.

4.5.6.4. Systemintegration heterogener Busschedulingkonzepte

Neben dem Einsatz rein applikationssynchroner oder applikationsasynchroner Systeme ergibt sich bei der Architekturentwicklung die Fragestellung hinsichtlich der Verwendung gemischt synchron/asynchroner FlexRay-Anwendungen. Zur Analyse des Verhaltens dieser heterogen ausgelegten Systeme innerhalb eines FlexRay-Clusters wird eine

Fallstudie mit folgendem Aufbau untersucht (s. Abb. 4.32).

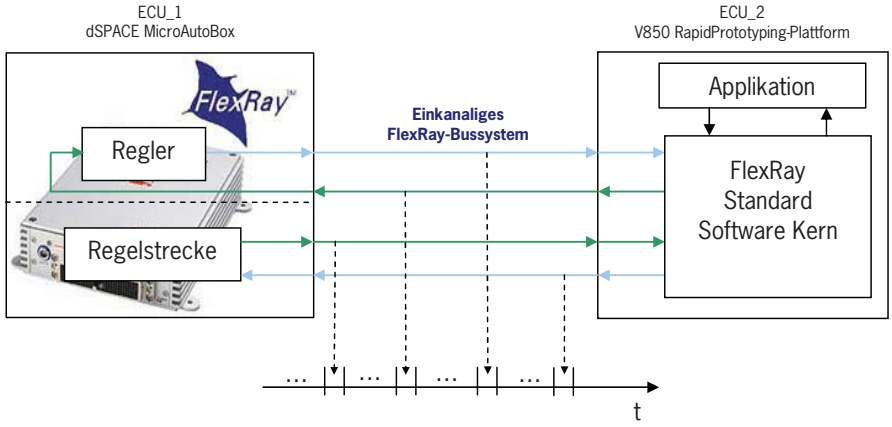


Abbildung 4.32.: Zwei-Knoten-Cluster zur abgleichenden Simulation von asynchronen und synchronen Regelung über FlexRay

Zur Vereinfachung basiert das System auf einer 2 Knoten-FlexRay-Topologie. Für die Analyse des Umfelds einer performanten Regelung innerhalb einer zeitgesteuerten Netzwerkarchitektur wird der praxisnahe Fall einer FlexRay-Applikation nachgebildet. Die beiden Knoten sind nach Tab. 4.8 spezifiziert:

Bezeichnung	ECU1	ECU2
Applikationskonzept	Synchron	Asynchron
Funktionsmodule	Regelmodell, Regler	Gateway
Botschaftstiming (*Cycle Offset*) (μs)	1344, 3840	320, 384, 2880
Maximale Ausführzeit (μs)	1024, 1960	1480, 1536

Tabelle 4.8.: Eigenschaften der Netzwerkknoten

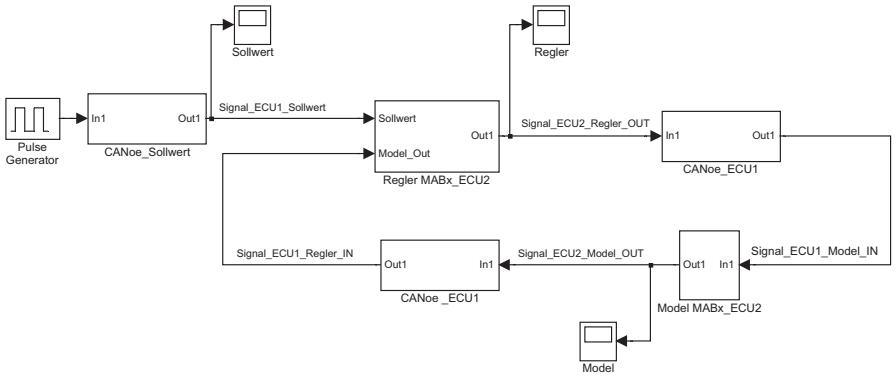


Abbildung 4.33.: Grundlegende Reglerstruktur mit Sollwert, Regelungsblock und Regelstrecke (Modell)

Ziel dieser Untersuchung ist die Auswirkung der Task- und Busschedules in gemischt synchron /asynchronen FlexRay-Applikationen auf die Qualität einer PID-Reglerumsetzung (s. Abb. 4.34). In dem Fallbeispiel wird das Regelmodell und der Regler

innerhalb einer Echtzeitumgebung mit einem Betriebssystem für harte Echtzeitanforderungen (*OSEKtime*) (*ECU1*) umgesetzt. Der Datenfluss zwischen der Regelstrecke und dem Regler wird dabei über das FlexRay-Cluster in einem verteilten System realisiert. Mit der Tunnelung der Daten über ein zweites FlexRay-Steuergerät mit einer vom Cluster entkoppelten Betriebssystemausführung (*osCAN*) (*ECU2*) entsteht die Abbildung eines asynchronen verteilten Systems. Durch die asynchrone Datenverarbeitung lassen sich dabei die Wechselwirkungen zwischen der asynchronen Regelung und dem zeitgesteuerten Bussystem nachweisen.

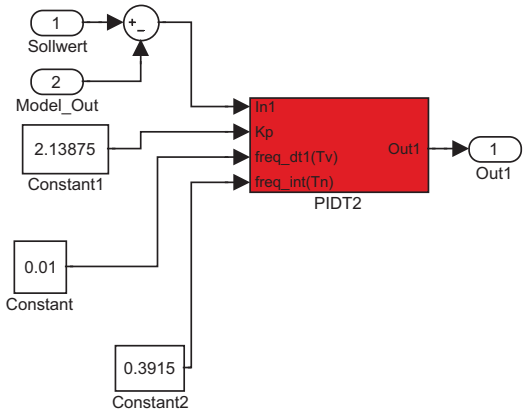


Abbildung 4.34.: PID2-Regelkonzept als Basis der implementierten Regelung

Während des Datenaustauschs zwischen *ECU1* und *ECU2* läuft in der Untersuchung eine Zählvariable in der Kommunikation mit, deren Werte im Intervall $[0;10]]$ verlaufen und nach einem Sollwert (*0* oder *10*), der von *ECU2* vorgegeben wird, nachgeregelt werden (s. Abb. 4.35).

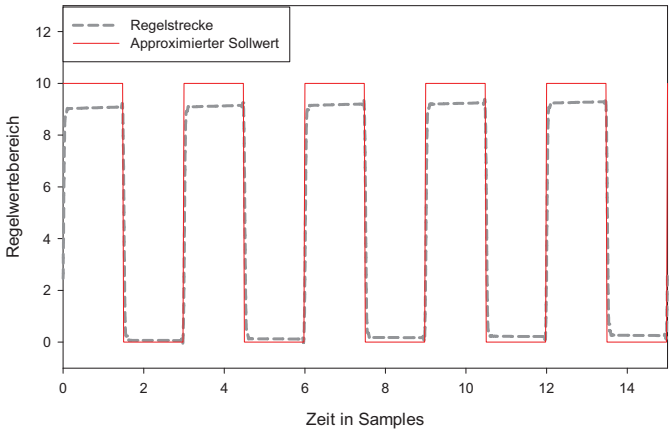


Abbildung 4.35.: Sollwert und darauf geregeltes Streckenmodell

In dem Zusammenhang erfolgt eine Inkrementierung der Zählvariable in *ECU1* in Abhängigkeit von dem vorgegebenen Zählvariablenwert der *ECU2* in einem zeitlichen Abstand von 2 Buszyklen. In Abb. 4.36 wird ein Ausschnitt über ca. 35 Buszyklen bei der aktiven Regelung dargestellt.

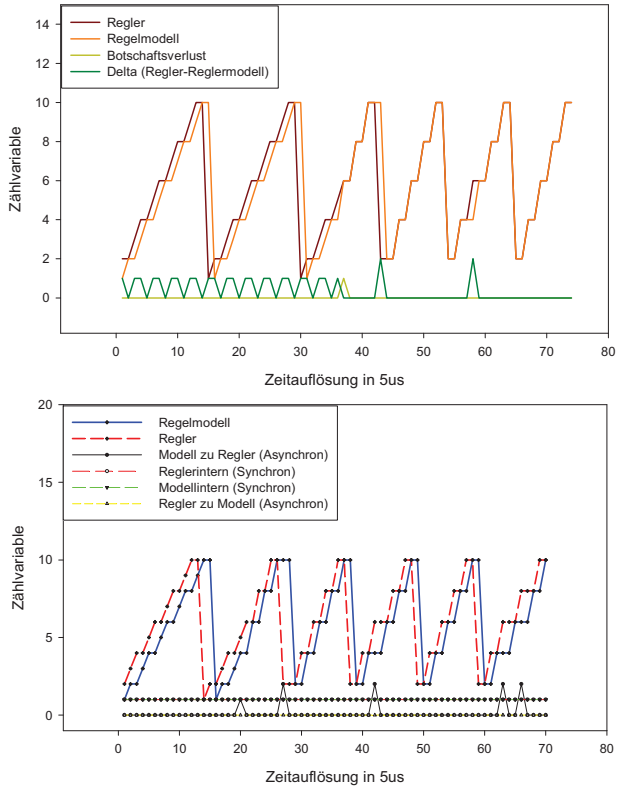


Abbildung 4.36.: Auswirkungen von Botschaftsverlusten bei einer partiell asynchron ausgelegten FlexRay-Systemkonfiguration

Beobachtung 1:

Es bestätigt sich, dass die Zählvariablenanpassung des Reglermodells cnt_{Modell} der Zählvariablenanpassung des Reglers cnt_{Regler} um zwei Buszyklen nacheilt. In Zyklus 37 wird offensichtlich, dass bei der asynchronen Systemauslegung *Jitter*-Effekte entstehen, die im Extremfall zum Botschaftsverlust führen (s. Abb. 4.37).

Erklärung:

Im Falles des Botschaftsverlusts springt die nacheilende Zählvariable cnt_{Modell} auf den identischen Wert von cnt_{Regler}. Auf diesen Vorgang reagiert der Regler korrekterweise zwei Buszyklen später, indem der verfrüht auftretende Wert $cnt_{Modell} = 10$ im Buszyklus 40 durch einen übersprungenen Wert $cnt_{Regler} = 1$ kompensiert wird. Das erklärt die Tatsache, dass sich der Botschaftsausfall in *ECU2* zeitlich versetzt auf *ECU1* propagiert. Entscheidend für die Entwicklung zeigt sich konsequenterweise, dass die Performanz und die Anordnung der Sende- und Empfangsprozesse im asynchronen FlexRay-Steuergerät signifikant Einfluss auf die Funktionsweise des verteilten Systems nehmen.

Beobachtung 2:

Neben dem in Abb. 4.37 dargestellten Botschaftsverlust kommt es, wie in Abb. 4.36 ersichtlich, zu einem komplementären Effekt, bei dem je nach Sende- und Empfangsverhalten eine Botschaft zweimal gesendet wird (*Wert 10 in Zyklus 27*).

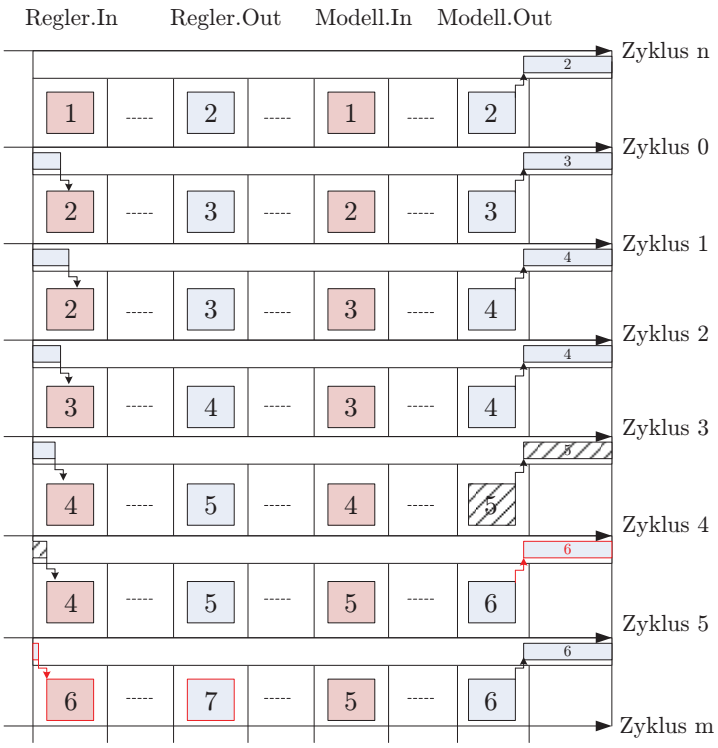


Abbildung 4.37.: Rekonstruktion des ermittelten Botschaftsverlust bei der Übertragung von Botschaften am asynchronen FlexRay-Steuergerät

Erklärung:

Aus dem doppelten Sendevorgang lässt sich schließen, dass speziell die Sende-Task mit einem *Jitter* behaftet ist. Durch das Driften der Steuergeräteapplikation gegenüber der Buszykluszeit werden die empfangenen Werte aus *Slot 65* in *Zyklus x* partiell bereits in *Slot 5* in *Zyklus x+1* auf dem Bus weiterversendet. Falls die Applikation stärker nacheilt wird die Sendetask erst nach *Slot 5* in *Zyklus x+1* beendet und die Information folglich erst in *Slot 5* in Zyklus *x+2* verschickt. Das Nacheilen wird in Abb. 4.36 durch den Wert *2* gekennzeichnet und ein Botschaftsverlust durch den Wert *1*[11].

Fazit:

Insgesamt lässt sich nachweisen, dass ein regelungstechnisches System mit bussynchron ablaufenden Applikationen deterministisch auf ein zeitgesteuertes verteiltes System abgebildet werden kann. Zum Bus asynchron laufende Applikationen bergen jedoch das Risiko, dass Seiteneffekte zu Botschaftsverlusten oder -wiederholungen führen, welche sich signifikant auf die Regelungsergebnisse auswirken. In der dargestellten Fallstudie wirkt sich die Problematik dadurch beispielsweise auf die Quantisierungsstufen in der Regelung aus (alle ungeraden Werte zwischen *1* und *10* werden nicht mehr abgebildet).

[11] Die Werte sind in Betragsform angegeben. Das bedeutet, dass die Datenweiterleitung im *Regler-Modell* bei Fehlern mit den Werten *1* und *2* versehen wird. Die modell- und reglerinterne Inkrementierung erfolgt dagegen innerhalb der synchronen Applikation der *ECU1* und ist fehlerfrei.

4.6. Optimierung im Systementwurf

„The First Rule of Program Optimization: Don't do it. The Second Rule of Program Optimization (for experts only!): Don't do it yet." - Michael A. Jackson

In diesem Kapitel werden Potentiale hinsichtlich der Einstufung und Bewertung von FlexRay-Konfigurationen dargestellt. Dabei lassen sich Aspekte eines optimierten Einsatzes der Technologie zur Applizierung in einer gegebenen E/E-Architektur aufzeigen.

4.6.1. Optimierungsziele in flexiblen zeitgesteuerten Architekturen

Aus Sicht der Netzwerkspezifikation muss beim Entwurf einer verteilten zeitgesteuerten Systemarchitektur ein heterogenes Feld an Designanforderungen (s. Abb. 4.38) berücksichtigt werden /[BMG07]/.

Leistung: Ein gemischt zeit-/ereignisgesteuertes Konzept erfordert ein gezieltes Erfassen von minimalen Abtastraten einer zyklisch zeitgesteuerten Kommunikation. Diese Einschränkung beeinflusst die Länge von Kommunikationszyklen und deren Slotverwaltung für die Inter- und die Intrakommunikation eines Mikrocontrollers.

Integrierbarkeit: Für einen langlebigen Systemeinsatz mit zusätzlichen Systemintegrationsaufgaben müssen Bandbreitenanteile auf Buskommunikationsebene vorgehalten werden. Da sich verschiedene Abschnitte eines Kommunikationszyklus unterschiedlich klassifizieren lassen und ein Vorhalt mit unterschiedlichen Konzepten erzeugt werden kann (Multiplexing, Botschaftsvervollständigung, Reserveslots), gilt eine Integrationsstrategie als hochkomplex.

Systemmanagement: Das Aktivitätsmanagement eines Busses, basierend auf der Konfiguration jedes einzelnen Teilnehmers, hat Einfluss auf das Buskommunikationsdesign. Welche Knoten eine Coldstart- oder ein Sync-Fähigkeit erhalten ist entscheidend. Optimierungsziele basieren auf niedrigem Energieverbrauch und geringer Fehleranfälligkeit während kritischer Betriebsphasen des Systems (StartUp/ShutDown).

Plattformkonzept: Kommunikationsbeziehungen auf Netzwerkebene werden in plattform- und knotenspezifische Anteile zusammengefasst. Die Idee der Slot- oder Signalklassifikation in Bezug auf Applikationsdomänen muss diesbezüglich berücksichtigt werden.

Datendurchsatz: Neben der Leistungsbetrachtung für performante Abtastraten auf Buskommunikationsebene interessiert vorrangig eine effiziente Bandbreitenvergabe mit hoher Nettodatenrate. Dazu zählt auch die Anforderung eines schnellstmöglichen Versendens von größeren Datenpaketen für die Zwecke einer performanten Steuergeräteprogrammierung (z.B. bei der Bandendeprogrammierung). Die Systemstabilität und -robustheit muss dabei garantiert bleiben.

Topologien: Eine Modifikation eines Steuergerätenetzwerks in der technischen Systemarchitektur erfordert in der Regel eine Neubewertung der physikalischen Ausprägung der Netzwerktopologie. Veränderte Leitungslängen oder das Einfügen von aktiven oder

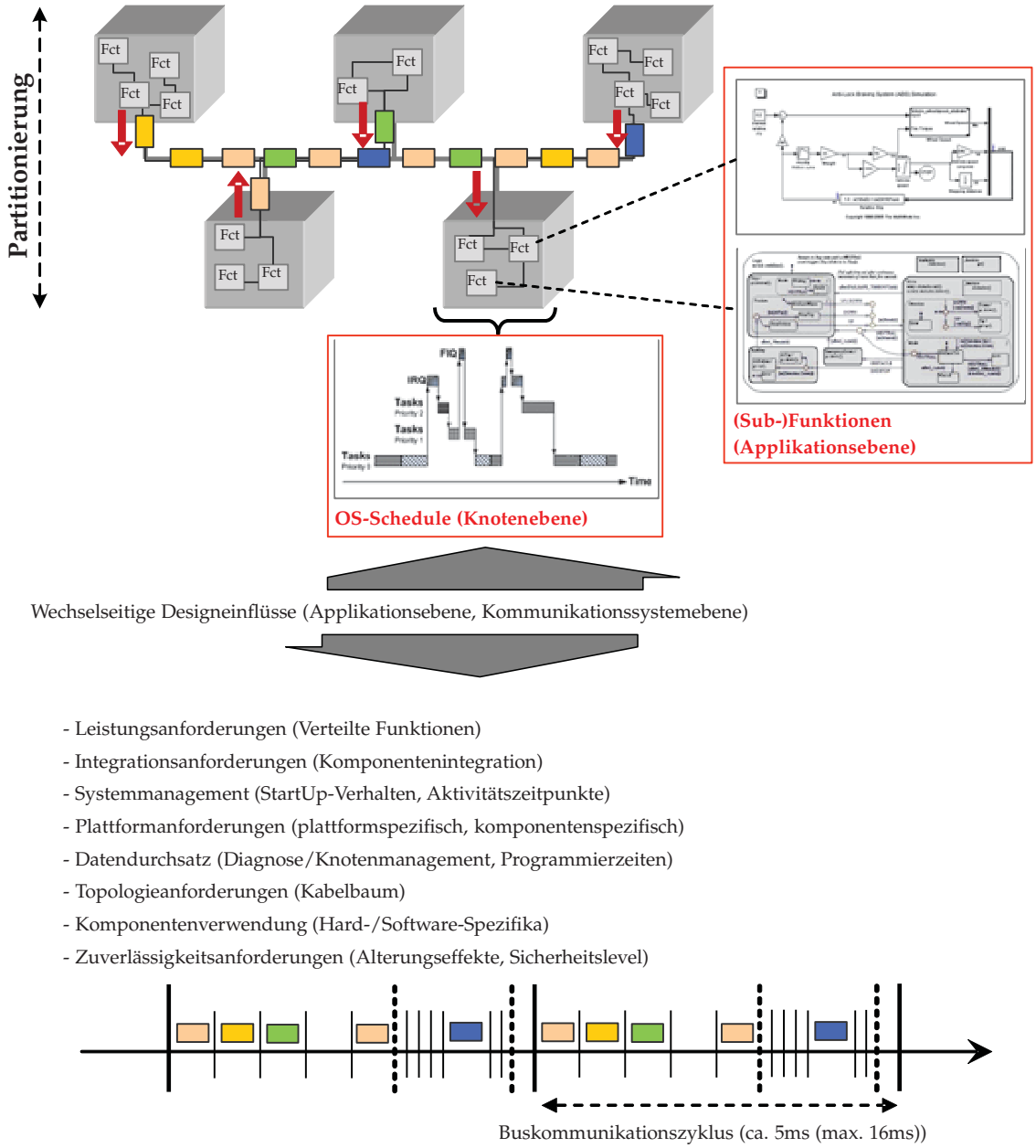


Abbildung 4.38.: Wechselseitige Abhängigkeit zwischen den allgemeinen Anforderungen an eine Systemarchitektur und einem FlexRay-Systemdesign

passiven Sternkopplern erzeugen Effekte im Bezug auf das Zeitverhalten des verteilten Systems. Daher müssen Strategien für zweckgebundene Modifikationen an technischen Systemarchitekturen mit flexiblem zeitgesteuerten Kommunikationssystem erarbeitet werden.

Komponentenverwendung: Normalerweise besitzt jede Hardwareeinheit (z.B. Transceiver oder Kommunikationscontroller) Einschränkungen in ihrem Anwendungsspektrum. Es müssen Restriktionen hinsichtlich der Konfigurierbarkeit von Spezifikationsparametern, der spezifischen Pufferhandhabung, den Kodierungs- oder Dekodierungseigenschaften oder für den Arbeitstemperaturbereich beachtet werden, welche den Zielraum

für Systemlösungen limitieren.

Zuverlässigkeit: Der Gebrauch von Kommunikationssystemen und die mit der Nutzung verbundene Kritikalität in Bezug auf die Benutzersicherheit erfordert eine gesonderte Betrachtung (vgl. Abs. 2.3.5).

4.6.2. Systemqualität

Die Erfassung der Qualitätseigenschaften einer flexiblen zeitgesteuerten Architektur ist zum aktuellen Zeitpunkt nicht als Standard definiert. Daher empfiehlt sich in erster Instanz eine Orientierung an etablierten Standards von verwandten Entwicklungsbereichen, beispielsweise dem ISO9126-Standard für Software-Qualität /[Int91]/ (s. Abb. 4.39). Eine Betrachtung der Hauptmerkmale lässt sich in mehreren Punkten auf eine flexible zeitgesteuerte und verteilte Architektur mit hohen Performanzansprüchen übertragen:

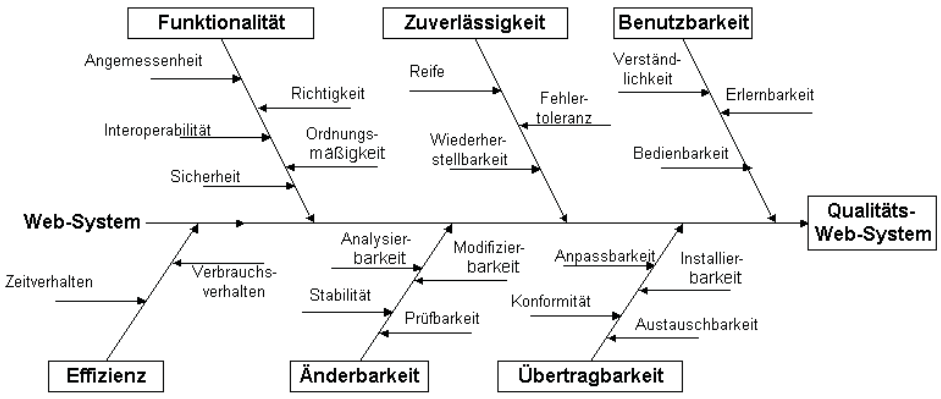


Abbildung 4.39.: Allgemeine Systemqualität nach dem ISO9126-Standard

Funktionalität: Aus Sicht der E/E-Architektur bezieht sich dieser Aspekt auf die Komponentenkonfiguration im Netzwerk und deren Kommunikationsprofil. Sämtliche zu verbauende Halbleiterkomponenten, ausgehend von Kommunikationscontrollern und Transceiver-Architekturen (aktive Sternkoppler) bis hin zu Bauteilen wie Oszillatoren und Konzepte aus der Schaltungstechnik (PLL, *Clock Tree*), müssen bei der Systemintegration berücksichtigt werden. Das Kommunikationsprofil entsteht aus den umzusetzenden Abtastraten, die sich aus den zu entwickelnden regelungstechnischen Systemen ableiten lassen, unter Berücksichtigung der quantitativen Ausprägung dieser zu übertragenden Daten. Entscheidend muss dabei auch das Thema aller potentiellen Kommunikationsbeziehungen $n : m$ im Netzwerk beachtet werden. Speziell das Thema *Interoperabilität* wird im Zusammenhang mit FlexRay zwischen den Fahrzeugherstellern analysiert /[RMBK07]/.

Zuverlässigkeit: Da zeitgesteuerte Bussysteme konzeptionell für den Einsatz in sicherheitskritischen Systemen entwickelt worden sind, stellt sich die Frage nach einer korrekten Umsetzung der skalierbar einzusetzenden Fehlertoleranzmechanismen. Diese ergeben sich bei FlexRay durch den optional implementierbaren redundanten Datenkanal, die abgestuften Fehlerbehandlung auf der Protokollebene POC, den NM-Vektor in den

Nutzdaten und die verteilte Synchronisation. In abgestufter Form stellt sich die Frage nach Wiederherstellbarkeit bei Fehlerzuständen (*Recovery Mechanismen*), beispielsweise nach byzantinischen Fehlern oder bei Datenverlust auf dem Kommunikationsmedium. Als eigenständiges Thema muss die Anforderung hinsichtlich der angestrebten Lebensdauer des Systems unter Berücksichtigung eventueller Alterungseffekte innerhalb verbauter Komponenten behandelt werden.

Benutzbarkeit: Der Einsatz von flexiblen Kommunikationsmechanismen im Rahmen von Multiplexing auf Zyklus-, Slot-, oder Pufferebenen erschwert die Anbindung und Interpretation innerhalb softwaretechnischer Anbindungen. Auch eine Ergänzung der Flexibilisierung mithilfe von logischem Multiplexing auf Botschaftsebene erhöht die Komplexität bei der Applikationsverknüpfung im Steuergerät und senkt den Grad an Verständlichkeit des gesamten Kommunikationskonzepts des Systems.

Effizienz: Die Systemeffizienz überträgt die Anforderungen aus dem Funktionalitätsbegriff, indem erstellte Kommunikationsprofile in eine effiziente Bedatung des Bussystems umgesetzt werden. Eine Herausforderung dabei liegt in der sinnvollen Verteilung von Signalen in den unterschiedlichen Buszyklen des Schedules sowie einer geeigneten Wahl von Zykluslänge und Multiplexing.

Änderbarkeit: Starre Konzepte bei der Implementierung von Bussystemen erzeugen in der Regel hohe Kosten bei der Gesamtsystementwicklung im Bereich der Wiederaufbereitung und Weiterverwendung (*Reengineering*) von Einzelkomponenten oder -modulen. Daher muss der Wunsch nach simplen Modifikationen im Zusammenhang mit Kommunikationserweiterungen oder bei Adaption der Sendeeigenschaften, beispielsweise durch Veränderung von Empfängeradressen oder Zykluszeiten, pragmatisch umsetzbar bleiben. Massive Eingriffe durch Austausch kompletter Komponenten (Steuergeräte) sollten unter Berücksichtigung von Kompatibilitätsanforderungen realisierbar bleiben.

Übertragbarkeit: Unter Übertragbarkeit versteht sich aus E/E-Architektursicht einerseits der Austausch von Komponenten im Entwicklungszyklus einer Fahrzeugbaureihe als auch die Möglichkeit der Plattform- und Variantenbildung auf Funktions-, Software- und Hardwareebene.

4.6.3. Grundlagen der mathematischen Optimierung

Das Gebiet der mathematischen Optimierung gilt als weitläufiger eigenständiger Forschungsbereich. Daher beschränkt sich dieser Abschnitt auf die wesentlichen Elemente zum Verständnis der anfolgend entwickelten Lösungen im Rahmen des FlexRay-Systemdesigns. Für eine weitere Vertiefung der adressierten Methoden und zum allgemeinen Überblick wird auf einschlägige Literatur weiterverwiesen /[Lue97], [Alt02], [BV04]/.

Definition 4.6.1 (Optimierungsproblem) *Sei $D \subset \mathbb{R}^n$ eine offene Menge $f : D \to \mathbb{R}$ und $F \subset D$. Dabei stellt sich das Problem, die Funktion f auf F zu minimieren in der Form $min_{x \in F} f(x)$. f wird in dem Zusammenhang als **Zielfunktion**, F als **zulässige Menge** und die Elemente $x \in F$ als **zulässige Punkte** bezeichnet. Wird die Menge F durch Nebenbedingungen (**Restriktionen**) beschränkt, so bezeichnet $min_{x \in F} f(x)$ ein **restringiertes Optimierungspro-***

***blem (OP)**. Die Menge F wird dabei durch Gleichungen und/oder Ungleichungen definiert.*

□

Die im Kontext der FlexRay-Parametrierung entwickelten OP basieren auf eigens entwickelten Zielfunktionen. Sie definieren eine Abbildung aus einem Vektorraum V in einen Zahlenkörper K. Der Vektorraum wird dabei individuell für jede Zielfunktion über die notwendig zu betrachteten FlexRay-Parametrierungsregeln gebildet. Daher werden die OP als *Funktionale* aufgefasst. Bei der Suche nach der Lösung eines OP stellt sich neben der Frage nach der Existenz einer Lösung für das OP auch die Frage nach der Eindeutigkeit einer gefundenen Lösung. Nach /[BV04]/ korreliert die Eigenschaft einer eindeutigen Lösung mit den Konvexitätseigenschaften der Zielfunktion des betrachteten OP.

Definition 4.6.2 (Konvexitätseigenschaft) *Die Menge $C \subset \mathbb{R}^n$ heißt konvex, falls für beliebige Punkte $x, y \in C$ gilt*

$$\{(1-t)x + ty | 0 \leq t \leq 1\} \subset C.$$

Für die Verbindungsstrecke

$$[x, y] := \{(1-t)x + ty | 0 \leq t \leq 1\},$$

gilt $[x, y] \subset C$. Der Konvexitätsbegriff lässt sich übertragen auf die Zielfunktion $f : D \to \mathbb{R}$ mit $D \subset \mathbb{R}^n$ und $D \subset C$ mit C als nichtleere, konvexe Menge, wobei gleichermaßen folgt

$$f((1-t) * x + t * y) < (1-t) * f(x) + t * f(y)$$
$$\textit{für alle } x, y \in C \textit{ und } t \in [0, 1].$$

□

Ergänzend zu den Konvexitätseigenschaften der Zielmenge und des Zielfunktionals eines OP, ist es entscheidend Aussagen über deren Stetigkeit formulieren zu können. Dies erfolgt per Analyse des vorliegenden Funktionals auf Differenzierbarkeit. Dabei muss beachtet werden, dass das vorliegende Funktional Einschränkungen aufweist, etwa mit Sprungstellen in seinem Wertebereich. Die Unterscheidung und Abgrenzung passender Differenzierbarkeitsbegriffe werden aus /[Lue97]/ übernommen und anfolgend definiert.

Definition 4.6.3 (Gateaux-Ableitung) *Sei X ein Vektorraum und $x \in D \subset X$, h sei frei wählbar in X. Wenn der Grenzwert*

$$\partial T(x; h) = \lim_{\alpha \to 0} \frac{1}{\alpha} [T(x + \alpha * h) - T(x)]$$

existiert, heißt dieser Grenzwert Gateaux-Ableitung der Transformation T an der Stelle x mit Richtung h. Wenn dieser Grenzwert für alle $h \in X$ existiert, dann heißt T Gateaux-differenzierbar an der Stelle x.

□

Das als *Richtungsableitung* bezeichnete Gateaux-Differential lässt sich weiter verstärken, falls es sich bei X um einen normierten Vektorraum handelt. Dieser stärkere Differenzierbarkeitsbegriff bezieht sich auf die *Frechet-Ableitung*.

Definition 4.6.4 (Frechet-Ableitung) *Sei T eine Transformation, die auf einer offenen Menge D in einem normierten Raum X mit Wertebereich im normierten Raum Y definiert ist. Existiert für ein festes $x \in D$ und alle $h \in D$ ein bezüglich h stetiger und linearer Grenzwert $\partial T(x;h) \in Y$, so dass*

$$\lim_{\|h\| \to 0} \frac{\|T(x+h) - T(x) - \partial T(x;h)\|}{\|h\|} = 0$$

gilt, dann heißt T Frechet-differenzierbar an der Stelle x. $\partial T(x;h)$ bezeichnet dabei das Frechet-Differential von T an der Stelle x in Richtung h.

□

Der nachfolgend beschriebene Zusammenhang zwischen Gateaux- und Frechet-Differenzierbarkeit wird aus /[Lue97]/ übernommen und aus Gründen der Vollständigkeit hier aufgeführt. Für den Beweis wird auf /[Lue97]/ weiterverwiesen.

Satz 4.6.5 *Wenn das Frechet-Differential von T an der Stelle x existiert, dann existiert auch das Gateau-Differential an der Stelle x und beide stimmen überein.*

Sämtliche hiermit definierten Grundlagen erweisen sich als notwendig in der Analyse der im nächsten Abschnitt entwickelten Funktionale in Bezug auf die Optimierung von FlexRay-Systemparametrierungen. Im Folgenden werden potentielle Optimierungsmöglichkeiten von FlexRay-Systemkonfigurationen vorgestellt.

4.6.4. Optimierungsprobleme in FlexRay-Parametersätzen

Die Festlegung der Werte für einen FlexRay-Parametersatz erfolgt nach den Berechnungsvorschriften der FlexRay-Spezifikation /[Fle05a]/. Diese setzen sich im Wesentlichen aus einer Vielzahl an Ungleichungen und Gleichungen zusammen, welche aufeinander aufbauen. Die Parameter definieren dabei sämtliche Konfigurationsaspekte zur Etablierung einer *stabilen Netzwerkkommunikation* unter den gegebenen Randbedingungen, etwa den konkreten gegebenen *physikalischen Topologieausprägungen* und den *dedizierten Hardwareeigenschaften* oder in abstrakter Form, beispielsweise über die Festlegung der *bevorzugten Botschaftsgröße* und *Slotanzahl* im statischen Segment.

Allgemeine Gütekriterien oder Kennzahlen eines für den Anwendungszweck optimierten Parametersatzes in einems Fahrzeugserienprojekt werden in /[Fle05a]/ nicht adressiert. Ein analoges Dokument zu /[Fle06b]/, einer Ergänzung der Spezifikation für die physikalische Schicht /[Fle04b]/, ist seitens des FlexRay-Konsortiums nicht standardisiert entwickelt worden. Konsequenterweise wird die FlexRay-Technologie innerhalb dieser Arbeit hinsichtlich allgemeiner Gütekriterien analysiert und die identifizierten Aspekte formalisiert. Auf Basis der beiden Grundspezifikationen /[Fle04b], [Fle05a]/ lassen sich in Ergänzung mit /[Rau07b]/ für Ebenen der physikalischen Übertragungsschicht und der Protokollschicht folgende Gütekriterien identifizieren, die es zu optimieren gilt. Folgende Optimierungsbereiche beziehen sich auf die elektrische Bitübertragungsschicht:

Flexibilität: Unter *Flexibilität* versteht sich eine topologisch modular auslegbare Busarchitektur, in der bestenfalls alle Permutationen einer gegebenen Steuergerätemenge ohne Modifikationen im Hard- und Softwarebereich realisiert werden können. Eine Vermeidung von Variantenanfertigungen für verschiedene Fahrzeugderivate stellt einen signifikanten Kostenvorteil in der Serienproduktion dar.

Erweiterbarkeit: Als Verschärfung des Flexibilitätsbegriffs gilt es die Integration neuer Systemkomponenten (ECUs) ohne Adaptionsaufwand der existierenden Lösungen zu berücksichtigen. Speziell die *Erweiterbarkeit* an verschiedenen Stellen des verbauten Bordnetzes erfordert eine Beachtung hardwarespezifischer Effekte, etwa der Oszillatorqualität oder der Transceivereigenschaften.

Robustheit: Die Funktionstüchtigkeit der Busarchitektur in verschiedenen physikalischen Grenzbereichen wird durch die *Robustheit* ausgedrückt. Temperatureinflüsse und Alterungsprozesse wirken dabei speziell als temporaler Einfluss durch eine Veränderung der Systemgenauigkeit.

Für die Protokollebene ergeben sich folgende Optimierungsbereiche:

Nutzdatendurchsatz/Zeiteinheit: Dieses Gütekriterium zielt auf die Bewertung des Netto-Nutzdatenanteil der über FlexRay übertragenen Daten bezogen auf ein bestimmtes Zeitintervall (Makrotick, Slot, Buszyklus) (*Nutzdateneffizienz*). Diese Wert bezieht sich ausschließlich auf die Konfiguration des statischen FlexRay-Segments.

Übertragungslatenz/Sendeauftrag: Durch die ereignisgesteuerten Eigenschaften des dynamischen FlexRay-Segments stellt sich die Frage nach einem anwendungskonformen Verhältnis zwischen der maximalen Übertragungslatenz eines Sendeauftrags, etwa bei *Burst*-Zuständen im System, und der dafür notwendigen zeitlichen Ausprägung des dynamischen Segments (*Dynamisches-Segment-Layout*).

Netzwerkruhephasen/Zeiteinheit: Durch das dynamische FlexRay-Segment, das Symbol-Fenster und die Netzwerkruhephase kommt es zu einer periodischen Unterbrechung des statischen Segments mit einer statisch festlegbaren Dauer. Eine Reduktion dieser Dauer ermöglicht eine Aussage über die maximal abbildbare Abtastrate für zyklische Signale in Botschaften des statischen Segments.

Resynchronisationsphasen/Zeiteinheit: Die Netzwerkruhephase dient der Korrektur der Offsets der jeweiligen Uhren der im Cluster integrierten FlexRay-Knoten. Die zeitliche Länge der Netzwerkruhephase korreliert dabei auch mit der maximal möglichen Korrektur des Offsets einer lokalen Zeit des FlexRay-Steuergeräts. Konsequenterweise bildet die statisch festgelegte Netzwerkruhephase durch das limitierte Intervall zur Offset-Korrektur auch eine Grenze für die maximal tolerierbare Alterung der Steuergeräteoszillatoren. Dadurch wird gleichermaßen die maximale Lebensdauer eines FlexRay-Systems mit einer dedizierten Parameterkonfiguration dediziert.

Aufstartdauer: Das Aufstartverhalten des FlexRay-Clusters und dessen Dauer hängt von mehreren Parametern ab. Ziel ist diese Ablaufsequenz mit der bestmöglichen Effizienz und Zuverlässigkeit für eine E/E-Architektur zu konfigurieren.

Neben den reinen bussystemspezifischen Aspekten interessieren auch die nachfolgend beschriebenen netzwerkübergreifenden Optimierungsbereiche:

Schedulability: Je nach Anforderung an das Clustern von Botschaften in einem FlexRay-Zyklus innerhalb des statischen Segments kann es zu Einschränkungen in der Generierung eines passenden Schedules für das Senden und Empfangen der Botschaften auf Betriebssystemebene eines Steuergeräts kommen. Ein geeignetes Schedulelayout muss diese Einschränkungen a priori als Restriktionen in der Botschaftsverteilung berücksichtigen.

Innerhalb dieser Arbeit haben sich die Gütekriterien auf der Protokollebene als komplexe Optimierungsprobleme dargestellt, die aus mathematischer Sicht nicht mehr statisch sondern mit numerischen Verfahren zu lösen sind. Die Fragestellungen im Bereich der Protokollkonfiguration erweisen sich dabei verstärkt unabhängig von externen Einflüssen im Vergleich zu den vorhandenen Gegebenheiten bei der physikalischen Übertragungsschicht im Bordnetzdesign (EMV, Energiemanagement, Verbauorte, Kabelbaumdesign, Steuergerätehardware, etc.). Speziell die Qualitätsgröße *Nutzdateneffizienz* in Kombination mit der Analyse des *Nutzdatenvolumens* im statischen Segment sowie die Dimensionierung des dynamischen Segments zur Garantie gegebener maximaler Übertragungslatenzen von Botschaften sind im Rahmen von /[Mai08]/ intensiv untersucht worden. Im Folgenden werden die entwickelten Ergebnisse zusammengefasst.

Im Rahmen einer optimierten FlexRay-Protokollauslegung wird die Konfiguration des Schedules hinsichtlich seiner Effizienz im Bereich der pro Zeiteinheit verfügbaren Nettonutzdatenrate untersucht. In /[Mai08]/ werden unterschiedliche Optimierungsalgorithmen appliziert, um einerseits zweckmäßige Botschaftsgrößen im statischen Segment des Schedules abzuleiten und zusätzlich eine ideale Größe des optionalen dynamischen Segments zu berechnen.

4.6.4.1. Formalisierung des Optimierungsbereichs „Statisches Segment“

Im Folgenden werden die erschlossenen Optimierungsbereiche formalisiert, um darauf aufbauend numerische Lösungsverfahren anzuwenden. Eine vollständige Formalisierung aller Optimierungskriterien impliziert umfangreiche Fragestellungen, welche für eine Lösungsfindung detailliert analysiert werden müssen. Konsequenterweise wird in dieser Arbeit lediglich eine Untermenge der identifizierten Optimierungsbereiche ausführlich untersucht. Der Fokus soll in dem Kontext auf die Themen *Nutzdateneffizienz* und *Dynamisches-Segment-Layout* beschränkt bleiben.

OP: Datendurchsatz (Nutzdatenvolumen $N_v(x)$ und Nutzdateneffizienz $N_e(x)$)

Der Begriff *Nutzdateneffizienz* erlangt seine Relevanz durch die a priori festzulegende statische Aufteilung der Netzwerkbandbreite im statischen FlexRay-Segment. Wie in Abb. 4.40 dargestellt, gilt es bei der Parameterfestlegung für eine E/E-Architektur einen hohen Anteil an reinen Nutzdaten im statischen Segment eines Zyklus zu generieren (*Nettodatenrate*).

Die Dauer des statischen Segments und eines Slots im statischen Segment wird durch einige in Abs. C.4 aufgeführte Restriktionen bestimmt. Die Nutzdateneffizienz basiert dabei auf dem im statischen Segment vorrätigen Nutzdatenvolumen innerhalb eines

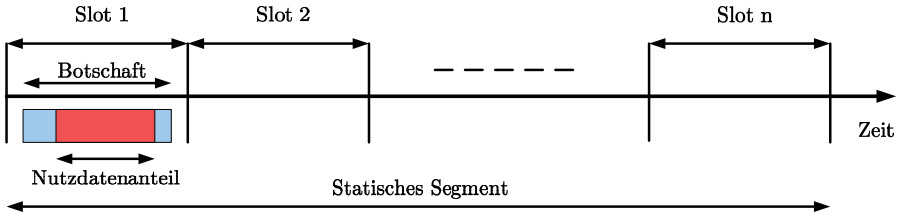


Abbildung 4.40.: Zusammenhang zwischen Nutzdaten-, Botschafts-, Slot-, und statische Segmentlänge eines FlexRay-Zyklus

FlexRay-Buszyklus. Dieses Nutzdatenvolumen lässt sich folgendermaßen auffassen[12]:

$$Nutzdaten_{StatischesSegment} = \frac{Länge_{StatischesSegment}}{Länge_{Slots}} * Nutzdaten_{Slot}$$

Entsprechend müssen die Berechungsformeln (*Restriktionen*) aus /[Fle05a]/ zur Berechnung der Länge eines statischen Slots und des statischen Segments herangezogen werden. Durch Umformungen ergeben sich folgende Definitionen[13]:

Definition 4.6.1 (Länge des statischen Segments) *Sei* $L_{Segment} : X \rightarrow Y$ *eine Abbildung. Dabei gelte* $X \subset \mathbb{R}^7$ *und* $Y = \mathbb{R}$ *. Dann ist*

$$\begin{aligned} L_{Segment}(x) &:= x_9 - \alpha_2 * (x_{18} + ceil(\frac{(\alpha_1 + \alpha_5)}{(x_7 * (1 - c_6))})) - x_{10} - (x_4 - x_{18}) \\ &- 2 * x_4 - ceil(\frac{(x_1 + c_{14} + c_5) * (c_4 * x_3 * (1 + c_6)) + \alpha_4 + \alpha_1 + \alpha_3}{x_7 * (1 - c_6)}) \end{aligned}$$

die Länge des statischen Segments.

□

Definition 4.6.2 (Länge eines Slots im statischen Segment) *Sei* $L_{Slot} : X \rightarrow Y$ *eine Abbildung. Dabei gelte* $X \subset \mathbb{R}^5$ *und* $Y = \mathbb{R}$*. Dann ist*

$$L_{Slot}(x) = 2 * x_4 + ceil(\frac{(x_1 + c_1 + c_2 + 20 * x_2 + c_3 + c_5) * (c_4 * x_3 * (1 + c_6)) + \alpha_1 + \alpha_3}{x_7 * (1 - c_6)})$$

die Länge eines Slots im statischen Segment.

□

Der Nutzdatengehalt einer Botschaft im statischen Segment wird durch die Variable x_2 in der Einheit *[Two-Byte-Words]* beeinflusst. Zur Bestimmung des Nutzdatengehalts im Bezug auf die Zeiteinheit $[\mu s]$ muss x_2 mit $16 * c_4 * x_3$ multipliziert werden. Einsetzen der Formeln $L_{Segment}(x)$, $L_{Slot}(x)$ und die Adaption von x_2 liefert das Nutzdatenvolumen $N_v(x)$ in Abhängigkeit zur Einheit $[\mu s]$ im statischen Segment:

[12] Eine graphische Analyse dieses Zusammenhangs ist im Anhang A aufgeführt.

[13] Für die relevanten FlexRay-Parameter werden im Anhang C Abkürzungen eingeführt. Die Umformungen basieren auf den Restriktionen aus /[Fle05a]/ und werden in /[Mai08]/ detailliert beschrieben.

$$N_v(x) = \frac{L_{Segment}(x)}{L_{Slot}(x)} * 16 * c_4 * x_3 * x_2$$

Die Nutzdateneffizienz $N_e(x)$ lässt sich als Nutzdatenvolumen pro zeitliche Länge interpretieren:

$$N_e(x) = \frac{N_v(x)}{x_7 * L_{Segment}(x)}$$

führt mit der Substitution des aufgestellten Nutzdatenvolumens $N_v(x)$ zu

$$N_e(x) = \frac{1}{x_7 * L_{Slot}(x)} * 16 * c_4 * x_3 * x_2$$

In Abb. 4.42 kann die Abhängigkeit zwischen der statischen Nutzdatenlänge und der Wahl des Makroticks als nominalen Zeitschritt eines FlexRay-Clusters nachvollzogen werden. Der Vorteil eines niederwertigen Makroticks (bei $1\mu s$ mit maximaler Nutzdatenlänge von 128 Byte-Wörtern pro Botschaft) wird hierbei ersichtlich. Allerdings gilt es entsprechende Nebenbedingungen zu determinieren, welche negative Aspekte, wie die relativ schlechte Skalierbarkeit von 256 Byte-Botschaften und die daraus resultierende minimale Slotanzahl im statischen Segment, miteinbeziehen. Bei der Analyse der Funktionale $N_v(x)$ und $N_e(x)$ muss beachtet werden, dass x in diesem Zusammenhang eine vektorielle Eingangsgröße darstellt ($x \in D \subset \mathbb{R}^8$ für $N_v(x), x \in D \subset \mathbb{R}^5$ für $N_e(x)$). Durch die Komponenten des Vektors $\vec{x} \in D \subset \mathbb{R}^8$ wird hauptsächlich das Kommunikationsverhalten im FlexRay-Cluster bestimmt, welche gleichzeitig Einfluss auf die Performanz des Bussystems ausüben. Daher lassen sich die Eingangsgrößen in diesem Zusammenhang auch als *Performanzparameter* auffassen:

Definition 4.6.3 (Performanz-Parameter) *Die Funktionale $N_v(x)$ bzw. $N_e(x)$ hängen von den Parametern*

- $x_1 = gdTSSTransmitter$
- $x_2 = gPayloadLengthStatic$
- $x_3 = gdSampleClockPeriod$
- $x_4 = gdActionPointOffset$
- $x_7 = gdMacrotick$
- $x_9 = gMacroPerCycle$
- $x_{10} = gdNIT$
- $x_{18} = gdMinislotActionPointOffset$

ab. Sämtliche Performanz-Parameter können innerhalb ihrer Definitionsbereiche nach den in /[Fle05a]/ aufgeführten Restriktionen gesetzt werden.

□

In Abbildung 4.41 werden die FlexRay Performanz-Parameter rot markiert dargestellt. Daraus lässt sich schließen, dass die Performanz eines FlexRay-Clusters von den Funktionsbereichen *Physical Layer*, *Coding/Decoding* und *Kommunikation* abhängt. N_v und N_v bilden frechet-differenzierbare Abbildungen in den Nicht-Sprungstellen, wodurch eine ordentliche Definition der Funktionale N_v und N_v erlaubt wird /[Mai08]/.

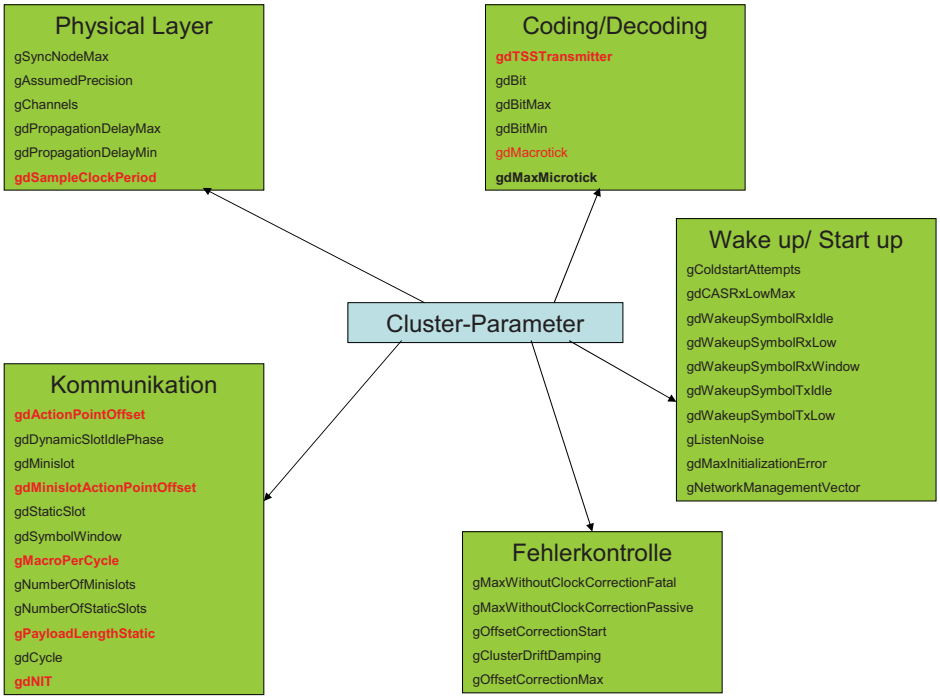


Abbildung 4.41.: Die Performanz-Parameter (rot) korrelieren mit drei unterschiedlichen Spezifikationsbereichen (logische und technische Systemaspekte)

Im nächsten Schritt soll versucht werden, die definierten Funktionale unter Beachtung relevanter Restriktionen der FlexRay-Spezifikation /[Fle05a]/ zu optimieren. In diesem Zusammenhang kommt dem Konvexitätsbegriff eine entscheidende Rolle zu. Für die Prüfung der Konvexitätseigenschaften gilt es die Hesse-Matrix zu untersuchen. Falls alle Eigenwerte der Matrix nicht negativ (*positiv-semidefinit*) sind, lässt sich die Annahme der Konvexitätseigenschaft der Funktionale verifizieren. Bei der Untersuchung der Hesse-Matrizen für N_v und N_v zeigt sich, dass in beiden Fällen negative Eigenwerte für jeweils mindestens einen Eingangsvektor aus den gültigen Definitionsbereichen gefunden werden kann (*Indefinitheit*) /[Mai08]/. Aus der Indefinitheit der Hesse-Matrizen lässt sich die Aussage folgern, dass die zugeordneten Funktionale keine konkaven Eigenschaften aufweisen. Reziprok lässt sich schliessen, dass die Negation dieses Funktionals somit keine konvexen Eigenschaften besitzt. Als Konsequenz lässt sich folgender Satz formulieren:

Satz 4.6.4 *Das Nutzdatenvolumen-Funktional $N_v(x)$ und das Nutzdateneffizienz-Funktional $N_e(x)$ aus den Definitionen A.3.1 und A.3.2 sind nicht konvex auf dem jeweils gültigen Definitionsbereich $\mathbb{D}_v \subset \mathbb{R}^8$ und $\mathbb{D}_e \subset \mathbb{R}^5$.*

□

Diese Erkenntnis hat weitreichende Konsequenzen im Kontext der Optimierung. Da die beiden Funktionale weder konvex noch konkav sind, kann nicht sichergestellt werden, dass für das Optimierungsproblem ein Minimum existiert. Die Tragweite der Ergebnisse der nachfolgend beschriebenen Optimierungen sind daher relativ einzuordnen. Das heißt, dass es sich bei jedem gefundenen Minimum des Optimierungsproblems[14] nicht strikt um ein globales Minimum handeln muss.

Beispielhaft zeigt die Abb. 4.42 das Nutzdateneffizienz-Funktional $N_e(x_2, x_7)$. Dabei werden die Eingabeparameter auf $x_1 = 14.625$, $x_3 = 0.0128$ und $x_4 = 4.0$ gesetzt.

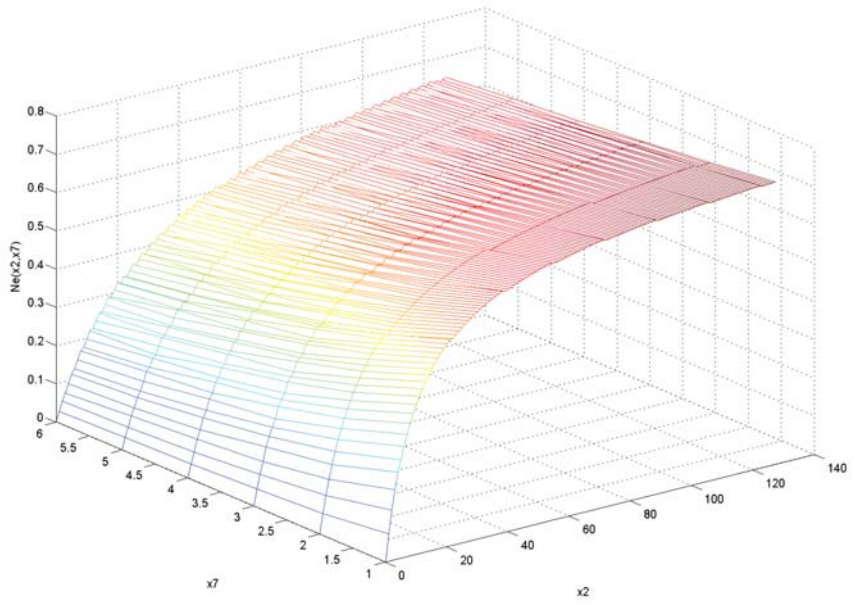


Abbildung 4.42.: Nutzdateneffizienz-Funktional in Abhängigkeit von den Eingangsparametern x_2 und x_7

Zur Lösung des Optimierungsproblems auf Basis des Funktionals N_e (OPN_e) müssen die Restriktionen der FlexRay-Spezifikation analysiert werden. Dabei lässt sich zwischen Parametern unterscheiden, die direkt das Funktional beeinflussen (α-Parameter) und Parametern bei denen keine direkte Abhängigkeit besteht (β-Parameter). Diese Parameter werden herangezogen, um mit zusätzlichen Funktionalen den Lösungsraum des OP einzuschränken. Neben dem Einsetzen und äquivalenten Umformen der FlexRay-Restriktionen werden sämtliche Nebenbedingungen in Form von Ungleichungen zusammengeführt[15].

Im Kontext von OPN_e werden sämtliche Restriktionen relevant, die in Abhängigkeit zu den Variablen aus N_e stehen. Als Einschränkung muss der existierende Definitionsbereich $\mathbb{D}_e$ beachtet und die Erfüllbarkeit der restlichen FlexRay-Restriktionen für einen Lösungsvektor verifiziert werden. Die Analyse der Nebenbedingungen führt zu einer Restriktionsabbildung $G_e(x)$, die insgesamt 15 FlexRay-Restriktionen, 10 Restriktionen

[14]Die Existenz eines Minimums für das OP kann unter den gegebenen Konvexitätseigenschaften nicht verifiziert werden.)

[15]Für alle relevanten FlexRay-Restriktionen ist die Frechet-Differenzierbarkeit in den Nicht-Sprungstellen gegeben /[Mai08]/.

zur Einhaltung von D_e und eine Zusatzrestriktion zur Beschränkung der Nutzdatenlänge der statischen Botschaften beinhaltet.

Mit der Definition von $N_e(x)$ und der Selektion relevanter Restriktionen $G_e(x)$ lässt sich das Optimierungsproblem vollständig formulieren. Dabei wird $N_e(x)$ unter der Prämisse maximiert, einen Lösungsvektor $x_0 \in \mathbb{D}_e$ mit gültigen Parameterwerten für eine FlexRay-Konfiguration zu erhalten. Ein dazu äquivalentes Optimierungsproblem ist die Minimierung von $-N_e(x)$ unter der Einhaltung aller in $G_e(x)$ definierten Restriktionen.

Definition 4.6.5 (Optimierungsproblem OPN_e) *Sei $N_e(x)$ wie in Definition A.1.1 und $G_e(x)$ wie in Definition A.2.1 beschrieben. Dann wird das Optimierungsproblem*

$$\begin{cases} min \ -N_e(x) \\ x \in \mathbb{D}_e, G_e(x) \leq 0 \end{cases}$$

OPN_e genannt. Ein Lösungsvektor für dieses Optimierungsproblem stellt somit gültige Parameterwerte für eine FlexRay-Konfiguration dar.

□

Zusammenfassend lässt sich OPN_e als *nichtlineares* und *nichtkonvexes Optimierungsproblem* auffassen. Diese Aussage begründet sich in dem offensichtlich nichtlinearen und nicht konvexen Nutzdatenfunktional N_e mit der ebenfalls nicht linearen Restriktionsabbildung *Ge(x)*, welche auf nichtlinearen Komponentenfunktionalen basiert. Die Implementierung von Optimierungsverfahren für OPN_e wird in Kapitel 5 beschrieben.

Eine Alternative zur Lösung von OPN_e basiert auf dem allgemeinen Satz von Kuhn-Tucker /[Lue97]/. Da bei der werkzeuggestützten Berechnung im Gegensatz zu *fmincon* keine Lösung gefunden werden konnte (s. Abs. 5), soll die Idee und das theoretische Vorgehen aus Vollstänigkeitsgründen nur knapp skizziert werden.

Das Funktional N_e bildet ein notwendiges Gateaux-differenzierbares, reellwertiges Funktional auf den Vektorraum X und G_e die Gateaux-differenzierbare Abbildung der Restriktionsgleichungen von X auf den normierten Raum Z mit dem positiven Kegel P[16]. Als Ziel wird versucht ein x_0 zu finden, welches N_e bzgl. $G_e(x) \leq \vec{0}$ minimiert, wobei x_0 einen regulären Punkt der Ungleichung $G_e(x) \leq \vec{0}$ darstellen muss. Dann gibt es ein $z_0^* \in Z^*$, $z_0^* \geq \vec{0}$ (Z^* ist der normierte Dualraum auf Z), so dass die *Lagrange-Gleichung*

$$-N_e(x) + \langle G_e(x), z_0^* \rangle ,$$
$$-N_e(x) + \sum_{j=1}^{m} z_0^* * g_j(x)$$

stationär an der Stelle x_0 ist. Weiterhin gilt $\langle G_e(x_0), z_0^* \rangle = 0$.

Zur Optimierung wird die Lagrange-Gleichung differenziert und dem Wert Null gleichgesetzt. Gleichzeitig wird G_e in Form des euklidischen Skalarprodukts ebenfalls dem Wert Null gleichgesetzt. Dadurch soll die stationäre Stelle für die Lagrange-Gleichung gefunden und die Gleichung $\langle G_e(x_0), z_0^* \rangle = 0$ gewährleistet bleiben.

[16] Für die Definitionen der Begriffe „*Normierter Vektorraum*“, „*Positiver Kegel*“ und „*regulärer Punkt*“ wird in diesem Zusammenhang auf /[Lue97]/ weiterverwiesen

4.6.4.2. Formalisierung des Optimierungsbereichs „Dynamisches Segment“

Das optional implementierbare dynamische FlexRay-Segment funktioniert im Gegensatz zum statischen Segment mit einem künstlich generierten ereignisgesteuerten Kommunikationsansatz. Die Bezeichnung „*künstlich*“ bezieht sich auf das feingranulare zeitscheibenbasierte Minislotprinzip, welches niedrige zeitliche Pufferabstände zwischen den aktiven Datenübertragungsvorgängen ermöglicht. Die Erzeugung der dynamischen Botschaften weist einen probabilistischen Charakter auf, da trotz der statischen Zuordnung von Sendeminislots zu Sendern die Netzwerkkommunikation nicht zwangsläufig zyklisch von den Sendern ausgeführt werden muss. Gleichzeitig führt die flexible Botschaftslänge zu Systeminterferenzen, da der relative Sendezeitpunkt im Buszyklus von der Sendeaktivität und der Nutzdatenlänge einer Botschaft abhängt. Im Extremfall kommt es zu zeitlichen Verschiebungen über mehrere Buszyklen.

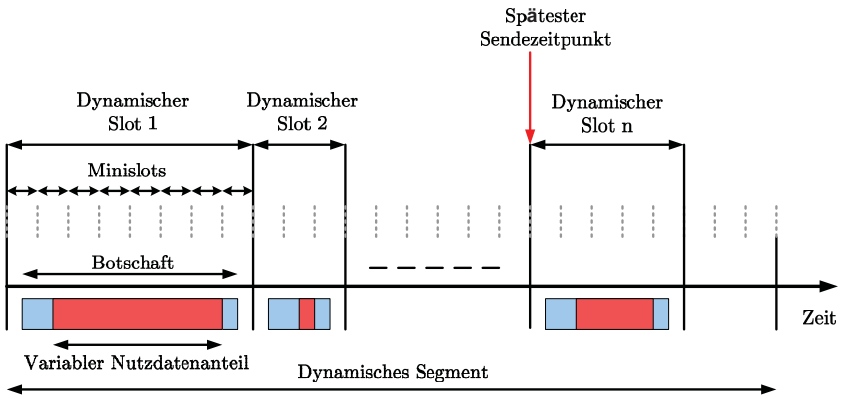


Abbildung 4.43.: Zusammenhang zwischen Minislot-, Nutzdaten-, Dynamische Slot- und dynamische Segmentlänge eines FlexRay-Zyklus

In Abb. 4.43 wird das Zusammenspiel der beschriebenen Systemaspekte im dynamischen FlexRay-Segment veranschaulicht. Die Darstellung zeigt $N \in \mathbb{N}$ Steuergeräte, die mit den Wahrscheinlichkeiten $p_i < 1$, $i \in 1, ..., M$ Botschaften im dynamischen Segment verschicken. Die M Botschaften werden injektiv auf verschiedene Minislots abgebildet. Dazu wird das dynamische Segment in insgesamt $M + r$ Minislots aufgeteilt. $r \in \mathbb{N}$ sei die Anzahl an Minislots, die zusätzlich nötig ist, um die zeitlichen Anforderungen der FlexRay-Spezifikation zu erfüllen und das einkalkulierte Datenvolumen übertragen zu können. Die Größe $M + r$ soll in diesem Abschnitt bestimmt werden. Folgende zwei Definitionen bilden die Basis für eine strukturierte Problemlösung.

Definition 4.6.1 (Ereignisgesteuerte Botschaft) *Eine ereignisgesteuerte Botschaft ist eine Botschaft, die mit einer Wahrscheinlichkeit $p < 1$ auf dem FlexRay-Cluster durch ein Steuergerät versendet wird.*

□

Definition 4.6.2 (Zufallsvariable X) *Sei $X \in \mathbb{R}^+$ eine stetige Zufallsvariable. X beziffert das zufällige Datenvolumen aller im dynamischen Segment verschickten Botschaften pro Zyklus.*

□

Im dynamischen Segment werden in der Regel genau die ereignisgesteuerten Botschaften aufgenommen, bei denen sich eine potentielle zeitliche Verzögerung beim Versenden nur eingeschränkt auf das funktionale Verhalten eines an der Kommunikation partizipierenden Steuergeräts auswirkt.

Berechnung der dynamischen Segmentgröße

Die Selektion der Botschaften für das dynamische Segment des FlexRay-Kommunikationssystems wird in dieser Arbeit auf Basis empirisch ermittelter Werte eines Serienfahrzeugs mit CAN-Vernetzung getroffen. Kennzahlen in diesem Zusammenhang beziehen sich auf die Erfassung der absoluten *Auftrittshäufigkeit* und des *Datenvolumens* der ausgewählten Botschaften innerhalb eigens definierter Zeitintervalle[17]. Grundlegend wird bei dem Ansatz der folgende Begriff der Dichte- und Verteilungsfunktion /[RWV97]/ herangezogen:

Definition 4.6.3 *(Dichte- und Verteilungsfunktion) Die Zufallsvariable X heißt stetig verteilt mit der Dichtefunktion f_X, falls $f_X \geq 0$ im Definitionsbereich $\mathbb{R}$ ist und sich die Verteilungsfunktion F_X von X folgendermaßen schreiben lässt:*

$$F_X(x) = \int_{-\infty}^{x} f_X(t)dt$$

□

In Abbildung 4.44 wird die im Fahrzeug ermittelte Punktmenge durch kubische Splines interpoliert. Dazu lässt sich in Analogie zu /[HS02]/ definieren:

Definition 4.6.4 *(Spline-Funktion) Sei $t \in \mathbb{R}$ und $s_m(t)$ ein Polynom 3. Grades mit $m \in 1, ..., M-1$ der Form*

$$s_m(t) = a_m + b_m * (t - t_m) + c_m * (t - t_m)^2 + d_m * (t - t_m)^3$$

Dann heißt

$$s(t) = \sum_{m=1}^{M-1} \chi_m(t) * s_m(t)$$

Spline-Funktion mit der Auswahlfunktion

$$\chi_m(t) = \begin{cases} 1 & t_m \leq t \leq t_{m+1} \\ 0 & sonst \end{cases}$$

□

Die ermittelten Splinekoeffizienten und Werte für die Auswahlfunktion werden in Abs. B dargestellt. Die stetige Zufallsvariable X definiert das zufällige Datenvolumen für alle

[17] Die ermittelten Daten basieren auf dem Fahrprofil eines Fahrers in einem Serien-SUV auf einer mittleren Distanz von ca. 600km. Für zuverlässige Ergebnisse wäre eine Erweiterung auf unterschiedliche Fahrer, Fahrzeuge und Streckenprofile sinnvoll.

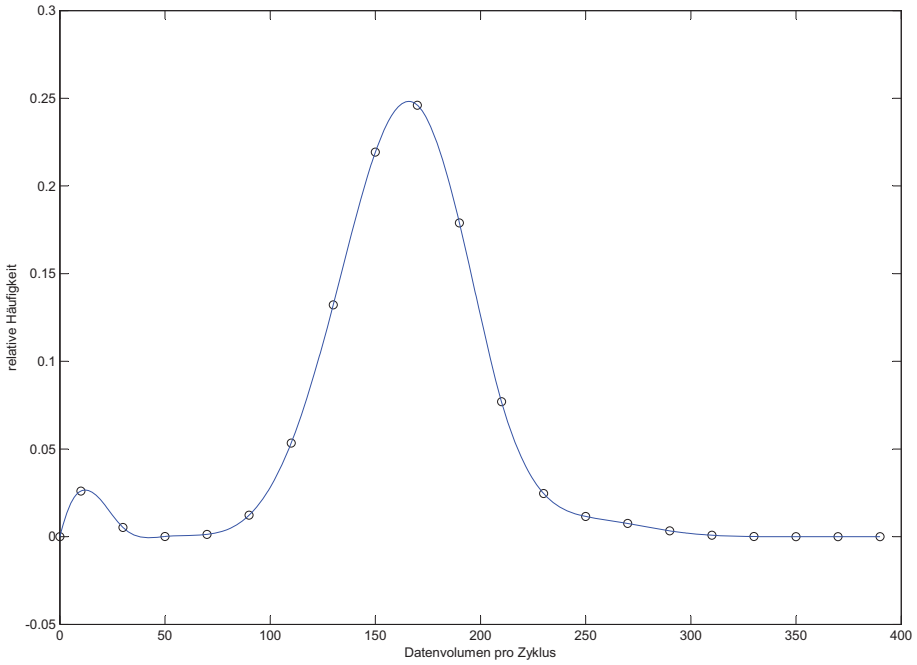


Abbildung 4.44.: Durch kubische Splines interpolierte Dichtefunktion $n_X(t)$ auf Basis der gemessenen Punktmengen

im dynamischen Segment verschickten Botschaften pro festgelegten Zyklus[18]. Dabei gilt die Dichtefunktion $n_X(t)$ zur Einschränkung des Definitionsbereichs:

$$n_X(t) = \begin{cases} \sum_{m=1}^{M-1} \chi_m(t) * s_m(t) & \text{für } 0 \leq t \leq 390 \\ 0 & \text{sonst} \end{cases}$$

Konsequenterweise lässt sich daraus die Verteilungsfunktion für die Zufallsvariable X notieren als:

$$N_X(x) = \begin{cases} 0 & \text{für } x \leq 0 \\ \int_0^x \sum_{m=1}^{M-1} \chi_m(t) * s_m(t) dt & \text{für } 0 \leq x \leq 390 \\ 1 & \text{für } x \geq 390 \end{cases}$$

Zur Berechnung des aufkommenden Datenvolumens muss der Erwartungswert der ermittelten Funktion $n_X(x)$ berechnet werden.

Definition 4.6.5 *(Erwartungswert) Sei X eine stetige Zufallsvariable mit der Dichte f_X. Dann heißt*

$$E(X) := \int_{-\infty}^{+\infty} x * f_X(x) dx$$

[18] Der Zyklus beträgt in diesem Fall 8ms.

Erwartungswert von X, falls gilt:

$$\int_{-\infty}^{+\infty} x * f_X(x)dx < \infty$$

□

Durch Einsetzten von $n_X(x)$ für $f_X(x)$ ergibt sich das erwartete Nutzdatenvolumen mit Einheit $[Byte]$ für das dynamische Segment.

$$E(X) = \int_{-\infty}^{+\infty} x * n_X(x)dx = \int_{0}^{390} x * n_X(x)dx = \int_{0}^{390} x * \sum_{m=1}^{M-1} \chi_m(x) * s_m(x)dx = 1.62e + 002$$

Das Ergebnis $1.62e + 002$ in $[Byte]$ entspricht einem Nutzdatenvolumen von 1296 $[Bit]$. Zur Berechnung der dynamischen Botschaftslänge müssen folgende zusätzliche Parameter hinzugenommen werden:

Definition 4.6.6 (Anzahl der ECUs) *Die Anzahl der ECUs, die im dynamischen Segment Botschaften verschicken dürfen, wird durch den Parameter $\eta \in \mathbb{N}_0$ festgelegt.*

□

Definition 4.6.7 (Anzahl Botschaften pro ECU) *Sei $i \in \mathbb{N}$. Dann ist $\xi_i \in \mathbb{N}_0$ die Anzahl der Botschaften, die die ECU i im dynamischen Segment verschicken darf.*

□

Definition 4.6.8 (Anzahl verschickter Botschaften im dynamischen Segment) *Der Parameter $\psi \in \mathbb{N}_0$ gibt die Anzahl aller verschickten Botschaften im dynamischen Segment pro Zyklus vor.*

□

Somit lässt sich die Anzahl der Botschaften, die im dynamischen Segment pro Zyklus verschickt werden, durch die Formel

$$\psi = \sum_{i}^{\eta} \xi_i$$

darstellen. Es wird angenommen, dass sich die versendete Nutzdatengröße pro Botschaft gleichmäßig verteilt. Deshalb errechnet sich der Parameter β_{11} (=dynamische Nutzdatenlänge einer Botschaft) zu

$$\beta_{11} = ceil(\frac{E(X) * c_4 * x_3}{\psi}) = ceil(\frac{1296 * c_4 * x_3}{\sum_{i}^{\eta} \xi_i})$$

mit Einheit $[gdBit]$. Jetzt lässt sich die Anzahl der benötigten Minislots im dynamischen FlexRay-Segment pro Botschaft berechnen.

Definition 4.6.9 (Anzahl Minislots pro Botschaft) *Die zeitliche Länge, die eine im dynamischen Segment verschickte Botschaft benötigt, wird mit ρ bezeichnet.*

□

Mit den obigen Größen wird neben der Länge einer dynamischen Botschaft auch die Länge des gesamten dynamischen Segments ermittelt. Die zeitliche Ausprägung einer Botschaft im dynamischen Segment wird mit einer Restriktion der FlexRay-Spezifikation /[Fle05a]/ bestimmt:

$$\rho = 1 + ceil(\frac{((\beta_{11}+1) * c_4 * x_3 * (1+c_6))}{(x_7 * (1-c_6) * x_{12})}) + \beta_{13}$$

Die Einheit ist $[Minislot]$.

4.6.4.3. Actionpointskalierung im statischen Segment

Eine Herausforderung in der Systemparametrierung basiert auf der zweckmäßigen Skalierung des Absatzpunkts (*Actionpoint Offset*)x_4 einer Botschaft im statischen Slot des FlexRay-Zyklus. Wesentliche Einflussgröße stellt die Gesamtsystempräzision dar, die im globalen Actionpoint jedes Slots der FlexRay-Architektur abgedeckt bleiben muss (s. Abb. 4.45).

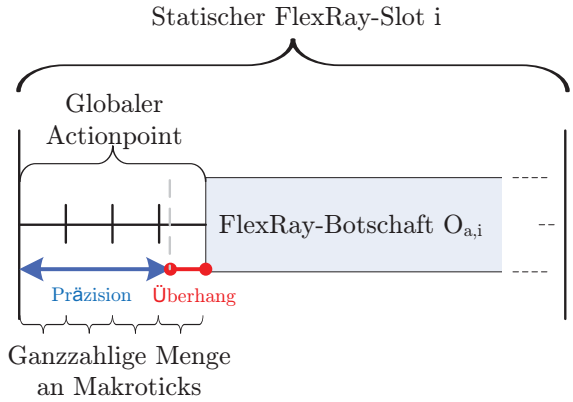


Abbildung 4.45.: Actionpoint-Skalierung zur Abdeckung der im Cluster vorhandenen Präzision

$$x_4 \geq \left\lceil \frac{\Delta T^{Assumed}_{N_{i,j}} - d^{max}_{EPL}}{MT * (1 - \Delta T^{max}_{Quarz})} \right\rceil \text{ wenn } \forall N_{i,j} : N_i \wedge N_j \in Cluster \text{ mit } N_i \neq N_j \qquad (4.1)$$

Aus 4.1 lässt sich ableiten, dass die Wahl von x_4 möglichst nah unterhalb einer ganzzahligen Makrotickgrenze liegen sollte. Daraus lässt sich folgern:

$$k * MT \geq \left\lceil \frac{\Delta T^{Assumed}_{N_{i,j}} - d^{max}_{EPL}}{MT * (1 - \Delta T^{max}_{Quarz})} \right\rceil$$

Aus Skalierungsgründen eignet sich eine MT-Größe in der Nähe des minimal zulässigen MT ($MT^{Min} = 1$).

$$k * MT \geq \left\lceil \frac{\Delta T_{N_{i,j}}^{Assumed} - d_{EPL}^{max}}{MT * (1 - \Delta T_{Quarz}^{max})} \right\rceil$$

$$k * MT \geq \frac{\Delta T_{N_{i,j}}^{Assumed} - d_{EPL}^{max}}{MT * (1 - \Delta T_{Quarz}^{max})}$$

$$MT \geq \sqrt{\frac{\Delta T_{N_{i,j}}^{Assumed} - d_{EPL}^{max}}{k * (1 - \Delta T_{Quarz}^{max})}}$$

$$MT^{Min} + x \geq \sqrt{\frac{\Delta T_{N_{i,j}}^{Assumed} - d_{EPL}^{max}}{k * (1 - \Delta T_{Quarz}^{max})}}$$

$$x \geq \sqrt{\frac{\Delta T_{N_{i,j}}^{Assumed} - d_{EPL}^{max}}{k * (1 - \Delta T_{Quarz}^{max})}} - MT^{Min}$$

In Abbildung 4.46 wird eine skalierte Darstellung des MT anhand seines Überhangs x zum minimalen MT in Relation zu den umsetzbaren Actionpointgrößen und der im Cluster vorhandenen Systempräzision visualisiert.

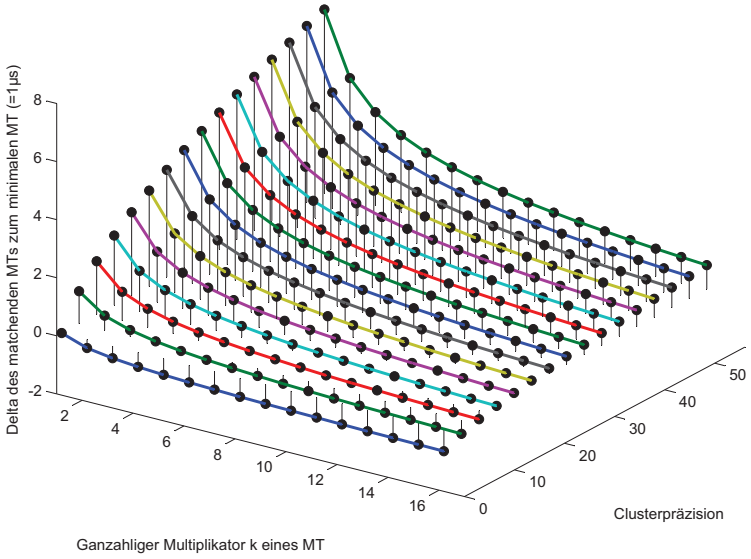


Abbildung 4.46.: Zusammenhang zwischen der MT-Skalierung und der Actionpointdefinition zur Abdeckung vorgegebener Präzisionswerte

4.6.4.4. Topologieabmessungen und Systemrobustheit

Topologievarianten beeinflussen die in FlexRay-Architekturen erreichbare Netzwerkpräzision. Dieser Sachverhalt wird anfolgend kurz erläutert. Die Präzision eines FlexRay-Clusters basiert partiell auf dem quantitativen Beitrag der Latenzen bei der Netzwerkkommunikation, die sich aus der Summe der an der Kommunikation beteiligten physi-

kalischen Komponenten ergeben. Durch Verzögerungen bei der Kommunikation zweier Netzwerkkomponenten n und m, welche durch die beteiligten Bustreiber, dem Netzwerkkommunikationsmedium und den potentiell verbauten aktiven Sternkopplern erzeugt werden, existiert ein wesentlicher Einflussfaktor hinsichtlich der in der Uhrensynchronisation zu tolerierenden Präzision des FlexRay-Clusters.

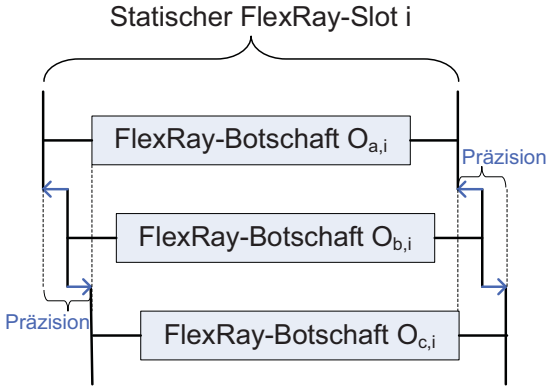


Abbildung 4.47.: Bedeutung der Netzwerkpräzision bezogen auf die Slotüberhänge eines FlexRay-Clusters

Wie in Abb. 4.47 dargestellt, steht die Präzision in Wechselwirkung mit der Protokollauslegung und dem zu entwickelnden Netzwerkschedule. Ein *Jitter*-Effekt auf Slotebene hinsichtlich einer positiven und negativen Abweichung vom normalen Absatzpunkt muss berücksichtigt bleiben, um Botschaftskollisionen bei der Verletzung von Slotgrenzen (*Boundary Violations*) in der Kommunikation zu verhindern. Die Netzwerkpräzision muss dabei in Abhängigkeit zur Granularität der existierenden minimalen diskreten Zeitschritte auf Controllerebene, den Mikroticks, betrachtet werden. Diese können in ihrer zeitlichen Ausprägung zwischen den Controllern variieren. Sogar bei einheitlichen Controllern gibt es Abweichungen, da die Schrittgröße als nominaler Wert zu betrachten ist und nicht mit dem jeweils aktuellen Wert am Controller identisch sein muss.

Jeder Kommunikationscontroller verfügt über eine Uhrenquelle mit einer potentiellen, nach oben beschränkten Abweichung von der Normfrequenz. Von dieser Uhrenquelle leitet sich für jeden Kommunikationscontroller der Mikrotick ab. Die nominale Länge eines derartigen Mikroticks muss nicht zwangsläufig in jedem Kommunikationscontroller identisch sein in einem Netzwerk. Drei definierte Längen sind prinzipiell möglich in einem Verhältnis $x_{41} = pdMicrotick = i * pdSmallestMicrotick$ mit $i \in \{2, 4\}$ /[Ung07]/.

Falls die Anzahl an Mikroticks pro Zyklus ($x_{45} = gMicroticksPerCycle$) nicht durch die Anzahl der Makroticks pro Zyklus $gMacroticksPerCycle$ teilbar ist, können zwei arbiträre Makroticks in ihrer zeitlichen Ausprägung in einem Zyklus im Kommunikationscontroller voneinanderabweichen, da Mikro- und Makrotickgrenzen stets zeitsynchron verlaufen.

Durch die gezielte Auswahl geeigneter aktiver Sternkoppler, Verkabelungsgrößen, Quarze zur Zeitgenerierung und EMV-Schutzmaßnahmen lassen sich die durch die Systempräzision verursachte Ungenauigkeiten bei der Signalausbreitung ($x_6 = gdMaxPropagationDelay$) beschränken. Zur Erhöhung der physikalischen Systemrobustheit und der Robustheit in der Systemkommunikation tragen weiterhin eine abgesenkte Baudra-

te sowie eine konservative Bedatung des dynamischen FlexRay-Segments bei.

4.6.4.5. Zielabtastraten und Minimierung von Slotüberhängen

Zum Erreichen der geforderten Regelgüte durch vorgegebene Abtastraten müssen die Anforderungen aller zu übertragenden Signale und deren Attribute im FlexRay-Parametersatz berücksichtigt werden. Analog zu Kapitel 4.2 lassen sich dabei unterschiedliche Aspekte der Netzwerkkommunikation betrachten, um grundlegende Eckdaten der FlexRay-Konfiguration zu identifizieren:

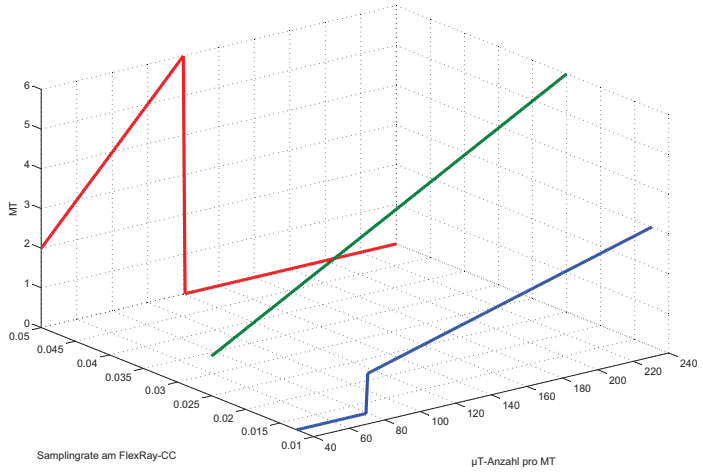


Abbildung 4.48.: Mikrotickauslegung anhand des vorgegebenen MTs in μs

Einflussfaktoren für die Definition der Buszykluslänge und -segmentierung:

- Zykluszeiten aus sämtlichen Signalen der Funktionsnetznotation,
- Anzahl der Signale/Signalgruppen mit einheitlicher Sendezykluszeit,
- Klassifikation der Kommunikationstypen (zyklisch/spontan/hybrid),
- Flexibilität in der Schedulebedatung durch Multiplexing (Erweiterbarkeitsvorteile),
- Buszyklusspezifische Paradigmen zur Abbildung bussynchroner/-asynchroner FlexRay-Applikationen (Antwortzeiten, Deadlines, Sperrzeiten).

Einflussfaktoren zur Reduktion von Slot-Überhängen (*Interframe-Gap-Reduction*):

- Systemgenauigkeit (Quarzqualität),
- Initialisierungsfehler bei der Knotenintegration in ein synchrones Cluster,
- Kompensation von Signalverkürzungen (*truncations*) während der Signalübertragung,
- Berücksichtigung von zuverlässig auftretenden Verzögerungen bei der Signalübertragung,

- Skalierung des Makroticks im FlexRay-Cluster.

Während die Systemgenauigkeit, der Initialisierungsfehler, die Signalverkürzungen und die Signalverzögerungen von der eingesetzten Hardware abhängen, bezieht sich die Makrotickskalierung auf eine Adaption des Busscheduledesigns für die vorliegende technische Systemarchitektur.

4.6.4.6. Makrotickskalierung im Cluster

Die Makrotickskalierung bezieht sich auf die Einpassung des MT auf Basis der individuellen Abtastfrequenz eines jeden FlexRay-Kommunikationscontrollers zur Ableitung der Anzahl an μTs pro MT. Als Restriktion für die Wahl des knotenspezifischen Mikroticks gilt die anliegende Abtastfrequenz in einem Bereich von 10-80MHz.

Wie in Abb. 4.48 dargestellt, lässt sich nicht der komplette Definitionsbereich für die MT-Länge (1-6μs) auf alle Abtastraten eines FlexRay-Clusters anwenden (s. Tab. 4.9). Durch die begrenzte Samplingrate pro Mikrotick (*1-4*) und das Intervall für die Anzahl an Mikroticks pro Makrotick (*40-240*) ergeben sich ungültige Konfigurationen, die in der Tabelle mit $\perp$ markiert sind.

Baudrate [Mbit/s]	Controllerfrequenz [MHz]	Makroticklänge [μs]					
2,5	10	$\perp$	$\perp$	$\perp$	40	50	60
2,5	20	$\perp$	40	60	80	100	120
2,5	40	40	80	120	160	200	240
2,5	80	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
5	10	$\perp$	$\perp$	$\perp$	40	50	60
5	20	$\perp$	40	60	80	100	120
5	40	40	80	120	160	200	240
5	80	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
10	10	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
10	20	$\perp$	40	60	80	100	120
10	40	40	80	120	160	200	240
10	80	80	160	240	$\perp$	$\perp$	$\perp$

Tabelle 4.9.: Mikro-pro-Makrotick-Verhältnisse für unterschiedliche Baudraten und Controllerfrequenzen

KAPITEL 5

Implementierung und Validierung

In diesem Kapitel wird das Konzept und die prototypische Umsetzung der entwickelten modellbasierten Entwicklungsmethodik für E/E-Architekturen auf Basis des flexiblen zeitgesteuerten Bussystems FlexRay vorgestellt. Dabei werden Erkenntnisse bei der Modellentwicklung zusammengefasst, die bei der Durchführung einer seriennahen Fallstudie gewonnen wurden. Das Vorgehen der FlexRay-Parameterentwicklung wird auf Basis der statischen und dynamischen Modellanalyse konkretisiert und dessen Vorteile und Grenzen aufgezeigt.

Während des Parametrierungsvorgangs eines FlexRay-Systems müssen sämtliche Systemeigenschaften der zu entwickelnden E/E-Architektur berücksichtigt bleiben, um eine zielgerechte, fehlerfreie Integration der FlexRay-Technologie ins Fahrzeug zu gewährleisten. Die in Kapitel 4 erarbeitete Entwicklungsmethodik wird dabei zur statischen Systemanalyse auf ein E/E-Architekturmodellierungswerkzeug übertragen. Für diesen Zweck wurde in einem Kooperationsprojekt ein Erweiterung eingeführt, um die komplexen Parameterberechungsregeln zweckmäßig in einer dedizierten Notationsform im Werkzeug zu integrieren. Die dynamische Systemanalyse erfolgt beispielhaft für die Untersuchung des Weck- und Startverhaltens in einem separaten freiverfügbaren Werkzeug zur Simulation gezeiteter Automaten. Zur Lösung der Optimierungsprobleme wird die Entwicklungsumgebung durch ein kommerzielles Werkzeug zur mathematischen Optimierung ergänzt. Die konzeptionelle Umsetzung und deren Anwendung werden zusammen mit den erhaltenen Ergebnissen für die durchgeführte Fallstudie im Folgenden erläutert.

5.1. FlexZOOMED-Implementierungskonzept

Die umgesetzte Entwicklungsmethodik des FlexZOOMED-Ansatzes beruht auf der Untergliederung der Prozessschritte in die Bereiche *abstrakte Systemspezifikation* (Ausstattungs- und Anforderungsanalyse), *konkrete Systemspezifkation* (statische und dynamische E/E-Architekturmodellierung), *Systemoptimierung*, *Systemparametrierung* und *Systemanalyse* (s. Abb. 5.1).

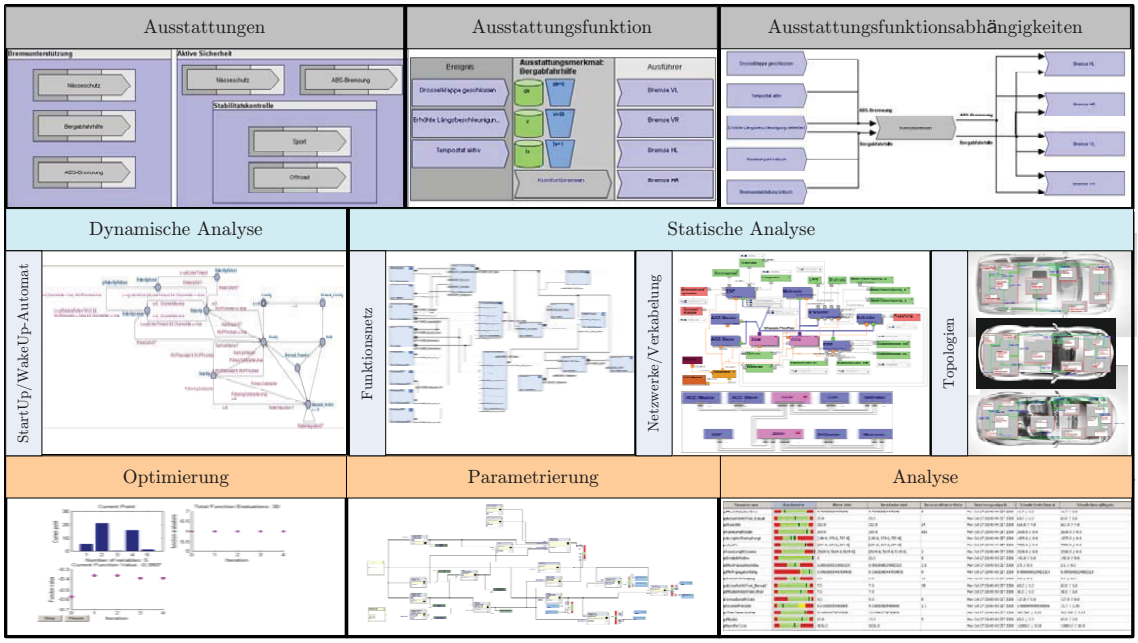


Abbildung 5.1.: Integrierte Darstellung des Entwicklungskonzepts FlexZOOMED

Die Umsetzung basiert für eine Validierung auf einer dreiteiligen Werkzeugumgebung. Die Grundlage bildet das E/E-Architekturmodellierungswerkzeug *PREEvision* /[Aqu07]/, welches für die Bereiche der abstrakten und konkreten statischen Systemspezifikation eingesetzt wird. Um den Bereich der Systemparametrierung für FlexRay--Systemarchitekturen umzusetzen, wurde das Basiswerkzeug um einen eigenständigen Parametrierungseditor innerhalb eines Forschungsprojekts erweitert. Die dynamische Systemspezifikation lässt sich nicht in PREEvision abbilden und wird daher in dem zusätzlichen Werkzeug *UPPAAL* /[BLP+96]/ zur Spezifikation, Simulation und Modellprüfung (*model checking*) von gezeiteten Automaten umgesetzt. Als weiterer Baustein wird der Bereich der Systemoptimierung mithilfe des Werkzeugs Matlab/Simulink /[The07a]/ realisiert. Eine Einbettung der umgesetzten Matlab-Optimierungsberechnungen in PREEvision generiert dabei funktional keinen entscheidenden Mehrwert und bleibt aus diesem Grund bei der technischen Implementierung unberücksichtigt. Die Gesamtsystemanalyse der Parametrierungsergebnisse und der Qualitätskennzahlen der FlexRay-Architekturen wird in einer weiteren Werkzeugergänzung innerhalb einer Metriktabelle visualisiert.

Während der Erstellung der Arbeit sind auch alternative Lösungen zur Umsetzung des FlexZOOMED-Ansatzes betrachtet worden. Beispielsweise wäre mit dem Werkzeug MetaEdit /[Met08]/ eine vollkommen freie domänenspezifische Umsetzung der FlexRay-Architekturmodellierung möglich. Allerdings würde eine Adaption auf die

E/E-Architekturentwicklung in diesem Zusammenhang einen sehr hohen Aufwand erfordern, der die Umsetzungstiefe innerhalb dieser Arbeit limitiert hätte. Zusätzlich zielt das Werkzeug MetaEdit auf eine verstärkt simulative Analyse und die Generierung von virtuellen Prototypen, was die Anforderungen der FlexRay-Systemparametrierung nicht ausreichend abdeckt. Einen ähnlicher Schluss lässt sich für das Werkzeug VisualSim /[Mir07]/ ziehen, welches gleichfalls für die dynamische Analyse des FlexRay-Verhaltens (dynamisches FlexRay-Segment, *Timing*-Analyse) interessant erscheint, aber ebenfalls mit einem hohen Aufwand-/Nutzenverhältnis verbunden ist. In diesem Fall würde sich eher die Integration eines eigenständigen *Timing*-Analysewerkzeugs, beispielsweise SymtaT/S /[HHJ$^+$]/, anbieten, was aber nicht den Schwerpunkt der vorliegenden Arbeit bildet und daher zurückgestellt wurde. Von einer proprietären Umsetzung des FlexZOOMED-Ansatzes auf einem herstellerinternen Werkzeug wird aufgrund eines fehlenden passenden Basissystems und hohen Kostenaufwendungen in dieser Arbeit abgesehen.

5.1.1. Designfluss

Die Idee des implementierten Ansatzes fundiert auf dem erläuterten fünfteiligen Designprozess. In Abb. 5.2 wird der Designfluss examplarisch in sequentieller Abfolge dargestellt. Initial werden die im Lastenheft festgelegten Ausstattungseigenschaften (*features*) des Fahrzeugs in ein Modell überführt. Dabei werden in diesem Beispiel spezielle Ausstattungsoptionen eines erweiterten Bremssystems (*Nässeschutz*, *Bergabfahrhilfe*, *ABS-Bremsung*) abgebildet. Jedes Ausstattungsmerkmal lässt sich nun in eine funktionale Beschreibungsform überführen. In diesem Fall wird beispielsweise die *Bergabfahrhilfe* auf eine mehrteilige Funktion *Komfortbremsen* abgebildet, die durch Ereignisse (*Drosselklappen geschlossen*, *erhöhte Geschwindigkeit*, *Tempostat aktiv*) ausgelöst und über zugeordnete Komponenten (*Bremse VL*, *Bremse VR*, *Bremse HL*, *Bremse HR*) ausgeführt wird. Im nächsten Schritt wird die abstrakte Ausstattungsfunktion *Komfortbremsen* einer konkreten Realisierung *ErweiterteBremsFkt* im Funktionsnetz zugeordnet. Dieses Funktionsnetzelement lässt sich anschließend auf eine technische Komponente, beispielsweise das *ESP-Steuergerät*, partitionieren. Nach der statischen Zuordnung der konkreten Funktion zu einer konkreten technischen Komponente erfolgt die Definition des Netzwerks sowie dessen Verkabelung, um darauf aufbauend die Topologien baureihenspezifisch (*Heckmotor-Sportwagen, Mittelmotor-Sportwagen, sportlicher Geländewagen (SUV)*) festzulegen. Der mit diesen Schritten festgelegte statische E/E-Architekturentwurf wird in Abb. 5.2 unter dem Kennzeichen *A* dargestellt.

Im nächsten übergeordneten Prozessschritt *B* erfolgt die dynamische und statische Bestimmung der festgelegten FlexRay-Parameter. Dabei werden die knoteninternen Parameter für den dynamischen Ablauf des Weck- und Startvorgangs des FlexRay-Clusters manuell in das Modell des gezeiteten Automaten im Werkzeug UPPAAL überführt. Die im dynamischen Modell ermittelten Ergebnisse werden im Verbund mit den restlichen Parametern aus der statischen Architekturbeschreibung in das FlexRay-Parametermodell eingebunden. Dieser Vorgang vollzieht sich automatisch während der Bedatung im ersten Prozessschritt *A*.

Die Optimierung erfolgt durch den dateibasierten Austausch der FlexRay-Parameter zwischen dem E/E-Architekturmodell in PREEvision und der Optimierungswerkzeugbox aus der Matlab/Simulink-Werkzeugkette (Prozessschritt *C*). Anschließend lassen sich die aus den drei Werkzeugen ermittelten FlexRay-Parametersätze in PREEvision

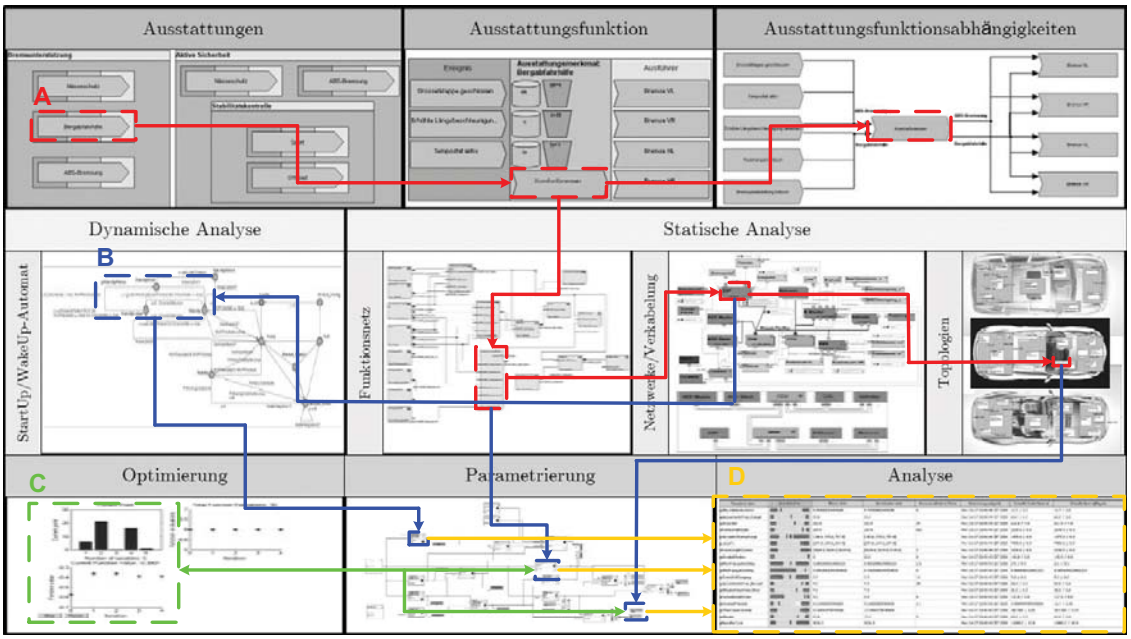


Abbildung 5.2.: Idee der durchgängigen Integration eines mehrteiligen Entwicklungskonzepts im Designfluss

in einer Metriktabelle visualisieren und variantenspezifisch vergleichen und bewerten (Prozessschritt *D*).

5.1.2. Modellierungsebenen

In Abs. 4.3.3 werden FlexRay-Parametergruppen den entsprechenden Modellierungsebenen auf E/E-Architekturebene zugeordnet. In /[Sch08]/ erfolgt eine Übertragung des Konzepts mithilfe einer Metamodelladaption auf das E/E-Architekturmodellierungswerkzeug PREEvision.

5.1.2.1. Anforderungsebene

Anstelle der in Abs. 2.2.4.1 vorgestellten Featurebäumen werden die funktionalen Anforderungen an eine E/E-Architektur auf die PREEvision-spezifischen Notationsformen abgebildet. Grundsätzlich werden dabei Ausstattungspakete konfektioniert und je nach Klassifikationsmethode partiell in hierarchischer Form spezifiziert (s. Abb. 5.3). So lässt sich die Ausstattungsoption *Stabilitätskontrolle* mit verschiedenen Abstufungen (*Sport, Offroad*) versehen. Zusammen mit den Ausstattungsoptionen *Nässeschutz* und *ABS-Bremsung* bildet die *Stabilitätskontrolle* zusätzlich das komponierte Ausstattungspaket *aktive Sicherheit*.

Die funktionale Beschreibung einer jeden Ausstattungsoption erfolgt auf einer eigenständigen Notationsebene. Dazu werden die in Abs. 2.5.2 beschriebenen Wirkketten über die Tupel (*Requestor,Functionality,Fulfiller*) definiert. Dabei werden alle anfordernde Ereignisse in funktionaler Form erfasst (*Drosselklappen geschlossen* $\rightarrow dk = 0$, *erhöhte Geschwindigkeit* $\rightarrow v > 50$, *Tempostat aktiv* $\rightarrow ts = 1$, ...) und einer übergeordneten Funktion *Komfortbremsen* zugeordnet. Die Reaktion auf die Ereignisse erfolgt über explizit

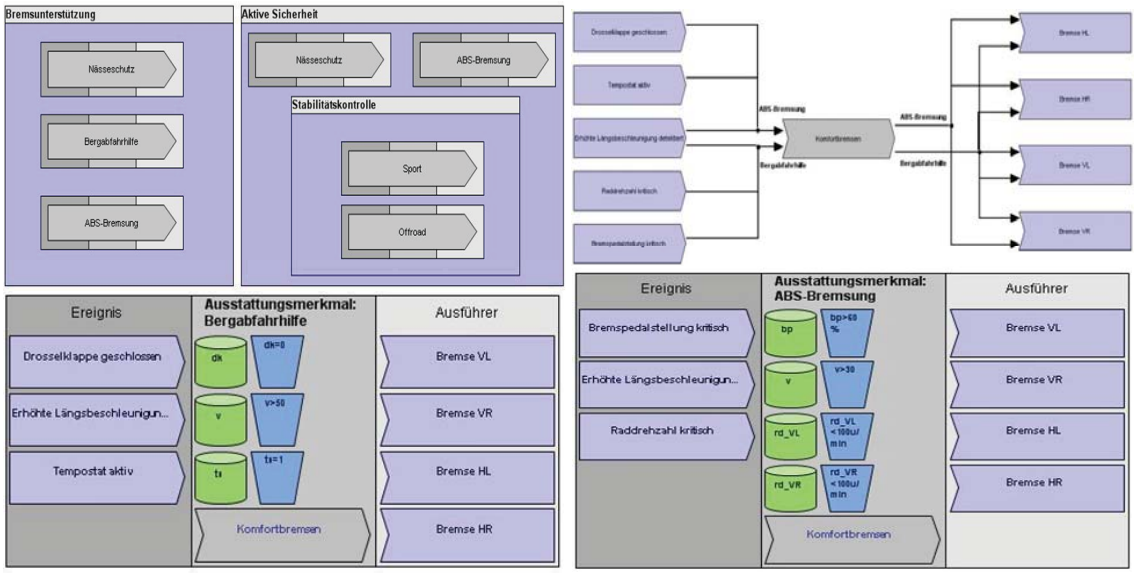


Abbildung 5.3.: Hierarchische Featuremodellierung aus Basis funktionaler Wirkketten

definierte Komponenten (in der Regel in mechanischer Form durch die Ansteuerung von Aktoren (*Bremse VL*, *Bremse VR*, *Bremse HL*, *Bremse HR*)). In einer separaten Darstellung lassen sich alle Abhängigkeiten zwischen den anfordernden Ereignissen, der Ausstattungsfunktion *Komfortbremsen* und den erfüllenden Komponenten in Graphenform integriert betrachten.

5.1.2.2. FN-Ebene

Bei der Funktionsnetzumsetzung werden analog zu Abs. 4.4.2 folgende Inhalte spezifiziert:

- Funktionskomponenten und komponierte Funktionskomponenten im Sinne einer hierarchischen Funktionsuntergliederung (Knoten),
- Signale und deren Aufbau (*Offsets*, Auflösung, Wertebereiche,...), die zwischen den Funktionskomponenten ausgetauscht werden (Kanten),
- Zuordnung von Sendearten (Kommunikationstypen) an den Schnittstellen zwischen Funktionskomponente und Sendesignal (*Sende- und Empfangsports*),
- Gruppierung von Signalen in Gruppen (funktional, modusabhängig, zustandsabhängig),
- Spezifikation der Sende-/Empfangsbeziehungen zwischen den Funktionen (*p2p*, *broadcast*).

In Abb. 5.4 wird das instanzierte Funktionsnetz aus der Fallstudie (s. Abs. 5.4) dargestellt. In der entwickelten Fallstudie wird beispielsweise auf logischer Ebene die Funktionskette *E-Gas* über das Tupel der Funktionsblöcke (*Gaspedalposition*, Verarbeitung in der *Motorsteuerung*, *Drosselklappenstellung*) spezifiziert. Zwischen den Modulen werden die Datensignale *MO1_Pedalwert* und *MO3_DKW* ausgetauscht. Für die Signale werden

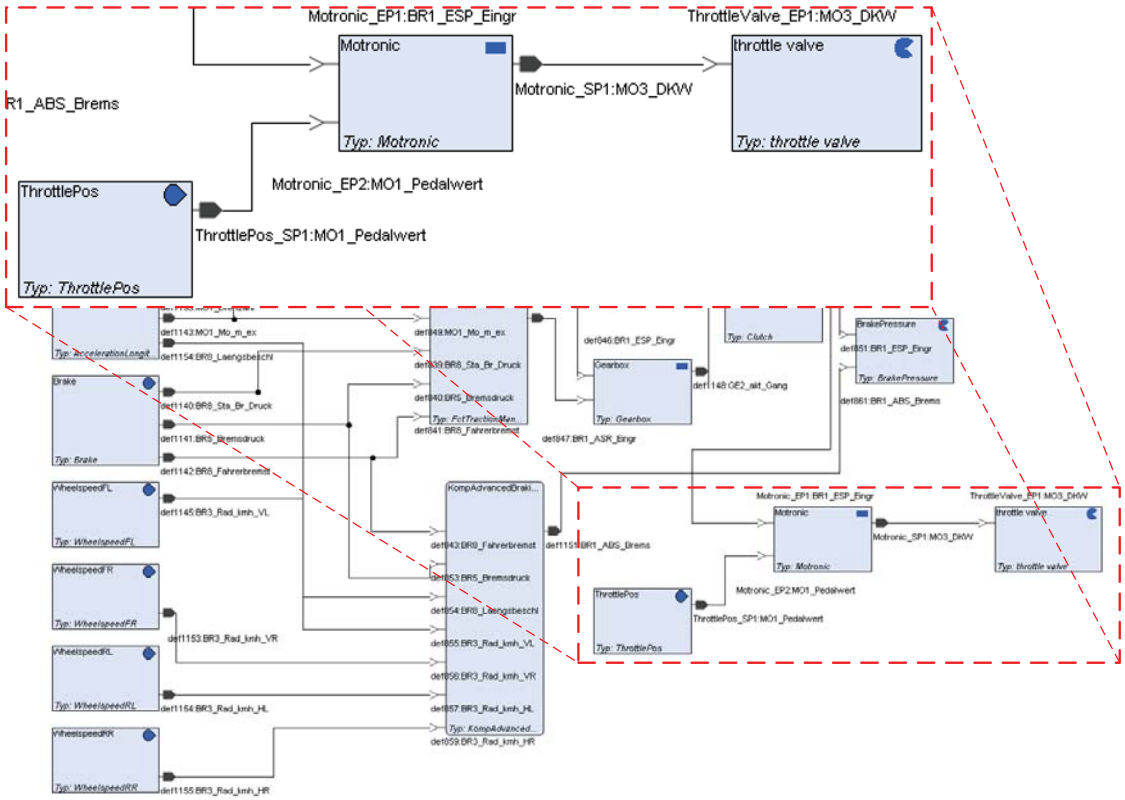


Abbildung 5.4.: Ausschnitt aus einem Funktionsnetz der Antriebs- und Fahrwerksdomäne

Länge, Einheit, Faktoren und Maximalbereiche definiert. Die Sendeports *ThrottlePos_SP1* und *Motronic_SP1* bekommen einen zyklischen Sendetyp mit Zykluszeit zugewiesen. Am Empfangsport *Motronic_EP1* wird das spontan gesendete Signal *BR1_ESP_Eingriff* in Abhängigkeit von der Aktivierungszeit empfangen. Je nach Signalgruppierung (bzw. Ausstattungsoption) könnte *BR1_ESP_Eingriff* funktional exklusiv einem Normalmodus zugeordnet und damit im Sportmodus nicht verarbeitet werden. In diesem Ausschnitt wird jedem Signal genau ein Empfänger zugeordnet, was einem *p2p*-Sende-/Empfangstyp entspricht. Die Information *BR6_Fahrer_bremst* wird hingegen mehreren Empfängern zugeordnet, was je nach Partitionierung zu einer *broadcast*-Botschaft führen kann.

5.1.2.3. PN-Ebene

Zur Partitionierung der logischen E/E-Architektur wird vorab, analog wie in Abs. 4.2.2.7 beschrieben, die statische technische E/E-Architektur entwickelt. Bei der Umsetzung der PN-Ebene lassen sich in dem Zusammenhang folgende Aspekte, die gleichermaßen mit der FlexRay-Systemparametrierung korrelieren, berücksichtigen:

- Verbaute FlexRay-Steuergeräte im Cluster (*blau*),
- Verbaute FlexRay-Steuergeräte mit aktiven Sternkopplern (*rosa*),
- Spezifikation der Netzwerkanbindungstypen (Stern-, Bustransceiver) (*blau*, *grün*),
- Systemumgebung (Sensorik/Aktorik) (*grün*, *rot*),

- Leistungsversorgung der Steuergeräte (Generatorversorgung) (*orange*, *gelb*, *dunkelrot*).

In Abb. 5.5 wird die physikalische Netzwerksicht aus der Fallstudie (s. Abs. 5.4) dargestellt.

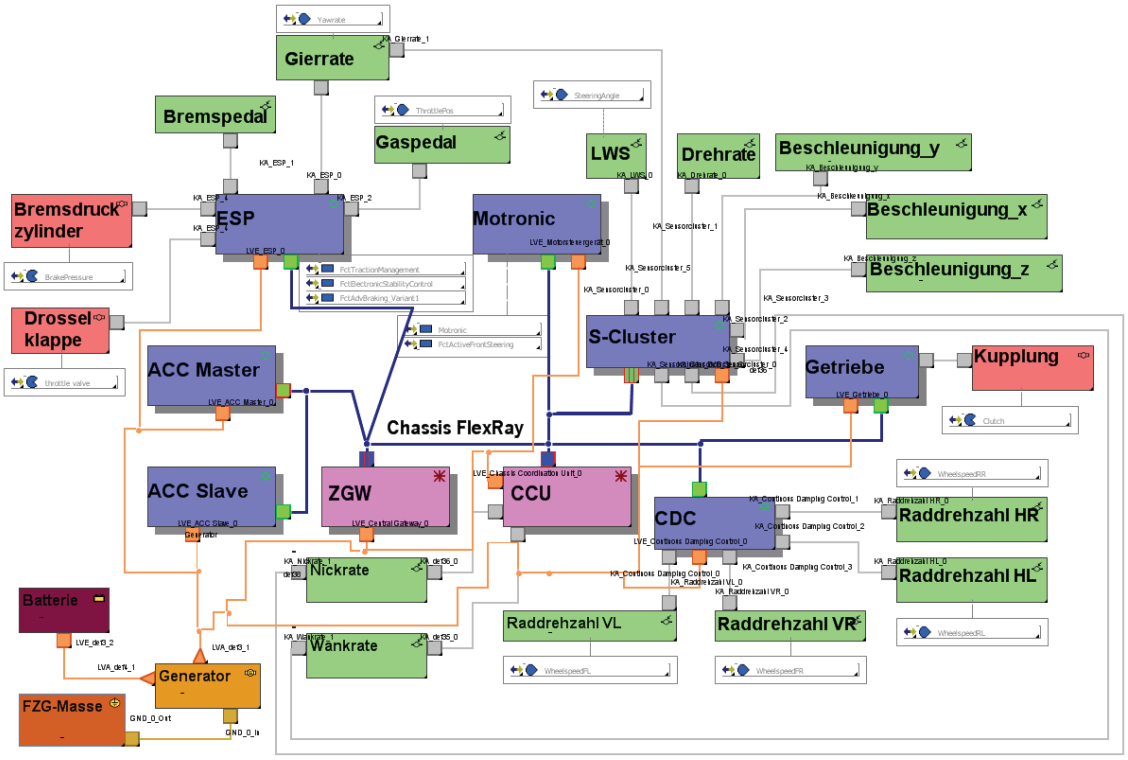


Abbildung 5.5.: Darstellung der physikalischen Netzwerksicht inklusive Aktorik/Sensorik und Leistungsversorgung

Die Grundidee besteht aus einem variablen Architekturansatz, der je nach Ausbaustufe bis zu neun FlexRay-Steuergeräte umfasst. Analog zu den ersten Serienentwicklungen /[NSF07]/ wird dabei auf modulare Architekturansätze zurückgeriffen, die sich aus aktiven Sternkopplersteuergeräten und konventionellen FlexRay-Steuergeräten zusammensetzen. StartUp-fähige Steuergeräte werden durch rote Schraffuren in Busanbindungen und Sync-Steuergeräte durch eine rote Umrandung der Busanbindung gekennzeichnet. In diesem Ansatz wird das *Gateway*-Steuergerät als zentrales Steuergerät aufgefasst, welches je nach Ausbaustufe mit den Steuergeräten *ACC Master, ACC Slave, Fahrwerkskoordinator, Dämpfungskontrolle, ESP, Getriebe, Motronic, Sensorcluster* verbunden wird. Zur Gewährleistung einer optimalen physikalischen Signalübertragung muss bei mehr als fünf Steuergeräten ein aktiver Sternkoppler eingesetzt werden, um Übertragungsdistanzen auf Basis passiver Bustopologien auf maximal fünf Teilnehmer zu begrenzen.

Die Sensoren und Aktoren werden vollständig außerhalb des FlexRay-Clusters mit diskreten Zugängen an die verschiedenen Steuergeräte angebracht. Je nach Architekturvariante werden verschiedene Sensoren integriert an das Sensorcluster angeschlossen oder verteilt mit unterschiedlichen Steuergeräten verbunden. Zusätzlich kann auf der PN-Ebene die Leistungsversorgung der Steuergeräte für die Fallstudie modelliert

werden, um beispielsweise DC/DC-wandlergestützte Steuergeräte oder das Klemmenkonzept im nächsten Schritt abzubilden. Zusätzlich wird unter anderem in Abb. 5.5 die Partitionierung der Funktion *FctActiveFrontSteering* auf das *Motronic*-Steuergerät beispielhaft angezeigt.

5.1.2.4. WIR-Ebene

Die konkrete Verkabelung des FlexRay-Clusters erfolgt auf der WIR-Ebene mit folgenden Schwerpunkten:

- Topologische Anordnung des Clusters (Bus, Stern, Hybrid)
- Stecker-/Pinbelegung des physikalischen Übertragungskanals (Erweiterbarkeit)
- Klemmenspezifkation (Kl.15/Kl.30-System)

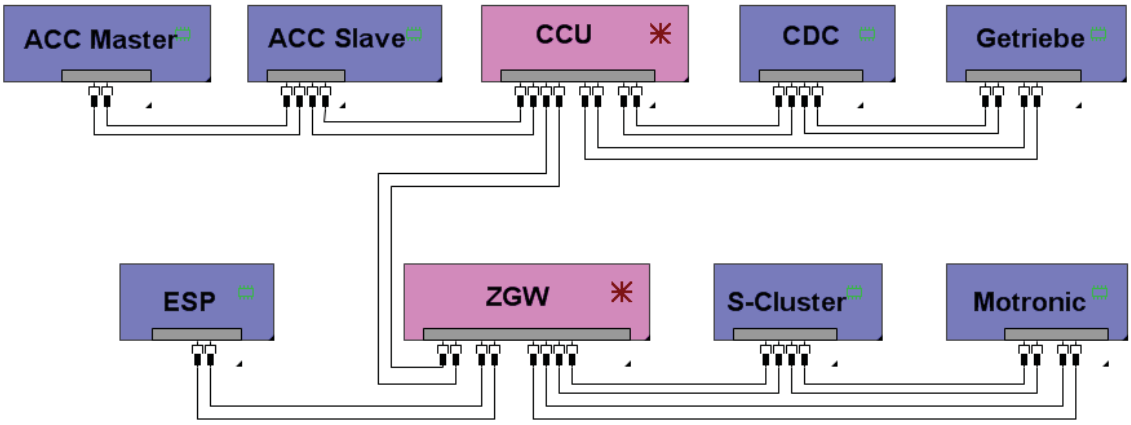


Abbildung 5.6.: Verkabelungskonzept in der WIR-Ebene inklusive Steckerbelegung

In der Fallstudie werden die Steuergeräte *ACC Master, ACC Slave, CDC* und *Getriebe* über den aktiven Sternkoppler *CCU* mit den restlichen Steuergeräten verkabelt. Die Herausforderung dabei ist, dass je nach Ausbaustufe die *CCU* entfallen kann, wodurch die einzelnen Steuergeräte *ACC Master, ACC Slave, CDC* und *Getriebe* je nach Steckerbelegung und Pinverfügbarkeit an den aktiven Sternkoppler *ZGW* angeschlossen werden können. In speziellen Ausstattungsvarianten lassen sich die aktiven Sternkoppler bei Verbindung mit maximal zwei weiteren FlexRay-Knoten auch als konventionelle FlexRay-Steuergeräte (mit Standard-Transceiver) auslegen.

5.1.2.5. TOP-Ebene

In der TOP-Ebene werden die Verbaubereiche der einzelnen Steuergeräte festgelegt. Aus Sicht der FlexRay-Systemparametrierung interessieren dabei folgende Aspekte:

- Geometrische Dimensionierung einer Topologie,
- Restriktionen für den Verbau im Fahrzeug (Topologieeinschränkungen).

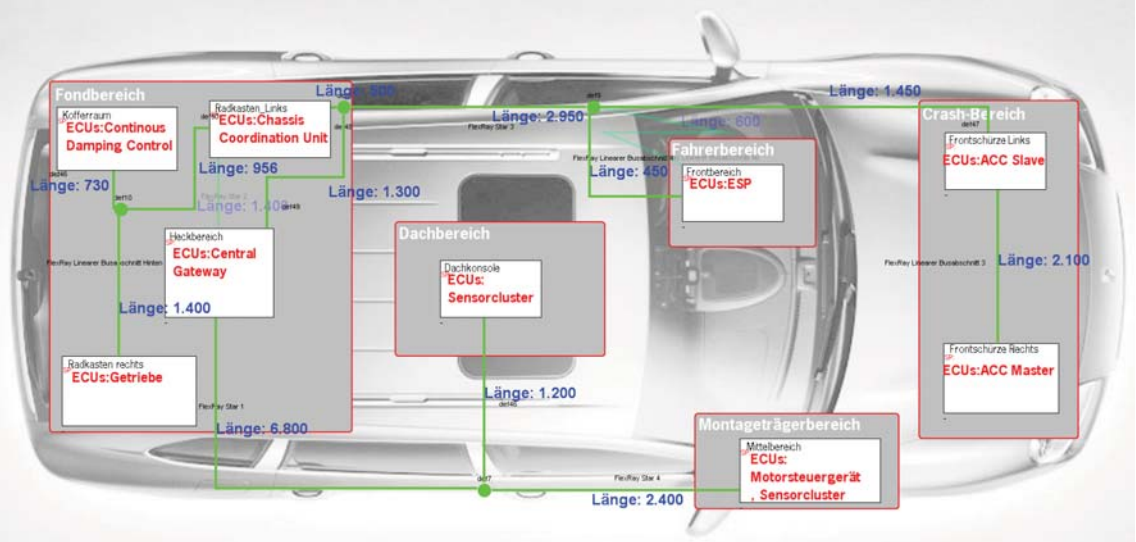


Abbildung 5.7.: Topologiemodell zur Auslegung der Verkabelungsgrößen des Bordnetzes

Bei der Betrachtung unterschiedlicher Fahrzeugbaureihen (*Sportwagen (Heckmotor), Sportwagen (Mittelmotor), SUV*) ergeben sich unterschiedliche Anforderungen an die Positionierung der Steuergeräte im Fahrzeug. In Abb. 5.7 wird die Topologie für einen *SUV* in den Verbauorten konkretisiert. Im FlexRay-Systemdesign müssen die Abstände zwischen den einzelnen Steuergeräten ermittelt werden, da sich diese direkt auf die physikalischen Verzögerungen im Netzwerk auswirken. Im weiteren Sinne lassen sich auch Einschränkungen im Bereich der Verbauumgebung identifizieren, etwa der vorkommende Temperaturbereich oder EMV-Einflüsse. In der Fallstudie liegt der Fokus vorrangig auf der Behandlung unterschiedlicher Topologievarianten und deren Auswirkungen im Wechselspiel mit den verbauten Halbleiterteilen (*Transceiver, Quarze*) auf die möglichen zulässigen FlexRay-Systemkonfigurationen.

5.1.2.6. PK-Ebene

Auf der PK-Ebene fliessen folgende Aspekte in den Parametrierungsprozess ein.

- Standardsoftwarespezifikationen (Varianten),
- Hardwarespezifikationen (Alterungseffekte, Performanz, Ressourcen).

Die physikalischen Komponenten werden modulorientiert gestaltet, wobei explizite Beziehungen zwischen den Modulen vernachlässigt bleiben. Auf der Hardwareseite interessieren vorrangig die Eigenschaften des Quarzes für die zeitgesteuerte Kommunikation sowie die physikalische Terminierung und auf Softwareseite der Typ der Standardsoftware[1]. In Abb. 5.8 wird ein einfaches PK-Modell mit seiner Busanbindung dargestellt.

[1]Die Eigenschaften der Standardsoftware bleiben in dieser Umsetzung unterspezifiziert und werden lediglich auf verschiedene SSW-Klassen (zuliefererspezifisch) reduziert.

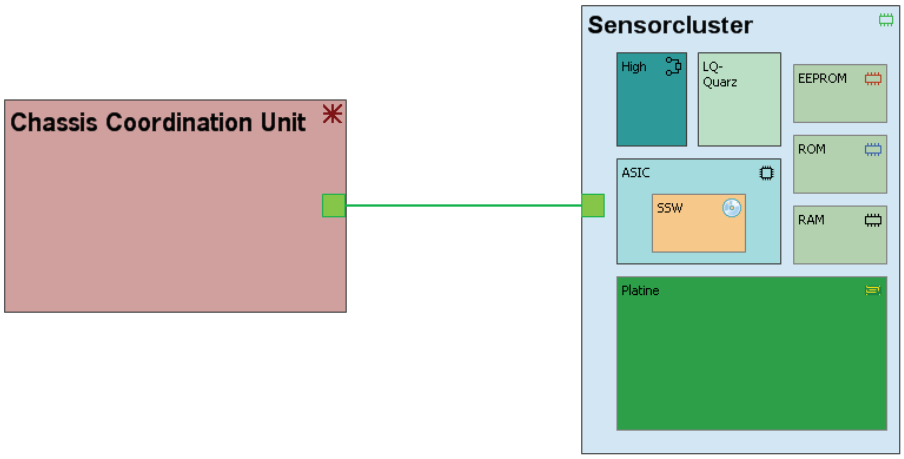


Abbildung 5.8.: Komponentenstruktur eines FlexRay-Steuergeräts inklusive seiner Hard- und Softwaremodule sowie der Busanbindung zu einem aktiven Sternkoppler

5.1.3. FlexRay-Parametrierungseditor (PE)

Die konventionelle Berechnung von E/E-Architekturmetriken basiert in dem gewählten Ansatz auf individuell erstellbaren Berechnungsskripten. Bei der Erfassung und Berechnung der FlexRay-Konfiguration führt dieses Vorgehen zu Einschränkungen, da die hohe Komplexität des FlexRay-Parametersatzes eine sinnvolle Dekomposition erfordert. Zusätzlich sollen neben der eigentlichen Systemparametrisierung auch benutzerspezifische Qualitätskennzahlen erfasst werden, die bei der Berechnung und Verwaltung inhaltlich von der Systemkonfiguration datentechnisch getrennt bleiben sollten. Eine statisch kodierte Umsetzung des vollständigen Parametersatzes in einem Werkzeug führt zu starken Einschränkungen bei der Hinzunahme von anwenderspezifischen Vorgaben (speziell bei der Substitution von Spezifikationskonstanten durch Parameter). Konsequenterweise muss der E/E-Architekt die Möglichkeit besitzen die Berechungsvorschriften zur Entwicklung des FlexRay-Designs explizit im Rahmen des Architekturdesigns spezifizieren zu können. In dieser Arbeit wird dafür auf die Entwicklung eines FlexRay-Parametrierungseditors (PE) zurückgegriffen, um die Eigenschaften der E/E-Architektur auf einen gültigen FlexRay-Parametersatz zu übertragen.

Das Grundkonzept des Editors basiert auf einem dreiteiligen Konzept. Dabei wird durch spezielle Modellabfrageregeln auf Basis des Metamodells auf arbiträre Artefakte der E/E-Architektur zugegriffen, etwa alle Steuergeräte mit einem aktiven Sterntransceiver[2]. Diese Artefakte und deren Attribute dienen dabei als Eingangsparameter zur Umsetzung der FlexRay-Spezifikationsrestriktionen. Diese werden dabei algorithmisch in Berechnungsblöcken skriptbasiert oder als JAVA-Code implementiert. Für den Sonderfall, dass anstatt eines berechneten Ergebnisses ein konstanter Wert für einen Berechnungsblock verwendet werden soll, kann der Benutzer diesen statischen Wert dem Berechnungsblock zuweisen und explizit im Graphen selektieren. Die Berechungsblöcke lassen sich über Datenverbindungen arbiträr verknüpfen. Eine Spezialfunktion liefert dabei die *Join*-Verknüpfung (vgl. /[KE04]/) zur Zusammenfassung mehrerer Modellabfrageergebnisse. Mit speziellen Analyseblöcken lassen sich visuelle Hilfen zur Interpretation von Berechnungsergebnissen integrieren (Aufzählungs-, Ampel-, Skalenblöcke).

[2]Die Umsetzung einer Modellabfrage wird in Anhang D.3 exemplarisch dargestellt.

In Abb. 5.9 wird beispielhaft ein Ausschnitt zur Berechnung der maximalen asymmetrischen Transceiververzögerung bei den beiden Gruppen der Sende- und Empfangssteuergeräte in Form eines Metrikdiagramms dargestellt.

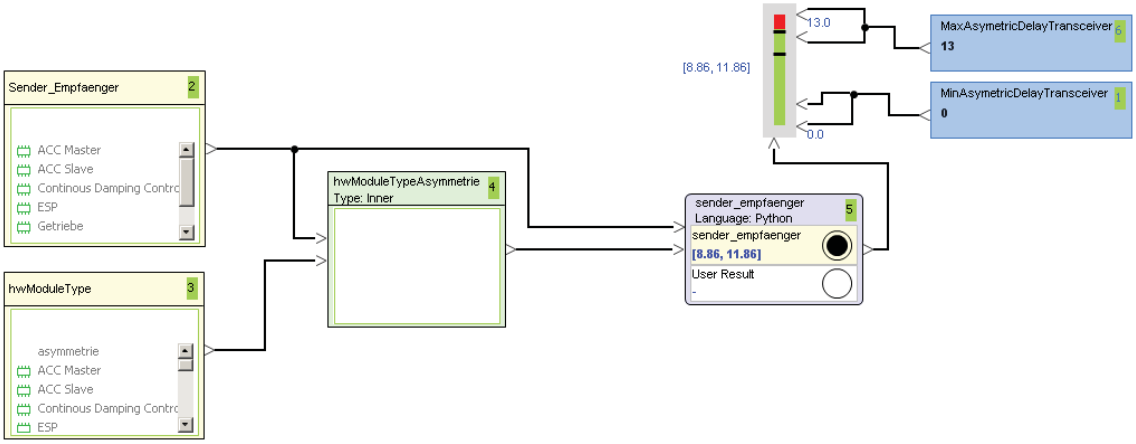


Abbildung 5.9.: Exemplarischer Ausschnitt eines Metrikmodells zur Berechnung der maximalen asymmetrischen Transceiververzögerung jeweils bei den Sender- und Empfängersteuergeräten

Durch eine variantenspezifsche Aktivierung von E/E-Artefakten lassen sich unterschiedliche Eingabevektoren bei der Durchführung von Modellabfragen erzeugen, die zu architekturspezifischen FlexRay-Parametersätzen führen. Dadurch entsteht die Möglichkeit verschiedene E/E-Architekturen und deren Varianten direkt miteinander zu vergleichen. Gleichermaßen wird im weiteren Sinn die Abhängigkeit zwischen den kundenspezifischen Fahrzeugausstattungsanforderungen und der technischen FlexRay-Parametrierung direkt nachvollziehbar.

Ein Vorteil bei der expliziten Umsetzung des FlexRay-Parametersatzes in Graphenform innerhalb des PE folgt aus den individuellen Strukturierungsformen und Dekompositionsmöglichkeiten. So lassen sich atomare Strukturen im Parametersatz erfassen und reinstanziert in diversen Kompositionen der Berechungsgraphen wiederverwenden. Dadurch bauen die Metriken aufeinander auf und die Auswirkung einer Modifikation eines Berechnungsblocks innerhalb der Parameterberechnung wird offensichtlich.

5.1.4. Roundtrip-Engineering

Um eine adaptierte FlexRay-Parametrierung zu erzielen ist es notwendig sämtliche Abhängigkeiten der Parameterberechnungsformeln einzeln modifizierbar zu gestalten. Daher kann die Definition der FlexRay-Parameter nicht statisch auf Basis fixer Formeln (*Constraints*) der offiziellen FlexRay-Spezifikationen erfolgen. Dies lässt sich dadurch begründen, dass in der Regel zusätzliche strikte Vorgaben, beispielsweise von einem Zulieferer, berücksichtigt werden, die sich in das starre Korsett der FlexRay-Berechnungsformeln einfügen müssen. Für diesen Zweck lässt die Validierung des FlexZOOMED-Ansatzes Freiräume für die Integration berechnungsunabhängiger Werte (s. Abb. 5.11). Im Sinne eines *Roundtrip-Engineering*-Ansatzes können dabei in den umgesetzten Berechnungsformeln bestimmte Attribute des E/E-Architekturmodells modifiziert und gleichzeitig neue Attribute berechnet werden. Dadurch wird die Modellentwicklung und die Systemparametrierung in einem parallel ablaufenden Entwicklungsstil vollzogen.

5.1.5. Parameteranalysesicht (PA)

Auf Basis arbiträrer Berechnungsergebnisse innerhalb des PE-Editors lassen sich die Werte der individuellen Qualitätsmerkmale einer FlexRay-basierten E/E-Architektur in Form von Analyseblöcken messen. Durch eine direkte Gegenüberstellung von skalierbaren Analyseblöcken wird die Möglichkeit zur zentralen Erfassung der Eigenschaften von Architekturvarianten geschaffen.

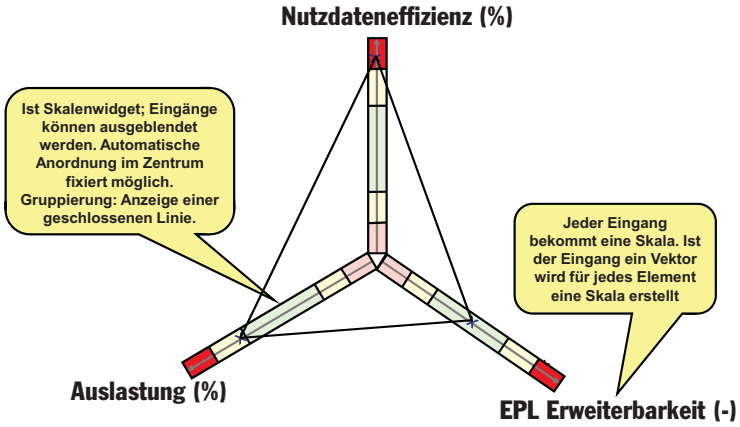


Abbildung 5.10.: Parameteranalyse auf Grundlage spezifisch skalier- und komponierbarer Diagramme

Durch die unterschiedlichen spezifisch definierbaren Grenzwerte werden die Skalenblöcke als Diagrammachsen spezifiziert, die zu Netzdiagrammen miteinander verknüpft werden können. Durch die Skalierung entsteht die Möglichkeit zur Abgrenzung und zum verbesserten Vergleich von Architekturalternativen (s. Abb. 5.10 und 5.11).

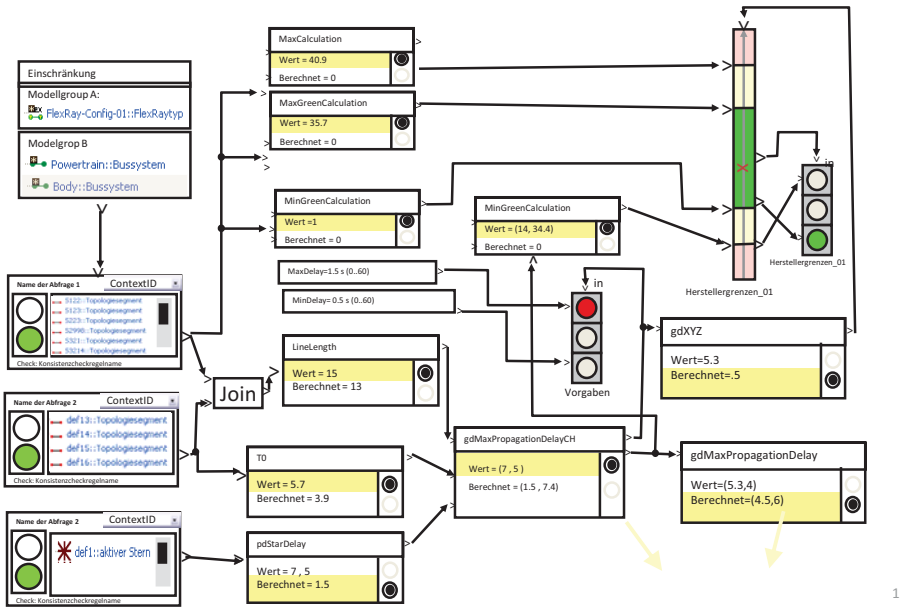


Abbildung 5.11.: Flexibel komponierbarer Parametrierungseditor mit Zugriff auf die Modellebene

Als Ergänzung zur Netzdiagrammdarstellung eignet sich die zentrale Ablage sämtlicher berechneter Aspekte des flexiblen zeitgesteuerten Systems in Tabellenform. Diese Metriktabelle filtert dabei die komplexen Diagramme des PE-Editors und unterstützt die integrierte Interpretation bei der automatischen FlexRay-Parametersatzgenerierung. In Abb. 5.12 wird der Auszug aus einer FlexRay-Parametrierungstabelle dargestellt.

Parametername	Grenzbereiche	Aktiver Wert	Berechneter Wert	Benutzerdefinierte Werte	Berechnungszeitpunkt	Schwelle Positiv/Neutral	Schwelle Neutral/Negativ
gdMaxInitializationError		9.769000004470348	9.769000004470348	8	Mon Oct 27 20:43:44 CET 2008	11.7 / 0.0	11.7 / 0.0
gdActionPointOffset_Robust		20.0	20.0		Mon Oct 27 20:43:44 CET 2008	63.0 / 1.0	63.0 / 1.0
gdStaticSlot		212.0	212.0	24	Mon Oct 27 20:43:44 CET 2008	661.0 / 4.0	661.0 / 4.0
pFrameLengthStatic		269.0	269.0	653	Mon Oct 27 20:43:44 CET 2008	2638.0 / 0.0	2638.0 / 0.0
pdAcceptedStartupRange		[190.0, 379.0, 757.0]	[190.0, 379.0, 757.0]		Mon Oct 27 20:43:44 CET 2008	1875.0 / 0.0	1875.0 / 0.0
pLatestTx		[234.0, 234.0, 234.0]	[234.0, 234.0, 234.0]		Mon Oct 27 20:43:44 CET 2008	7900.0 / 0.0	7900.0 / 0.0
pFrameLengthDynamic		[5169.0, 5169.0, 5169.0]	[5169.0, 5169.0, 5169.0]	2	Mon Oct 27 20:43:44 CET 2008	2638.0 / 0.0	2638.0 / 0.0
gdSymbolWindow		0	22.0	0	Mon Oct 27 20:43:43 CET 2008	142.0 / 0.0	142.0 / 0.0
gdMaxPropagationDelay		0.855000002980232	0.855000002980232	2.5	Mon Oct 27 20:43:43 CET 2008	2.5 / 0.0	2.5 / 0.0
gdMinPropagationDelay		0.19600000447034838	0.19600000447034838	0	Mon Oct 27 20:43:43 CET 2008	0.8550000029802323	0.8550000029802323
gClusterDriftDamping		2.0	2.0	12	Mon Oct 27 20:43:43 CET 2008	5.0 / 0.0	5.0 / 0.0
gdActionPointOffset_Normal2		7.0	7.0	39	Mon Oct 27 20:43:43 CET 2008	63.0 / 1.0	63.0 / 1.0
gdMinislotActionPointOffset		7.0	7.0		Mon Oct 27 20:43:43 CET 2008	31.0 / 1.0	31.0 / 1.0
gPayloadLengthStatic		9.0	9.0	8	Mon Oct 27 20:43:43 CET 2008	127.0 / 0.0	127.0 / 0.0
gAssumedPrecision		9.110000005960465	9.110000005960465	2.1	Mon Oct 27 20:43:43 CET 2008	3.0589999985098846	11.7 / 0.15
gOffsetCorrectionMax		10.690437507450 50	10.690437507450 50		Mon Oct 27 20:43:43 CET 2008	303.567 / 0.15	303.567 / 0.15
gdMinislot		15.0	15.0	5	Mon Oct 27 20:43:43 CET 2008	63.0 / 2.0	63.0 / 2.0
gMacroPerCycle		3636.0	3636.0		Mon Oct 27 20:43:43 CET 2008	16000.0 / 10.0	16000.0 / 10.0

Abbildung 5.12.: Ausschnitt aus einer FlexRay-PE-Metriktabelle zur Interpretation einer E/E-Architekturvariante

In diesem Beispiel gibt es nur einen Ergebnisblock. Deshalb wird nur eine Zeile in der Tabelle angezeigt. Bei einem Skalen- oder einem Aufzählungsblock wird das Ergebnis an Stelle des Ampelsymbols in der zweiten Spalte dargestellt.

5.2. Vorgehen bei der E/E-Architekturentwicklung

Wie in Abs. 2.5 beschrieben, wird die eigenständige Disziplin der E/E-Architekturentwicklung durch das Bestreben der integrierten Entwicklung der Bereiche Hardwaretopologien, Mechanik, Regelung, Funktionalität oder Software motiviert. In diesem Zusammenhang steht die Abbildung der E/E-Architektur auf mehrere Fahrzeugbaureihen und -derivate mit stringenten Anforderungen (Kosten, Gewicht, Verbrauch, Zuverlässigkeit) im Vordergrund (vgl. Abb. 5.13).

Unter diesen Prämissen werden sämtliche Modifikationen an der Architektur bei der Entwicklung durchgeführt. Als Modifikation versteht sich vor allem die Architekturerweiterung durch die Integration einzelner Komponenten, Steuergeräte und -verbünde, deren Auswirkungen am Architekturmodell analysiert werden.

5.2.1. Systemintegration

Bei der Systemintegration stehen Aspekte der technischen und der logischen E/E-Architekturentwicklung im Vordergrund. Dabei kommt den folgenden Fragestellungen die grösste Aufmerksamkeit zu:

Technische Systemarchitektur:

- Integration eines neuen Steuergeräts,
- Veränderung am Verkabelungskonzept (Stichleitungen, Zusatzleitungen, Leitungslängen, Leitungstypen, Aktive Sterne),

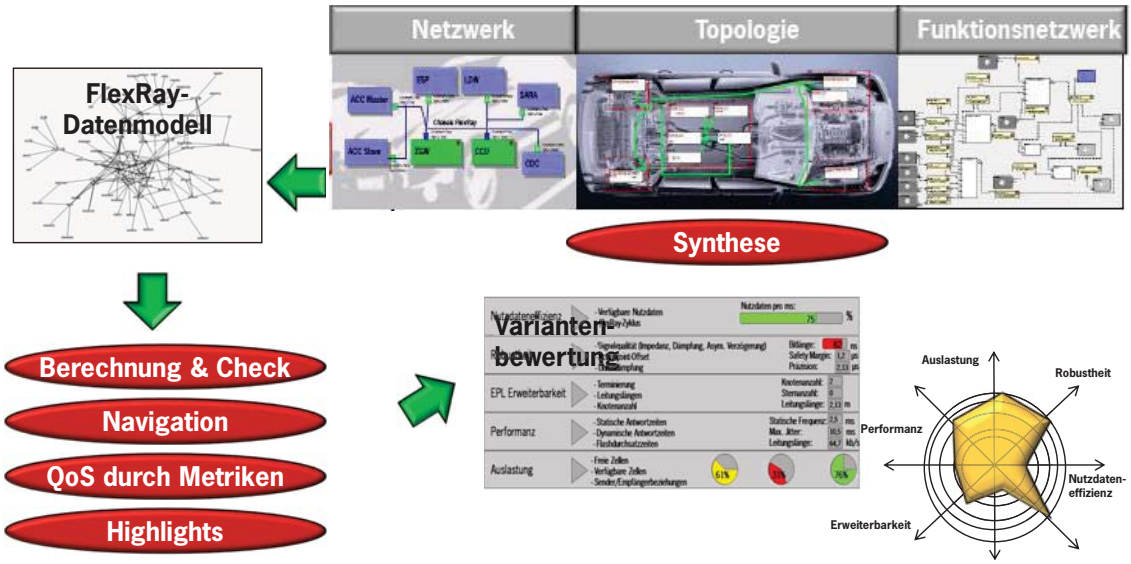


Abbildung 5.13.: Validierung des FlexZOOMED-Ansatzes

- Substitution von Subkomponenten (μC, CC, Transceiver, Quarze),
- Migration der Leistungsversorgung (Generatorstützung, Versorgungsklemmenkonzept (Kl.15/Kl.30)),
- Globale Modifikation der Systemanforderungen (Performanz, Zuverlässigkeit).

Logische Systemarchitektur:

- Integration neuer Komponenten (Standardsoftware),
- Integration neuer Funktionalität (Applikationsebene, Systemebene (z.B. Diagnose, TP, NM)),
- Datenveränderungen (Botschaftslayout, Schedulelayout),
- Globale Modifikation der Systemanforderungen (Performanz, Zuverlässigkeit).

Ergänzend müssen übergreifende Modifikationen, etwa ein verändertes Plattform- und Variantenkonzept auf beiden Ebenen berücksichtigt bleiben. Im nächsten Schritt gilt es diese Themen bei der Systemintegration im Rahmen der statischen und dynamischen Systemanalyse zu betrachten.

5.2.2. Statische Analyse

Die statische Systemanalyse verzichtet auf Systemsimulationen und lässt sich demzufolge idealweise regelbasiert durch Entwicklung und Pflege von Metriken, Modellsuchregeln und dem FlexRay-Parametrierungsmodells umsetzen.

5.2.2.1. Metrikkonzept

Zur Bewertung und Analyse der E/E-Architekturmodellvarianten werden Metrikskripte entwickelt, deren logische Struktur beispielhaft in Abb. 5.14 dargestellt ist.

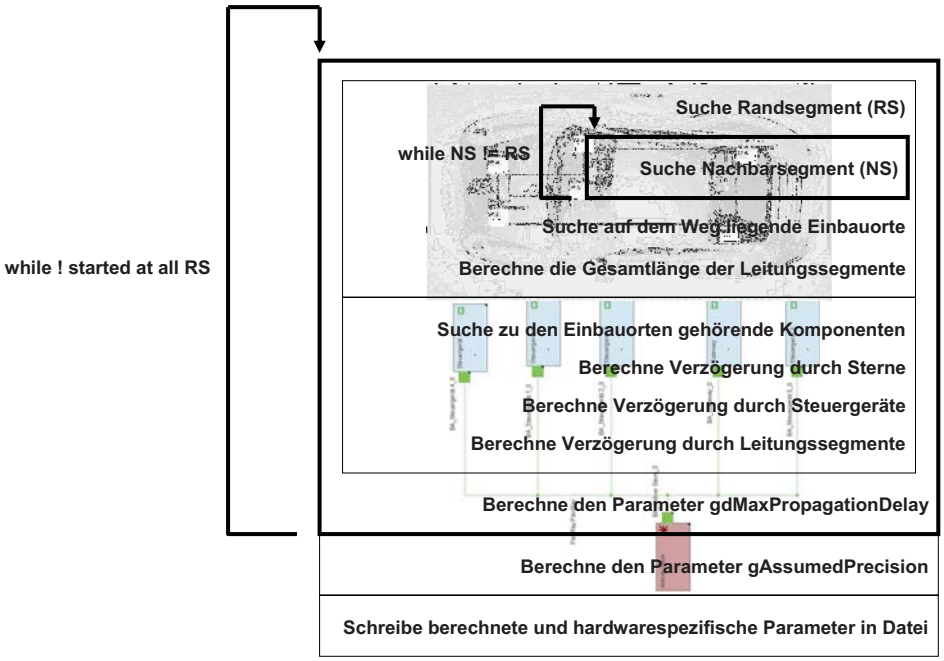


Abbildung 5.14.: Struktur eines Metrikskripts zur hardwarespezifischen Berechnung von FlexRay-Parametern

In diesem Beispiel werden die beiden hardwarespezifischen FlexRay-Parameter α_1 =*gdMaxPropagationDelay* und α_5 =*gAssumedPrecision* auf Basis der modellierten Topologie und der verbauten Bauteile berechnet. Als nichtfunktionales Qualitätskriterium lassen sich dabei parallel Kostenabschätzungen durchführen, die aufgrund der verbauten Hardwarekomponenten anfallen. Die berechneten Parameterwerte, die Kosten und die hardwarespezifischen FlexRay-Parameter können prozessabhängig zur Dokumentation oder zum Datenaustausch extrahiert werden, beispielsweise in einem zwischengelagerten Optimierungsschritt (s. Abs. 5.1). Aus der TOP-Ebene werden die Längen der einzelnen Verbausegmente ermittelt. Dadurch lässt sich der FlexRay-Parameter

- $LineLength_{M,N}$

aus /[Fle05b]/ zur Spezifikation der gesamten Segmentlänge zwischen Knoten M und Knoten N durch eine Metrikberechnung erfassen. Aus der PN-Ebene werden zusätzliche globale FlexRay-Parameter bzw. Konstanten hinzugenommen:

- x_{32} =*gClusterDriftDamping*
- β_1 =*gdMaxMicrotick*
- α_3 =*gdMinPropagationDelay*
- c_6 =*cClockDeviationMax*

Neben den globalen Parametern ergänzen knotenlokale Parameter die Berechnung. Dabei spielen vorrangig die Verzögerungsparameter an den Bustransceivern beim Senden und beim Empfangen sowie die auftretenden Latenzen bei der Signalweiterleitung im aktiven Sternkoppler

- *pdBDTx*
- *pdBDRx*
- *pdStarDelay*

eine entscheidende Rolle (vgl. /[Fle05a]/).

5.2.2.2. Regelbasierte Prüfung

Neben dem Metrikkonzept profitiert die Validierung der E/E-Architekturmodellierung von der werkzeuggestützten regelbasierten Prüfung des zugrunde gelegten Werkzeugs. In diesem Kontext orientieren sich Suchmuster am vorliegenden Metamodell und unterstützen bei der Objektidentifikation für die Metrikberechnung oder verifizieren den Modellstand auf Konsistenz (s. Abb. D.2). Speziell bei der Umsetzung komplexer Plattform- und Variantenstrukturen im Modell helfen zusätzliche Propagationsregeln zum automatisierten Gruppieren von Modellartefakten.

5.2.3. Dynamische Analyse

Der primäre Einsatzzweck der dynamischen Architekturanalyse im Kontext der FlexRay-Systemparametrierung konzentriert sich auf die Parameterauslegung für das Weck- und Startverhalten des Bussystems und die Parametrierung des Fehlerdegradierungsmodells (Übergange zwischen aktiver, passiver und angehaltener Kommunikation). Ergänzend lässt sich die Auslegung des dynamischen FlexRay-Segments per dynamischer Analyse umsetzen. Anfolgend wird eine einfache Umsetzung der Bereiche des Weck-/Startvorgangs und des Fehlerdegradierungsmodells erläutert.

Das Weckverhalten ist getrennt von dem Startvorgang zu betrachten, da im Fahrzeug in der Regel nicht alle Steuergeräte weckfähig ausgelegt werden, sondern teilweise auch hart mit Anliegen des Zündungsplus gestartet werden. In Abb. 5.15 wird die Protokollausführungskontrolle eines FlexRay-Kommunikationscontrollers als gezeiteter Automat im Werkzeug UPPAAL dargestellt[3].

Beim Weckvorgang stehen vordergründig die Phasen der Weckvorbereitung (*WakeUp-Listen*-Phase), des eigentlichen Weckvorgangs (*WakeUp-Send*-Phase) und der Detektion von Kommunikationskollisionen beim Weckvorgang (*WakeUp-Detection*-Phase) im Fokus. Diese Phasen hängen im Parametrierungsprozess direkt von der Auslegung der Länge der Weckvorbereitungsphase (*pdListenTimeout*) in Kombination mit detektierter Kommunkation auf dem Bus (*gListenNoise*) und der Anzahl der zu sendenden Wecksymbolen auf dem Bus (*pWakeUpPattern*) ab. Diese Parameter verzweigen selbst auf dritte Parameter im Bereich des maximalen Uhrendrifts, der Buszykluslänge, der Mikroticklänge im FlexRay-Kommunikationscontroller und der maximalen Uhrenabweichung im

[3]Die Umsetzung der POC fokussiert auf den Bereich des Weck- und Startvorgangs und beinhaltet ergänzend partielle logische Anteile, die aus den SDL-Diagrammen der FlexRay-Spezifikation /[Fle05a]/ entnommen wurden.

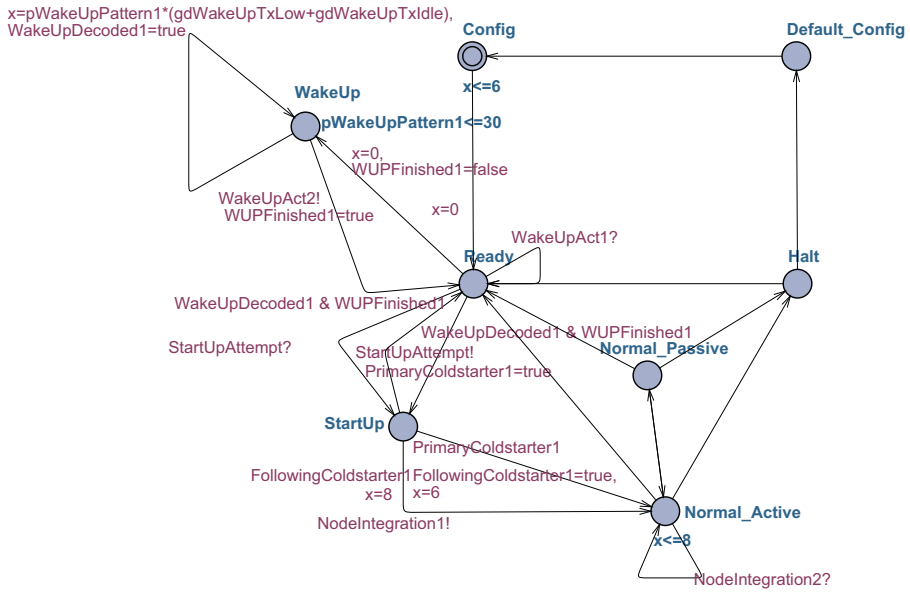


Abbildung 5.15.: Simulation des dynamischen Verhaltens der FlexRay-Protokollengine zweier Netzknoten auf Basis gezeiteter Automaten

Cluster. Die Komplexität in der Analyse der zeitlichen Ausprägung des Weckvorgangs ergibt sich aus den bei der Berechnung potentiell unterschiedlich auslegbaren Knotenparametern sowie den probabilistischen Effekten beim Startvorgang, etwa für unterschiedliche WakeUp-Initiatoren, detektierten Kommunikationsgeräuschen auf dem Bus oder auftretenden Weckkollisionen zwischen den FlexRay-Kommunikationscontrollern.

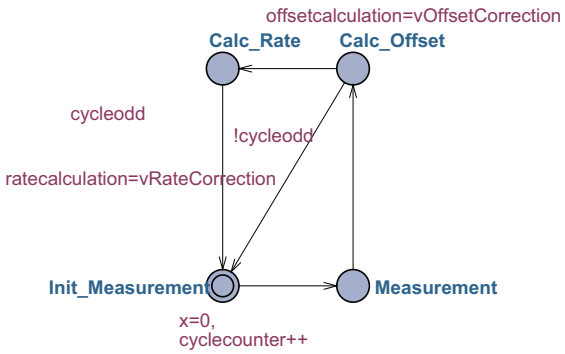


Abbildung 5.16.: Simulation des dynamischen Verhaltens des verteilten FlexRay-Synchronisationsprozesses zweier Netzknoten auf Basis gezeiteter Automaten

Bei der Umsetzung des Ablaufs der FlexRay-Protokollablaufkontrolle in zeitgesteuerte Automaten fällt auf, dass die semantischen Eigenschaften der Automaten im Bezug auf die verteilten zeitgesteuerten Architekturen mit Einschränkungen behaftet sind. So lassen sich die *Sync*-Transitionen zur Synchronisation mehrerer Teilnehmer nur an ausgewiesenen Stellen einsetzen, etwa bei der Auslösung eines WUP auf dem Bus, welches ein dritter Knoten unmittelbar empfängt und dabei im Ablauf mit dem Sender eine Synchronisationsschnittstelle bildet. Allerdings ist es kaum möglich die probabilistischen Effekte während des dynamischen Vorgangs im Detail umzusetzen. Als Beispiel ist die Phase zur Erkennung des exakten Zeitpunkts bei der Detektion eines WUP-Patterns

oder einer WUP-Kollision zu nennen. Dieser Problematik wird in dem Konzept durch die Annahme der maximalen Latenzen bei den jeweiligen Transitionen und der Analyse von tolerierbaren *Worst-Case*-Szenarien entgegengewirkt. Theoretisch mögliche Extremfälle, beispielsweise bei einem fehlerhaften permanenten Wechsel zwischen kurzzeitiger Busruhe und Busaktivität (Dekodierung von WUP-Symbolen), können zu arbiträren Iterationen innerhalb der Weckvorbereitungsphase führen, was den Weckvorgang gleichermaßen unbeschränkt verzögert.

5.3. Optimierung

Während der Entstehung dieser Arbeit sind in einer Studienarbeit /[Mai08]/ Methoden zur Optimierung der Nutzdateneffizienz (respektive des Nutzdatenvolumens) im statischen FlexRay-Segment und zur Auslegung des dynamischen FlexRay-Segments erarbeitet worden. Die Ergebnisse der Umsetzung und Analyse der Optimierungskonzepte mit der Werkzeugkette Matlab/Simulink /[The07a]/ werden in diesem Abschnitt zusammengefasst.

5.3.1. Statische Segmentauslegung (Nutzdateneffizienz)

Für die Formulierung der Optimierungsprobleme der Nutzdateneffizienz für ein FlexRay-System werden die Funktionale N_e und N_v definiert. Diese Funktionale haben die Eigenschaften, dass deren Werteverlauf mit Sprungstellen behaftet ist, wodurch mit einem der beiden untersuchten Optimierungsverfahren (nach *Kuhn-Tucker*) werkzeuggestützt kein Ergebnis gefunden wurde.

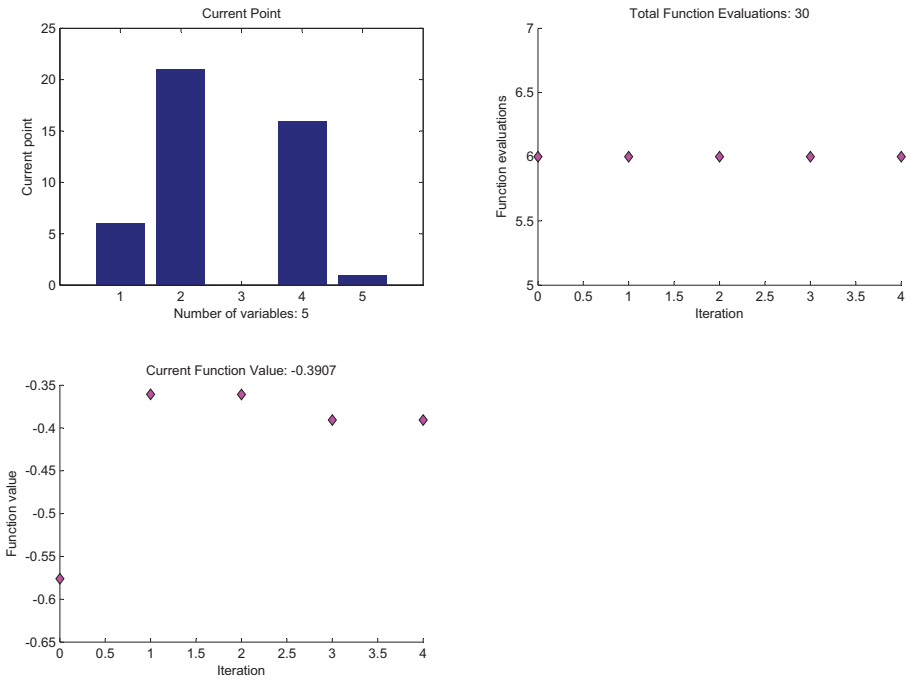


Abbildung 5.17.: Graphische Veranschaulichung des errechneten Lösungsvektors $x_{opt,1}$ mit der Optimierungstoolbox aus der Matlab/Simulink-Werkzeugkette

Im zweiten Ansatz wird der Optimierungsalgorithmus *fmincon* (s. Abb. 5.17) aus dem Optimierungswerkzeugsatz von Matlab/Simulink verwendet. Für die Untersuchungen wird ein Startvektor $x_{start,i}$ definiert, welcher als Eingabevektor in *fmincon* den optimierten Ausgabevektor $x_{opt,i}$ erzeugt. Der Parameter i beschreibt die jeweilige Iterationsstufe der sequentiellen Optimierungsläufe.

$$x_{start,i} = \begin{pmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ x_{i4} \\ x_{i7} \end{pmatrix} \overset{i=1}{\rightarrow} \begin{pmatrix} 3 \\ 18 \\ 0.0125 \\ 1 \\ 1 \end{pmatrix} \overset{i=2}{\rightarrow} \begin{pmatrix} 4 \\ 2 \\ 0.04 \\ 15 \\ 6 \end{pmatrix} \overset{i=3}{\rightarrow} \begin{pmatrix} 3 \\ 50 \\ 0.0125 \\ 3 \\ 1 \end{pmatrix}$$

$$x_{opt,i} = \begin{pmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ x_{i4} \\ x_{i7} \end{pmatrix} \overset{i=1}{\rightarrow} \begin{pmatrix} 6.0 \\ 21.0 \\ 0.0125 \\ 16.0 \\ 1.0 \end{pmatrix} \overset{i=2}{\rightarrow} \begin{pmatrix} 6.0 \\ 21.0 \\ 0.0125 \\ 5.0 \\ 1.0 \end{pmatrix} \overset{i=3}{\rightarrow} \begin{pmatrix} 4 \\ 21 \\ 0.05 \\ 2 \\ 4 \end{pmatrix} \overset{i=4}{\rightarrow} \begin{pmatrix} 6.0 \\ 127 \\ 0.0125 \\ 5 \\ 1 \end{pmatrix}$$

Dabei fällt auf, dass bei der Umsetzung benutzerspezifische Verfeinerungen des Startvektors benötigt werden oder manche Restriktionen nicht zwangsläufig eingehalten werden müssen. Beispielsweise kann das Restriktionsfunktional $g_6(x)$ unberücksichtigt bleiben, falls kein optionaler zeitlicher Sicherheitsabstand (*safety margin*) zur *Cliquenvermeidung* beim FlexRay-StartUp in der Systemparametrierung addiert werden soll.

Bei der Modifikation des Startvektors im Bereich der *Abtastperiode* und des *Makroticks* ergeben sich mit x_{i4} und x_{i7} in $x(opt, 3)$ bessere Werte für $N_e(x)$, was allerdings zu Lasten der maximalen Baudrate und damit des verfügbaren Nutzdatenvolumens $N_v(x)$ im System geht. Aus diesem Schritt leitet sich ab, dass die buszyklusbezogene Nutzdateneffizienz nicht strikt mit der höchsten Baudrate (10Mbit/s) erreicht wird. Als weitere Option lässt sich die potentielle Nutzdatenlänge auf das Maximum von 127 Bytewörtern erhöhen, dessen Wert im finalen Lösungsvektor $x(opt, 4)$ auch bestätigt wird. Der eingeschränkten Skalierbarkeit des statischen Segments durch die maximal umsetzbaren Botschaftsgrößen mit der Konsequenz einer minimalen Slotanzahl muss daher durch E/E-Architekturspezifische Anforderungen einer Mindestsendeslotanzahl mit Zusatzrestriktionen entgegengewirkt werden[4].

$$N_e(x_{opt,i}) \overset{i=1}{\rightarrow} 39,1\% \overset{i=2}{\rightarrow} 52,5\% \overset{i=3}{\rightarrow} 58,9\% \overset{i=4}{\rightarrow} 73,4\%$$

Bei der Betrachtung der erzielten Nutzdateneffizienz ergeben die Analysen mit *fmincon* eine zulässige Steigerung um 34,3%, was einer Erhöhung des ersten ermittelten Nutzdateneffizienzwerts $N_e(x_{opt,1})$ um 87,7% entspricht.

5.3.2. Dynamische Segmentgröße

Im Abs. 4.6.4.2 wird die Theorie zur Berechnung des dynamischen FlexRay-Segments auf Basis einer empirischen Erfassung des Datendurchsatzes pro Zeitintervall im Fahrzeug mit einem probabilistischen Ansatz dargestellt. Zur Veranschaulichung wird angenommen, dass 3 Steuergeräte in einem FlexRay-Cluster im dynamischen Segment

[4]Beispielsweise lassen sich aus der Funktionsnetznotation und der Partitionierung der logischen auf die technische Architektur Rückschlüsse auf die Mindestsendeslotanzahl ziehen

kommunizieren. Dabei sendet jedes Steuergerät maximal 5 Botschaften pro Buszyklus. Die Anzahl der maximal auftretenden Botschaften im dynamischen Segment entspricht folglich $\psi = 15$. Durch Einsetzen in die Gleichungen für β_{11} und ρ ergibt sich dabei

- eine normalverteilte Nutzdatenlänge $\beta_{11} = 90$ [Bit], was einem zeitlichen Wert $\beta_{11} = 9$ [μs] entspricht,
- eine Anzahl $\rho = 3$ [Minislot] $= 3 * x_{12} * x_7$ [μs] $= 36$ [μs] an Minislots, die pro Botschaft benötigt werden.

Dadurch wird für eine im dynamischen Segment verschickte Botschaft mit einem Nutzdatenanteil von $\beta_{11} = 9$ [μs] die entsprechende zeitliche Belegung $\rho = 3$ [Minislots] $= 3 * x_{12} * x_7$ [μs] $= 36$ [μs] im Buszyklus erwartet. Konsequenterweise müssen clusterweit mindestens $\alpha_2 = \psi * \rho = 15 * 3 = 45$ [Minislot] Minislots pro Buszyklus im dynamischen Segment bei der Systemparametrierung einkalkuliert werden.

5.3.3. Buszyklusauslegung (Nutzdatenvolumen)

Durch die Betrachtung des statischen Segments bei der Optimierung der Nutzdateneffizienz $N_e(x)$ und der separaten Analyse des dynamischen Segments lassen sich die Eingangsparameter des Funktionals zur Nutzdatenvolumenberechnung weitestgehend eliminieren (s. Tab. 5.1).

Parameter	Wert
x_1	6.0
x_2	127.0
x_3	0.0125
x_4	5.0
x_7	1.0
x_{18}	5.0
α_2	45.0
β_{11}	9.0
β_{13}	1.0
β_{15}	12.0

Tabelle 5.1.: Übersicht der aus Abs. 5.3.1 und Abs. 5.3.2 bestimmten Parameterwerte

Dadurch ergibt sich die lineare Zielfunktion $N_v(x_9, x_{10})$:

$$N_v(x_9, x_{10}) = \frac{1016}{1385} * (x_9 - x_{10}) - \frac{109728}{277}$$

In /[Mai08]/ wird $N_v(x_9, x_{10})$ mit den teilweise nichtlinearen Komponentenfunktionalen der zugeordneten Restriktionsabbildung $G_v(x)$ als nichtlineares Optimierungsproblem im *Solver fmincon* gelöst. Im numerischen Lösungverfahren wird die in der Restriktion $g_8(x)$ auftretende Buszykluszeit auf 5000 μs begrenzt und folgender Startvektor gewählt:

$$x_{start} = \begin{pmatrix} 1000 \\ 100 \end{pmatrix}$$

Daraus ergibt sich folgender Lösungsvektor:

$$x_{opt} = \begin{pmatrix} 5000.0 \\ 14.0 \end{pmatrix}$$

x_{opt} liegt auf dem Rand des gültigen Wertebereichs und ergibt für das modifizierte Nutzdatenvolumenfunktional einen Wert $N_v(x_{opt}) = 3261.5\ [\mu s]$. Da sich in den vorhergehenden Optimierungsschritten bereits ein Makrotick $x_7 = 1\ \mu$s ergeben hat, entspricht die optimierte Zykluszeit $x_9 = 5000\ [MT]$ der maximal zulässigen Vorgabe von 5000 μs. Die Netzwerkruhephase im Buszyklus zur Synchronisationskorrektur wird mit mindestens $x_{10} = 14.0$ [MT] berechnet[5].

5.4. Fallstudie

Begleitend zur Entwicklung des FlexZOOMED-Ansatzes wird die Tragfähigkeit des Konzepts durch eine Fallstudie mit einem eigens aufgebauten E/E-Architekturmodell untersucht. Die Inhalte der entwickelten Modelle orientieren sich pragmatisch an den Erkenntnissen, die sich aus der Mitarbeit an serienvorentwickelnden und -begleitenden Themen ergeben. Sämtliche Inhalte sind daher seriennah angelegt, allerdings aus Wettbewerbsgründen ohne direkten Zusammenhang zu einer real umgesetzten E/E-Architektur eines Fahrzeugherstellers.

5.4.1. Modellübersicht

Die erstellten Modelle erstrecken sich über sämtliche Bereiche des in Abs. 5.1 vorgestellten Validierungskonzepts. Die Grundidee basiert auf der durchgängigen Abbildung zweier Features eines Bremsassistenten auf eine logische E/E-Architektur in Form eines einfachen Funktionsnetzwerks aus dem Bereich der Fahrwerks- und Antriebssysteme. Die logische Architektur wird fahrzeugvariantenabhängig auf unterschiedlich dimensionierte technische FlexRay-Architekturen partitioniert. Ziel dabei ist die architekturvariantenspezifische Erfassung der FlexRay-Parametrierung. Die Erkenntnisse bei der Umsetzung und Anwendung der Fallstudie werden anfolgend zusammengefasst.

5.4.1.1. E/E-Architekturmodell

Die den Modellen zugrunde gelegte E/E-Architektur umfasst alle in Abs. 5.1 dargestellten Modellierungsebenen. Die einkanalige FlexRay-Architektur bildet dabei die Infrastruktur zur Vernetzung von neun Steuergeräten aus dem Antriebs-, Fahrwerks- und Fahrerassistenzbereich. Es werden dabei variantenabhängig verschiedene Funktionspartitionierungen umgesetzt, die auf unterschiedlichen Vernetzungstopologien beruhen (*Bus-, Stern-, Doppelsterntopologie*). Für jede Baureihe werden indidividuelle *Package*-Anforderungen berücksichtigt. Tab. 5.2 gibt einen Überblick über die Größen der relevanten Modellelementmengen im E/E-Architekturmodell[6].

[5] In Abs. A.3 wird das Nutzdatenvolumenfunktional dargestellt.

[6] Die quantitativ größeren Mengen an *Lowlevel*-Modellelementen, beispielsweise die Hardwarekomponenten, werden dabei nicht erfasst.

Logische Architektur				
|FC|	|SRC|	|PO|	|SP|	|RP|
15	75	150	60	90
Technische Architektur				
|ECU|	|PC|, |NC|	|CL|	|WS|	|PP|
9	9	16	30	18

Tabelle 5.2.: Kardinalitäten der Modellelemente auf logischer und technischer Architekturebene

5.4.1.2. Variantenmanagement

Für die Generierung von Architekturvarianten werden im ersten Schritt alle Artefakte eines Modells in disjunkte Mengen gruppiert, die einer arbiträr definierbaren Klassifikation angehören, beispielsweise als Gruppen der *ESP-Bauteile* oder des *Bremsassistenten*. Vereinzelte Elemente der Architektur lassen sich dabei allerdings nicht eindeutig einer Gruppe zuordnen, was zu potentiellen Inkonsistenzen im Modell führen kann und fallspezifisch behoben werden muss. Exemplarisch lässt sich die Zuordnung konventionell angebundener Sensoren in der Architektur anführen. Ein Gierratensensor lässt sich einerseits der Gruppe *ESP-Elemente* als zugehöriges Teil des ESP-Steuergeräts zuordnen. Andererseits ist der Gierratensensor auch ein Bestandteil der Gruppe *Dezentrale Sensorik*, falls in der Architektur kein zentrales Sensorcluster verbaut sein sollte.

Die entwickelten Modellgruppen werden den verschiedenen Konzeptvorlagen zugeordnet. Entsprechend wäre die Zuordnung einer Modellgruppe *FlexRay-Vernetzung* zu einer Vorlage *Fahrwerksysteme* plausibel, falls diese auf den Inhalt der Modellgruppe, einem FlexRay-Cluster, aufbaut. In dieser Fallstudie werden folgende Konzeptvorlagen definiert (s. Tab. 5.3):

Konzeptvorlage: Softwarefunktionen			
Integriert	Bremsen	Fahrwerk	Traktionsmanagement
Konzeptvorlage: Sensorik			
High-End-Cluster	Low-End-Cluster	Dezentral	-
Konzeptvorlage: Energieversorgung			
DC/DC-Wandler	Unabhängig	-	-
Konzeptvorlage: Steuergeräte			
Vollständig	Gruppe A	Gruppe B	Gruppe C
Konzeptvorlage: Vernetzungstopologie			
8-fach Stern	4-fach Stern	Passiver Bus	-
Konzeptvorlage: Halbleiterbauteile			
HQ-Quarz	HQ-Quarz (ink. PLL)	LQ-Quarz	LQ-Quarz (ink. PLL)
Konzeptvorlage: Standardsoftware			
Zulieferer A	Zulieferer B	Proprietär	-
Konzeptvorlage: Verkabelung			
Klemmenkonzept A	Klemmenkonzept B	-	-

Tabelle 5.3.: E/E-Konzeptvorlagen und -instanzen des Variantenmanagements

Softwarefunktionen: Diese Gruppe zielt auf die Bündelung von Strukturen aus der logischen Architektursicht. Dabei werden neben den Funktionsblöcken auch deren Sende- und Empfangsports sowie verbundene Konnektoren zwischen den Ports miteinbezogen.

Senorik: Auf Seite der technischen Architekturentwicklung können Sensoren unterschiedlich angebunden sein. Es wird dabei zwischen einer rein dezentralen Anbindung der Sensoren an verschiedene Steuergeräte und einer Bündelung innerhalb einer Einheit

als Sensorcluster unterschieden. Die Größe des Sensorclusters lässt sich in verschiedene Ausbaustufen einteilen.

Energieversorgung: Bei der Energieversorgung ergibt sich eine Differenzierung zwischen Komponenten, welche über die Batteriespannung ohne oder mit dazwischengeschalteten Gleichstromkonverter (DC/DC-Wandler) betrieben werden. Dieser Aspekt wirkt sich bei Spannungseinbrüchen im Bordnetz aus, bei dem nicht DC/DC-Wandlergestützte Komponenten für eine kurze Dauer ausfallen können.

Steuergeräte: Aus technischer Sicht ist eine Gruppierung auf Steuergeräteebene sinnvoll. Beispielsweise wird in dieser Fallstudie ein sekundärer ACC-Sensor stets nur im Verbund mit einem primären Radarsensor verbaut.

Vernetzungstopologie: Aufgrund der physikalischer Eigenschaften und erhöhten Kosten muss der Einsatz aktiver Sternkoppler im Fahrzeug genau geplant werden. Daher wird zwischen passiven Bustopologien, 4-fach- und 8-fach-Sterntopologien unterschieden.

Halbleiterbauteile: Die in der Architektur eingesetzten Halbleiterbauteile lassen sich je nach Kostenanforderungen in unterschiedliche Qualitätsstufen untergliedern. In der betrachteten Fallstudie fokussiert dieser Bereich auf die Eigenschaften der Quarze für die Takterzeugung in den FlexRay-Steuergeräten, wobei zwischen hochqualitativen Quarzen und Standardquarzen (HQ und LQ) inklusive oder ohne Verwendung von PLLs differenziert wird.

Standardsoftware: Eine detaillierte Modellierung und Spezifikation der Eigenschaften von Standardsoftwaremodulen eines FlexRay-Steuergeräts liegt außerhalb des Fokus dieser Analyse. Daher beschränkt sich das Konzept der Standardsoftware auf eine abstrakte Unterscheidung der erreichten Sende- und Empfangslatenzen bei der Netzwerkkommunikation, die auf die Implementierungseigenschaften der Basissoftware in den Steuergeräten zurückgeführt werden kann, beispielsweise die Dauer für die Abarbeitung eines Sendeaufrufs im Bustreiber. Zur Vereinfachung wird die Standardsoftware je nach Zulieferer oder Eigenentwicklung mit jeweils einheitlichen Werten für das Sendezeitverhalten attributiert.

Verkabelung: Bei der Verkabelung interessieren vorrangig die Versorgungsklemmen der verschiedenen Steuergeräte. Durch die Klassifizierung der FlexRay-Busteilnehmer in Sync- und Non-Sync-Knoten muss gewährleistet bleiben, dass je nach Konzept die FlexRay-Kommunikation gegebenenfalls bereits im Vorlauf eines Motorstarts im Fahrzeug etabliert werden kann (*Klemme-30-Steuergeräte*).

Parallel zu der Entwicklung und Befüllung der Konzeptvorlagen (vgl. Tab. 5.4) werden die entwickelten Ausstattungsvorlagen den Fahrzeugbaureihen sowie deren Ausstattungsvarianten zugeordnet. Die Fallstudie setzt sich dabei aus den drei Fahrzeugbaureihen *Heckmotor-Sportwagen, Mittelmotor-Sportwagen und sportlicher Geländewagen (SUV)* zusammen, für die jeweils zwei Ausstattungspakete aus den Bereichen (*Basis, Komfort, Sport, Premium*) definiert werden. In Tab. 5.4 wird die Zuordnung der Konzepte zu den Baureihen zur Erzeugung sechs unterschiedlicher Architekturvarianten dargestellt.

Sportwagen (HM) (Basis)	Sportwagen (HM) (Premium)	Sportwagen (MM) (Sport)	Sportwagen (MM) (Premium)	SUV (Komfort)	SUV (Sport)
Fahrwerk	Integriert	Fahrwerk	Integriert	Bremsen	Traktions-management
Dezentral	Low-End-Cluster	Low-End-Cluster	High-End-Cluster	Low-End-Cluster	Low-End-Cluster
Unabhängig	Unabhängig	Unabhängig	Unabhängig	DC/DC-Wandler	Unabhängig
Gruppe B	Vollständig	Gruppe B	Vollständig	Gruppe A	Gruppe B
Passiver Bus	4-fach Stern	4-fach Stern	8-fach Stern	4-fach Stern	4-fach Stern
LQ-Quarz (ink.PLL)	HQ-Quarz (ink.PLL)	LQ-Quarz	Plattformquarz	HQ-Quarz (ink.PLL)	HQ-Quarz
Zulieferer B	Zulieferer B	Zulieferer A	Zulieferer A	Zulieferer B	Zulieferer A
Klemmen-konzept B	Klemmen-konzept B	Klemmen-konzept B	Klemmen-konzept A	Klemmen-konzept A	Klemmen-konzept B

Tabelle 5.4.: Variantenerzeugung für die drei Fahrzeugderivate auf Basis zweier Ausstattungspakete pro Fahrzeugklasse

5.4.2. Modellanalyse

Die Untersuchung der spezifizierten Architekturvarianten basiert auf dem in Abs. 5.1.3 erläuterten Parametrierungseditor. Für den Zugriff auf die Architekturvarianten dienen dabei Modellstrukturen, die regelbasiert innerhalb des Gesamtmodells identifiziert werden. Um die Anzahl der Modellsuchregeln überschaubar zu gestalten, werden Basisregelstrukturen geschaffen, die bei Bedarf per *Join*-Verknüpfung transformiert werden können. Die Zusammenführung und Weiterverarbeitung der Eingangsdaten zur Ableitung der Architektureigenschaften durch Berechnung der konkreten FlexRay-Parameter und der entwickelten Qualitätsmerkmalen erfolgt innerhalb von 134 programmierten Berechnungsblöcken. Zur Analyse der erhobenen und berechneten Daten dient eine Vielzahl an Auswertungsstrukturen in Form von Warnschaltern und kontinuierlichen oder diskreten Wertebereichen. In Tab. 5.5 werden die Kennzahlen zum entwickelten Parametrierungsmodell zusammengefasst.

Modellsuchregeln	*Join*-Verknüpfungen	Berechnungsblöcke	Auswertungsstrukturen	LoC
33	13	134	253	5758

Tabelle 5.5.: Kennzahlen zum entwickelten FlexRay-Parametrierungsmodell

5.4.2.1. Basiseigenschaften

Bei der Analyse der Grundeigenschaften der aus den Fallstudienmodellen ermittelten FlexRay-Parameter fällt auf, dass sich die Architekturderivate am umfangreichsten im Bereich der Konfiguration des physikalischen Netzwerks und bei den erfassten Qualitätseigenschaften unterscheiden. Gleichzeitig ergeben sich Interferenzen bei vereinzelten Parametern bei der Mikro-/Makrotickskalierung (Abweichungen bei der Ausbreitungsverzögerung im System) oder der Uhrensynchronisation (unterschiedliche Anzahl verfügbarer Sync-Knoten).

Die jeweils zwei unterschiedlich ausgestatteten Fahrzeugderivate der drei untersuchten Baureihen weisen bereits für einheitliche Eingangsparameter (s. Abs. 4.6) unterschiedliche Ausprägungen im Bereich der ableitbaren FlexRay-Parameter auf. Auch die bei der Analyse der konzipierten Qualitätskriterien ermittelten Werte auf E/E-Architekturebene zeigen bereits Abweichungen zwischen den spezifizierten Varianten.

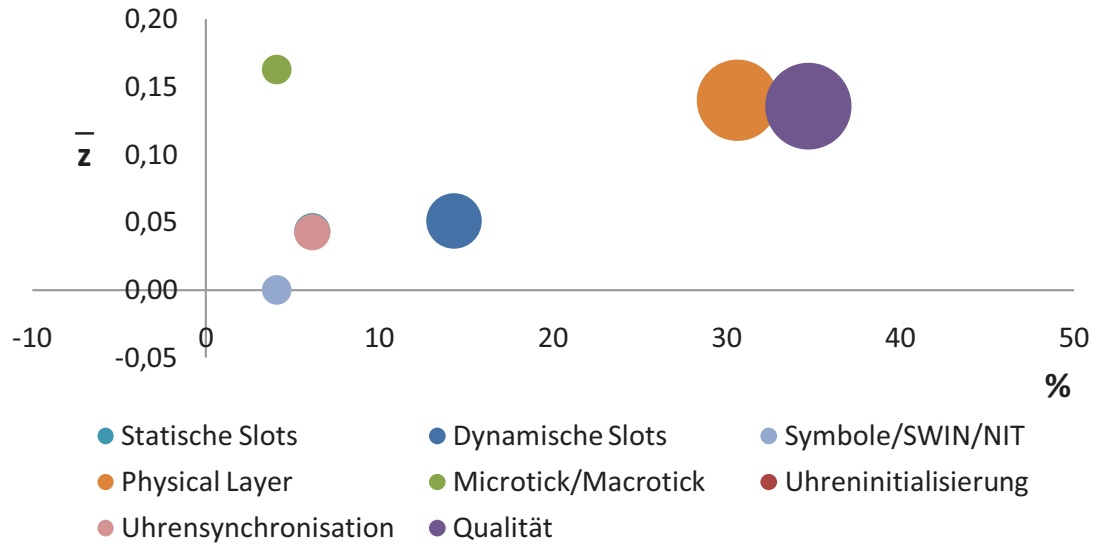


Abbildung 5.18.: Gegenüberstellung der prozentualen Verteilung der abweichenden FlexRay-Parameter nach Klassifikationsgruppe und der durchschnittlichen z-Standardabweichung

Abb. 5.22 visualisiert ausgewählte Parameter in einer Netzdiagrammdarstellung. Um die Ergebnisse in Relation setzen zu können werden die stark unterschiedlichen Wertebereiche der betrachteten Parameter (10^{-3}-10^{5}) mittels einer z-Standardisierung vergleichbar gemacht[7] (vgl. /[Nac08]/). Bei der Betrachtung des vollständigen Parametersatzes werden neben den notwendig konfigurierbaren Parametern auch die Kennzahlen für die Qualitätsanalyse, Minimal-/Durchschnitts-/Maximalwerte sowie Konstanten, die durch Variation zur Systemoptimierung beitragen, berücksichtigt. Dadurch ergeben sich die in Abb. 5.22 dargestellten Eigenschaften. In dem Zusammenhang lassen sich folgende Erkenntnisse feststellen:

Allgemeine Konfigurationsabhängigkeiten:

- (statische Slotlänge x_{19}, Minislotlänge x_8, physikalische Erweiterbarkeit) $\leftrightarrow$ Topologieart x_{67}) $\leftrightarrow$ Verkabelungsparameter,
- Maximale Sync-Knotenanzahl x_{29} $\leftrightarrow$ Anzahl verbauter Steuergeräte,
- Maximaler Initialisierungsfehler x_{36} $\leftrightarrow$ (angenommene Systempräzision x_{30}, maximale Ausbreitungsverzögerung x_6).

Informelle Erkenntnisse:

- Signalübertragungslatenzen variieren in den Empfangsknoten aufgrund unterschiedlicher Leiterplattenverzögerungen in den Steuergeräten,
- Signalverzögerungen in *DaisyChain*-angebundenen Steuergeräten variieren je nach Steckerspezifikation,

[7] *Anmerkung:* Die Streuung liegt in dem Zusammenhang im Spitzenbereich bei bis zu 70% des arithmetischen Mittelwerts für vereinzelte Parameter!

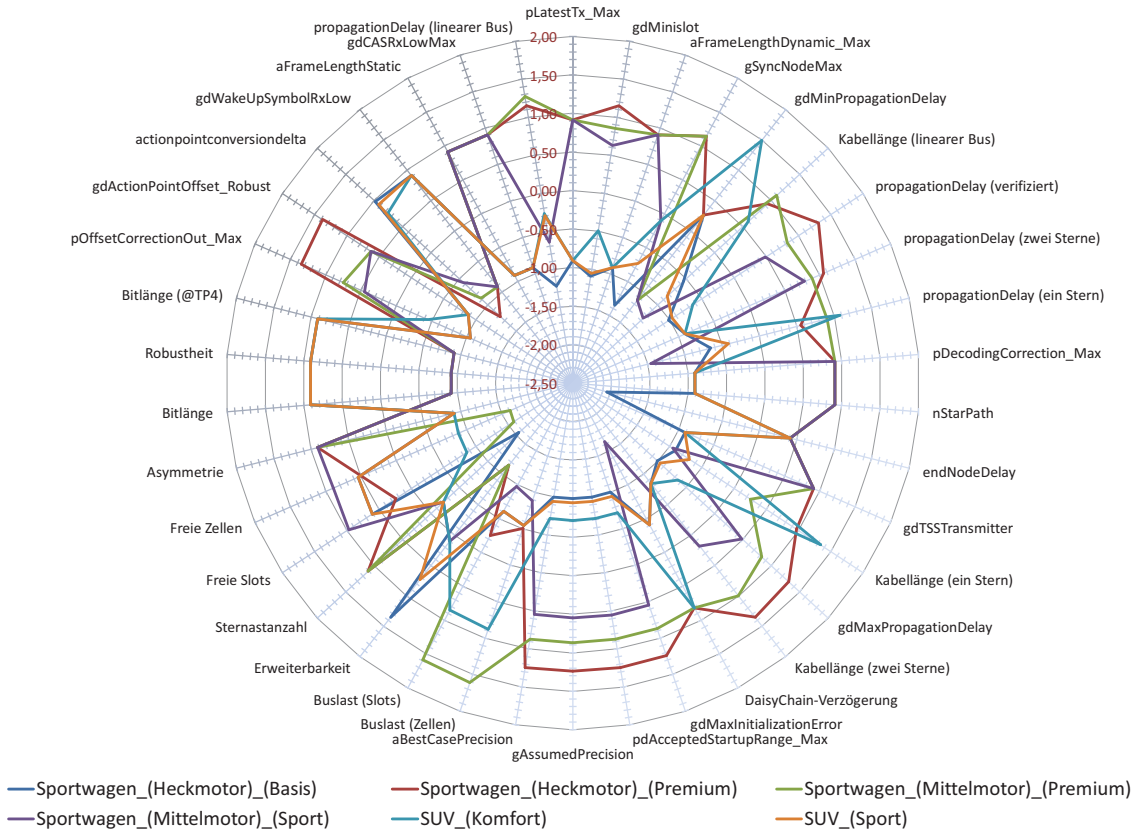


Abbildung 5.19.: Prozentuale Abweichungen zwischen ausgewählten FlexRay- und Qualitätsparametern für sechs untersuchte Fahrzeugederivate

- Bei den Derivaten mit einem aktiven Stern gibt es Bandbreitenverluste bei der Umrechnung auf den *ActionPoint* gegenüber den Architekturen mit zwei aktiven Sternen,
- Die maximale Ausbreitungsverzögerung, die Robustheit (Signalasymmetrie) und die Synchronisationskorrekturen werden stärker durch die Sternanzahl als durch Verkabelungseigenschaften beeinflusst,
- Die Sensordatenfusion (Sensorcluster und Umfelderkennung) trägt zu wesentlichen Buslastunterschieden bei,
- Unterschiede zwischen den Buslasten auf Slotbasis lassen sich nicht strikt auf Buslasten auf Zellenbasis im FlexRay-Buszyklus übertragen.

Ein Teilbereich der Modellanalyse konzentriert sich auf die Berechnung der physikalischen Parameter der drei FlexRay-Architekturvarianten zur Ermittlung der maximalen Ausbreitungsverzögerung $\alpha_1 = gdMaxPropagationDelay$, welche sich im Design des Schedules und bei der zulässigen Synchronisationskorrektur auf Basis der Systempräzision signifikant auswirkt. Dabei spielen die auftretenden Topologievarianten (*passiver Bus,*

aktiver Stern, zwei aktive Sterne) und die Hardwareeigenschaften (*Transceiver-, Quarzeigenschaften*) eine wichtige Rolle.

5.4.2.2. Topologien

In dieser Betrachtung stehen Modifikationen im Bereich der Fahrzeugtopologie der untersuchten Fahrzeugderivate im Vordergrund. Dabei werden folgenden Vorbedingungen angenommen.

- Die Anzahl statischer Kommunikationsslots wird architekturübergreifend festgelegt,
- Der Signalaufbau der verwendeten Transceiver ist architekturübergreifend identisch.

Während der Fahrzeugentwicklung ergeben sich bei der Definition des Kabelbaums potentielle Variationen zwischen den einzelnen Segmenten des konzipierten physikalischen Bordnetzes (s. Abb. 5.20). In diesem Fall werden Modifikationen an der Länge einzelner Kabelsegmente sowie ein Wechsel des verwendeten Kabeltyps mit veränderten Signalverzögerungseigenschaften betrachtet[8].

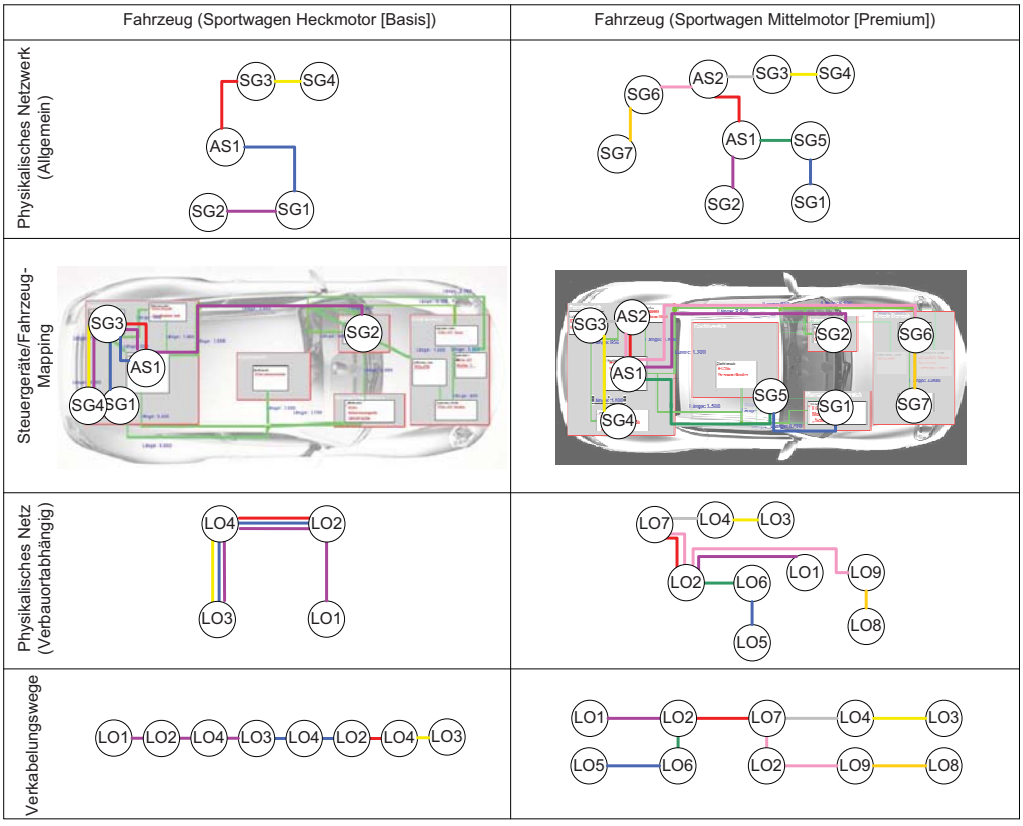


Abbildung 5.20.: Analyse der physikalischen Bordnetzabmessungen auf Basis des Steuergeräte/-Fahrzeug-Mappings

[8]Dämpfungseigenschaften des Kabels und Temperatureinflüsse werden in diesem Zusammenhang nicht weiter berücksichtigt.

1. Kabellängenmodifikation: In dieser Untersuchung werden sechs arbiträre Elemente zur Verlegung der Kabelbaumelemente im Fahrzeug modifiziert, wobei drei Elemente verkürzt und drei weitere Elemente verlängert werden (s. Abb. 5.21).

Topologiesegment	Längenmodifikation (mm)	Fahrzeugbaureihen [Varianten]
FlexRay Star 4a	2400 → 1700	(Sportwagen HM [Basis]/HM [Premium])
FlexRay Star 4b	2500 → 4700	(Sportwagen MM [Premium, Sport])
FlexRay Star 6	5200 → 6100	(Sportwagen HM [Premium]/MM [Premium], SUV[Komfort])
FlexRay Star 7	4200 → 3000	(Sportwagen HM [Basis])
FlexRay Star 8	1000 → 2000	(Sportwagen HM [Basis]/MM [Premium])
FlexRay Star 9	1400 → 600	(Sportwagen HM [Premium], SUV [Sport])

Tabelle 5.6.: Übersicht zu den baureihen- und variantenabhängigen Modifikationen am Kabelbaum

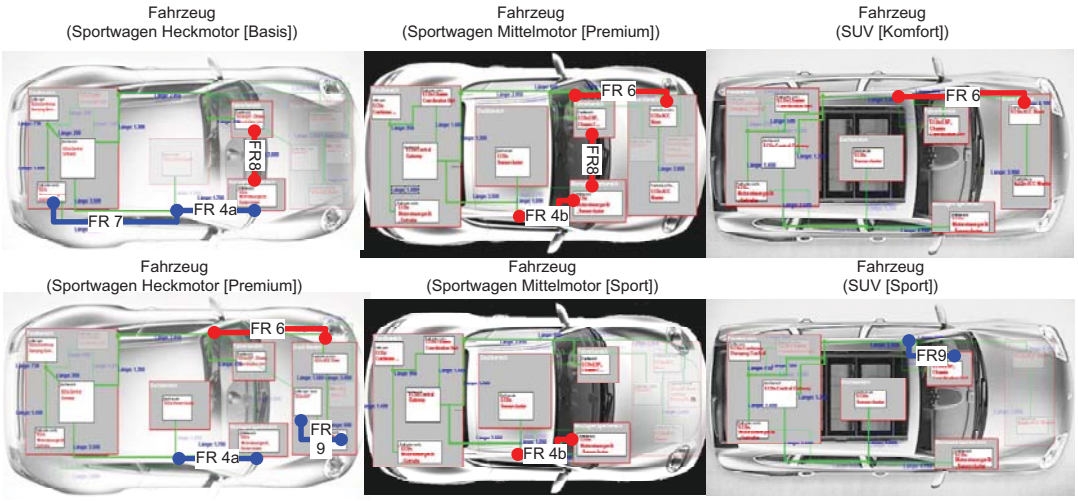


Abbildung 5.21.: Topologieschaubilder zu den baureihen- und variantenabhängigen Modifikationen am Kabelbaum

2. Kabeltypmodifikation: Im Sinne einer Kostenoptimierung wird ein Wechsel auf einen günstigeren Kabeltyp mit einem erhöhten Signalverzögerungswert ($0,1ns/m \rightarrow 0,5ns/m$) betrachtet.

3. Steuergerätehinzunahme (Mittelknoten): Einfügen eines Steuergeräts zur Dämpferregelung als Mittelknoten auf einem Sternast, beispielsweise zwischen dem Gatewaysteuergerät als zentralen Sternkoppler und dem Getriebesteuergerät als Endknoten in der Baureihe *Sportwagen HM* mit der Basisausstattung. Dieser Eingriff wirkt sich stark auf die Performanz- und Robustheitswerte des Clusters aus. Die statischen Slotgröße erhöht sich um ca. *7%* und die Minislotgröße des dynamischen Segments um *15%*, welches zum Teil durch die deutlich schlechtere Actionpointskalierung und dem daraus resultierenden größeren Actionpoint (*+23%*) bedingt ist. Aber auch die um *1,5m* und damit *13,5%* längere Verkabelung des Clusters in Sterntopologie trägt zur schlechteren Performanz bei. Die Gesamtpräzision erhöht sich deutlich um *7,7%*, was letztendlich zu einer um *6,3%* niedrigeren statischen Nutzdateneffizienz führt mit einer Absenkung der statischen Slotanzahl von

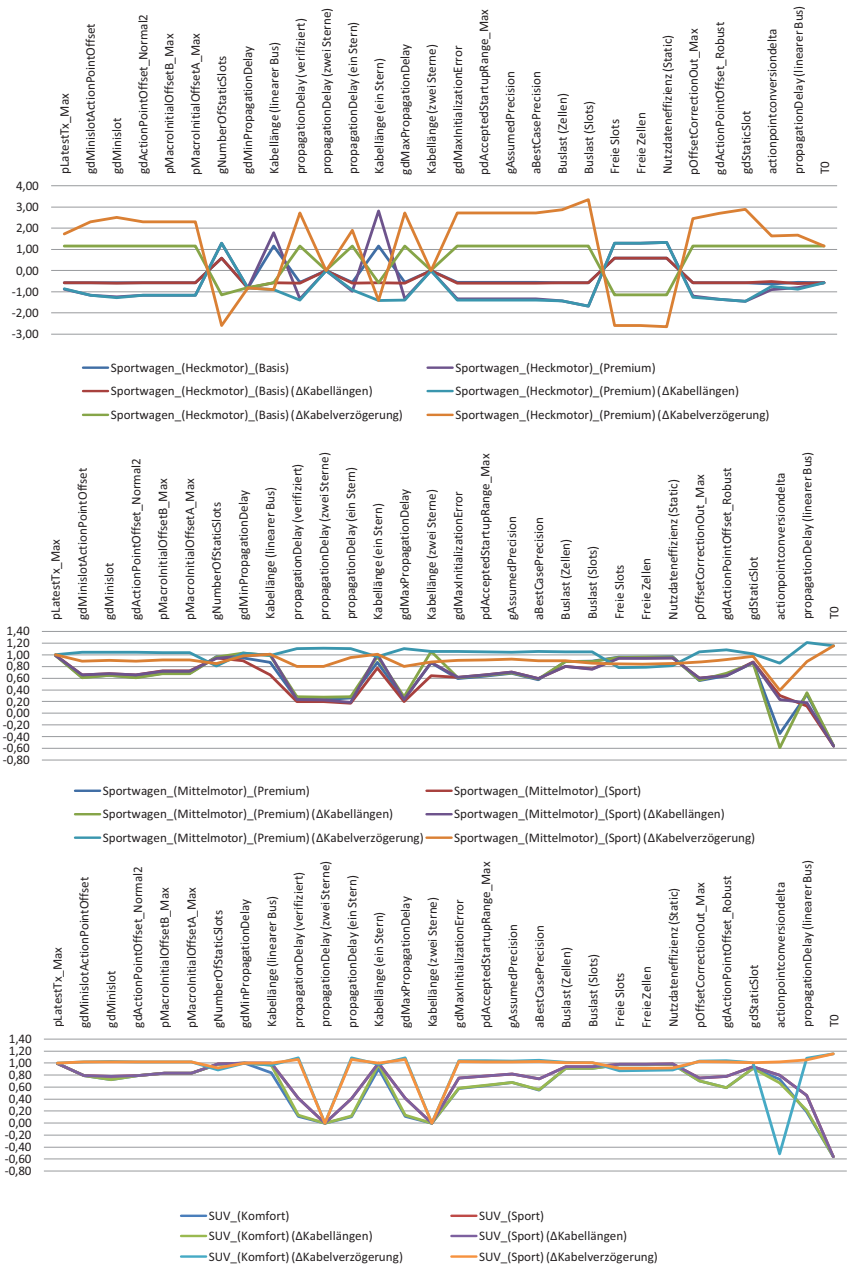


Abbildung 5.22.: Variantenabhängige, BR-bezogene standardisierte Abweichungen ausgewählter FlexRay-Parameter bei Modifikationen am physikalischen Bordnetz

121 auf *113*. Dementsprechend steigt die Buslast neben den zu versendenden Botschaften des Dämpfersteuergeräts um 7% auf Zellen- und Slotbasis an. Da sich die asymmetrische Bitverkürzung um *8%* aufgrund der Wandlung des *p2p*-Sternastes zu einem zusätzlichen linearen passiven Bus erhöht, sinkt die Systemrobustheit um *8,9%* ab. Da aber in diesem Bereich keine negativen Werte erreicht werden, kann die entstehende Sterntopologie mit zwei linearen passiven Bussen ohne Bedenken eingesetzt werden.

4. Steuergerätehinzunahme (Endknoten): Hinzunahme eines einzelnen ACC-Steuer-

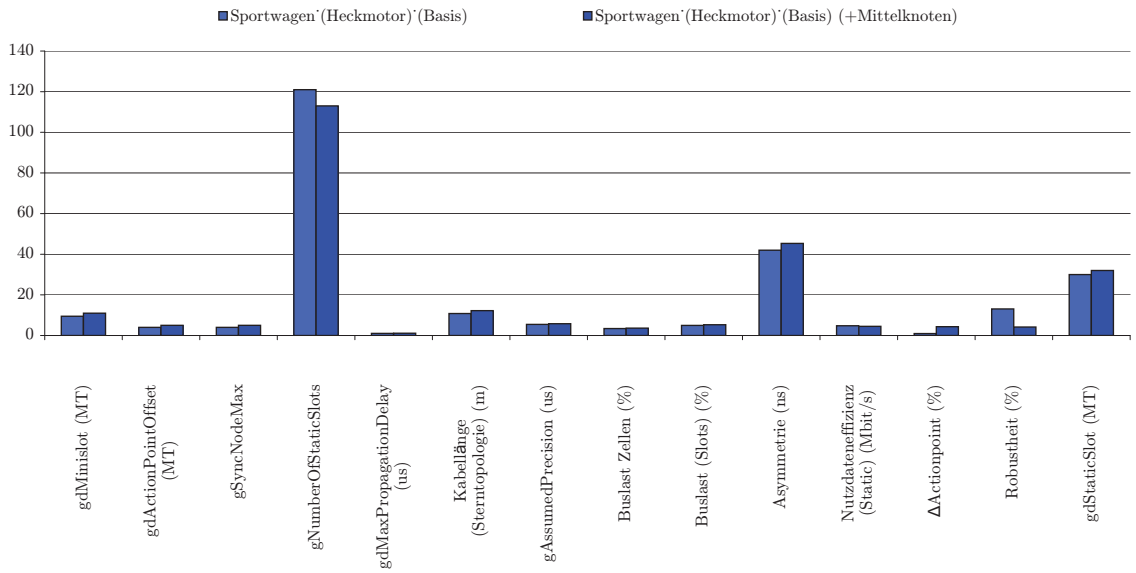


Abbildung 5.23.: Auswirkung der Integration eines DaisyChain-Mittelknotens zwischen Sternkoppler und Endknoten auf den FlexRay-Parametersatz in einer Architekturvariante

geräts für die Baureihe *SUV* in der Ausstattungsvariante *Sport* und Anhängen des Knotens an einen der bisherigen Endknoten in der Topologie, der ohne Mittelknoten mit dem zentralen Sternkoppler verbunden ist, beispielsweise das ESP-Steuergerät. Diese Ergänzung wirkt sich in diesem Fall nicht auf den FlexRay-Parametersatz aus, da die zusätzliche Verkabelung und das hinzugefügte Steuergerät die vorliegende Doppelsterntopologie nicht modifiziert und auch die maximale Kabellänge zwischen den am weitesten voneinander entfernten Endknoten nicht weiter erhöht wird (*Getriebe, Motorsteuergerät*).

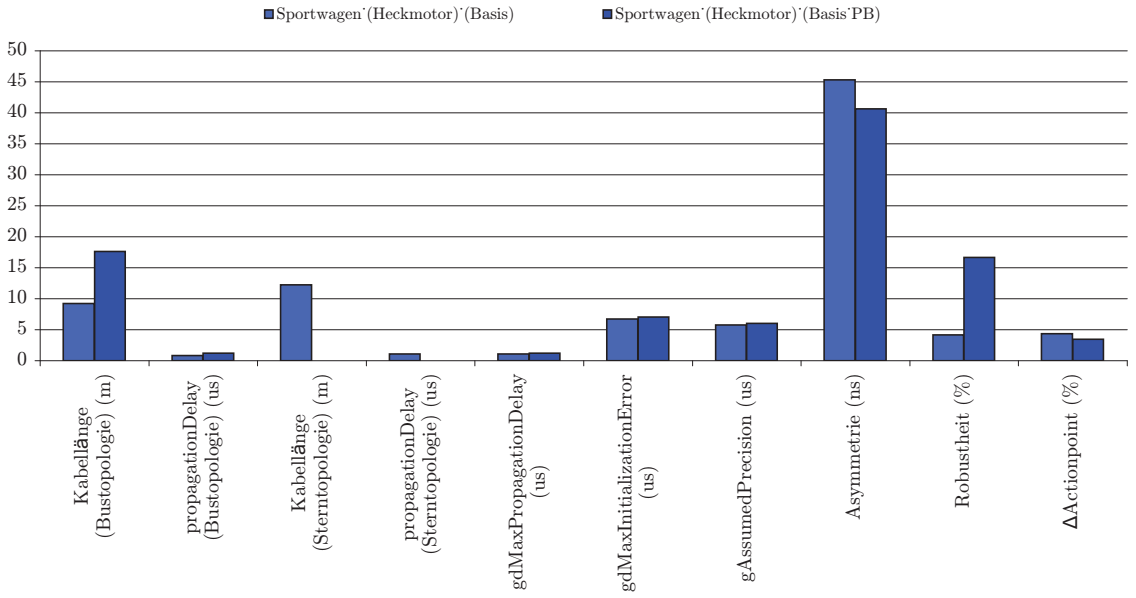


Abbildung 5.24.: Veränderungen im Parametersatz bei der Eliminierung des Sternkopplers in einer Architekturvariante

5. Eliminierung des aktiven Sternkopplers: Reduzierung eines Sternasts mit den beiden Steuergeräten für das Getriebe und die Dämpferregelung in der Baureihe *Sportwagen HM* mit der Basisausstattung. Durch die Eingliederung der beiden Knoten auf dem verbliebenen Sternast entsteht aus der vorliegenden aktiven Sterntopologie ein linearer passiver Bus mit vier Teilnehmern. Der Topologiewechsel bewirkt eine deutliche Vergrösserung der längsten zusammenhängenden Busstruktur um *8,4m* (*91,6%*) (s. Abb. 5.24). Die Kabellänge der passiven Bustopologie übersteigt somit die maximale Kabellänge der einfachen Sterntopologie um *5,4m*. Dadurch erhöht sich die im System auftretende gesamte Ausbreitungsverzögerung um weitere *10,7%* trotz des Wegfalls des aktiven Sternkopplers. Allerdings verbessert sich das System ohne den Sternkoppler im Bereich der asymmetrischen Bitverkürzung um *4,7ns* (*10,3%*). Dadurch lässt sich die Robustheit deutlich erhöhen bei einer sich nur minimal verschlechternden Actionpointübersetzung.

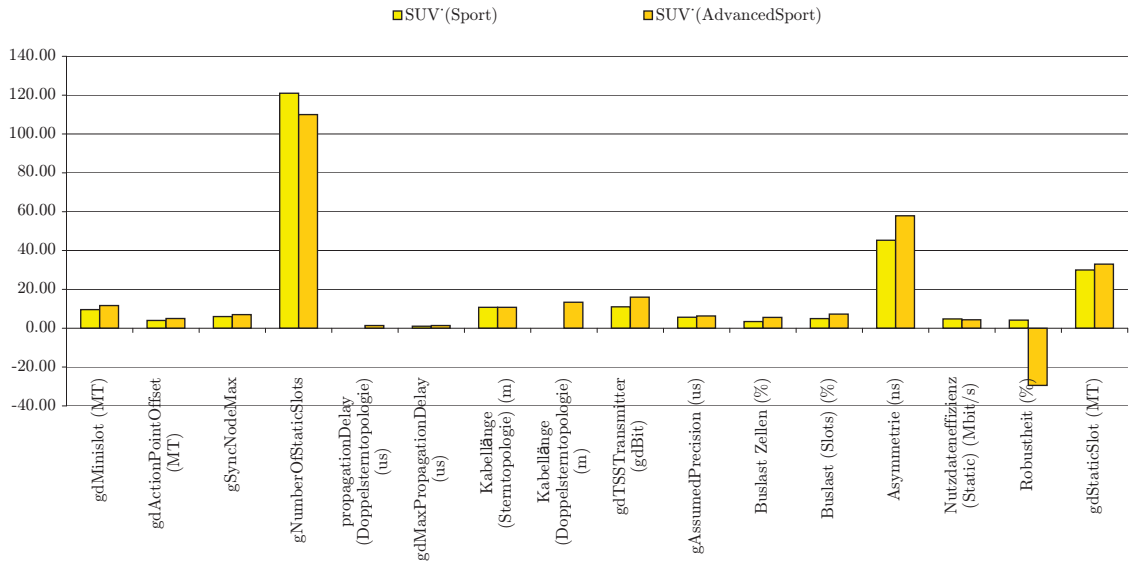


Abbildung 5.25.: Einflüsse der Integration eines zweiten FlexRay-Sternkopplers auf den Parametersatz einer Architekturvariante

6. Integration eines zusätzlichen Sternkopplers: Erweiterung der existierenden Ausstattungsvariante *Sport* der Baureihe *SUV* um ein zentrales Steuergerät zur Fahrwerkskoordination. Dieses Steuergerät ist standardmäßig als aktiver Sternkoppler ausgelegt. Mit diesem Schritt wird die Netzwerktopologie von einer einfachen zur einer doppelten Sterntopologie umkonfiguriert. Diese Veränderung an der Topologie bewirkt eine Reihe an Änderungen im FlexRay-Parametersatz. So erfordert die um *32%* höhere Ausbreitungsverzögerung einen *25%* längeren Actionpoint und führt daher zu *10%* größeren statischen Slots und *20%* größeren Minislots (s. Abb. 5.25). Dadurch senkt sich die Nutzdateneffzienz um *9,2%*, was einer Reduzierung der statischen Slots von *121* auf *119* entspricht. Anhand der zusätzlichen asymmetrischen Bitverkürzung (*27,8%*) und der daraus resultierenden Robustheitsverschlechterung von knapp *35%* lässt sich ableiten, dass eine Doppelsterntopologie keinesfalls die Anforderungen eines Serieneinsatzes in diesem Fall erfüllen kann.

5.4.2.3. Partitionierung

1. Funktionsverschiebung: In der Baureihe *Sportwagen MM* wird für die *Premium*-Ausstattungsvariante eine Funktion des ESP-Steuergeräts in den Fahrwerkskoordinator integriert. In diesem Zusammenhang werden Signale aus einer Botschaft des ESP-Steuergeräts in eine Botschaft des Fahrwerkskoordinators umpartitioniert (*Botschaft ESP*_01 $\rightarrow$ *Botschaft CCU*_02). Diese Veränderung bleibt ohne Auswirkungen am FlexRay-Parametersatz und am Busschedule, da die Botschaften *ESP*_01 und *CCU*_02 weiterhin mit identischen Zeitverhalten versendet werden und lediglich Signale innerhalb der Botschaften vertauscht worden sind.

2. Integration neuer Funktionalität: Zur Verbesserung der Fahrdynamik wird in der Baureihe *Sportwagen MM* für die *Sport*-Ausstattungsvariante eine elektrische Aktivlenkung konzipiert. Für die Umsetzung wird eine neue Funktion in die Architektur integriert, die im Zusammenspiel mit zwei weiteren bereits existierenden Funktionen (*FctActiveFrontSteering*, *FctChassisStabilization*) über zwei Empfangs- und einem Sendeport interagiert. Für die Umsetzung werden zwei Signale aus der Botschaft *CCU*_02 empfangen. Da die neue Funktion ins ESP-Steuergeräts partitioniert wird, wird eine neue Botschaft *ESP*_03 versendet. Zusätzlich wird eine zeitliche Adaption einer Botschaft des Motorsteuergeräts ($Botschaft\ Mot_1 \rightarrow 5ms$) erforderlich. Durch die logische Erweiterung wird der Parametersatz beibehalten, wobei sich die Kennzahlen für die Buslasten verändern. Dabei fällt auf, dass die Buslast auf Slotebene stärker ansteigt als auf Zellenebene (*2,7%* zu *1,0%*). Dies liegt daran, dass durch die Zykluszeitanpassung die Multiplexbotschaft des Motorsteuergeräts auf mehrere Slots verteilt werden muss, die sich allesamt allerdings nur partiell befüllen lassen.

3. Sensorclusterung: In der Baureihe *Sportwagen HM* wird für die *Basis*-Ausstattungsvariante ein Umstieg des dezentralen konventionellen Sensorikansatzes auf ein *Low-End*-Sensorcluster untersucht. Für dieses Vorhaben ersetzt ein eigenes am FlexRay-Bus hängendes Sensorcluster-Steuergerät die bisher verteilt angebundenen Beschleunigungs- und Ratensensoren. Dabei wird neben dem Steuergerät auch sein Transceiver und der Stecker für die neue Architekturvariante berücksichtigt. Die Anbindung erfolgt über jeweils zwei Pins zwischen Sensorcluster und Gatewaysteuergerät. Das verbindende Kabelsegment muss aufgrund der Partitionierung im Dachbereich des Fahrzeugs um zwei Topologiesegmente (*FlexRay_Star_1*, *FlexRay_Star_Kabel5b*) eine Länge von 5000mm besitzen. Auf der logischen Ebene lassen sich zwei Funktionen des Sensorclusters (*FctAccelerationSensorAnalysis*, *FctAngleSensorAnalysis*) verwenden, die über neun zusätzliche Sendeports Signale in vier zusätzlichen Botschaften mit spezifischen Zeitverhalten auf dem FlexRay-Bus verschicken. Gleichzeitig entfällt eine Botschaft des ESP-Steuergeräts, da deren Signale jetzt innerhalb der Sensorclusterbotschaften übertragen werden. In Abb. 5.26 werden die beiden Architekturkonzepte gegenübergestellt. Bei der Umstellung auf das Sensorcluster ergeben sich aufgrund des zusätzlichen Steuergeräts Veränderungen im Bereich der Kabellänge (*24%*), der Ausbreitungsverzögerung (*13%*) und damit auch bei der Systempräzision (*5%*). Die Anzahl der zur Verfügung stehenden statischen und dynamischen Slots bleibt durch die Veränderung unberührt, wobei die Buslast aufgrund der neuen Botschaften des Senorclusters um *1,8%* auf Zellenbasis und um *2,65%* auf Slotebene ansteigt.

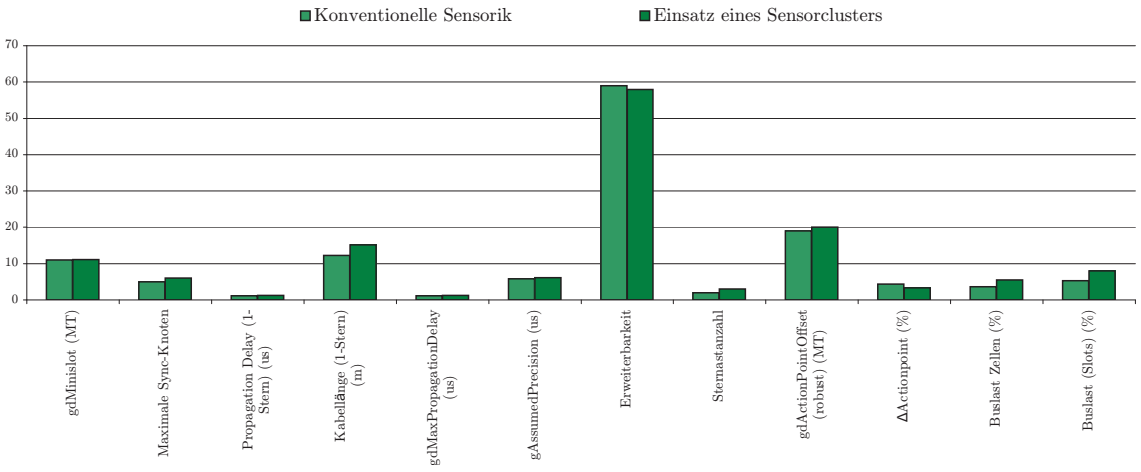


Abbildung 5.26.: Veränderungen im Parametersatz bei einer Modifikation des Sensorikkonzepts

5.4.2.4. Systemoptimierung

Neben der Untersuchung der physikalischen Eigenschaften der E/E-Architektur interessiert in der Fallstudie auch die Effizienz des Busses auf logischer Ebene. Da die verfügbare Nettodatenrate des Bussystems direkt mit der Flexibilität in der Auslegung des Bordnetzes korreliert, wird in diesem Abschnitt die vorher statisch festgelegte Slot-Anzahl spezifisch für jedes zur analysierende Fahrzeugderivat berechnet. Dabei wird vorausgesetzt, dass die Topologieabmessungen in der jeweiligen Ausprägung permanent gültig bleiben.

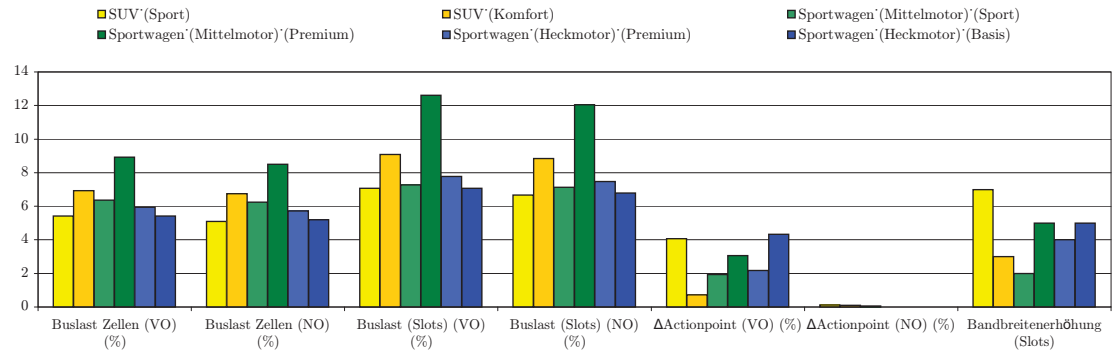


Abbildung 5.27.: Buslasten im statischen Segment für alle Architekturvarianten vor und nach der Makrotickskalierung

1. Performanzsteigerung: Eine Performanzsteigerung lässt sich durch eine Abstimmung des zu konfigurierenden Makroticks mit der in der Architekturvariante vorherrschenden Systempräzision erreichen. Für diesen Zweck werden die errechneten Daten für die relevante Präzision für jede Ausstattungsvariante einer Baureihe in der Berechnung des Makroticks berücksichtigt. Ziel ist es die zeitliche Differenz zwischen dem Actionpoint und der Systempräzision ($\Delta Actionpoint$) zu minimieren. Dabei wird ersichtlich, wie in Abb. 5.27 dargestellt, dass sich die Buslast in

der Fallstudie mit einem optimierten Makrotick im statischen Segment um bis zu *5%* absenken lässt. Dies entspricht durchschnittlich *4,3 Slots* für die sechs Architekturvarianten.

2. Robustheitssteigerung: Als Referenzbeispiel für eine allgemeine Erhöhung der Gesamtsystemrobustheit wird an dieser Stelle die Möglichkeit zur Reduzierung der Bitlängenverkürzung untersucht. Für diesen Zweck werden die FlexRay-Parametersätze für alle Architekturvarianten mit der standardmäßig vorgeschriebenen Quarzgenauigkeit für FlexRay-Systeme (1500ppm) berechnet. In einer ersten Optimierungsstufe substituieren genauere Quarze (300ppm) die Standardquarze in allen Steuergeräten. Als zweite Modifikation werden anschließend alle PLL-Uhrenkonzepte in den Knoten eliminiert. Bei der Umsetzung zeigt sich, dass sich die asymmetrische Bitverkürzung beim Quarzwechsel um durchschnittlich *4,4%* und bei einem PLL-Verbot um durchschnittlich *5,8%* in den angelegten Architekturvarianten reduzieren lässt (s. Abb. 5.28). Grundsätzlich erfüllt keine der definierten Topologien im Worst-Case die gegebenen Robustheitsanforderungen, wobei bereits die Uhrenmodifikation dazu führt, dass drei Varianten die Robustheitsvorgaben für die Bitlänge erreichen. Mit dem PLL-Verbot verbessert sich die Robustheit aller Varianten, wobei sich die Anzahl der validen Topologien nicht erhöht[9].

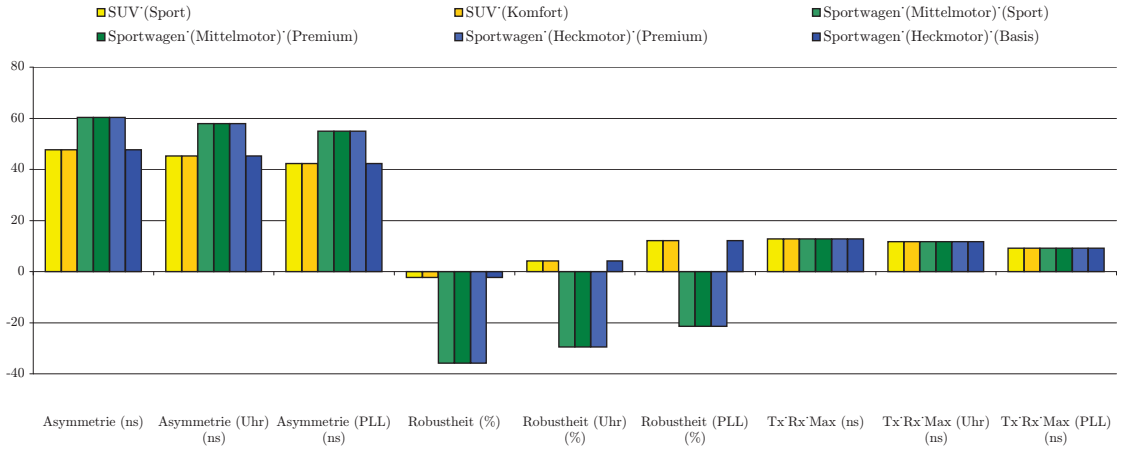


Abbildung 5.28.: Veränderung der Systemrobustheit anhand der asymmetrischen Bitlängenverkürzung unter Einsatz unterschiedlicher Uhrenkonzepte

5.4.3. Bewertung

Im Folgenden werden die grundlegenden Erkenntnisse aus der Fallstudie zusammengefasst. Die automatisierte Berechnung variantenabhängiger FlexRay-Parametersätze auf E/E-Architekturebene lässt sich im FlexZOOMED-Konzept vollständig valideren. Während der aktiven Modellierung und Parametrierung haben sich folgende Vorteile gezeigt:

[9] Durch die durchgeführten Optimierungen lässt sich die an den Transceivern enstehende Verzögerung um ca. 25% reduzieren!

- Integrierte Betrachtung heterogener Systemaspekte (Logische/Technische Architekturen),
- Unterstützung statischer und dynamischer Systemanalysen,
- Feingranulare variantenabhängige Optimierung des Architekturdesigns,
- Automatisierte Entwicklung von FlexRay-Konfigurationen (Parametersatz und Busschedule)

Die vielversprechenden Vorteile müssen jedoch eine Reihe an Einschränkungen kompensieren:

- Initialaufwand bei der Modellierung FlexRay-konformer Architekturen,
- Hohe Komplexität durch wechselseitige Verknüpfung der Berechnungsblöcke und Modellartefakte,
- Trennung von Berechnungs- und Architekturmodellen,
- Hoher Aufwand für Modellkonsistenzsicherung und -weiterentwicklung,
- Hoher Zeit-/Rechenaufwand bei der Systemanalyse,
- Sehr hohes Expertenwissen notwendig.

Während der Implementierung des E/E-Architekturmodells, des FlexRay-Berechnungsmodells und bei der Durchführung der Systemanalyse haben sich eine Vielzahl an potentiellen Weiterentwicklungsmöglichkeiten gezeigt, welche im Kap. 6 im Rahmen eines allgemeinen Ausblicks erläutert werden.

KAPITEL 6

Zusammenfassung und Ausblick

In diesem letzten Abschnitt werden die in der Arbeit erzielten Ergebnisse zusammengefasst und die aus den unterschiedlichen Forschungstätigkeiten gewonnenen wissenschaftlichen Erkenntnisse dargestellt. Nach einer kritischen Bewertung der erarbeiteten Methode lassen sich Schlüsse über Potentiale für zukünftige Entwicklungen ziehen, die in einem Ausblick integriert betrachtet werden.

Die vorliegende Arbeit befasst sich mit der Frage einer zweckmäßigen Entwicklungsmethodik für flexible zeitgesteuerten Bussysteme in der Fahrzeugserientwicklung am Beispiel der FlexRay-Technologie. Das Umfeld der Bussystementwicklung im Fahrzeug wird umfassend untersucht und dabei die zukünftigen Schwerpunkte der *modellgetriebenen Entwicklung*, der *Systemoptimierung* und der *Systemanalyse* abgeleitet. Diese drei Aspekte stellen die zentralen Themen der Arbeit, deren Ergebnisse zusammengefasst werden.

6.1. Ergebnisse

In der Grundlagenforschung zur Entwicklung eingebetteter Systeme im Automobil hat sich gezeigt, dass in naher Zukunft das Potential eines erhöhten nutzbaren Bandbreitenbedarfs eine treibende Kraft hochvernetzer Fahrerassistenzsysteme im PKW-Bereich darstellen wird. Die Vorteile einer flexiblen Zeitsteuerung mit ihrer deterministischen zuverlässigen Datenübertragung sind dabei bislang als sekundär einzustufen. Diese Erkenntnis beruht auf der Tatsache, dass flexible zeitgesteuerte Architekturen im direkten Vergleich mit den im Serienbereich etablierten ereignisgesteuerten Systemen mit einem Komplexitätsanstieg einhergehen. Speziell die statischen Eigenschaften und die damit verbundene umfangreiche und mit Wechselwirkung behaftete Systemkonfiguration erfordern ein verstärktes Verständnis für integrierte Systementwicklung. Eine

anwendungsbezogene allgemeine Strategie zur Systemparametrierung des Feldbussystems fehlt. Eine Analyse existierender Entwicklungsmethoden und -werkzeuge verstärkt die Erkenntnis, dass ein globales architekturbezogenes Systemdesign für flexible zeitgesteuerte Architekturen bisher nur unzureichend auf den Stand der Technik abgebildet wird. Eine technisch fundierte Argumentation zur Herleitung einer E/E-Architekurbezogenen FlexRay-Systemspezifikation steht bisher aus. Daraus resultiert die Zurückhaltung einer Vielzahl an Fahrzeugherstellern im Kontext eines FlexRay-Serieneinsatzes.

Die Analyse der Einflussfaktoren für den Entwurf eines FlexRay-Systems führt auf allgemeine Fragestellungen der E/E-Architekturentwicklung, etwa bei Systemvarianten in der logischen und technischen Systemkonfiguration. Dieser Erkenntnis wird in der vorliegenden Arbeit Rechnung getragen. In Abgrenzung zu existierenden Ansätzen ist eine neuartige durchgehende Einbettung der Anforderungen für die Entwicklung einer Querschnittstechnologie erzielt worden, wobei spezifische Anforderungen im Kontext der modellbasierten E/E-Architekturentwicklung berücksichtigt werden. Dabei werden die logischen und technischen Aspekte auf eine akzeptable Eingangskomplexität für die Gesamtsystementwicklung heruntergebrochen sowie das Abstraktionsniveau auf der Systemspezifikationsebene erhöht. Dadurch lassen sich Fragestellungen des technischen Systementwurfs im Entwicklungsprozess zeitlich nach vorne verlagern (*Front Loading*).

Durch die Überführung des FlexRay-Entwurfs auf eine abstrakte Spezifikationsebene ergeben sich zusätzliche Fragestellungen für eine optimierte Systemauslegung und Analysierbarkeit der Architekturen. Bei der Anwendung ausgewählter Optimierungsverfahren zeigt sich, dass der Freiheitsgrad in der Systemparametrierung nicht eindeutig einschränkbar ist. Die systematische Hinzunahme eigens definierter Restriktionen zur Einschränkung des Lösungsraums, etwa im Bereich der Skalierung der Anzahl an gewünschter Slots im statischen FlexRay-Segment, ist dabei entscheidend. Speziell die Ableitung dieser Restriktionen aus dem E/E-Architekturmodell bleibt eine weiterführende Aufgabe. Die Integration der FlexRay-Systemparameter in statische und dynamische Architekturmodelle zeigt sich in der Arbeit als komplexe Aufgabe, etwa bei der Analyse der netzwerktopologieabhängigen Parameter. Die Methoden zur Modellprüfung und Parameterberechnung gilt es daher weiterzuentwickeln, um den damit verbundenden Arbeitsaufwand zu verringern.

Zusammenfassend lässt sich folgern, dass die architekturorientierte FlexRay-Systementwicklung ein sinnvolles Konzept zur integrierten Fahrzeugentwicklung und -pflege darstellt, welches allerdings mit hohem Aufwand einhergeht. Doch gerade die damit abbildbare Systemkomplexität eröffnet die Möglichkeit für ein grundlegendes Systemverständnis für die Anforderungen einer zielorientierten E/E-Architekturentwicklung und -optimierung. Die in dieser Arbeit entwickelte FlexZOOMED-Methode konzentriet sich auf die systematische Herleitung FlexRay-basierter E/E-Architekturen. Im Rahmen künftiger industrieller Entwicklungen werden partielle Konzepte der entstandenden Arbeit zur Weiterentwicklung existierender E/E-Plattformen eingesetzt werden. Eine tiefe Integration und weitergeführte Entwicklung der erarbeiteten Ergebnisse stellt eher eine längerfristige Herausforderung dar und ist mit einem zusätzlichen Forschungsaufwand verbunden.

Die in dieser Arbeit erzielten Ergebnisse bilden einen Ausschnitt aus einer dreijährigen industriellen Forschungstätigkeit im direkten Umfeld der Fahrzeugserienentwicklung. Sämtliche Arbeitsinhalte beziehen sich auf Querschnittsumfänge aus der Fahr-

zeugserienentwicklung, die in keinster Weise als vollständig zu betrachten sind. Ausblickend werden die interessantesten Ansatzpunkte für potentielle Weiterentwicklungen im nächsten Abschnitt erläutert.

6.2. Ausblick

Gezwungenermaßen muss sich diese Arbeit auf wesentliche Gesichtspunkte beschränken. Die Durchschlagskraft der entwickelten Methodik würde sich bei einer Ausweitung des Gesamtkonzepts auf folgende Themen erhöhen:

Busscheduling: Eine erweiterte Umsetzung des automatisierten Busschedulings auf Basis vorliegender Funktionsnetze kann die Pflege des Bussystems auf dem abstrakten Level der Funktionsbeschreibung verbessern. Gleichzeitig wird die Integration neuer Steuergeräte oder Steuergerätfunktionen vom Aufwand der manuellen Bedatung des Busschedules befreit.

Restfahrzeug: Während in dieser Arbeit der Fokus weitestgehend auf flexible zeitgesteuerte Architekturen autark vom Restfahrzeug gelegt worden ist, erfordert eine ganzheitliche Entwicklung der E/E-Architekturen die Betrachtung sämtlicher technischer Aspekte und Technologien. Speziell die Analyse heterogener Architekturen auf Basis eines gemischt agierenden CAN-FlexRay-Systemverbunds ist bei der Verschiebung von Funktionen zwischen CAN- und FlexRay-Steuergeräten von hoher Bedeutung.

Komplexitätsreduktion: Die Entwicklung und Pflege modellbasierter Architekturen im Kontext der FlexRay-Systementwicklung benötigt eine zeitlich und inhaltlich intensive Vorlaufphase zur Erhebung sinnvoller und valider Daten. Weiterhin stellt die Integration dieser Daten in die E/E-Architekurmodelle eine anspruchsvolle Aufgabe dar, die sich durch die Erhöhung des Reifegrads von Metamodellen messen lässt. Eine ausgereifte Bedatung der E/E-Architektur führt aus Netzwerksicht zu kritischen Modellgrößen, die bei der Erstellung und Pflege hohen Aufwand erfordern. Auch die Fehlerfreiheit in den Modellen lässt sich durch die Restriktionen der Semantik eines Metamodells nur teilweise erfüllen. Dies resultiert aus der Tatsache, dass sich ein Metamodell in einer permanenten Entwicklung befindet und dem Entwickler einen gewissen Spielraum in der Modellierung einräumen muss. Um die Abbildung diverser Entwicklungskonzepte verschiedener Anwendungsbereiche in unterschiedlichen Industriepositionen nicht zu stark durch den Designflow des Werkzeugs einschränken zu müssen, birgt diese Offenheit im Entwicklungsprozess gewisse Risiken. Eine Maßnahme zur Steigerung der Qualität wäre die Identifikation von Klonen in einem Modell, um die Strukturierung und Wiederverwendung der Modelle zu verbessern.

Sensitivitätsanalyse: Durch den hohen Grad an wechselseitigen Abhängigkeiten zwischen den FlexRay-Parametern kommt es bei Modifikationen in den E/E-Architekturmodellen zu partiellen Veränderungen im FlexRay-Parametersatz, welche in der Analysesicht zu untersuchen sind. Um die Zusammenhänge zwischen einzelnen Modifikationsschritten am Modell und deren Auswirkungen auf den Parametersatz besser zu verstehen, bieten sich Methoden zur Sensitivitätsanalyse an, um

die Eigenschaften der E/E-Architektur und die Auswirkungen von Modelländerungen werkzeuggestützt nachvollziehen zu können.

***Timing*-Visualisierung:** Eine explizite Darstellung der Zeitabhängigkeiten im Konzept der Zeitkettenbildung innerhalb des Funktionsnetzwerks bildet einen weiteren Schritt zur Interpretation von funktionalen Erweiterungsmöglichkeiten der E/E-Architektur. Zusätzlich wäre es interessant, die Abhängigkeiten zwischen Teilgraphen des Funktionsnetzwerks und deren Abbildung im Steuergeräte- und im Busschedule in einer eigenständigen Darstellung zu pflegen. Speziell bei der bussynchronen Applikationsentwicklung bietet sich dies an. Jedoch zeigt sich in dieser Arbeit auch die Kritikalität der Entwicklung vernetzter busasynchroner Applikationen. Eine systematische Herleitung eines stabilen busasynchronen Funkionsverbunds stellt in gleicher Weise eine Herausforderung, die in einem Zeitmodell zu analysieren ist.

Integrierter *Schedulability*-Test: In Ergänzung einer *Timing*-Visualisierung bietet es sich an auf Basis des modellierten Zeitverhaltens eine holistische *Scheduling*-Analyse durchzuführen. Dazu müssten allerdings die internen zeitlichen Abläufe, die während der Steuergeräteausführung auftreten, explizit spezifiziert werden. Auf dieser Basis wäre ein *Schedulability*-Test aufsetzbar.

Automatisierte Modellierung: Die Erstellung und Pflege eines FlexRay-Architekturmodells ist mit einem beträchtlichen Aufwand verbunden. Eine sukzessive Automatisierung in der Modellentwicklung würde dabei neben einer Effizienzsteigerung in der Entwicklung des Modells auch zu einer reduzierten Komplexität und Fehleranfälligkeit bei der Modellierung führen. In dieser Arbeit sind die entwickelten Modellartefakte regelbasiert auf Konsistenz geprüft worden. Speziell im Bereich der logischen und technischen Architekturbeschreibung wären vorkonfektionierte Modellkomponenten hilfreich. Dadurch würde sich der Entwicklungsstil stärker vertikal über die Modellierungsebenen hinweg für einzelne Architekturkomponenten gestalten, was einer Abkehr von dem Konzept der horizontalen Entwicklung einzelner Ebenen mit anschließender Ebenenverknüpfung entspricht.

Automatisierte Testfallgenerierung: Durch die vollständige Beschreibung des FlexRay-Clusters in der Systemarchitektur lassen sich je nach Notationsform aus den modellierten Artefakten Testfälle ableiten, welche durch Überführung in nachgelagerte Werkzeugketten und Prüfständen weiterverarbeitet werden.

Weiterentwickeltes Variantenmanagement: Während der Umsetzung der FlexZOOM-ED-Methode in einer integrierten Entwicklungsumgebung ist in dieser Arbeit der Entwicklungsstand des Variantenmanagements innerhalb der verfügbaren Werkzeugkette eingesetzt worden. In diesem Ansatz lassen sich die Varianten durch angelegte Schablonen, welche ein Gesamtmodell filtern, darstellen. Der Nachteil dieser Methodik liegt in der expliziten Auswahl einzelner Fahrzeugbaureihen im E/E-Architekturmodell, die anschließend unabhängig voneinander analysiert werden. Dadurch kann ein direkter Vergleich unterschiedlicher Architekturvarianten nur sequentiell durchgeführt werden. Interessant wäre eine Weiterentwicklung zur simultanen Betrachtung von n Architekturvarianten, um die Unterschiede werkzeuggestützt erfassen und interpretieren zu können.

Kostenfunktionen: Das Thema *Kostenkalkulation* wird in dieser Arbeit nicht gesondert berücksichtigt. Neben der technischen Untersuchung einer integrierten modellbasierten E/E-Architekturentwicklung für FlexRay-Systeme bietet es sich durchaus an die entwickelten Architekturen hinsichtlich der erwarteten Kosten zu analysieren. Dies kann direkt über die Betrachtung unterschiedlicher Hardwarealternativen oder abstrakt, etwa über die Gewichtung von Bandbreitenverbrauch oder der Aufwände für Systemmodifikationen, erfolgen. Für diese Belange wäre allerdings ein eigenständiges Kostenmodell inklusive spezifischer Kostenfunktionen zu entwickeln.

Probabilistische Verfeinerung der dynamischen Architekturanalyse: Einige Aspekte der FlexRay-Systembetrachtung korrelieren mit probabilistischen Annahmen, beispielsweise bei der Betrachtung der Netzwerkkommunikation im dynamischen FlexRay-Segment oder den Abläufen beim Netzwerkweck- und -startvorgang. Innerhalb der erarbeiteten Ergebnisse ist für die Netzwerkkommunikation eine Abstraktion durch die Annahme einer Normalverteilung im Auftritt der ereignisgesteuerten Botschaften geschaffen worden. Für den Weck- und Startvorgang bleiben die Auswirkungen von potentiellen Kollisionen bei der Kommunikation auf dem Übertragungskanal unberücksichtigt und werden durch *Worst-Case*-Annahmen ersetzt. Die Analyse der Auswirkungen der probabilistischen Einflüsse auf das Gesamtsystem bildet eine Erweiterung.

Verfeinerung der FlexRay-Abbildungen: Während der Fokus in dieser Arbeit auf Restriktionen in der Netzwerkkonfiguration abzielt, wäre eine verfeinerte Abbildung der FlexRay-Protokollalgorithmen speziell für die dynamische Architekturanalyse denkbar. Eine formale Darstellung dafür ist auf Basis von SDL-Notationen bereits in der FlexRay-Spezifikation vorhanden. Dabei muss allerdings beachtet werden, dass dieser Ansatz neben einer beträchtlichen Komplexitätssteigerung stark von einer sauberen Implementierung der verwendeten FlexRay-Bauteilen abhängt, um die erbrachten Analyseergebnisse zuverlässig verifizieren zu können.

Erweiterung der angewendeten Optimierungsverfahren: Zur qualitativen Verbesserung der angewandten Optimierungsverfahren müssen weitere Restriktionen formuliert werden, um den existierenden Freiheitsgrad für die Wahl der Eingangsparameter in der Optimierung zur verringern und die Aussagekraft der ermittelten Ergebnisse zu verbessern.

Schnittstellenverhalten in der Komponentensicht: Während die Thematik der Entwicklung bussynchroner Applikationen vorrangig auf das Ausführungsverhalten der Steuergeräte auf Betriebssystemebene beschränkt wird, zeigt sich mit der Entwicklung des AUTOSAR-Standards /[AUT07a]/ die Notwendigkeit der fundierten Betrachtung treibernaher Standardsoftware. Dafür wird eine explizite Spezifikationsebene in der E/E-Architekturmodellierung erforderlich. Bei der Umsetzung dieser Arbeit auf eine integrierte Entwicklungsumgebung wird bisher nur zwischen Standardsoftwarevarianten als monolithische Einheiten unterschieden. Eine erweiterte Sicht zur Standardsoftwarespezifkation wäre auf Komponentenebene zukünftig vorteilhaft.

Integrierte dynamische Architekturanalyse: Die in dieser Arbeit betrachteten Aspekte der dynamischen Analyse sind getrennt von der statischen Analyse in dem Werk-

zeug UPPAAL /[BLP+96]/ untersucht worden. Dadurch wird eine Kopplung mit sämtlichen Attributen aus dem E/E-Architekturmodell verhindert. Eine sukzessive Integration einer formalisierten Notation für die dynamische Architekuranalyse innerhalb des modellierten E/E-Architekturmodells bildet eine evolutionäre Weiterentwicklung für die vorgestellte integrierte Entwicklungsumgebung.

Funktionale Sicherheit: Der funktionalen Sicherheit wird aufgrund des Anwendungsfelds der fortgeschrittenen Fahrerassistenzsysteme bei der FlexRay-Vernetzung zukünftig mehr Aufmerksamkeit zukommen. Eine spannende Frage ist die integrierte Betrachtung notwendiger Absicherungskonzepte für sicherheitskritische Architekuren im System, etwa bei der Injektion von Fehlerzuständen zur Modellanalyse und den Methoden für Fehlerbehandlungen.

Korrektheitsnachweis der Parameterentwicklung: Eine zertifizierter Parametersatz ist unerlässlich für die Freigabe von FlexRay-Systemen in Fahrzeugserienprojekten. Bisher werden entsprechende Testverdikte zur Zertifizierung hardwarebasiert von unabhängigen Institutionen betreut und auf die herstellerspezifischen Parametersätze angewendet. Mittelfristig sollte eine Verifikation der entwickelten Parametersätze bereits beim Hersteller erfolgen, um den Aufwand zur Freigabe zu reduzieren. Eine interessante Fragestellung liegt im Korrektheitsnachweis durch die automatisierte Prüfung des FlexRay-Parametersatzes als Teil der vorgestellten Werkzeugumgebung.

Werkzeugkopplung zur Optimierung: Die Untersuchungen zur Optimierung von FlexRay-Parametersätzen mittels numerischer Optimierungsverfahren erfolgt in der vorgestellten Methode auf Basis einer autarken Untersuchung im Werkzeug Matlab/Simulink. Eine Werkzeugkopplung zwischen Matlab/Simulink und dem E/E-Architekturmodell in PREEvision wäre zur integrierten Optimierung des E/E-Architekturmodells vorteilhaft, um die manuellen Schritte zwischen Modellierung und Optimierung zu umgehen. Damit wäre es möglich die Methode der Architektursimulation /[Sch06]/ mit der statischen Architekturanalyse zu koppeln.

Glossar

Application Data
Daten, welche in *Tasks* erzeugt und/oder verwendet werden. Im Kontext der *Automotive Systems* werden oft Signale als Format für die Kommunikation von *Tasks* untereinander verwendet..

Backbone
Ein Übertragungsmedium, auf dem der gesamte Datenverkehr zwischen mehreren Systemen ablaufen kann. In der Vernetzung könnte ein *Backbone* einen schnellen Bus darstellen, an dem die *Gateways* sämtlicher anderer Bussysteme hängen, um sämtliche Daten untereinander austauschen zu können..

Bitstopfen
Unter Bitstopfen (bit stuffing) versteht man im Zusammenhang mit der CAN-Technolgie das Einfügen eines Bits in eine monotone Datensequenz (5 gleiche Bits mit dem logischen Wert 0 oder 1) bei der Übertragung, um die Korrektheit des übertragenen Bitstroms an den Empfänger zu signalisieren. Dieser Mechanismus garantiert einen regelmäßigen Flankenwechsel auf der Bitebene, der zur Synchronisation der Netzwerknoten erforderlich ist. Der Empfänger detektiert und entfernt das zusätzlich eingefügte Bit, um die eigentlichen Nutzdaten wiederherzustellen..

Botschafts-ID
Dieser Identifkator definiert die *Slot*-Position im statischen Segment und die Priorisierung des Rahmens für das dynamische Segment. Ein niedriger Identifikator signalisiert eine höhere Priorität. Es ist zwei Kommunikations-*Controllern* nicht gestattet Rahmen mit dem gleichen Identifikator auf dem gleichen Kanal zu verschicken..

Bus
Eine physikalische Topologie für ein Kommunikationssystem, in der alle Knoten über eigene Zugangsleitungen (*Stichleitungen*) an ein gemeinsames Übertragungsmedium direkt angeschlossen werden (ohne die Verwendung von Sternen, *Gate-*

ways, etc.). Den Begriff Bus assoziert man zusätzlich mit dem Übertragungsmedium ansich..

Bus driver
Eine elektronische Komponente, bestehend aus einem *Transmitter* und einem *Receiver*, der den Kommunikations-*Controller* mit einem Kommunikationskanal verbindet..

Bus guardian
Eine elektronische Komponente, die den Kommunikationskanal vor nicht authorisierten Eingriffen (*Interferenzen*) schützt. Ein nicht authorisierter Eingriff bezieht sich auf einen Sendezugriff zu einem arbiträren nicht vorgesehenen Zeitpunkt im *Schedule* des *Clusters*. Dies wird erreicht durch die Freischaltung des Kanals für den Kommunikations-*Controller* ausschließlich zu den für ihn reservierten Zeitpunkten im *Schedule*..

Bustreiber
= *Bus Driver*.

Channel
Siehe Kommunikationskanal.

Channel idle
Der Zustand der Ruhe auf einem Kommunikationskanal, welcher von jedem angeschlossenen Knoten erkannt wird..

Clique
Eine Menge von Kommunikations-*Controllern*, welche die selbe Sicht auf bestimmte Systemeinstellungen haben, beispielsweise die globale Zeit oder den Aktivitätszustand von Kommunikations-*Controller n*. Für die Zwecke eines konsistenten *Cluster-StartUps* dürfen sich in keinem Fall zwei konkurrierende Cliquen bilden, die durch einen simultanen *StartUp* versuchen jeweils einen eigenen *Schedule* zu initialisieren..

Cluster
Ein aus mehreren Knoten bestehendes Kommunikationssystem, dessen Teilnehmer mindestens mit einem gemeinsamen Kanal verbunden sind (Bus Topologie) oder über Sternkoppler (Sterntopologie) zusammengeführt werden. Falls alle Knoten zweikanalig verbunden sind werden die Kommunikationspfade zwischen arbiträren Knotenpaaren dupliziert und die Übertragungszuverlässigkeit erhöht..

Coldstart node
Ein spezieller Knotentyp, welcher die Fähigkeit besitzt eine *StartUp*-Botschaft zu versenden, um eine *Cluster-StartUp*-Prozedur einzuleiten. Eine Anforderung an diese Knoten ist, dass sie in einem 2-Kanalnetzwerk in jedem Fall zweikanalig ausgelegt sein müssen..

Communication controller (CC)
Eine elektronische Komponente in einem Knoten, der für die Implementierung der Protokolle des FlexRay-Kommunikationssystems verantwortlich ist..

Communication slot
Ein Zeitintervall, indem der Zugriff auf dem Kommunikationskanal exklusiv einem Knoten gewährt wird zur Übertragung eines Rahmens mit einer *Frame*-ID, welche korrespondiert mit der *Slot*-ID. FlexRay unterscheidet zwischen statischen und dynamischen *Slots*..

Cycle counter
Der Zähler, welcher die aktuelle Nummer des Kommunikationszyklus beinhaltet..

Cycle time
Die Zeit, innerhalb des Kommunikationszyklus, ausgedrückt in *Makrotick*-Einheiten. Zu Beginn jedes Kommunikationszyklus wird die Zykluszeit auf Null zurückgesetzt..

Cyclic Redundancy Check
Ein Datensicherungsmechanismus, mit dem durch einen an die Botschaft angehängten Prüfcode, beim Empfänger die Korrektheit der eingetroffenen Datentelegramme verifiziert werden kann. Dieser Prüf-*Code* stellt dabei den Rest der Divison des Datentelegramms durch ein speziell definiertes Generatorpolynom dar. Dadurch kann mit einer erneuten Divison des Datentelegramms mit angefügten Prüf-*Code* durch das Generatorpolynom beim Empfänger Fehler entdeckt werden und gegebenenfalls die Botschaft verworfen werden..

Dynamischer Slot
Ein Zeitintervall innerhalb des dynamischen Segments eines Kommunikationszyklus, welches je nach Übertragungsdauer eines Rahmens einen oder mehrere *Minislots* andauern kann. Eine Zugriff erfolgt wie im statischen Teil über Frame-IDs, allerdings muß man die Identifikatoren hier als Priorisierungsnummer betrachten, da ein *Slot* nach Belieben ausgedehnt werden kann bis der nächste Teilnehmer mit der nächsthöheren ID an der Reihe ist. Ein lineares *Mapping* von *Frame*-ID zum Sendezeitpunkt im *Schedule* wie für den statischen Bereich kann daher hier nicht beobachtet werden. Falls in einem *Slot* nicht übertragen wird entspricht die Länge eines dynamischen *Slots* der eines *Minislots*..

Dynamisches Segment
Ein Abschnitt des Kommunikationszyklus, in dem der MAC-Zugriff über ein *Minislotting*-Schema kontrolliert wird, welches nach dem FTDMA-Prinzip (Flexible Time Division Multiple Access) arbeitet. Innerhalb diesem Segments erfolgt der Zugriff auf das Übertragungsmedium dynamisch anhand einer Priorisierung der zu übertragenden Botschaften zwischen den Knoten..

E/E-Architektur
Die logische und technische Ausprägung aller zusammenhängenden Elektrik-/Elektronikkomponenten von Fahrzeugbaureihen. In der Regel bildet die E/E-Architektur die Übermenge aller potentiell verbaubarer Elektrik-/Elektronikkomponenten. Die Bezeichnung Architektur entstammt der Idee der strukturierten, integrierten Planung und Entwicklung aller E/E-Anteile im Fahrzeug. Aktuell verbindet man dem Begriff in erster Instanz die physikalische Ausprägung, wobei die logische Sicht ein wesentliches Element im Entwurf darstellt..

Electronical Control Unit (ECU)
Die technische und praktische Umsetzung eines
Knotens in einem FlexRay-System. Dazu gehört die Stromversorgung, der Host, der Kommunikations-*Controller*, der elektrische Bustreiber und der *Bus Guardian*..

Flashen
Der Vorgang, indem ein ausführbarer übersetzter *Code* auf ein Steuergerät überführt wird, um ihn im *Cluster* starten und stoppen zu lassen..

Frame
Eine Struktur zum Austauschen von Informationen in einem Kommunikationssystem. Ein *Frame* besteht aus einem *Header-*, *Payload-*, und *Trailer*-Segment. Der *Header* überträgt Statusinformationen des gesamten Rahmens, die Payload beinhaltet die Nutzdaten und das *Trailer*-Segment wird zur Rahmenverifikation mittels eines CRCs verwendet..

Gateway
Ein Knoten, an dem zwei oder mehrere unabhängige Kommunikationsnetze hängen und deren Informationen über diese Komponente hinweg zwischen den Netzen fliessen können..

Globale Zeit
Eine Kombination von Zykluszähler und Zykluszeit..

Hamming Abstand
Der minimale Abstand (Anzahl der von einander abweichenden Bits) zwischen zwei beliebigen Code-Wörten in binärer Schreibweise..

Host
Der Host ist die Komponente einer ECU, an der die Applikations-*Software* ausgeführt wird. Getrennt wird der Host durch die CHI von der Protokoll-*Engine*..

Knoten
= Node.

Kommunikationskanal
Die Verbindung, zwischen der die Knoten Botschaften oder Signale austauschen können für die Zwecke einer Kommunikation miteinander. Diese Bezeichnung wird abstrahiert von der Topologie (Bus,Stern) und vom physikalischen Übertragungsmedium (elektrisch/optisch) benutzt..

Kommunikationszyklus
Eine komplette Instanz der Kommunikationsstruktur, welche periodisch wiederholt wird, um die Zugriffsmethode auf das FlexRay-System komprimieren zu können. Ein Kommunikationszyklus besteht aus einem statischen Segement, einem optionalen dynamischen Segment, einem optionalen Fenster für die Übertragung von Symbolen und ein abschließendes Segment für die Netzwerkruhe (*Network Idle Time*)..

Konnektor
Eine Steckverbindung, um Netzwerkkomponenten miteinander zu verknüpfen..

Makrotick
Ein Zeitintervall, dessen Dauer von dem *cluster*-weiten Taktsynchronisationsalgorithnmus abgeleitet wird. Ein *Makrotick* besteht aus einer ganzzahligen Menge von *Mikroticks*. Ein *Makrotick* repräsentiert die feingranularste Einheit für die Zwecke der globalen Zeit..

Medium idle
Der Zustand des physikalischen Übertragungsmediums, indem kein Knoten aktiv sendet..

Microtick
Ein Zeitintervall, direkt übernommen vom Oszillator des Kommunikations-*Controllers* (möglicherweise durch Verwendung eines Vorverteilers). Der Mikrotick wird nicht durch den Taktsynchronisationsmechanismus beeinflusst, da er ein knotenlokales Medium darstellt. Unterschiedliche Knoten können mit unterschiedlichen *Mikrotick*-Perioden arbeiten..

Minislot
Ein Zeitintervall innerhalb des dynamischen Segments des Kommunikationszyklus, welches von konstanter Dauer (in Bezug auf *Makroticks*) ist und von dem synchronisierten FTDMA MAC-Schema für den Zweck der Arbitrierung auf dem Übertragungsmedium verwendet wird..

Netzwerk
Die Kombination von Kommunikationskanälen, welche die Knoten in einem *Cluster* verbinden..

Netzwerktopologie
Die Anordnung der Verbindungen zwischen den Knoten eines Netzwerks. FlexRay unterstützt Bus-, Stern-, kaskadierte Stern-, und hybride Netzwerktopologien..

Node
Eine logische Einheit, verbunden in einem Netzwerk, welches in der Lage ist Rahmen zu senden und/oder zu empfangen. Ein *Node* ist eine Abstraktion eines Steuergeräts..

Null frame
Ein Rahmen ohne verwendbare Daten innerhalb des Nutzdatensegments. Ein *Null Frame* wird über ein speziell ausgezeichnetes Indikatorbit des *Header* Segments spezifiziert und hat sämtliche Bytes des *Payload*-Segments auf Null gesetzt..

Physikalisches Übertragungskanal
Eine Knotenverbindung zur Übertragung von Signalen für die Zwecke einer Kommunikation zwischen den Knoten. Alle über diese Verbindung zusammengeschlossenen Knoten teilen sich die identischen elektrischen oder optischen Signale (demzufolge ist ihre Verbindung nicht unterbrochen durch Signalverstärker, Sterne, Gateways, etc.). Beispiele für eine physikalische Verbindungsleitung wären ein

einzelnes *Backbone* oder eine Punkt-zu-Punkt-Verbindung zwischen einem Knoten und einem Stern. Ein Kommunikationskanal kann durch die Kombination mehrerer physikalischer Verbindungsleitungen zusammen mit Sternen gebildet werden..

Präzision
Die *Worst-Case*-Abweichung zwischen korrespondierenden *Makroticks* zweier beliebiger synchronisierter Knoten in einem *Cluster*..

Schedule
Ein zeitlich bedingter Ablaufplan, in dem die Zugriffszeitpunkte der Kommunikations-*Controller* in den einzelnen Knoten geregelt ist, um eine Koordination der Kommunikation und ungewollte Konflikte bei dem Zugriff auf das Übertragungsmedium verhindern zu können..

Slot
= Communication slot..

Startup frame
FlexRay-Rahmen, dessen Header Segment einen speziell dafür ausgezeichneten Indikator beinhaltet in Form eines *StartUp*-Bits. Sich integrierende Knoten nützen die zeitlich relevante Information dieses Rahmens für die Initialisierung während des *StartUp*-Prozesses. *StartUp-Frames* sind immer gleichzeitig auch *Sync-Frames*..

Startup slot
Kommunikationsslot, indem der der *StartUp-Frame* gesendet wird..

Statischer Slot
Ein Zeitintervall innerhalb des statischen Segment eines Kommunikationszyklus, welches eine konstante Anzahl von *Makrotricks* beinhaltet. Für jeden dieser *Slots* wird der Zugriff exklusiv einem einzigen Knoten gewährt für die Übertragung eines Rahmens mit einer zu der *Slot*-Nummer korrespondierenden Rahmen-ID. Im Gegensatz zu dem dynamischen *Slot* beinhaltet jeder statische *Slot* eine konstante Anzahl von *Makroticks* unabhängig davon ob der Rahmen in dem *Slot* eines Zyklus gesendet wurde oder nicht..

Statisches Segment
Abschnitt eines Kommunikationszyklus, indem der MAC-Zugriff per zeitgesteuerten TDMA-Schema stattfindet. In diesem Segment ist der Buszugriff fest geregelt und deterministisch durch die Zuteilung von *Slots* anhand einer zeitlichen Regelung. Dadurch entsteht in jedem Fall ein determiniertes Verhalten für diesen Bereich..

Stern
Eine Komponente, die einen Informationstransfer zwischen mehreren physikalischen Übertragungsmedien implementiert. Ein Stern spiegelt die Informationen eines Übertragungsmediums auf alle anderen angeschlossenen Übertragungsmedien. Ein Stern kann aktiv oder auch passiv fungieren, also auch falls gewünscht die Eigenschaften eines Steuergeräts annehmen..

Steuergerät
= ECU.

Stichleitung
= Stub.

Stub
Dies bezeichnet die Verbindung von einem Knoten in einem Netzwerk zum Übertragungsmedium (z.B. einem Bus)..

Sync frame
FlexRay-Rahmen, in dessen *Header*-Segment ein gesetztes Indikatorbit signalisiert, dass die zeitliche Abweichung zwischen dem erwarteten Rahmensankunftszeitpunkt und dem realen Ankunftszeitpunkt gemessen und für die Berechnung der Taktsynchronisationsalgorithmen als Eingabewerte herangezogen werden sollen..

Sync node
Diese Knoten verfügen über die Eigenschaft *Sync-Frames* verschicken zu können, welche für die Taktsynchronisation im *Cluster* verwendet werden. In einem *Cluster* ist die Anzahl an vorkommenden *Sync Nodes* auf 16 beschränkt..

Sync slot
Zeitschlitz im *Schedule*, indem der *Sync*-Rahmen verschickt wird..

Task
Eine gekapselte Aufgabe, in deren Rahmen eine gewisse Funktionalität bereitgestellt und umgesetzt werden muß. Diese Aufgabe kann in FlexRay zeitlich limitiert oder/und periodisch abgearbeitet werden..

Topologie
= Netzwerktopologie.

Verbindungskabel
= physikalische Verbindungsleitung.

WCET
Die *Task*-spezifische maximal benötigte Zeit zur Ausführung seiner implementierten Funktionalität (*Worst Case Execution Time*)..

Abkürzungsverzeichnis

ΔT . . . . *Jitter* zwischen zwei Komponenten

$\perp$ . . . . Nicht definiert

ϵ . . . . . Fehler (*error*)

μT . . . . Mikrotick auf Controllerebene

A . . . . Alter eines im Netzwerk empfangenen Signals

A_{FN} . . . Konnektivitätsmatrix zu einem Funktionsnetz

ACI . . . Actuator Interface

AMOC . Application Mode Control

APL . . . Application

ASIC . . Application Specific Integrated Circuit

BD . . . . Physikalischer Bustreiber (Bus Driver)

BRC . . . Basisempfangskomponente

BSC . . . Basissendekomponente

C . . . . . Systemkondition

c . . . . . Kosten/Komplexität

CAE . . . Computer Aided Design

CB . . . . Berechnungsblock

CC . . . . Kommunikationscontroller (Communication Controller)

CMM(I) . Capability Maturity Model (Integration)

CONF . . Configuration

CR . . . . Zykluswiederholrate (Cycle Repetition)

CSMA(D) Carrier Sense Multiple Access (Detection)

d . . . . . Verzögerung (Delay)

DEM . . Discrete Event Machine

DL . . . . Länge eines zu verarbeiteten Datensignals (Data Length)

E/E . . . Elektrik/Elektronik (EE)

ECU . . . Steuergerät

EPL . . . Electrical Physical Layer

FC . . . . Funktionskomponente

FC_In$_i$ . . Eingangsport einer Funktionskomponente

FC_Out$_i$. Ausgangsport einer Funktionskomponente

FFA . . . Functional Failure Analysis

FMEA . . Failure Mode and Effects Analysis

FN . . . . Funktionsnetz

FPGA . . Field Programmable Gate Array

FTA . . . Fault Tree Analysis

g . . . . . Netzwerkglobal

Ge . . . . Generalisierung

HAL . . . Hardware Abstraction Layer

HIL . . . Hardware in the Loop

LL . . . . Länge einer Verbindungsleitung

MD . . . Modelldaten

MT . . . Macrotick auf Netzwerkebene

N . . . . Netzwerkteilnehmer (logisch → Knoten, physikalisch → Steuergerät)

NAI . . . Network Application Interface

O . . . . Zeitlicher Abstand relativ zum Buszyklusstart (Offset)

$O_{a,i}$ . . . Slot mit Offset-Nummer i $\in \mathbb{N}$ aus Sicht des Netzwerkteilnehmers a

P . . . . . Prozess

p . . . . . Knotenlokal

PC . . . . Parameterberechung

PCH . . . Physikalischer Kanal

PCHP . . Physikalischer Kanalteil

PLL . . . Phase-Locked Loop

POC . . . Protocol Operation Control

PP . . . . Package Position

Re . . . . Verfeinerung

RP . . . . Rapid Prototyping

RTOS . . Real Time Operating System

S . . . . . Signal

SCS . . . Safety Control Services

SFD . . . Signal Flow Diagram

SIL . . . . Software in the Loop

SL . . . . Signallänge

SMOC . . System Mode Control

SNI . . . Sensor Interface

SOP . . . Start Of Production

SP . . . . Signalphase

SRC . . . Sende-Empfangsbeziehung

SSL . . . Safety/Security Layer

SST . . . Signalsubtyp

ST . . . . Signaltyp

SUV . . . Sports Utility Vehicle

SYS . . . System

T . . . . . Signalgenerierungszyklus (Applikationsspezifisch)

T_{BUS} . . . Übertragungszyklus (Busspezifisch)

TBTF . . Timing Budget Task Force

TD . . . . Temporäre Daten

TDMA . Time Division Multiple Access

TTNCONF Time Triggered Network Configuration

TTNI . . Time Triggered Network Interface

VHDL . . Very High Speed Integrated Circuit Hardware Description Language

WCET . . Worst Case Execution Time

$x \leftrightarrow y$ . . Parameter x korreliert mit Parameter y

Literaturverzeichnis

[ABR+93] Audsley, N. ; Burns, A. ; Richardson, M. ; Tindell, K. ; Wellings, A. J.: Applying New Scheduling Theory to Static Priority Pre-Emptive Scheduling. In: *Software Engineering Journal* 8 (1993), S. 284–292

[ADA07] ADAC: *ADAC - Pannenstatistik 2007*. http://www.adac.de, 2007

[AHP99] Atanassov, P. ; Haberl, S. ; Puschner, P.: Heuristic Worst-Case Execution Time Analysis. In: *Proc. 10th European Workshop on Dependable Computing*, Austrian Computer Society (OCG), May 1999, S. 109–114

[AJ78] A.Korn ; J.V.Wait: *Digital Continuous System Simulation*. Prentice Hall, 1978

[Alt02] Alt, Walter: *Nichtlineare Optimierung. Eine Einführung in Theorie, Verfahren und Anwendungen*. Vieweg, 2002 http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/352803193X. – ISBN 352803193X

[AQU06] AQUINTOS GmbH ; AQUINTOS GmbH (Hrsg.): *Benutzerhandbuch aquintos.EE*. V.1.7. Lammstraße 21, 76133 Karlsruhe: AQUINTOS GmbH, Mai 2006

[Aqu07] Aquintos GmbH: *PREEvision*. http://www.aquintos.com/de/products/ee.php, 2007

[ASA08] ASAM: FIBEX - Field Bus Exchange Format / Association for Standardisation of Automation and Measuring Systems - ASAM e.V. 2008 (Version 3.0). – Forschungsbericht

[ASH06] Armengaud, E. ; Steininger, A. ; Horauer, M.: Automatic Parameter Identification in FlexRay Based Automotive Communication Networks. In: *11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06)* (2006), Sep.

[Aut06] AutomotiveSPICE Initiative: *Automotive SPICE*. http://www.automotivespice.com, 2006

[AUT07a] AUTOSAR Konsortium: *Automotive Open Systems Architecture - Development Partnership -*. http://www.autosar.org, 2007

[AUT07b] AUTOSAR Konsortium: *Specification of the Virtual Function Bus*. http://www.autosar.org, Februar 2008 2007. – Version 1.0.1

[Avi95] Avizienis, A.: *The Methodology of N-Version Programming*. citeseer.ist.psu.edu/avizienis95methodology.html. Version: 1995

[AZL+06] Armbruster, M. ; Zimmer, E. ; Lehmann, M. ; Reichel, R. ; Sieglin, E. ; Spiegelberg, G. ; Sulzmann, A.: Affordable X-By-Wire technology based on an innovative, scalable E/E platform-concept. In: *VTC Spring*, IEEE, 2006, S. 3016–3020

[BBG+05] Beyer, S. ; Bohm, P. ; Gerke, M. ; Hillebrand, M. ; Rieden, T. I. ; Knapp, S. ; Leinenbach, D. ; Paul, W. J.: Towards the Formal Verification of Lower System Layers in Automotive Systems. In: *ICCD '05: Proceedings of the 2005 International Conference on Computer Design*. Washington, DC, USA : IEEE Computer Society, 2005. – ISBN 0–7695–2451–6, S. 317–326

[Bee07] Beeck, M. von d.: Development of logical and technical architectures for automotive systems. In: *Software and System Modeling* 6 (2007), Nr. 2, S. 205–219

[Ber89] Berry, G.: Real Time Programming: Special Purpose or General Purpose Languages. In: *IFIP Congress*, 1989, S. 11–17

[Ber00] Berry, G.: *The Foundations of Esterel*. MIT Press, 2000 citeseer.ist.psu.edu/62412.html

[Ber02] Berger, A.: *Embedded systems design*. CMP Books, 2002. – ISBN 1–57820–073–3

[BFM05] Belschner, R. ; Freess, J. ; Mrossko, M.: Gesamtheitlicher Entwicklungsansatz für Entwurf, Dokumentation und Bewertung von E/E-Architekturen. In: Ingenieure, VDI Verein D. (Hrsg.): *VDI Berichte Nr. 1907*. Düsseldorf : VDI Verlag, 2005. – ISBN 3–18–091907–8, S. 511–521

[BGG+04] Barry, A. ; Gisiger, A. ; Guido, N. ; Haak, G. ; Hartung, I. ; Heid, B. ; Huber, U. ; Näher, U. ; Ollig, W. ; Radtke, P. ; Vlasits, M. ; Weber, M.: Levers for Trouble-free Electronics. In: *McKinsey Automotive & Assembly* (2004), September. http://autoassembly.mckinsey.com

[BGH+06] Botaschanjan, J. ; Gruler, A. ; Harhurin, A. ; Kof, L. ; Spichkova, M. ; Trachtenherz, D.: Towards Modularized Verification of Distributed Time-Triggered Systems. In: *FM 2006: Formal Methods, 14th International Symposium on Formal Methods, Hamilton, Canada, August 21-27, 2006, Proceedings* Bd. 4085, Springer, 2006 (Lecture Notes in Computer Science). – ISBN 3–540–37215–6, S. 163–178

[BHS99] Broy, M. ; Huber, F. ; Schätz, B.: AutoFocus - Ein Werkzeugprototyp zur Entwicklung eingebetteter Systeme. In: *Inform., Forsch. Entwickl.* 14 (1999), Nr. 3, S. 121–134

[BHWJ07] Bonschab, S. ; Hermann, T. ; Wirrer, G. ; Jausel, H.: FlexRay aus der Sicht eines Elektronik-Suppliers / Euroforum 2007. Stuttgart, 8.-9. Mai 2007. – Forschungsbericht

[Bil08] Biller, Sven: *Vernetzung von Mikrocontrollern mit dem Ethernet: TCP/IP Kommunikation für Mikrocontroller und Embedded-Systeme*. Vdm Verlag Dr. Müller, 2008. – 978-3639014419

[BLP+96] Bengtsson, J. ; Larsson, F. ; Pettersson, P. ; Yi, W. ; Christensen, P. ; Jensen, J. ; Jensen, P. ; Larsen, K. ; Sorensen, T.: *Uppaal: a Tool Suite for Validation and Verification of RealTime Systems*. `http://www.docs.uu.se/rtmv/uppaal/-uppaal-guide.ps.gz`, 1996

[BMG07] Broy, J. ; Müller-Glaser, K. D.: The impact of time-triggered communication in automotive embedded systems. In: IEEE (Hrsg.): *SIES '07 - 2nd IEEE International Symposium on Industrial Embedded Systems*. Costa da Caparica, Portugal : IEEE, Juli 2007, S. 353–356

[Boo91] Booch, G.: *Object oriented design with applications*. Redwood City, CA, USA : Benjamin-Cummings Publishing Co., Inc., 1991. – ISBN 0–8053–0091–0

[Boo04] Booch, G.: *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Redwood City, CA, USA : Addison Wesley Longman Publishing Co., Inc., 2004. – ISBN 020189551X

[Bor94] Bortolazzi, J.: *Untersuchungen zur rechnergestützten Erfassung, Verwaltung und Prüfung von Anforderungsspezifikationen und Einsatzbedingungen elektronischer Steuerungs- und Regelungssysteme*. Erlangen-Nürnberg, Universität Erlangen-Nürnberg, Dissertation, April 1994

[BPG04] Berwanger, J. ; Peller, M. ; Griessbach, R.: *Byteflight-A New High-Performance Data Bus System for Safety-Related Applications*. `http://www.byteflight.com/`, 2004

[BR07] Broy, J. ; Roppel, R.: Challenges for the integration of FlexRay systems in future automotive electric/electronic architectures. In: *13. Internationaler Kongress Elektronik im Kraftfahrzeug*. Baden-Baden : VDI Fahrzeug- und Verkehrstechnik, Oktober 2007

[Bro98] Broy, M.: *Informatik - eine grundlegende Einführung*. Zweite. Springer, 1998

[Bro03] Broy, M.: Software im Automobil - Potentiale, Herausforderungen, Trends. In: *GI Jahrestagung*, September, 2003

[Bro05] Broy, J.: *Implementierung und Analyse eines FlexRay-Clusters zum Kosten- und Komplexitätsvergleich mit heutigen Bustechnologien*. München, Technische Universität München - Fakultät für Informatik - Lehrstuhl für Systemarchitektur: Betriebssysteme, Kommunikationssysteme und Rechnernetze, Diplomarbeit, Juli 2005

[Bro06] Broy, M.: Challenges in automotive software engineering. In: *ICSE '06: Proceeding of the 28th international conference on Software engineering*. New York, NY, USA : ACM Press, 2006. – ISBN 1–59593–375–1, S. 33–42

[Bro07] Broy, J.: *Herausforderung FlexRay System - Konfiguration und Test*. IIR Forum - ModellbasierterTest von Kfz-Steuergeräten 2007, Dezember 2007. – München

[BS01] Broy, M. ; Stoelen, K.: *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. 2001

[BS03] Broy, M. ; Steinbrüggen, R.: *Modellbildung in der Informatik*. Springer-Verlag, 2003

[BS06] Büttner, F. ; Schmid, T.: BMW: Seriencode für dezentrale Regler. In: *dSPACE GmbH - dSpace News* (Februar, 2006), Nr. 2

[BTB91] Behroozi-Toosi, A. ; Bhatia, S.: Modelling of computer networks and systems using SES/workbench. In: *Circuits and Systems, 1991., Proceedings of the 34th Midwest Symposium on* (1991), May, S. 992–996 vol.2. `http://dx.doi.org/10.1109/MWSCAS.1991.251969`. – DOI 10.1109/MWSCAS.1991.251969

[BV04] Boyd, S. ; Vandenberghe, L.: *Convex Optimization*. Cambridge University Press, 2004 `http://www.amazon.com/exec/obidos/redirect?tag=citeulike-20\&path=ASIN/0521833787`. – ISBN 0521833787

[BY04] Bengtsson, J. ; Yi, W.: *Timed Automata: Semantics, Algorithms and Tools*. `citeseer.ist.psu.edu/703625.html`. Version: 2004

[BZ08] Bortolazzi, J. ; Zöller, R.: Der Mechatronikentwicklungsprozess bei Porsche - Ziele, Herausforderungen, Implementierung / Dr. Ing. h. c. F. Porsche AG. Vector Kongress, 2008 2008. – Forschungsbericht

[Car06] Car 2 Car Communication Consortium: *Mission and Objectives*. `http://www.car-to-car.org`, 2006

[CCG+07] Cuenot, P. ; Chen, D. ; Gérard, S. ; Lonn, H. ; Reiser, M.-O. ; Servat, D. ; Sjostedt, C.-J. ; Kolagari, R. T. ; Torngren, M. ; Weber, M.: Managing Complexity of Automotive Electronics Using the EAST-ADL. In: *ICECCS '07: Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*. Washington, DC, USA : IEEE Computer Society, 2007. – ISBN 0–7695–2895–3, S. 353–358

[CDK01] Coulouris, G. ; Dollimore, J. ; Kindberg, T.: *Distributed systems (3rd ed.): concepts and design*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2001. – ISBN 0–201–61918–0

[Che08] Chen, X.: *Requirements and concepts for future automotive electronic architectures from the view of integrated safety*, Universität Karlsruhe (TH) - Fakultät für Elektrotechnik und Informationstechnik - Institut für Technik der Informationsverarbeitung (ITIV), Diss., April 2008

[Cor08] Cornelius, F.: EE-Architecture and Network Design from an OEM Perspective / Daimler AG. Synergy Forum Äircraft meets AutomotiveSSystem Architectures, Mai 2008. – Forschungsbericht

[DeM79] DeMarco, T.: Structured analysis and system specification. (1979), S. 409–424. ISBN 0–917072–14–6

[Dij72] Dijkstra, Edsger W.: Chapter I: Notes on structured programming. (1972), S. 1–82. ISBN 0–12–200550–3

[DIN90] DIN 40041 (12/90): *Zuverlässigkeit: Begriffe*. Beuth Verlag, Berlin, 1990

[DIN97] DIN 19226-1: *Leittechnik - Regelungstechnik und Steuerungstechnik - Stichwortverzeichnis*. Beuth Verlag, Berlin, 1997

[DMR77] Dickover, M. E. ; McGowan, C. L. ; Ross, D. T.: Software design using: SADT. In: *ACM '77: Proceedings of the 1977 annual conference*. New York, NY, USA : ACM, 1977, S. 125–133

[dSp07] dSpace GmbH: *dSpace SystemDesk*. `http://www.dspace.de/ww/de/gmb/home/products/sw/system_architecture_software/systemdesk.cfm?nv=n2`, 2007

[dSp08] dSpace GmbH: *SystemDesk - For planning, implementing and integrating complex system architectures and distributed software systems*. `http://www.dspace.co.uk`, Oktober 2008

[Ech90] Echtle, K.: *Fehlertoleranzverfahren*. Berlin : Springer-Verlag, 1990

[EE84] Electrical, Institute of ; Engineers, Electronics: IEEE guide to software requirements specifications. In: *IEEE Std 830-1984* (1984), Feb, S. –

[EEE98] Electrical, The I. ; Electronics Engineers, Inc.: IEEE Recommended Practice for Software Requirements Specifications / The Institute of Electrical and Electronics Engineers, Inc. New York, Juni 1998 (IEEE Std 830-1998). – Forschungsbericht

[ELLSV02] Edwards, S. ; Lavagno, L. ; Lee, E. A. ; Sangiovanni-Vincentelli, A.: Design of embedded systems: formal models, validation, and synthesis. (2002), S. 86–107. ISBN 1–55860–702–1

[ELM07] ELMOS Semiconductor AG: *E910.56 star coupler*. `http://www.elmos.de/german/produkte/assps/detail/flexray.html`. Version: 2007

[ERG08] Espinozaa, H. ; Richter, K. ; Gérard, S.: Evaluating MARTE in an Industry-Driven Environment: TIMMO's Challenges for AUTOSAR Timing Modeling. In: *Design Automation and Test in Europe (DATE)*. München, März 2008

[Ets09] Etschberger, K.: *CAN Controller Area Network - Grundlagen, Protokolle, Bausteine, Anwendungen*. 3. Auflage. Fachbuchverlag Leipzig, 2009. – ISBN 3-446-19431-2

[FBR+06] Freund, U. ; Braun, P. ; Romberg, J. ; Bauer, A. ; Mai, P. ; Ziegenbein, D.: AutoMoDe—A Transformation Based Approach for the Model-based Design of Embedded Automotive Software. In: *Proceedings of the 2006 European Congress on Embedded Real Time Software (ERTS)*. Toulouse, France, Januar 2006

[FBSG07] Faugère, M. ; Bourbeau, T. ; Simone, R. de ; Gérard, S.: MARTE: Also an UML Profile for Modeling AADL Applications. In: *iceccs* 00 (2007), S. 359–364. `http://dx.doi.org/http:`

//doi.ieeecomputersociety.org/10.1109/ICECCS.2007.29. – DOI http://doi.ieeecomputersociety.org/10.1109/ICECCS.2007.29. ISBN 0–7695–2895–3

[FFPT05] Farcas, E. ; Farcas, C. ; Pree, W. ; Templ, J.: Transparent distribution of real-time components based on logical execution time. In: *LCTES '05: Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*. New York, NY, USA : ACM Press, 2005. – ISBN 1–59593–018–3, S. 31–39

[FGZA03] Falk, C. ; Grechenig, T. ; Zuser, W. ; (Austria), R. B.: A Vehicle Structured based Software Architecture. In: *Software Engineering and Applications*. Marina del Rey, USA, November 2003

[FHS02] Fröhlich, P. ; Hu, Z. ; Schoelzke, M.: Imposing Modeling Rules on Industrial Applications through Meta-modeling. In: *Revised Papers from the HUMACS, DASWIS, ECOMO, and DAMA on ER 2001 Workshops*. London, UK : Springer-Verlag, 2002. – ISBN 3–540–44122–0, S. 166–182

[Föl05] Föllinger, O. ; Hüthig (Hrsg.): *Regelungstechnik. Einführung in die Methoden und ihre Anwendung*. 8. Auflage. Hüthig, 2005

[Fla07] Flammer, M.: *Untersuchungen zur Integration einer AUTOSAR-Architektur und deren Systemdienste in einem FlexRay Cluster auf Basis einer V850-Plattform*. Stuttgart, Dr. Ing. h.c. F. Porsche AG, Diplomarbeit, Januar 2007

[Fle04a] FlexRay Konsortium: FlexRay Communications System Bus Guardian Specification / FlexRay Konsortium. Version 2.0. http://www.flexray.com : FlexRay Konsortium, Juni, 2004. – Forschungsbericht

[Fle04b] FlexRay Konsortium: FlexRay Communications System Electrical Physical Layer Specification / FlexRay Konsortium. Version 2.0. http://www.flexray.com : FlexRay Konsortium, Juni, 2004. – Forschungsbericht

[Fle05a] FlexRay Konsortium: *FlexRay Communications System Protocol Specification*. Version 2.1 Revision A. http://www.flexray.com, Dezember 2005

[Fle05b] FlexRay Konsortium: FlexRay Communications System Protocol Specification Version 2.1 Revision A / FlexRay Konsortium. http://www.flexray.com, Dezember Dezember, 2005. – Forschungsbericht

[Fle06a] FlexRay Konsortium: *Time Budget Taskforce - Status Report 2006-02-23*. Internes Dokument, September 2006

[Fle06b] FlexRay Konsortium: FlexRay Electrical Physical Layer Application Notes / FlexRay Konsortium. Version 2.0. http://www.flexray.com : FlexRay Konsortium, November, 2006. – Forschungsbericht

[FM98] Fraboul, C. ; Martin, F.: Modeling and simulation of integrated modular avionics. In: *In Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Processing*, 1998, S. 102–110

[For02] Forsight: *Foresight Users Guide*. 5. Auflage. McLean, USA, 2000 - 2002

[Fre05] Freymann, R.: Connectivity and Safety. In: *6th European Congress on Intelligent Transport Systems and Services* BMW Group Forschung und Technik, 2005

[FSV99] Ferrari, A. ; Sangiovanni-Vincentelli, A.: System Design: Traditional Concepts and New Paradigms. In: *ICCD '99: Proceedings of the 1999 IEEE International Conference on Computer Design*. Washington, DC, USA : IEEE Computer Society, 1999. – ISBN 0–7695–0406–X, S. 2

[FW99] Ferdinand, C. ; Wilhelm, R.: Efficient and Precise Cache Behavior Prediction for Real-TimeSystems. In: *Real-Time Syst.* 17 (1999), Nr. 2-3, S. 131–181. – ISSN 0922–6443

[GJ75] Garey, M. R. ; Johnson, D. S.: Complexity Results for Multiprocessor Scheduling under Resource Constraints. In: *SIAM J. Comput.* 4 (1975), Nr. 4, S. 397–411

[GJ79] Garey, M. R. ; Johnson, D. S.: *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, 1979 `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/0716710455`. – ISBN 0716710455

[Gri01] Grimm, C.: *Systemtheorie für Informatiker*. Einführung in die Systemtheorie - Vorlesungsunterlagen - Technische Informatik J. W. Goethe-Universität Frankfurt am Main, 2001

[Gri03] Grimm, K.: Software technology in an automotive company: major challenges. In: *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*. Washington, DC, USA : IEEE Computer Society, 2003. – ISBN 0–7695–1877–X, S. 498–503

[Gup95] Gupta, Rajesh K.: *Co-Synthesis of Hardware and Software for Digital Embedded Systems*. Norwell, MA, USA : Kluwer Academic Publishers, 1995. – ISBN 0792396138. – Foreword By-Micheli,, Giovanni De

[H. 06] H. Heinecke et al. - AUTOSAR Development Partnership: AUTOSAR - Current results and preparations for exploitation / 7th EUROFORUM conference SSoftware in the vehicle". Stuttgart, Germany, 3-4 May 2006. – Forschungsbericht

[Har84] Harel, D.: Statecharts: A visual approach to complex systems / The Weizmann Institute of Science - Department of Applied Mathematics. 1984 (CS84-05). – Forschungsbericht

[Har87] Harel, D.: Statecharts: A Visual Formalism for Complex Systems. In: *Science of Computer Programming* 8 (1987), June, Nr. 3, 231–274. `citeseer.ist.psu.edu/harel87statecharts.html`

[Har06] Harrschar, P.: *Future Trends of Embedded Software for Automotive Systems*. 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, März 2006. – Potsdam

[HCRP94] Halbwachs, N. ; Caspi, P. ; Raymond, P. ; Pilaud, D.: The synchronous dataflow programming language LUSTRE / IMAG/LGI, VERILOG. 1994. – Forschungsbericht

[HD92] Hoyme, K. ; Driscoll, K.: SAFEbusTM. In: *11th AIAA/IEEE Digital Avionics Systems Conference* (1992), October, S. 68–73. – Seattle, WA

[HFP+06] Hartmann, J. ; Fleischmann, A. ; Pfaller, C. ; Rappl, M. ; Rittmann, S. ; Wild, D.: Feature Net - ein Ansatz zur Modellierung von automobilspezifischem Domänenwissen und Anforderungen. In: Hochberger, Christian (Hrsg.) ; Liskowsky, Rüdiger (Hrsg.): *GI Jahrestagung (1)* Bd. 93, GI, 2006 (LNI). – ISBN 978–3–88579–187–4, 761-765

[HHJ+] Henia, R. ; Hamann, A. ; Jersak, M. ; Racu, R. ; Richter, K. ; Ernst, R.: *System Level Performance Analysis - the SymTA/S Approach*. `citeseer.ist.psu.edu/745912.html`

[HHK01] Henzinger, T. A. ; Horowitz, B. ; Kirsch, C. M.: Embedded Control Systems Development with Giotto. In: *LCTES/OM*, 2001, 64–72

[HIS07] HIS Konsortium: *Herstellerinitiative Software - Development Partnership -*. `http://www.automotive-his.de`, 2007

[HKL07] Hietl, H. ; Kötz, J. ; Linn, G.: Bereit für FlexRay - FlexRay-Serieneinführung bei Audi. In: *Elektronik automotive* 01 (2007), S. S.32–36

[HOP05] Huber, B. ; Obermaisser, R. ; Peti, P.: MDA-Based Development in the DECOS Integrated Architecture - Modeling the Hardware Platform / Vienna University of Technology - Real-Time Systems Group. Wien, 2005. – Forschungsbericht

[HS02] Huckle, T. ; Schneider, S.: *Numerik für Informatiker*. Berlin : Springer, 2002

[HW04] Honekamp, U. ; Wernicke, M.: Development of Distributed Automotive Software: The DaVinci Methodology. In: *DIPES*, 2004, S. 93–102

[IBM06] IBM Corporation: *Rational RequisitePro*. `http://www-306.ibm.com/software/awdtools/reqpro/`, 2006

[IEC05] IEC/EN: 61508 - Functional safety of E/E/PE safety-related systems / IEC/TR - International Electrotechnical Commission. 1998-2005 (Teil 1 - 5). – Forschungsbericht

[IKB03] IKB Deutsche Industriebank AG: Märkte im Fokus - Automobilindustrie -zunehmender Investitions- und Finanzierungsbedarf / IKB Deutsche Industriebank. Wilhelm-Bötzkes-Straße 1 40474 Düsseldorf, Juni Juni, 2003 (1. Auflage). – IKB Report

[INC07] INCOSE: *INCOSE Systems engineering handbook*. INCOSE, 2007

[Ini07] Initiative, ATESST: The Modelling Approach in ATESST - Overview / Advancing Traffic Efficiency and Safety through Software Technology (ATESST). `http://www.atesst.org`, März 2007 (1.0). – Informeller Report

[Int91] International Standards Organisation (ISO): *International Standard ISO/IEC 9126. Information technology: Software product evaluation: Quality characteristics and guidelines for their use*. 1991

[Int99a] International Telecommunication Union: Series Z: Languages and general Software aspects for telecommunication systems - Formal description techniques (FDT) - Message Sequence Chart (MSC) / ITU-T Telecommunication Standardization Sector of ITU. 1999 (Z.120). – Recommendation

[Int99b] International Telecommunication Union: Series Z: Languages and general Software aspects for telecommunication systems - Formal description techniques (FDT) - Specification and description language (SDL) / ITU-T Telecommunication Standardization Sector of ITU. 1999 (Z.100). – Recommendation

[Int08] International Business Machines Corp.: *IBM Requirements Management - Rational RequisitePro*. `http://www-01.ibm.com/software/awdtools/reqpro/`, 2008

[ISE93] ISE Software: *Building bug-free O-O software: An introduction to design by contract(TM).* `http://archive.eiffel.com/doc/manuals/technology/contract/page.html`, 1993

[ISO93] ISO/IEC 2382-1: *Informationstechnik - Begriffe - Teil 1: Grundbegriffe*. Beuth Verlag, Berlin, 1993

[ISO04] ISO ; ISO (Hrsg.): *Road vehicles – Diagnostics on Controller Area Networks (CAN) – Part 2: Network layer services*. ISO, 2004

[ISO07] ISO/WD 26262: Road Vehicles - Functional Safety / International Standard Organization. 2007. – Arbeitsstand

[Jac75] Jackson, M. A.: *Principles of Program Design*. Orlando, FL, USA : Academic Press, Inc., 1975. – ISBN 0123790506

[JDU+74] Johnson, D. S. ; Demers, A. ; Ullman, J. D. ; Garey, M. R. ; Graham, R. L.: Worst-case Performance Bounds for Simple One-dimensional Packing Algorithms. 3 (1974), Dezember, Nr. 4, S. 299–325. – ISSN 0097–5397 (print), 1095–7111 (electronic)

[Jer03] Jernst: What are the differences between a vocabulary, a taxonomy, a thesaurus, an ontology, and a meta-model? (2003). `http://www.metamodel.com/article.php?story=20030115211223271`

[JHQ+03] Jeckle, M. ; Hahn, J. ; Queins, S. ; Rupp, C. ; Zengler, B. ; Fachbuchverlag, Hanser (Hrsg.): *UML 2 glasklar*. 1. Hanser Fachbuchverlag, 2003. – ISBN 3446225757

[JKR63] Johnson, R. A. ; Kast, F. ; Rosenzweig, J. E.: *The Theory and Management of Systems*. 1. Edition. New York, London : McGraw-Hill, 1963

[Kah74] Kahn, G.: The semantics of a simple language for parallel programming. In: *IFIP 74*. Holland, 1974

[KE04] Kemper, Alfons ; Eickler, Andre: *Datenbanksysteme. Eine Einführung*. 5. Oldenbourg, 2004. – ISBN-10: 3486273922ISBN-13: 978-3486273922

[KG94] Kopetz, H. ; Grünsteidl, G.r: TTP-A Protocol for Fault-Tolerant Real-Time Systems. In: *Computer* 27 (1994), Nr. 1, S. 14–23. http://dx.doi.org/http://dx.doi.org/10.1109/2.248873. – DOI http://dx.doi.org/10.1109/2.248873. – ISSN 0018–9162

[KL91] Kenny, K.B. ; Lin, K.-J.: Building Flexible Real-Time Systems Using the Flex Language. In: *Computer* 24 (1991), Nr. 5, S. 70–78. http://dx.doi.org/http://dx.doi.org/10.1109/2.76288. – DOI http://dx.doi.org/10.1109/2.76288. – ISSN 0018–9162

[KL02] Kalavade, Asawaree ; Lee, Edward A.: The extended partitioning problem: hardware/software mapping, scheduling, and implementation-bin selection. (2002), S. 293–312. ISBN 1–55860–702–1

[Kle06] Kleinod, Ekkart: Modellbasierte Systementwicklung in der Automobilindustrie – Das MOSES-Projekt / Fraunhofer-Institut für Software- und Systemtechnik, Abt. Verlässliche Technische Systeme. Version: April 2006. http://veia.isst.fraunhofer.de/. 2006 (ISST-Bericht 77/06). – Forschungsbericht

[Kle08] Klegraf, A.: Produkt- und Releasemanagement mit eASEE / Vector Informatik GmbH. Vector Kongress 2008, Oktober 2008. – Forschungsbericht

[KLFP02] Kirner, R. ; Lang, R. ; Freiberger, G. ; Puschner, P.: Fully Automatic Worst-Case Execution Time Analysis for Matlab/Simulink Models. In: *ECRTS '02: Proceedings of the 14th Euromicro Conference on Real-Time Systems*. Washington, DC, USA : IEEE Computer Society, 2002. – ISBN 0–7695–1665–3, S. 31

[KNR05] Kuhrmann, M. ; Niebuhr, D. ; Rausch, A.: Application of the V-Modell XT - Report from A Pilot Project. In: Li, Mingshu (Hrsg.) ; Boehm, Barry (Hrsg.) ; Osterweil, Leon J. (Hrsg.): *Unifying the Software Process Spectrum, International Software Process Workshop, SPW 2005, Beijing, China, May 25-27*, Springer, 2005 (Lecture Notes in Computer Science), S. 463–473

[Kop97] Kopetz, H. ; Publishers, Kluwer A. (Hrsg.): *Real-Time Systems - Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, 1997. – ISBN 0–7923–9894–7

[Kop98] Kopetz, H.: The Time-Triggered Architecture. In: *Proceedings of the First International Symposium on Object-Oriented Real-Time Distributed Computing*. Kyoto, Japan, April 1998, S. 22–29

[KP96] Koschke, R. ; Plödereder, E.: Ansätze des Programmverstehens. In: *Softwarewartung und Reengineering - Erfahrungen und Entwicklungen* (1996), S. 159–176

[KP05] Kirner, R. ; Puschner, P.: Classification of WCET Analysis Techniques. In: *Proc. 8th IEEE International Symposium on Object-oriented Real-time distributed Computing*, 2005, S. 190–199

[Kra05] Kraftfahrt-Bundesamt: Jahresbericht 2005 / Kraftfahrt-Bundesamt. 2005. – Forschungsbericht

[Kra07] Kraftfahrt-Bundesamt: Jahresbericht 2007 / Kraftfahrt-Bundesamt. 2007. – Forschungsbericht

[Krü01] Krüger, Ingolf: *Architekturbeschreibung mit UML-RT*. Workshop Architektur eingebetteter Systeme - Technische Universität München - Institut für Informatik - Software & Systems Engineering, Februar 2001

[KT08] Kelly, Steven ; Tolvanen, Juha-Pekka: *Domain-Specific Modeling*. Wiley-IEEE Computer Society Press, 2008

[Kün06] Künzli, S.: *Efficient Design Space Exploration for Embedded Systems*. Zürich, Eidgenössische Technische Hochschule Zürich - Information Technology and Electrical Engineering Department - Computer Engineering and Networks Laboratory, Dissertation, April 2006

[Lam78] Lamport, L.: Time, clocks, and the ordering of events in a distributed system. In: *Commun. ACM* 21 (1978), Nr. 7, S. 558–565. http://dx.doi.org/http://doi.acm.org/10.1145/359545.359563. – DOI http://doi.acm.org/10.1145/359545.359563. – ISSN 0001–0782

[Lap91] Laprie, Jean C.: *Dependability: Basic concepts and terminology in English, French, German, Italian, and Japanese (Dependable computing and fault-tolerant systems)*. Springer-Verlag, 1991 http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/3211822968. – ISBN 3211822968

[Law07] Lawrenz, W. ; Hüthig (Hrsg.): *CAN Controller Area Network - Grundlagen und Praxis*. 5. Auflage. Hüthig, 2007. – ISBN-10: 3778529064

[LEk04] Larses, O. ; El-khoury, J.: Multidisciplinary Modeling and Tool Support for EE Architecture Design. In: *Proceedings FISITA 2004 30th World Automotive Congress,*. Barcelona, Spanien, 23 - 27 Mai, 2004 2004

[LIN03] LIN Consortium ; Report, Technical (Hrsg.): LIN Specification Package 2.0 / LIN Consortium. September, 2003. – Technical Report

[LL73] Liu, C. L. ; Layland, James W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. In: *J. ACM* 20 (1973), Nr. 1, 46-61. http://dblp.uni-trier.de/db/journals/jacm/jacm20.html#LiuL73

[LME98] Li, Y. ; Malik, S. ; Ehrenberg, B.: *Performance Analysis of Real-Time Embeded Software*. Norwell, MA, USA : Kluwer Academic Publishers, 1998. – ISBN 0792383826

[LSP82] Lamport, L. ; Shostak, R. ; Pease, M.: The Byzantine Generals Problem. In: *ACM Trans. Program. Lang. Syst.* 4 (1982), Nr. 3, S. 382–401. http://dx.doi.org/http://doi.acm.org/10.1145/357172.357176. – DOI http://doi.acm.org/10.1145/357172.357176

[Lue97] Luenberger, David G.: *Optimization by Vector Space Methods*. New York, NY, USA : John Wiley & Sons, Inc., 1997. – ISBN 047118117X

[Mai08] Mair, T.: *Modellbasierte Spezifikationstechniken zur Entwicklung und Optimierung von FlexRay-Systemen auf Basis von Metrikanalysen*. Stuttgart, Dr. Ing. h.c. F. Porsche AG, Diplomarbeit, 2008

[Mar97] Marchetto, A.: Structured Definition of Modular Avionics Architectures Using Blueprints. In: *Digital Avionics Systems Conference* Bd. Volume 1, 1997, S. 3.2–24–31

[MB97] Mosnier, F. ; Bortolazzi, J.: Prototyping Car-Embedded Applications. In: Press, IOS (Hrsg.): *Advances in Information Technologies: The Business Challenge*. Amsterdam, 1997, S. 744–751

[McG82] McGraw, J. R.: The VAL language: Description and analysis. In: *ACM TOPLAS*, Januar 1982

[Mea55] Mealy, G. H.: A Method for Synthesizing Sequential Circuits. In: *Bell System Technical Journal* 34 (1955), Nr. 5, S. 1045–1079

[Men08] Mentor Graphics Corp.: *SystemVision - System Integration, Simulation, and Analysis*. `http://www.mentor.com/products/sm/system_integration_simulation_analysis/systemvision/`, Juni 2008

[Met08] MetaCase: *MetaCase - Domain-Specific Modeling with MetaEdit+*. `http://www.metacase.com`, 2008

[MG06] Müller-Glaser, K.D.: *Systems and Software Engineering (SSE) - Vorlesungsunterlagen*. Universität Karlsruhe (TH) - Fakultät für Elektrotechnik & Informationstechnik - Institut für Technik der Informationsverarbeitung, `http://www.itiv.uni-karlsruhe.de/opencms/de/institute/staff/index.html` 2005/2006

[MG07] Müller-Glaser, K.D.: Challenges for reliable software design in automotive electronic control units. In: *12th International Conference on Reliable Software Technologies*. Genua, Juni 2007

[Mic07] Michael, U.: *Prioritäten einer Strategie - Elektronik-Entwicklung bei Porsche*. Fachkongress Automobil-Elektronik, Juli 2007. – Dr. Ing. h.c. F. Porsche AG

[Mil73] Miles, R. F.: *System Concepts: Lectures on Contemporary Approaches to Systems*. New York : Wiley, 1973

[Mir07] Mirabilis Design: *VisualSim Auto/Avionics Bus Modeling Toolkit - FlexRay Network System*. `http://www.mirabilisdesign.com/Pages/Demonstrations/automotive/FlexRay/FlexRay.html`, 2007

[MIS04] MISRA: *MISRA-C:2004 Guidelines for the Use of the C Language in Vehicle Based Software*. Motor Industry Research Association, Nuneaton CV10 0TU, UK, 2004

[MMP01] Murdoch, J. ; McDermid, J.A. ; P.Wilkinson: Failure Modes and Effects Analysis (FMEA) and Systematic Design / University of York - High Integrity Systems Engineering Group; Rolls Royce plc. High Integrity Systems Engineering Group;University of York, York, YO10 5DD UK P.Wilkinson; Rolls Royce plc; PO, 2001. – Forschungsbericht

[Moo56] Moore, E. F.: Gedanken-Experiments on Sequential Machines. In: Shannon, Claude (Hrsg.) ; McCarthy, John (Hrsg.): *Automata Studies*. Princeton, NJ : Princeton University Press, 1956, S. 129–153

[Moo65] Moore, G. E.: Cramming more components onto integrated circuits. In: *Electronics* 38 (1965), April, Nr. 8. ISBN 1–55860–539–8

[MOS04] MOST Cooperation: MOST Specification Rev 2.3 / MOST Cooperation. August, 2004. – Forschungsbericht

[MSH08] Milbredt, P. ; Steininger, A. ; Horauer, M.: Automated Testing of FlexRay Clusters for System Inconsistencies in Automotive Networks. In: *DELTA*, IEEE Computer Society, 2008, 533-538

[Nac08] Nachtigall, Wirtz: *Deskriptive Statistik. Statistische Methoden für Psychologen*. 5. überarbeitete Auflage. Juventa Verlag GmbH, 2008. – ISBN 3779910519

[NAS07] NASA Formal Methods Group: *SPIDER homepage*. `http://shemesh.larc.nasa.gov/fm/spider/`, 2007

[NG97] Nossal, R. ; Galla, T.: Solving NP-Complete Problems in Real-Time System Design by Multichromosome Genetic Algorithms. In: *In Proceedings of the SIGPLAN 1997 Workshop on Languages, Compilers, and Tools for Real-Time Systems, pages 68-76, ACM SIGPLAN, June 1997* (1997), Jun.

[NGH08] Navet, N. ; Grenier, M. ; Havet, L.: Configuring the communication on FlexRay: the case of the static segment. In: *4th European Congress Embedded Real Time Software (ERTS 2008)*. Toulouse, France, 29. Januar - 1. February 2008. 2008

[Nos96] Nossal, R.: Pre-Runtime Planning of a Real-Time Communication System / Technische Universität Wien, Institut für Technische Informatik. Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 1996 (1/1996). – Research Report

[NR05] Näher, U. ; Radtke, P.: Automotive Electronics I: Managing Innovations on the Road. In: *McKinsey Automotive & Assembly* (2005), März, 1-24. `http://autoassembly.mckinsey.com`

[NSF07] Nyenhuis, M. ; Steiner, F. ; Fröhlich, M.: *Das semiaktive Verstelldämpfersystem des BMX X5 mit verteilter Systemarchitektur*. chassis.tech 2007, März 2007. – Technische Universität München, Garching

[OMG01] OMG: Model Driven Architecture (MDA): A technical perspective / Object Management Group. Juli, 2001 (OMG Document No. ormsc/2001-07-01). – Forschungsbericht

[OMG02] OMG: *Meta Object Facility (MOF) Specification*. `http://www.omg.com`, April 2002

[OMG06] OMG: Unified Modeling Language: Superstructure / OMG - Object Management Group. 2006 (V.2.1). – Forschungsbericht

[OSE01] OSEK/VDX: OSEK/VDX Time Triggered Operating System Specification 1.0 / OSEK/VDX. 2001. – Forschungsbericht

[OSE07] OSEK/VDX: *Open systems and the corresponding interfaces for automotive electronics*. http://www.osek-vdx.org, 2001-2007

[Par92] Park, C.Y.: *Predicting deterministic execution times of real-time programs*. Seattle, WA, USA, Diss., 1992

[PASH01] Papadopoulos, Y. ; A.McDermid, J. ; Sasse, R. ; Heiner, G.: Analysis and Synthesis of the Behaviour of Complex Programmable Electronic Systems in Conditions of Failure. In: *Reliability Engineering and System Safety*, Elsevier Science, June 2001, S. 71(3):229–247

[PEPP06] Pop, P. ; Eles, P. ; Peng, Z. ; Pop, T.: Analysis and optimization of distributed real-time embedded systems. In: *ACM Trans. Des. Autom. Electron. Syst.* 11 (2006), Nr. 3, S. 593–625. `http://dx.doi.org/http://doi.acm.org/10.1145/1142980.1142984`. – DOI http://doi.acm.org/10.1145/1142980.1142984. – ISSN 1084–4309

[PF99] Petters, S. M. ; Färber, G.: Making Worst Case Execution Time Analysis for Hard Real-Time Tasks on State of the Art Processors Feasible. In: *RTCSA '99: Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*. Washington, DC, USA : IEEE Computer Society, 1999. – ISBN 0–7695–0306–3, S. 442

[Pik06] Pike, L.: *Formal Verification of Time-Triggered Systems*, Indiana University, Diss., 2006. `citeseer.ist.psu.edu/pike06formal.html`

[PK89] Puschner, P. ; Koza, Ch.: Calculating the maximum, execution time of real-time programs. In: *Real-Time Syst.* 1 (1989), Nr. 2, S. 159–176. `http://dx.doi.org/http://dx.doi.org/10.1007/BF00571421`. – DOI http://dx.doi.org/10.1007/BF00571421. – ISSN 0922–6443

[PK02] Pohjonen, R. ; Kelly, S.: Domain-Specific Modeling. In: *Dr. Dobb's Journal* 27 (2002), August, Nr. 8, `http://www.ddj.com`. `http://www.ddj.com/ftp/2002/2002_08/dsm.txt;http://www.ddj.com/ftp/2002/2002_08/dsm.zip`. – ISSN 1044–789X

[PPE+06] Pop, T. ; Pop, P. ; Eles, P. ; Peng, Z. ; Andrei, A.: Timing Analysis of the FlexRay Communication Protocol. In: *ECRTS '06: Proceedings of the 18th Euromicro Conference on Real-Time Systems*. Washington, DC, USA : IEEE Computer Society, 2006. – ISBN 0–7695–2619–5, S. 203–216

[PT07] Platzner, M. ; Thiele, L.: Skriptum zur Vorlesung Hardware/Software Codesign / ETH Zürich - Institut für Technische Informatik und Kommunikationsnetze. 2007. – Forschungsbericht

[Pus93] Puschner, P.: *Zeitanalyse von Echtzeitprogrammen (Timing Analysis for Real-Time Programs)*. Treitlstr. 1-3/3/182, 1040 Vienna, Austria, Technische Universität Wien, Institut für Technische Informatik, Diss., 1993

[PV97a] Puschner, P. ; Vrchoticky, A.: Problems in Static Worst-Case Execution Time Analysis. In: Irmscher, Klaus (Hrsg.) ; Mittasch, Christian (Hrsg.) ; Richter, Klaus (Hrsg.): *MMB (Kurzbeiträge)*, TU Bergakademie Freiberg, 1997. – ISBN 3–86012–045–X, 18-25

[PV97b] Puschner, Peter P. ; Vrchoticky, Alexander: Problems in Static Worst-Case Execution Time Analysis. In: *MMB (Kurzbeiträge)*, 1997, S. 18–25

[Que06] Quecke, H. ; Vector Informatik GmbH (Hrsg.): *DaVinci Network Designer CAN - Message Timing User Documentation*. Version 1.0. Vector Informatik GmbH, Februar 2006

[R. 95] R. Shishko et al. ; Aeronautics, Headquarters: N. (Hrsg.) ; Administration., Space (Hrsg.): *NASA Systems Engineering Handbook*. SP-6105. Shishko, R. (1995). NASA Systems Engineering Handbook, SP-6105. Headquarters: National Aeronautics and Space Administration., 1995

[Rai02] Raisch, A.: *Calculation of Signal Latencies and Bus Loads in Vehicle Networks*. Vector Congress 2002, September 2002. – Vector Informatik GmbH

[Raj05] Rajamani, R.: *Vehicle Dynamics and Control*. Springer Science+Business Media, 2005 (Mechanical Engineering)

[Rau07a] Rausch, M. ; Fachbuchverlag, Hanser (Hrsg.): *FlexRay*. Bd. 1. Auflage. Hanser Fachbuchverlag, 2007

[Rau07b] Rausch., M.: *FlexRay: Grundlagen, Funktionsweise, Anwendung*. 1. Auflage. München : Hanser, 2007

[RE02] Richter, K. ; Ernst, R.: *Event model interfaces for heterogeneous system analysis*. `citeseer.ist.psu.edu/richter02event.html`. Version: 2002

[Ric07] Richter, K.: Schedulinganalysen für FlexRay - Warum? In: *DESIGN und ELEKTRONIK Entwicklerforum 07*. Ludwigsburg, 2007

[Ric08] Richter, K.: Defining a Timing Model for AUTOSAR - Status and Challenges. In: *Software Engineering Conference - Workshop Automotive Software Engineering: Forschung, Lehre, Industrielle Praxis*. München, Februar 2008

[Rin02] Ringler, T.: *Entwicklung und Analyse zeitgesteuerter Systeme*. Stuttgart, Universität Stuttgart, Fakultät für Informatik, Elektrotechnik und Informationstechnik, Institut für Automatisierungs- und Softwaretechnik, Dissertation, November 2002

[RK07] Rajnak, A. ; Kumar, A.: Computer-aided architecture design & optimized implementation of distributed automotive EE systems. In: *DAC '07: Proceedings of the 44th annual conference on Design automation*. New York, NY, USA : ACM, 2007. – ISBN 978–1–59593–627–1, S. 556–561

[RMBK07] Rzepka, M. ; Mickisch, W. ; Brandstätter, W. ; Kaindl, M.: *Validation of FlexRay(tm) networks - interoperability test concepts and experiences*. FlexRay Product Day 2007, November 2007. – Fellbach, Stuttgart

[Rob91] Robert Bosch GmbH ; ISO11898, Technical R. (Hrsg.): CAN Specification Version 2.0 / Robert Bosch GmbH. Robert Bosch GmbH, 1991. – Technical Report ISO11898

[Rob02] Robert Bosch GmbH - Automotive Electronics Semiconductors and Integrated Circuits - Digital CMOS Design Group: *TTCAN IP Module User's Manual*. Revision 1.6, November, 2002

[Rom06] Romberg, Jan: *Synthesis of distributed systems from synchronous dataflow programs*. München, Technische Universität München - Fakultät für Informatik - Lehrstuhl für Software- und Systems Engineering, Dissertation, Juni 2006

[RWV97] Råde, L. ; Westergren, B. ; Vachenauer, P.: *Springers mathematische Formeln : Taschenbuch für Ingenieure, Naturwissenschaftler, Wirtschaftswissenschaftler*. 2., korrigierte und erw. Aufl. Springer, 1997 `http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+227443535&sourceid=fbw_bibsonomy`. – ISBN 3–540–62829–0

[RZE+02] Richter, K. ; Ziegenbein, D. ; Ernst, R. ; Thiele, L. ; Teich, J.: *Model composition for scheduling analysis in platform design*. `citeseer.ist.psu.edu/richter02model.html`. Version: 2002

[SAHP97] Stappert, Friedhelm ; Altenbernd, Peter ; Hard, Straight line ; Programs, Real time: Complete Worst-Case Execution Time Analysis of Straight-line Hard Real-Time Programs. In: *Journal of Systems Architecture* 46 (1997), S. 200–0

[Sal06] Salzmann, C.: AUTOSAR - First Experiences and the Migration Strategy of the BMW Group / BMW Car IT GmbH. ARTIST2 WS - `http://www.artist-embedded.org`, März 2006. – Forschungsbericht

[Saw] Sawodny, O.: *Systeme mit verteilten Parametern - Systemeigenschaften - Klassifikation von Systemen*. – Universität Stuttgart - ISYS

[SBV+02] Scheidler, C. ; Boutin, S. ; Virnich, U. ; Rennhack, J. ; Grünsteidl, G. ; Pisecky, M. ; Lang, R. ; Kirner, R. ; Papadopoulos, Y.: Systems Engineering von zeitgesteuerten Systemen - die SETTA Methodik. In: *VDI/VDE GMA Fachtagung, Steuerung und Regelung von Fahrzeugen und Motoren - AutoReg 2002*. Mannheim, Germany, April 2002, S. 663–676

[Sch96] Schedl, A.: *Design and Simulation of Clock Synchronization in Distributed Systems*, TU Wien, Dissertation, 1996

[Sch06] Schlosser, J.: *Architektursimulation von verteilten Steuergerätesystemen*. Berlin, Technische Universität München, Fakultät für Informatik, Diss., 2006

[Sch07] Schedl, A.: *Goals and Architecture of FlexRay at BMW*. Vector FlexRay Symposium Stuttgart, März 2007

[Sch08] Schneider, T.: *Analyse und Konzeption von Berechnungsvorschriften zur Parametrisierung der FlexRay-Bussysteme*, Universität Karlsruhe (TH), Diplomarbeit, 2008

[Sha89] Shaw, A.: Reasoning about time in higher-level language software. In: *IEEE Transactions on Software Engineering* 15 (1989), S. 875–889

[SHH+03] Schätz, B. ; Hain, T. ; Houdek, F. ; Prenninger, W. ; Rappl, M. ; Romberg, J. ; Slotosch, O. ; Strecker, M. ; Wisspeintner, A.: CASE Tools for Embedded Systems / Technische Universität München. 2003 (TUM-I0309). – Forschungsbericht

[Sim99] Simon, D. E.: *An Embedded Software Primer*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1999. – ISBN 020161569X

[SK06] Steiner, W. ; Kopetz, H.: The Startup Problem in Fault-Tolerant Time-Triggered Communication, 2006

[SN03] Saket, R. ; Navet, N.: Frame Packing Algorithms for Automotive Applications, / Rapport de recherche de l'INRIA. 2003 (INRIA RR-4998). – Forschungsbericht

[SNA00] Sandstrom, K. ; Norstom, C. ; Ahlmark, M.: Frame packing in real-time communication. In: *rtcsa* 00 (2000), S. 399. `http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/RTCSA.2000.896418`. – DOI http://doi.ieeecomputersociety.org/10.1109/RTCSA.2000.896418. – ISSN 1533–2306

[Som01] Sommerville, I.: *Software Engineering*. Bd. 6. Auflage. Pearson Studium, 2001. – ISBN 3827370019

[SP96] Srinivas, M. ; Patnaik, L. m.: Genetic Search: Analysis Using Fitness Moments. In: *IEEE Trans. on Knowl. and Data Eng.* 8 (1996), Nr. 1, S. 120–133. `http://dx.doi.org/http://dx.doi.org/10.1109/69.485641`. – DOI http://dx.doi.org/10.1109/69.485641. – ISSN 1041–4347

[Spr96] Spreng, M.: *Rapid Prototyping elektronischer Steuerungssysteme in der Automobilentwicklung*. Karlsruhe, Universität Karlsruhe (TH) - Fakultät für Elektrotechnik - Institut für Technik der Informationsverarbeitung, Dissertation, 1996

[SSV06] Scharnhorst, T. ; Schwab, G. ; Valentini, H.-D.: *Anforderungen an robuste E/E-Systeme im Fahrzeug*. 10. Jahrestagung Ëlektronik-Systeme im Automobil", Februar 2006

[Sta73] Stachowiak, H.: *Allgemeine Modelltheorie*. 1973 `http://www.amazon.de/Allgemeine-Modelltheorie-Herbert-Stachowiak/dp/3211811060/ref=sr_1_2/028-1073608-4157317?ie=UTF8&s=books&qid=1190302420&sr=1-2`

[Sta01] Stauner, T.: *Systematic Development of Hybrid Systems*. München, Technische Universität München - Fakultät für Informatik - Lehrstuhl für Software- und Systems Engineering, Dissertation, 2001

[Sto07] Storjohann, K.: Durchgängiges E/E-Produktdokumentations- und Change-Management - (DEEP-C) / DaimlerChrysler AG. VECTOR FORUM, 2007. – Forschungsbericht

[Sur08] Surmann, T.: *Konfigurierung, Implementierung und Analyse eines FlexRay-Netzwerkes*. Stuttgart, Dr. Ing. h.c. F. Porsche AG, Studienarbeit, 2008

[Sus07] Sussman, G. J.: *Building Robust Systems an essay*. Massachusetts Institute of Technology, Januar 2007

[SW07] Scharnhorst, T. ; Wiechmann, S.: *Zuverlässigkeit von Fahrzeugelektronik*. 11. Euroforum Jahrestagung Ëlektronik-Systeme im Automobil", Februar 2007. – München

[Syn08] Synopsis Corp.: *Saber Design and Simulation*. `http://www.synopsys.com/products/mixedsignal/saber/saber.html`, Juni 2008

[Sys06] SysML Partners: *OMG SysML Specification V.1.0 (Final Adopted Specification)*. `http://www.sysml.org`, Mai 2006

[SZ03] Schäuffele, J. ; Zurawka, T.: *Automotive Software Engineering*. 1. Auflage. Vieweg Verlag, 2003. – ISBN 3–528–01040–1

[Tan00] Tanenbaum, A. S.: *Computernetzwerke*. 3. revidierte Auflage. München : Pearson Studium, 2000. – ISBN 3–8273–7011–6

[TB94] Tindell, K. ; Burns, A.: Guaranteed Message Latencies for Distributed Safety-Critical Hard Real Time Control Networks / Dept. Computer Science, Univ. of York. York, UK, 1994. – Forschungsbericht

[TC94] Tindell, K. ; Clark, J.: Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. In: *Microprocessing and Microprogramming* 40 (1994), S. 117–134

[Tec08] Technologies, Esterel: *Scade Suite*. `http://www.esterel-technologies.com/products/scade-suite/`, Oktober 2008

[Tel06] Telelogic AB: *Telelogic DOORS*. `http://www.telelogic.de/products/doors/index.cfm`, 2006

[Teu08] Teuchert, S.: Technik treibt die Organisation - der ganzheitlich implementierte funktionsorientierte Systementwicklungsprozess bei MAN / MAN Nutzfahrzeuge AG. Vector Kongress, 2008 2008. – Forschungsbericht

[Tha00] Thane, H.: *Monitoring, Testing and Debugging of Distributed Real-Time Systems*. Stockholm, Royal Institute of Technology Sweden - Department of Machine Design - Mechatronics Laboratory, Dissertation, Mai 2000. `http://www.mrtc.mdh.se/index.php?choice=publications&id=0242`

[The07a] The MathWorks Inc.: *Matlab*. `http://www.mathworks.com/products/matlab/`, 2007

[The07b] The MathWorks Inc.: *Stateflow*. `http://www.mathworks.com/products/stateflow/`, 2007

[Trö05] Tröster, F.: *Steuerungs- und Regelungstechnik für Ingenieure*. 2. Auflage. Oldenbourg Verlag, 2005

[TS06] Tanenbaum, A. S. ; Steen, M. van ; Hall, Prentice (Hrsg.): *Distributed Systems. Principles and Paradigms.* 2. Auflage. Prentice Hall, 2006. – ISBN 0132392275

[TTA07] TTAutomotive Software GmbH: *TTXPlan - The FlexRay Cluster Design Tool*. `http://www.ttautomotive.de/produkte/ttx-plan.htm`, April 2007. – Version 1.2.46

[TTT03] TTTech: *TTP Time-Triggered Protocol TTP/C - High-Level Specification Document - Protocol Version 1.1 - Spezifikation Edition 1.4.3*. `http://www.ttagroup.com/technology/specification.htm`, November 2003

[Ung07] Ungermann, J.: Calculation of the FlexRay Clock Precision / Philips Research. Aachen, Dezember 2007 (TN-2007-00904). – Technical note. – Unclassified Internal Document

[Vec07] Vector Informatik GmbH ; Vector Informatik GmbH (Hrsg.): *DaVinci Network Designer FlexRay Object Properties*. Version 1.1. Vector Informatik GmbH, 2007

[VG92] Vahid, F. ; Gajski, D. D.: Specification partitioning for system design. In: *DAC '92: Proceedings of the 29th ACM/IEEE conference on Design automation*. Los Alamitos, CA, USA : IEEE Computer Society Press, 1992. – ISBN 0–89791–516–X, S. 219–224

[VHHM03] Versteegen, G. ; Hesseler, A. ; Hood, C. ; Missling, C. ; Verlag, Springer (Hrsg.): *Anforderungsmanagement*. 1. Auflage. Berlin : Springer Verlag, 2003

[VR01] Verissimo, P. ; Rodrigues, L.: *Distributed Systems for System Architects*. Norwell, MA, USA : Kluwer Academic Publishers, 2001. – ISBN 0792372662

[Wal08] Walz, S.: Integrated EE-Architecture Design / Mentor Graphics (Deutschland) GmbH. Synergy Forum Äircraft meets AutomotiveSSystem Architectures, Mai 2008. – Forschungsbericht

[Web94] Weber, M.: *Verteilte Systeme*. Spektrum Akademischer Verlag, 1994. – ISBN 0–07–113688–1

[WEE+08] Wilhelm, R. ; Engblom, J. ; Ermedahl, A. ; Holsti, N. ; Thesing, S. ; Whalley, D. ; Bernat, G. ; Ferdinand, C. ; Heckmann, R. ; Mueller, F. ; Puaut, I. ; Puschner, P. ; Staschulat, J. ; Stenström, P.: The Determination of Worst-Case Execution Times—Overview of the Methods and Survey of Tools. 7 (2008), Nr. 3. – ACM Transactions on Embedded Computing Systems (TECS)

[Wei06] Weilkiens, T.: *Systems Engineering mit SysML/UML Modellierung, Analyse, Design*. Dpunkt.Verlag GmbH, 2006. – ISBN 3–89864–409–X

[Wie02] Wieringa, R. J.: *Design Methods for Software Systems: YOURDON, Statemate and Uml*. Science & Technology Books, 2002. – ISBN 1558607552

[WL88] Welch, J. L. ; Lynch, N.: A new fault-tolerant algorithm for clock synchronization. In: *Inf. Comput.* 77 (1988), Nr. 1, S. 1–36. `http://dx.doi.org/http://dx.doi.org/10.1016/0890-5401(88)90043-0`. – DOI http://dx.doi.org/10.1016/0890–5401(88)90043–0. – ISSN 0890–5401

[WP07] W.Böge ; Plassmann, W. ; Vieweg+Teubner (Hrsg.): *Vieweg Handbuch Elektrotechnik: Grundlagen und Anwendungen für Elektrotechniker*. 4. überarb. A. Vieweg+Teubner, 2007

[WW03] Weber, M. ; Weisbrod, J.: Requirements Engineering in Automotive Development: Experiences and Challenges. In: *IEEE Software* 20 (2003), Nr. 1, S. 16–24. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/MS.2003.1159025. – DOI http://doi.ieeecomputersociety.org/10.1109/MS.2003.1159025. – ISSN 0740–7459

[WW04] Wenzel, M. ; Wille, M.: Bedien- und Anzeigeprotokoll trennt Funktion und Gestaltung. In: *Elektronik Automotive* 6 (2004), S. 30–32

[Wym76] Wymore, W. ; Wiley (Hrsg.): *Systems Engineering Methodology for Interdisciplinary Teams*. New York, 1976

[YC75] Yourdon, E. ; Constantine, L. L.: *Structured Design*. Yourdon Press, 1975

[zei08] zeitware: *TDL Technology*. http://www.zeitware.org/, Oktober 2008

[Zim88] Zimmermann, H.: OSI reference model—The ISO model of architecture for open systems interconnection. (1988), S. 2–9. ISBN 0–89006–337–0

[ZS06] Zimmermann, W. ; Schmidgall, R.: *Bussysteme in der Fahrzeugtechnik*. Vieweg Verlag, April, 2006

ANHANG A

Graphische Analyse des Nutzdatenvolumens $N_v(x)$ im statischen FlexRay-Segment

Sei $f : D \in R \rightarrow R$ eine Abbildung mit $f(x) := \frac{L}{x} * (x - c(x))$ und $D = [a, b]$. Die Funktion $f(x)$ stellt die Menge der Nutzdaten im statischen Segment pro Zyklus dar. Die Variable x sei die Länge eines Slots im statischen Segment, der fixe Parameter L die Länge des statischen Segments pro Zyklus und c der Overhead, der pro Slot im statischen Segment anfällt. Damit bildet $x - c(x)$ die Menge der Nutzdaten pro Slot. Man nehme einen linearen Verlauf von $c(x)$ an und definiere $c(x) := 1 + \frac{x}{10}$.

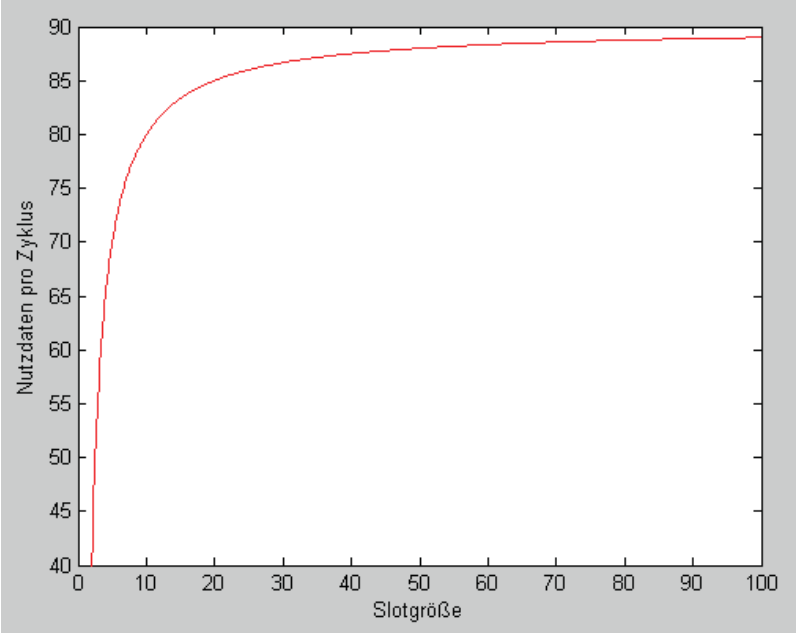


Abbildung A.1.: Die Funktion $f(x) = \frac{L}{x} * (x - c(x))$

Zur Veranschaulichung wird die Funktion $f(x)$ im Definitionsintervall $D = [2, 100]$ in Abb. A.1) gezeichnet.Der Verlauf des Graphen erscheint plausibel. Je größer ein Slot gewählt wird, desto mehr Nutzdaten können pro Zyklus im statischen Segment übertragen werden.

Sei nun $f : D \in R^2 \rightarrow R$ eine Abbilung mit $f(x, L) := \frac{L}{x} * (x - c(x))$ und $D = [a, b]^2$. Die Funktion $f(x, L)$ stellt die Menge der Nutzdaten im statischen Segment pro Zyklus dar. Zu beachten ist dabei, dass hier x und L variabel sind. Die Bedeutung der Variablen x und L behalten ihre Gültigkeit. Man definiere wieder $c(x) := 1 + \frac{x}{10}$.

Zur Veranschaulichung wird die Funktion $f(x, L)$ im Definitionsbereich $D = [2, 100]^2$ in Abb. A.2) gezeichnet. Der Verlauf des Graphen erscheint auch hier plausibel, da abhängig von der Zykluslänge L bei konstanter Slotgröße x die Nutzdaten pro Zyklus $f(x, L)$ ansteigen. Werden die Zykluslänge L und die Slotgröße x möglichst groß gewählt, so können auch am meisten Nutzdaten pro Zyklus $f(x, L)$ übertragen werden.

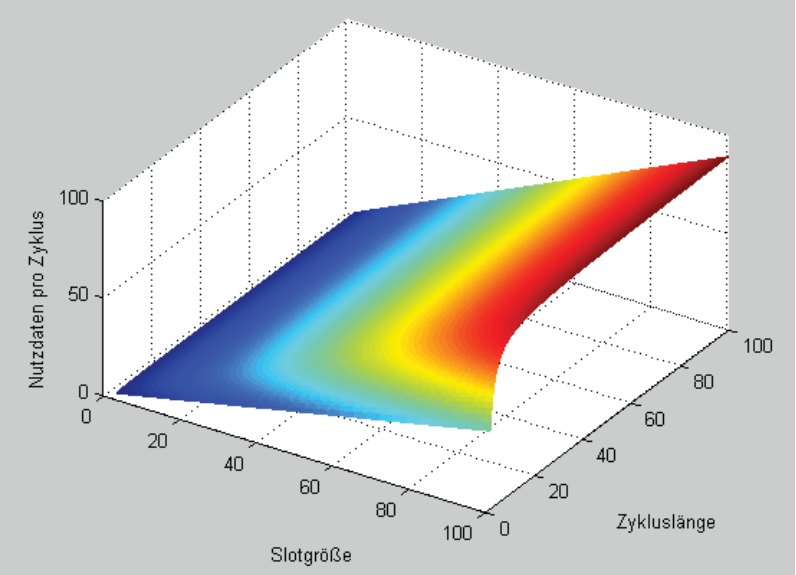


Abbildung A.2.: Die Funktion $f(x, L) = \frac{L}{x} * (x - c(x))$

A.1. Definition des Nutzdateneffizienz-Funktionals $N_e(x)$

Definition A.1.1 (Nutzdateneffizienz-Funktional) *Sei $X = \mathbb{R}^5$ ein normierter Vektorraum und $N_e : \mathbb{D}_e{}^1 \subset X \rightarrow Y = \mathbb{R}$ in den Nicht-Sprungstellen eine Frechet-differenzierbare Abbildung. Und es gilt:*

$$N_e(x) = \frac{1}{x_7 * (2 * x_4 + ceil(\frac{(x_1+c_1+c_2+20*x_2+c_3+c_5)*(c_4*x_3*(1+c_6))+\alpha_1+\alpha_3}{x_7*(1-c_6)}))} * 16 * c_4 * x_3 * x_2$$

Dann heißt $N_e(x)$ Nutzdateneffizienz-Funktional für eine FlexRay-Parameterkonfiguration. Ausgedrückt mit dem Funktional $k_3(x)$ lässt sich $N_e(x)$ auch schreiben als :

[1]Der Definitionsbereich $\mathbb{D}_e$ wird in Definition A.3.2 erklärt

$$N_e(x) = \frac{1}{x_7 * (2 * x_4 + k_3(x))} * 16 * c_4 * x_3 * x_2$$

□

A.2. Definition der Restriktionsabbildung $G_e(x)$

Sämtliche Restriktionen zur Eingrenzung der Berechnung von $N_e(x)$ werden in der folgenden Definition in einer Abbildung zusammengefasst.

Definition A.2.1 (Restriktionsabbildung $G_e(x)$) *Sei $x \in \mathbb{D}_e \subset \mathbb{R}^5$. Dann heißt die in den Nicht-Sprungstellen Frechet-differenzierbare Abbildung $G_e : \mathbb{D}_e \to \mathbb{R}^{26}$ mit*

$$G_e(x) = \begin{pmatrix} g_1(x) \\ g_2(x) \\ g_3(x) \\ \vdots \\ g_{26}(x) \end{pmatrix}$$

Restriktionsabbildung des Nutzdateneffizienz-Funktionals $N_e(x)$.

□

A.3. Definitionsbereich für N_v und N_e

Definition A.3.1 (Definitionsbereich $\mathbb{D}_v$) *Der Definitionsbereich des Nutzdatenvolumen-Funktionals $N_v(x)$ wird mit $\mathbb{D}_v \subset \mathbb{R}^8$ bezeichnet. Es gilt:*

$$\mathbb{D}_v = [3, 15] \times [0, 127] \times [0.0125, 0.05] \times [1, 63] \times [1, 6] \times [10, 16000] \times [2, 805] \times [1, 31]$$

□

Definition A.3.2 (Definitionsbereich $\mathbb{D}_e$) *Der Definitionsbereich des Nutzdateneffizienz-Funktionals $N_e(x)$ wird mit $\mathbb{D}_e \subset \mathbb{R}^5$ bezeichnet. Es gilt:*

$$\mathbb{D}_e = [3, 15] \times [0, 127] \times [0.0125, 0.05] \times [1, 63] \times [1, 6]$$

□

Bemerkung: Der Definitionsbereich $\mathbb{D}_v$ wird durch die FlexRay-Spezifikation /[Fle05a]/ vorgeschrieben.

A.4. Graphische Darstellung des modifizierten N_v

In Abb. A.3 wird das modifizierte Nutzdatenvolumen-Funktional $N_v(x_9, x_{10})$ für den errechneten optimalen Lösungsvektor x_{opt}, der alle FlexRay-Restriktionen erfüllt, dargestellt.

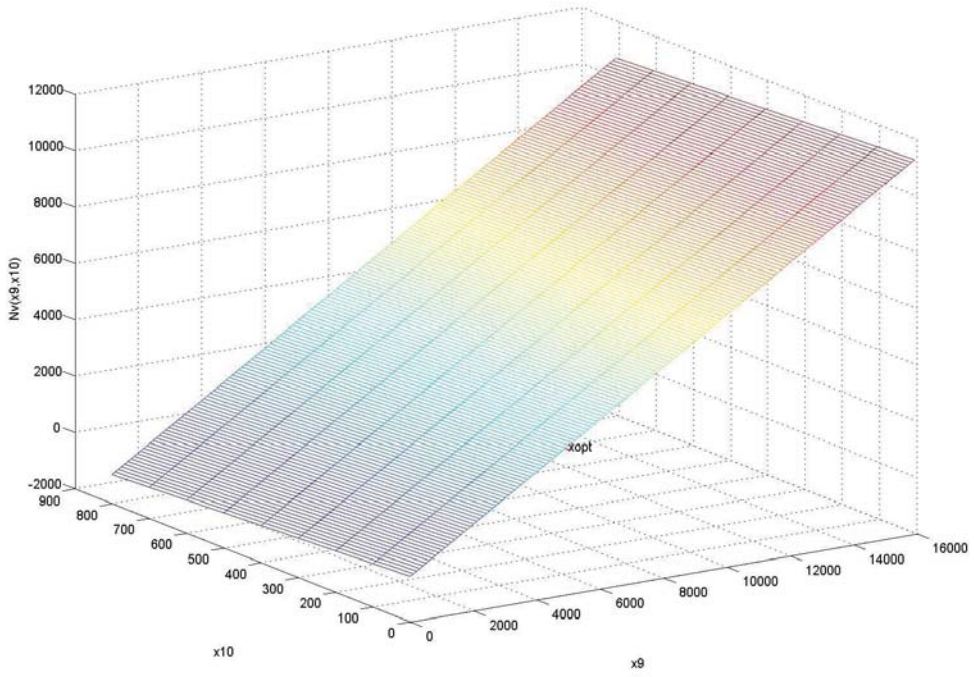


Abbildung A.3.: Das Nutzdatenvolumen-Funktional $N_v(x_9, x_{10})$ mit dem gefundenen Lösungsvektor x_{opt}

ANHANG B

Spline-Koeffizienten und Funktionswerte der Auswahlfunktion

Bei der konkrete Problemstellung mit $M = 21$ erhält man die in Tabelle B.1 aufgelisteten Koeffizienten für die Spline-Funktion $s_m(t)$. Die Tabelle B.2 zeigt die Funktionswerte der Auswahlfunktion $\chi_m(t)$.

Spline	a_m	b_m	c_m	d_m
$s_1(t)$	0.0000	-0.0003	0.0050	0
$s_2(t)$	0.0000	-0.0002	0.0006	0.0260
$s_3(t)$	-0.0000	0.0001	-0.0012	0.0053
$s_4(t)$	0.0000	-0.0000	0.0001	0.0001
$s_5(t)$	0.0000	0.0000	0.0001	0.0013
$s_6(t)$	0.0000	0.0000	0.0012	0.0123
$s_7(t)$	-0.0000	0.0001	0.0031	0.0534
$s_8(t)$	-0.0000	0.0000	0.0046	0.1322
$s_9(t)$	-0.0000	-0.0001	0.0034	0.2193
$s_{10}(t)$	0.0000	-0.0001	-0.0011	0.2460
$s_{11}(t)$	0.0000	-0.0001	-0.0050	0.1789
$s_{12}(t)$	-0.0000	0.0001	-0.0042	0.0770
$s_{13}(t)$	-0.0000	0.0000	-0.0013	0.0247
$s_{14}(t)$	-0.0000	0.0000	-0.0003	0.0116
$s_{15}(t)$	0.0000	-0.0000	-0.0002	0.0075
$s_{16}(t)$	-0.0000	0.0000	-0.0002	0.0033
$s_{17}(t)$	-0.0000	0.0000	-0.0001	0.0008
$s_{18}(t)$	-0.0000	0.0000	-0.0000	0.0001
$s_{19}(t)$	0.0000	-0.0000	0.0000	0.0000
$s_{20}(t)$	0.0000	0.0000	-0.0000	0

Tabelle B.1.: Spline-Koeffizienten von $s_m(t)$

Auswahlfunktion	Gültigkeitsintervall
$\chi_1(t)$	$0 \leq t \leq 10$
$\chi_2(t)$	$10 \leq t \leq 30$
$\chi_3(t)$	$30 \leq t \leq 50$
$\chi_4(t)$	$50 \leq t \leq 70$
$\chi_5(t)$	$70 \leq t \leq 90$
$\chi_6(t)$	$90 \leq t \leq 110$
$\chi_7(t)$	$110 \leq t \leq 130$
$\chi_8(t)$	$130 \leq t \leq 150$
$\chi_9(t)$	$150 \leq t \leq 170$
$\chi_{10}(t)$	$170 \leq t \leq 190$
$\chi_{11}(t)$	$190 \leq t \leq 210$
$\chi_{12}(t)$	$210 \leq t \leq 230$
$\chi_{13}(t)$	$230 \leq t \leq 250$
$\chi_{14}(t)$	$250 \leq t \leq 270$
$\chi_{15}(t)$	$270 \leq t \leq 290$
$\chi_{16}(t)$	$290 \leq t \leq 310$
$\chi_{17}(t)$	$310 \leq t \leq 330$
$\chi_{18}(t)$	$330 \leq t \leq 350$
$\chi_{19}(t)$	$350 \leq t \leq 370$
$\chi_{20}(t)$	$370 \leq t \leq 390$

Tabelle B.2.: Für den Auswahlfunktionswert $\chi_m(t)$ gilt innerhalb des jeweiligen Gültigkeitsintervalls $\chi_m(t) = 1$, sonst $\chi_m(t) = 0$

ANHANG C

Parameter Formalisierung

C.1. Variablendefinitionen

Variablendefinition zur Abbildung von FlexRay-Parameterbezeichnungen in mathematische Berechnungsterme.

$$x_1 := gdTSSTransmitter \tag{C.1}$$

$$x_2 := gPayloadLengthStatic \tag{C.2}$$

$$x_3 := gdSampleClockPeriod \tag{C.3}$$

$$x_4 := gdActionPointOffset \tag{C.4}$$

$$x_5 := gdBitMax \tag{C.5}$$

$$x_6 := gdMaxPropagationDelay \tag{C.6}$$

$$x_7 := gdMacrotick \tag{C.7}$$

$$x_8 := gNumberOfMinislots \tag{C.8}$$

$$x_9 := gMacroPerCycle \tag{C.9}$$

$$x_{10} := gdNIT \tag{C.10}$$

$$x_{11} := gNumberOfStaticSlots \tag{C.11}$$

$$x_{12} := gdMinislot \tag{C.12}$$

$$x_{13} := adActionPointDifference \tag{C.13}$$

$$x_{14} := gdSymbolWindow \tag{C.14}$$

$$x_{15} := gColdStartAttempts \tag{C.15}$$

$$x_{16} := gdCASRxLowMax \tag{C.16}$$

$$x_{17} := gdDynamicSlotIdlePhase \tag{C.17}$$

$$x_{18} := gdMinislotActionPointOffset \tag{C.18}$$

$$x_{19} := gdStaticSlot \tag{C.19}$$

$$x_{20} := gdWakeupSymbolRxIdle \tag{C.20}$$

$$x_{21} := gdWakeupSymbolRxLow \tag{C.21}$$

$$x_{22} := gdWakeupSymbolRxWindow \tag{C.22}$$

$$x_{23} := gdWakeupSymbolTxIdle \tag{C.23}$$

$$x_{24} := gdWakeupSymbolTxLow \tag{C.24}$$

$$x_{25} := gListenNoise \tag{C.25}$$

$$x_{26} := gMaxWithoutClockCorrectionFatal \tag{C.26}$$

$$x_{27} := gMaxWithoutClockCorrectionPassive \tag{C.27}$$

$$x_{28} := gOffsetCorrectionStart \tag{C.28}$$

$$x_{29} := gSyncNodeMax \tag{C.29}$$

$$x_{30} := gAssumedPrecision \tag{C.30}$$

$$x_{31} := gChannels \tag{C.31}$$

$$x_{32} := gClusterDriftDamping \tag{C.32}$$

$$x_{33} := gdBit \tag{C.33}$$

$$x_{34} := gdBitMin \tag{C.34}$$

$$x_{35} := gdCycle \tag{C.35}$$

$$x_{36} := gdMaxInitializationError \tag{C.36}$$

$$x_{37} := gdMaxMicrotick \tag{C.37}$$

$$x_{38} := gdMinPropagationDelay \tag{C.38}$$

$$x_{39} := gNetworkManagementVectorLength \tag{C.39}$$

$$x_{40} := gOffsetCorrectionMax \tag{C.40}$$

$$x_{41} := pdMicrotick \tag{C.41}$$

$$x_{42} := pdAcceptedStartupRange \tag{C.42}$$

$$x_{43} := pClusterDriftDamping \tag{C.43}$$

$$x_{44} := dBDRxai \tag{C.44}$$

$$x_{45} := pMicroPerCycle \tag{C.45}$$

$$x_{46} := pRateCorrectionOut \tag{C.46}$$

$$x_{47} := pOffsetCorrectionOut \tag{C.47}$$

$$x_{48} := pExternRateCorrection \tag{C.48}$$

$$x_{49} := pExternOffsetCorrection \tag{C.49}$$

$$x_{50} := pdMaxDrift \tag{C.50}$$

$$x_{51} := pdListenTimeout \tag{C.51}$$

$$x_{52} := pDecodingCorrection \tag{C.52}$$

$$x_{53} := pSamplesPerMicrotick \tag{C.53}$$

$$x_{54} := pMacroInitialOffset \tag{C.54}$$

$$x_{55} := pDelayCompensation \tag{C.55}$$

$$x_{56} := pMicroPerMacroNom \tag{C.56}$$

$$x_{57} := pMicroInitialOffset \tag{C.57}$$

$$x_{58} := pLatestTx \tag{C.58}$$

$$x_{59} := dBDRxia \tag{C.59}$$

$$x_{60} := max(x|x = x_{67}) \tag{C.60}$$

$$x_{61} := dStarTruncation \tag{C.61}$$

$$x_{62} := adOffsetCorrection \tag{C.62}$$

$$x_{63} := adRemRateCalculation \tag{C.63}$$

$$x_{64} := adRemOffsetCalculation \tag{C.64}$$

$$x_{65} := adTxMax \tag{C.65}$$

$$x_{66} := aFrameLengthDynamic \tag{C.66}$$

$$x_{67} := nStarPath_{M,N} \tag{C.67}$$

C.2. Konstantendefinitionen

Konstantendefinition zur Abbildung von FlexRay-Konstantenbezeichnungen in mathematische Berechnungsterme.

$$c_1 := cdFSS \tag{C.68}$$

$$c_2 := 80 \tag{C.69}$$

$$c_3 := cdFES \tag{C.70}$$

$$c_4 := cSamplesPerBit \tag{C.71}$$

$$c_5 := cChannelIdleDelimiter \tag{C.72}$$

$$c_6 := cClockDeviationMax \tag{C.73}$$

$$c_7 := cCASActionPointOffset \tag{C.74}$$

$$c_8 := cdMaxOffsetCalculation \tag{C.75}$$

$$c_9 := cCrcInit[A] \tag{C.76}$$

$$c_{10} := cCrcInit[B] \tag{C.77}$$

$$c_{11} := cCrcPolynomial \tag{C.78}$$

$$c_{12} := cCycleCountMax \tag{C.79}$$

$$c_{13} := cdBSS \tag{C.80}$$

$$c_{14} := cdCAS \tag{C.81}$$

$$c_{15} := cdCASRxLowMin \tag{C.82}$$

$$c_{16} := cdCycleMax \tag{C.83}$$

$$c_{17} := cdMaxMTNom \tag{C.84}$$

$$c_{18} := cdMinMTNom \tag{C.85}$$

$$c_{19} := cdWakeupMaxCollision \tag{C.86}$$

$$c_{20} := cdWakeupSymbolTxLow \tag{C.87}$$

$$c_{21} := cdWakeupSymbolTxIdle \tag{C.88}$$

$$c_{22} := cHCrcInit \tag{C.89}$$

$$c_{23} := cHCrcPolynomial \tag{C.90}$$

$$c_{24} := cPayloadLengthMax \tag{C.91}$$

$$c_{25} := cMicroPerMacroMin \tag{C.92}$$

$$c_{26} := cMicroPerMacroNomMin \tag{C.93}$$

$$c_{27} := cSlotIDMax \tag{C.94}$$

$$c_{28} := cStaticSlotIDMax \tag{C.95}$$

$$c_{29} := cStrobeOffset \tag{C.96}$$

$$c_{30} := cSyncNodeMax \tag{C.97}$$

$$c_{31} := cVotingDelay \tag{C.98}$$

$$c_{32} := cVotingSamples \tag{C.99}$$

$$c_{33} := cPropagationDelayMax \tag{C.100}$$

$$c_{34} := cdTxMax \tag{C.101}$$

C.3. α- und β-Parameter (N_e)

In Tabelle C.1 und Tabelle C.2 werden die α- und β- Parameter auf realistische (serienahe) Werte gesetzt. Einige davon beziehen sich auf das dynamische Segment und werden an anderer Stelle berechnet.

α-Parameter	Wert
$\alpha_1 := x_6$	1.42
$\alpha_2 := x_8$	siehe Kapitel 5.3.2
$\alpha_3 := x_{38}$	0.15
$\alpha_4 := x_{44}$	0.4
$\alpha_5 := x_{30}$	4.715

Tabelle C.1.: α-Parameter

β-Parameter	Wert
$\beta_1 := x_{37}$	0.025
$\beta_2 := x_{53}$	2
$\beta_3 := x_{58}$	100
$\beta_4 := x_{55}$	1
$\beta_5 := x_{56}$	40
$\beta_6 := x_{24}$	60
$\beta_7 := h_2$	0
$\beta_8 := x_{61}$	0
$\beta_9 := x_{59}$	0.45
$\beta_{10} := x_{41}$	0.025
$\beta_{11} := x_{66}$	siehe Kapitel 5.3.2
$\beta_{12} := x_{36}$	3
$\beta_{13} := x_{17}$	siehe Kapitel 5.3.2
$\beta_{14} := x_{40}$	6
$\beta_{15} := x_{12}$	siehe Kapitel 5.3.2
$\beta_{16} := x_{14}$	0
$\beta_{17} := x_{57}$	2

Tabelle C.2.: β-Parameter

C.4. FlexRay-Parametrierungsrestriktionen

Anfolgend werden die wichtigsten der für Abs. 4.6 relevanten FlexRay-Restriktionen (zur Berechnung von x_1, x_2, x_3, x_4 und x_7) aufgeführt.

Restriktion 5:

$$g_1(x) := x_7 - c_{17} \leq 0$$

$$g_2(x) := c_{18} - x_7 \leq 0$$

Die Funktionale $g_1(x)$ und $g_2(x)$ sind stetig und Frechet-differenzierbar für alle $x \in \mathbb{D}_e$.

Restriktion 6:

Der Parameter β_{10} kann mit Hilfe der Gleichung $\beta_{10} = \beta_2 * x_3$ ersetzt werden. Man erhält dadurch

$$g_3(x) := c_{26} * (\beta_2 * x_3) - x_7 \leq 0$$

Das Funktional $g_3(x)$ ist stetig und Frechet-differenzierbar für alle $x \in \mathbb{D}_e$.

Restriktion 7:

$$g_4(x) := x_7 - (x_1 + c_1 + c_2 + 20 * x_2 + c_3) * (c_4 * x_3 * (1 - c_6)) \leq 0$$

Das Funktional $g_4(x)$ ist stetig und Frechet-differenzierbar für alle $x \in \mathbb{D}_e$.

Restriktion 11:

$$g_5(x) := ceil(\frac{(\alpha_5 - \alpha_3)}{(x_7 * (1 - c_6))}) - x_4 \leq 0$$

Das Funktional $g_5(x)$ ist stückweise stetig für alle $x \in \mathbb{D}_e$ und Frechet-differenzierbar in den Nicht-Sprungstellen.

Restriktion 12:

$$g_6(x) := ceil(\frac{(2 * \alpha_5 - \alpha_3 + 2 * \beta_{12})}{(x_7 * (1 - c_6))}) - x_4 \leq 0$$

Das Funktional $g_6(x)$ ist stückweise stetig für alle $x \in \mathbb{D}_e$ und Frechet-differenzierbar in den Nicht-Sprungstellen.

Restriktion 24:

$$g_7(x) := \beta_{14} - (x_4 * x_7 * (1 + c_6) + \alpha_1 + \alpha_3) \leq 0$$

Das Funktional $g_7(x)$ ist stetig und Frechet-differenzierbar für alle $x \in \mathbb{D}_e$.

Restriktion 32:

$$g_8(x) := round(\frac{((x_1 + c_1 + 0.5 * c_{13}) * c_4 + c_{29} + c_{31})}{\beta_2}) - 143 \leq 0$$

$$g_9(x) := 14 - round(\frac{((x_1 + c_1 + 0.5 * c_{13}) * c_4 + c_{29} + c_{31})}{\beta_2}) \leq 0$$

Die Funktionale $g_8(x)$ und $g_9(x)$ sind stückweise stetig für alle $x \in \mathbb{D}_e$ und Frechet-differenzierbar in den Nicht-Sprungstellen.

Restriktion 33:

$$g_{10}(x) := x_4 + ceil(\frac{((round(\frac{((x_1 + c_1 + 0.5 * c_{13}) * c_4 + c_{29} + c_{31})}{\beta_2})) + \beta_4)}{\beta_5}) - 68 \leq 0$$

$$g_{11}(x) := 2 - (x_4 + ceil(\frac{((round(\frac{((x_1 + c_1 + 0.5 * c_{13}) * c_4 + c_{29} + c_{31})}{\beta_2})) + \beta_4)}{\beta_5})) \leq 0$$

Die Funktionale $g_{10}(x)$ und $g_{11}(x)$ sind stückweise stetig für alle $x \in \mathbb{D}_e$ und Frechet-differenzierbar in den Nicht-Sprungstellen.

Restriktion 35:

Der Parameter β_{10} kann ersetzt werden durch die Gleichung $\beta_{10} = \beta_2 * x_3$. Man erhält dadurch

$$g_{12}(x) := \beta_{17} - floor(\frac{(((5+2*x_2+3)*10-2)*(c_4*x_3*(1-c_6)))}{((\beta_2*x_3)*(1+c_6))}) \leq 0$$

Das Funktional $g_{12}(x)$ ist stückweise stetig für alle $x \in \mathbb{D}_e$ und Frechet-differenzierbar in den Nicht-Sprungstellen.

Restriktion 37:

$$g_{13}(x) := ceil(\frac{((c_4*x_3*(1+c_6))+\beta_9+\beta_7*\beta_8)}{(c_4*x_3*(1-c_6))}) - x_1 \leq 0$$

Das Funktional $g_{13}(x)$ ist stückweise stetig für alle $x \in \mathbb{D}_e$ und Frechet-differenzierbar in den Nicht-Sprungstellen.

Restriktion 38:

$$g_{14}(x) := ceil(2 * (x_1 + c_{14}) * \frac{(1+c_6)}{(1-c_6)} + 2 * \frac{\alpha_4}{(c_4*x_3*(1-c_6))}) - 99 \leq 0$$

$$g_{15}(x) := 67 - ceil(2 * (x_1 + c_{14}) * \frac{(1+c_6)}{(1-c_6)} + 2 * \frac{\alpha_4}{(c_4*x_3*(1-c_6))}) \leq 0$$

Die Funktionale $g_{14}(x)$ und $g_{15}(x)$ sind stückweise stetig für alle $x \in \mathbb{D}_e$ und Frechet-differenzierbar in den Nicht-Sprungstellen.

ANHANG D

Fallstudien

D.1. Eingangsparametersätze

	Konstr A	Konstr B	Konstr C
dBDRxia	0.3	0.45	0.35
dBDRxai	0.4	0.3	0.2
pdBDRx	0.075	0.1	0.075
pdBDTx	0.1	0.1	0.1
pDelayCompensationA	1	6	3
pDelayCompensationB	1	3	6
pExternRateCorrection	0	0	0
pExternOffsetCorrection	0	0	0
pPayloadDynMax	254	254	254
pSamplesPerMicrotick	2	1	4
gClusterDriftDamping	2	-	-
gdCycle	5000	-	-
gdMacrotick	1.375	-	-
gdSamplesClockPeriod	0.0125	-	-
gNumberOfStaticSlots	91	-	-
gNumberOfMinislots	286	-	-
gPayloadLengthStatic	9	-	-
cClockDeviationMax	0.0015	-	-
cdWakeUpSymbolTxLow	6	-	-
cdWakeUpSymbolTxIdle	0.3	0.45	0.35
cVotingDelay	2	-	-
cStrobeOffset	5	-	-
cSamplesPerBit	8	-	-
cChannelDelimiter	11	-	-
dStarTruncation	0.15	0.25	-
pdStarDelay	0.25	0.25	-

Tabelle D.1.: Eingangsparametersatz bei der E/E-Architekturmodellierung

D.2. FlexRay-Parametersätze

Parametername	Boundary	active value
gdCycle		5000.0
pMicroPerCycle		[100000.0, 200000.0, 50000.0]
pdMicrotick		[0.05, 0.025, 0.1]
gdBit		200.0
cSamplesPerBit		8.0
cClockDeviationMax		0.0015
pMicroPerMacroNom		[28.0, 55.0, 14.0]
pMicroPerMacroNom		[28.0, 55.0, 14.0]
gdMacrotick		1.375
gdSampleClockPeriod		0.025
gMacroPerCycle		3636.0
pSamplesPerMicrotick		[2.0, 1.0, 4.0]
gPayloadLengthStatic		9.0
aFrameLengthStatic		269.0
gdActionPointOffset_...		7.0
gdActionPointOffset_...		28.0
gdStaticSlot		212.0
gClusterDriftDamping		2.0
gAssumedPrecision		9.149999994039536
gdMaxInitializationEr		9.828999986588954
pdAcceptedStartupRa...		[381.0, 761.0, 191.0]
gdMaxPropagationDel...		0.8749999970197679
gdTSSTransmitter		6.0
pDecodingCorrection		[11.0, 22.0, 6.0]
gdMinPropagationDel...		0.19600000447034838
gdMinislotActionPoin...		7.0
gdMinislot		15.0
aFrameLengthDynamic		[5169.0, 5169.0, 5169.0]
pLatestTx		[234.0, 234.0, 234.0]
gdDynamicSlotIdlePha...		1.0
gdNIT		1.0
gdSymbolWindow		0
pMacroInitialOffsetB		[8.0, 8.0, 8.0]
pMacroInitialOffsetA		[8.0, 8.0, 8.0]
pDelayCompensationA		[1.0, 6.0, 3.0]
pDelayCompensationB		[1.0, 3.0, 6.0]
pdMaxDrift		[301.0, 601.0, 151.0]
pdListenTimeout		[200602.0, 401202.0, 100302.0]
pMicroInitialOffsetA		[16.0, 27.0, 5.0]
pMicroInitialOffsetB		[16.0, 30.0, 2.0]
pClusterDriftDamping		[4.0, 8.0, 2.0]
gOffsetCorrectionStar		3636.0
gOffsetCorrectionMax		10.710437501490116
pRateCorrectionOut		[301, 2, 345]
pOffsetCorrectionOut		[215.0, 430.0, 108.0]
pExternOffsetCorrect...		[0.0, 0.0, 0.0]
pExternRateCorrection		[0.0, 0.0, 0.0]
gdWakeUpSymbolRxL...		25.0
gdWakeUpSymbolTxL...		30.0
gdWakeUpSymbolTxI...		90.0
gdCASRxLowMax		77.0
cdCASRxLowMin		29.0
gdWakeUpSymbolRxI...		29.0
gdWakeUpSymbolRx...		151.0
Robustheit		0.0
slots_cells		2
Erweiterbarkeit		1.0
nutzDatenEffizienz		57.599999999999994

Parametername	Boundary	active value
gdCycle		5000.0
pMicroPerCycle		[100000.0, 200000.0, 50000.0]
pdMicrotick		[0.05, 0.025, 0.1]
gdBit		200.0
cSamplesPerBit		8.0
cClockDeviationMax		0.0015
pMicroPerMacroNom		[28.0, 55.0, 14.0]
pMicroPerMacroNom		[28.0, 55.0, 14.0]
gdMacrotick		1.375
gdSampleClockPeriod		0.025
gMacroPerCycle		3636.0
pSamplesPerMicrotick		[2.0, 1.0, 4.0]
gPayloadLengthStatic		9.0
aFrameLengthStatic		269.0
gdActionPointOffset_...		7.0
gdActionPointOffset_...		27.0
gdStaticSlot		212.0
gClusterDriftDamping		2.0
gAssumedPrecision		9.009120008940696
gdMaxInitializationEr		9.249680008940697
pdAcceptedStartupRa...		[366.0, 732.0, 183.0]
gdMaxPropagationDel...		0.8045600044703484
gdTSSTransmitter		6.0
pDecodingCorrection		[11.0, 22.0, 6.0]
gdMinPropagationDel...		0.5640000044703484
gdMinislotActionPoin...		7.0
gdMinislot		15.0
aFrameLengthDynamic		[5169.0, 5169.0, 5169.0]
pLatestTx		[234.0, 234.0, 234.0]
gdDynamicSlotIdlePha...		1.0
gdNIT		1.0
gdSymbolWindow		0
pMacroInitialOffsetB		[8.0, 8.0, 8.0]
pMacroInitialOffsetA		[8.0, 8.0, 8.0]
pDelayCompensationA		[1.0, 6.0, 3.0]
pDelayCompensationB		[1.0, 3.0, 6.0]
pdMaxDrift		[301.0, 601.0, 151.0]
pdListenTimeout		[200602.0, 401202.0, 100302.0]
pMicroInitialOffsetA		[16.0, 27.0, 5.0]
pMicroInitialOffsetB		[16.0, 30.0, 2.0]
pClusterDriftDamping		[4.0, 8.0, 2.0]
gOffsetCorrectionStar		3636.0
gOffsetCorrectionMax		11.007997508940695
pRateCorrectionOut		[301, 2, 345]
pOffsetCorrectionOut		[221.0, 441.0, 111.0]
pExternOffsetCorrect...		[0.0, 0.0, 0.0]
pExternRateCorrection		[0.0, 0.0, 0.0]
gdWakeUpSymbolRxL...		25.0
gdWakeUpSymbolTxL...		30.0
gdWakeUpSymbolTxI...		90.0
gdCASRxLowMax		77.0
cdCASRxLowMin		29.0
gdWakeUpSymbolRxI...		29.0
gdWakeUpSymbolRx...		151.0
Robustheit		0.0
slots_cells		2
Erweiterbarkeit		1.0
nutzDatenEffizienz		57.599999999999994

Abbildung D.1.: Tabellarische Gegenüberstellung zweier FlexRay-Parametersätze als Ergebnis zweier unterschiedlicher E/E-Architekurvarianten.

D.3. Umsetzung von Modellabfragen

Die Modellabfrage stützt sich auf eine Mustersuche im Modell. Die Suche erfordert ein Quellobjekt (beispielweise ein Steuergerät), welches gemäß den Vorgaben des zugrunde gelegten Metamodells nach dessen Propagationen auf andere Objekte im System untersucht wird. So lassen sich Zielobjekte (beispielsweise ein Sternkoppler und der FlexRay-Konnektortyp) festlegen, die in Relation zu den gesuchten Quellobjekten stehen müssen. Durch eine Markierung wird determiniert, welche Objekte bei der Modellabfrage zurückgeliefert werden sollen. Eventuell interessieren neben den Quellobjekten auch weitere Aspekte, etwa eine spezifische Relation zwischen Quell- und Zielobjekt oder auch das Zielobjekt selbst.

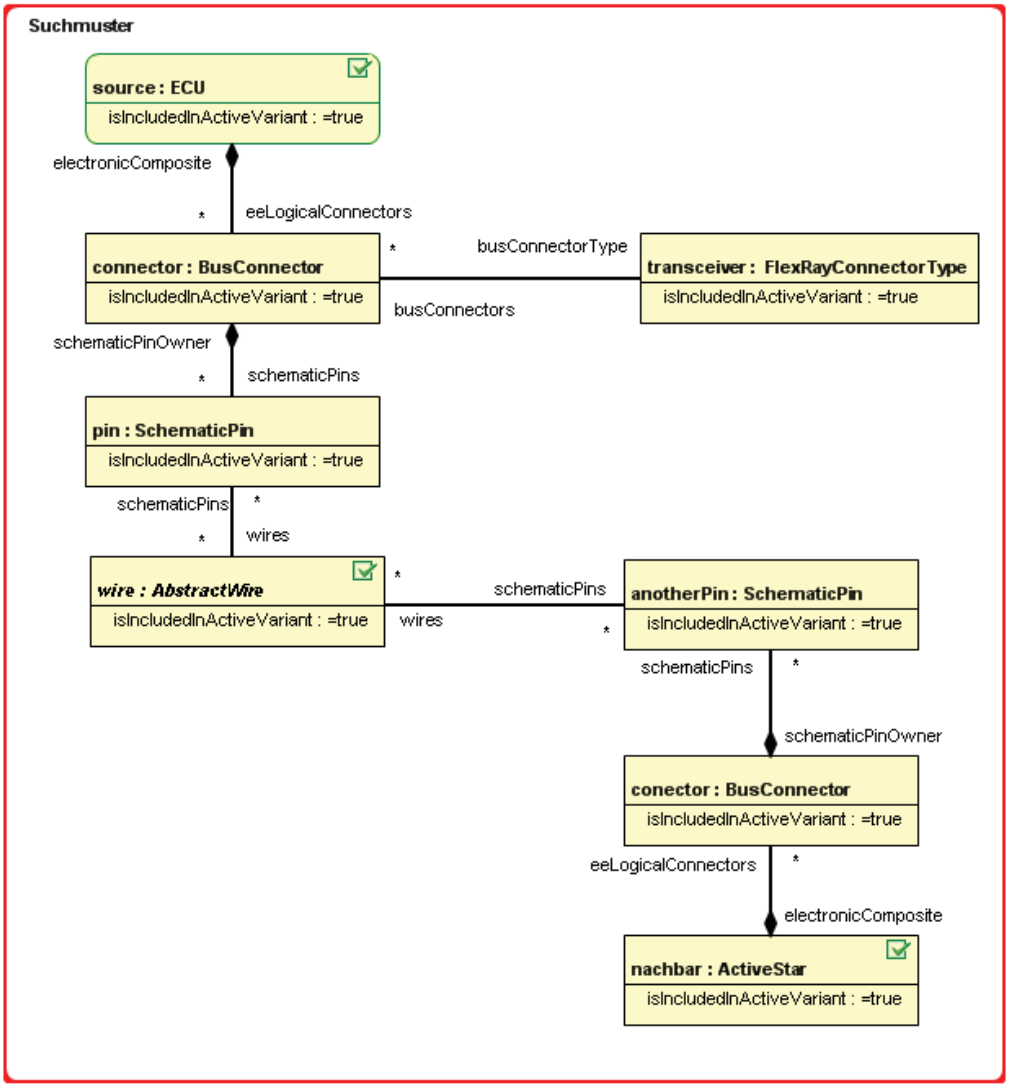


Abbildung D.2.: Suchmuster zur Identifikation von Steuergeräten mit FlexRay-Anschluss, die eine physikalische Verbindung zu aktiven Sternkopplern im System aufweisen.

D.4. Automatisches Busscheduling eines Funktionsnetzes

Listing D.1: PseudoCode zum Erzeugen des Busschedules

```
proc generateBusschedule(ArchitectureVariant architectureList[],
                         Frame packedFunctionNetSignals[], int scheduleLayout[]):

   ArchitectureVariant activeArchitectureVariant =
   dataBase.getActiveElementFrom(architectureList)

   Frame validFrameList[] =
   dataBase.getActiveElementsFrom(packedFunctionNetSignals)

   int staticSlots = scheduleLayout[gdStaticSlotsParameter]
   int busCyclePeriod = scheduleLayout[gdCycle]
   int element = 0
   int lastFilledSlot = 0
   int lastScheduledSender = 0
   boolean result = false

   // aktuellen Slotbelegungszustand mit keinem Sender und 64 freie Zellen
   // pro Slot initialisieren
   char slotUsed[][] = []
   int baseValue[2] = [0,64]
   for int slot = 0 to staticSlots:
   do
      slotUsed.append(baseValue)
   od
   // prüfen ob Slotanzahl der Architektur ermittelt wurden und ob gültige
   // Botschaften existieren
   if staticSlots != 0
    if validFrameList != []
        result = true
        char scheduleTable[64][staticSlots] = []
        for int cycle = 0 to 63:
        do
            for slot to staticSlots:
            do
                scheduleTable[cycle][staticSlots] = "0"
            od
        od
        open_gen_file(dir + "/busSchedule.csv")

            // die Botschaften werden sequentiell entlang der Slots im Schedule
                platziert
        for Frame flexrayFrame in validFrameList:
        do
            // determinieren, welche Sendezyklen pro Botschaft gültig sind
            Signal signals[] = getSignalTimings(flexrayFrame)
            for flexraySignal in signals:
            do
               signalCycleTime = 64*busCyclePeriod
               try:
                     TransmissionPort sendPorts[] = flexraySignal.
                         mappedTransmissionPorts
                     FNElement logicalSender = sendPorts[0].mappedFNElement
                     ECU networkSender = getHWDeployment(logicalSender)
               except:
                    print "Kein Sender wurde bisher definiert!"
               if flexraySignal.cycleTime<signalCycleTime:
                    signalCycleTime = flexraySignal.cycleTime
               fi
            od
            // Botschaftswiederholungsrate bezogen auf Buszykluslänge ermitteln
            repetition = getFrameCycleTime(signalCycleTime)
            element = lastFilledSlot
                        int freecells = 0
```

```
            int column = 0
            while element =< staticSlots:
            do
                // prüfen, ob freie Zellen des aktuellen Slots ausreichen
                // zum Einfügen der aktuellen Botschaft
                int freeCells = (int)slotUsed[element][1]-(64/repetition)
                if (int)slotUsed[element][0] == 0:
                    // falls Botschaft i und Botschaft i+1 vom gleichen Sender,
                    // dann wird der Frame zum Balancieren der Slots weiter
                    // hinten im Schedule eingefügt
                    if networksender.ToString() == lastScheduledSender.ToString():
                        element = lastFilledSlot+10
                        lastFilledSlot = element
                    else:
                        lastFilledSlot = element
                    fi
                    newElement[] = []
                    //Sender und restliche freie Zellen des Slots ermitteln
                    newElement.append(networkSender)
                    newElement.append(freeCells)
                    slotUsed[element] = newElement
                    lastScheduledSender = networkSender
                    break
                elif  networkSender.ToString() == slotUsed[element][0].ToString() &&
                    freecells > -1:
                    slotUsed[element][0] = networkSender
                    slotUsed[element][1] = slotUsed[element][1]-(64/repetition)
                    break
                fi
                element = element+1
            od
            int i = 0
            while i < 63:
            do
                scheduleTable =
                writescheduleTable(i,lastFilledSlot,flexrayFrame,scheduleTable)
                i = i+repetition
            od
        od
        write_gen_file(scheduleTable)
        close_gen_file()
        if(result == true):
            dataBase.setOutput(scheduleTable)
        else:
            dataBase.setOutput([])
        fi
    else:
        // falls keine gültigen Botschaften detektiert werden können
        dataBase.setOutput([])
    fi
else:
    // falls keine FlexRay-Parameter zum Schedule-Layout detektiert werden können
    dataBase.setOutput([])
fi

fct int getFrameCycleTime(signalCycleTime):
   int repetition = 0
   for int cycleRepetition = 6 to 0:
   do
      if signalCycleTime >= busCyclePeriod*2^cycleRepetition:
         repetition = 2^cycleRepetition
      fi
   return repetition

fct int[][] writeScheduleTable(int line, int column, Frame frame, char inputTable[][]):
   char table[][] = inputTable
   // Variable zum Anzeigen eines bereits belegten Slots
   int blocked = 0
```

```
if ((int)table[line][column] != 0):
    blocked = 1
    if line<63 && blocked == 1:
        // aktuelle FlexRay-Zelle ist belegt => Funktionsrekursion mit
        // nächster FlexRay-Zelle
        line = line+1
        writeScheduleTable(line, column, frame, table)
    fi
else:
    // Name der in dieser FlexRay-Zelle versendeten Botschaft wird eingefügt
    table[line][column] = frame.getName()
fi
return table
```

D.5. Einschränkungen im Parametrierungsprozess

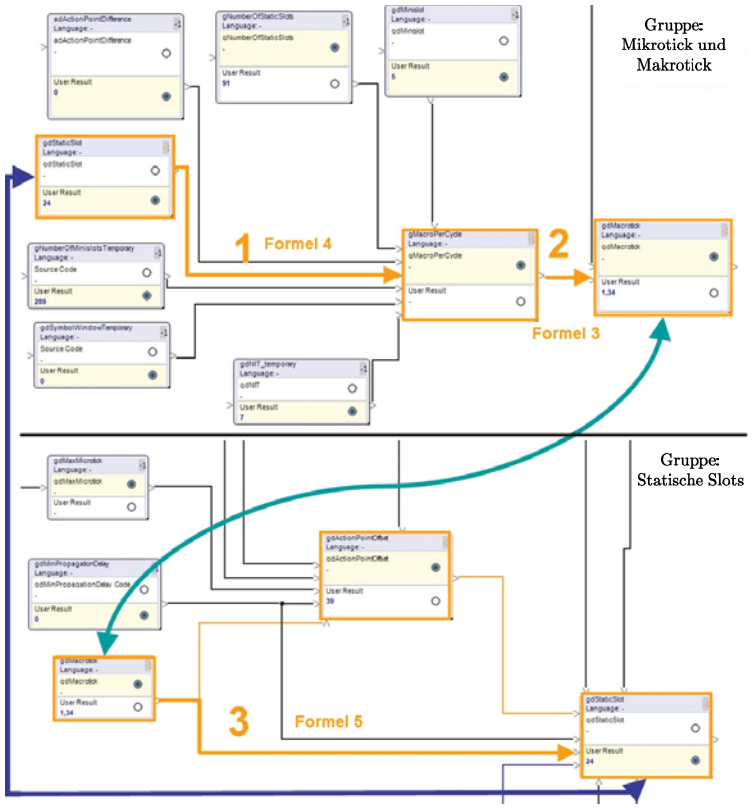


Abbildung D.3.: Mathematischer Zyklus bei der Berechnung (*gdStaticSlot* → *gMacroPerCycle* → *gdMacrotick* → *gdStaticSlot*).

Lebenslauf

Zur Person

Julian Broy

Derzeitige Stellung: **Entwicklungsingenieur**

geboren am **30. April 1981** in Dachau, Deutschland

verheiratet mit Mariana Broy

Wohnsitz: 71634 Ludwigsburg

E-Mail: **julian.broy@gmx.de**

Ausbildung

2005–2008	**Promotion,** Fakultät für Elektrotechnik der Universität Karlsruhe (TH) in Zusammenarbeit mit der Dr. Ing. h.c. F. Porsche AG, Entwicklungszentrum Weissach
2000–2005	**Studium der Informatik,** Technische Universität München Abschluß mit **Diplom mit Auszeichnung**
1991–2000	**Gymnasium Oberhaching** Abschluß mit **Abitur**

Berufliche Laufbahn

2008–heute	**Projektingenieur Vernetzung/Diagnose/Gateways,** Dr. Ing. h.c. F. Porsche AG, Entwicklungszentrum Weissach
2004-2005	**Diplomarbeit,** Dr. Ing. h.c. F. Porsche AG, Entwicklungszentrum Weissach
2003–2004	**Werkstudententätigkeit,** Fakultät für Informatik, Technische Universität München
2003–2004	**Werkstudententätigkeiten,** Pontifíca Universidade Católica Rio de Janeiro, Brasilien
2000–2003	**Praktikum und Werkstudententätigkeiten,** sd&m AG, München