Hierarchische Versionierung in der Entwicklung von Fahrzeugnetzwerken

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der KIT-Fakultät für Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)

angenommene

Dissertation

von

M. Sc.

Andreas Florian Vetter

Tag der mündlichen Prüfung:	10. 4. 2025
Hauptreferent:	Prof. Dr.-Ing. Eric Sax
Korreferent:	Prof. Dr. Ralf Reussner

Abstract

Modern premium vehicles contain up to 150 electronic control units (ECUs), which communicate with each other via various data buses. During development, numerous parameters must be defined and coordinated with all involved hardware and software suppliers. For microcontroller-based ECUs, data formats and transmission paths are planned, statically and bit-accurately, in a so-called communication matrix (K-matrix). The parameters of the K-matrix are an essential part of the ECU software.

Due to megatrends such as electrification, automated driving, and connectivity (between vehicle and the Internet, apps, and infrastructure like traffic lights or parking garages), the amount of information to be transmitted within a vehicle is rapidly increasing. As the number of ECUs and the amount of information to be processed grow, so does the challenge of planning this communication. Additionally, the ability to retrofit a vehicle with additional software functions is gaining more importance. Software-based innovations need to be integrated into vehicles much faster, and customers expect new software functions even after purchasing a vehicle. At the same time, there are domains where changes are rarely necessary due to mature technology.

To avoid unnecessary development costs, larger development cycles (i.e., longer intervals between software versions) and one-to-one reuse between model series are pursued for mature ECUs. In contrast, shorter development cycles are sought in competitive, more innovative domains to maintain a high pace of innovation. Due to the strong dependency between the vehicle global communication matrix and the ECU software, a common development speed for all nodes in the vehicle network is currently necessary. This forms a compromise between the two sensible but contradictory requirements for development speed.

This dissertation developed a method for K-matrix development that resolves this contradiction and enables different development cycles for ECUs within a common vehicle network. By converting incompatible changes in communication elements into backward-compatible extensions, ECUs with software, based on different K-matrix versions, can be meaningfully combined. This allows for development in different, needs-based cycles.

However, the extension of communication elements leads to an increase in the required bandwidth on the vehicle buses. The duration for which compatibility with older states is maintained is selectable, with the bandwidth demand increasing with longer compatibility durations. By making extensions at different levels of the hierarchically structured communication, the additional bandwidth requirement is minimized.

A model is established to estimate the additional bandwidth demand based on the chosen compatibility durations. The model was verified through simulations based on all K-matrix development stages of a real vehicle development process. It was found that the increase in required bandwidth is within an acceptable range, with optimal results achieved by adjusting the compatibility durations to the respective phases of the development process. The reduced additional bandwidth requirement due to the compatible extensions at multiple hierarchical levels is confirmed by both the model and the simulations.

The practical applicability of the developed method is discussed, considering current and future networking technologies. This discussion also addresses the impact of potential bandwidth increases and the challenges of adaptation by a real vehicle manufacturer.

Kurzfassung

Moderne Premiumfahrzeuge enthalten bis zu 150 elektronischer Steuergeräte, welche über verschiedene Datenbusse miteinander kommunizieren. In der Entwicklung müssen hierzu diverse Parameter festgelegt und mit allen beteiligten Hard- und Softwarezulieferern abgestimmt werden. Für die Mikrocontroller-basierten Steuergeräte werden Datenformate und Übertragungspfade bei der Planung, statisch und Bit-genau in einer sogenannten Kommunikationsmatrix (K-Matrix) festgelegt. Deren Parameter sind ein wesentlicher Bestandteil der Steuergerätesoftware.

Durch die Megatrends Elektrifizierung, automatisiertes Fahren und Konnektivität (zwischen Fahrzeug und Internet, Apps und Infrastruktur wie z. B. Ampeln oder Tiefgaragen) steigt die Menge der innerhalb eines Fahrzeugs zu übertragenden Informationen rapide an. Mit der Anzahl der Steuergeräte und der Menge der zu verarbeitenden Informationen wächst die Herausforderung diese Kommunikation zu planen. Hinzu kommt, dass die nachträgliche Erweiterbarkeit eines Fahrzeugs durch zusätzliche Softwarefunktionen immer mehr Bedeutung gewinnt. Softwarebasierte Innovationen sollen deutlich schneller in Fahrzeuge eingebracht werden und Kundinnen und Kunden erwarten neue Softwarefunktionen auch nach dem Kauf eines Fahrzeugs. Gleichzeitig bestehen Domänen, in denen, aufgrund ausgereifter Technologie, nur selten Änderungen notwendig sind.

Zur Vermeidung unnötiger Entwicklungskosten werden bei ausgereiften Steuergeräten größere Entwicklungszyklen (d. h. größere Intervalle zwischen zwei Softwareversionen) und eins-zu-eins-Übernahmen zwischen Baureihen angestrebt. In den wettbewerbsrelevanten, innovativeren Domänen werden dagegen kürzere Entwicklungszyklen angestrebt, um ein hohes Innovationstempo zu erreichen. Durch die starke Abhängigkeit zwischen der (Fahrzeug-globalen) Kommunikationsmatrix und der Steuergerätesoftware muss aktuell auf eine einheitliche Geschwindigkeit für Änderungen am Fahrzeugnetz gesetzt werden, die einen Kompromiss zwischen den beiden sinnvollen, aber widersprüchlichen Anforderungen bildet.

Im Rahmen dieser Dissertation wurde ein Verfahren zur K-Matrix-Entwicklung erarbeitet, welches den Widerspruch aufhebt und unterschiedliche Entwicklungszyklen für Steuergeräte eines gemeinsamen Fahrzeugnetzwerkes ermöglicht. Durch die Umwandlung inkompatibler Änderungen von Kommunikationselementen in rückwärtskompatible Erweiterungen können Steuergeräte, deren Softwarestände auf unterschiedlichen K-Matrix-Versionen basieren, sinnvoll kombiniert werden. Dies ermöglicht eine Entwicklung in unterschiedlichen, bedarfsgerechten Zyklen. Durch die Erweiterung der Kommunikationselemente kommt es zu einem Anstieg der benötigten Bandbreite auf den Fahrzeugbussen. Die Dauer, über die die Kompatibilität zu älteren Ständen gewahrt bleibt, ist wählbar, wobei mit zunehmender Kompatibilitätsdauer der Bandbreitenbedarf steigt. Durch Erweiterungen auf unterschiedlichen Ebenen der hierarchisch strukturierten Kommunikation wird der zusätzliche Bandbreitenbedarf minimiert.

Es wird ein Modell aufgestellt, mit dem sich der zusätzliche Bandbreitenbedarf in Abhängigkeit von den gewählten Kompatibilitätsdauern abschätzen lässt. Durch Simulationen auf Basis sämtlicher K-Matrix Entwicklungsstände eines realen Fahrzeugentwicklungsprozesses wurde das Modell überprüft. Es zeigte sich, dass der Zuwachs an benötigter Bandbreite in einem akzeptablen Bereich liegt, wobei für optimale Ergebnisse die Kompatibilitätsdauern an die jeweiligen Phasen des Entwicklungsprozesses angepasst werden können. Der reduzierte, zusätzliche Bandbreitenbedarf durch die kompatiblen Erweiterungen auf mehreren Hierarchieebenen wird sowohl durch das Modell als auch die Simulationen bestätigt.

Die praktische Anwendbarkeit des entwickelten Verfahrens wird unter Berücksichtigung der aktuell eingesetzten und zukünftigen Vernetzungstechnologien diskutiert. Dabei werden auch die Auswirkungen des potenziellen Bandbreitenzuwachses sowie die Herausforderungen bei der Adaption durch einen realen Fahrzeughersteller diskutiert.

Danksagung

Zuallererst möchte ich mich herzlichst bei meinem Doktorvater Prof. Dr.-Ing. Eric Sax für die exzellente Betreuung bedanken. Die jederzeit schnellen, ausführlichen und konstruktiven Rückmeldungen waren eine wertvolle Hilfe auf dem Weg vom ersten Entwurf bis zur vorliegenden Fassung. Herzlichen Dank auch für die Durchführung des Zweitgutachtens an Prof. Dr. Ralf Reussner und an die weiteren Mitglieder der Prüfungskommission: Prof. Dr. Sören Hohmann, Prof. Dr. Aghassi-Hagmann und Prof. Dr.-Ing. Sebastian Randel.

Die in der vorliegenden Dissertation vorgestellte Forschung erfolgte am Entwicklungsstandort der Daimler AG bzw. später der Mercedes-Benz AG in Sindelfingen. Dort bedanke ich mich bei Lorenz Slansky und Marco Maniscalco in deren Abteilungen und Dr. Christian Kühn, Dr. Steffen Goerzig und Adriana Onet in deren Teams ich meiner Forschung nachgehen durfte. Ganz besonders möchte ich mich bei Philipp Schumacher für die fachliche Betreuung, das vermittelte Wissen über AUTOSAR, XDIS und die Mercedes-internen Prozesse und die zahlreichen Diskussionen über mögliche Herangehensweisen und Lösungen bedanken. Herzlichen Dank auch an Robert Sakretz und Holger Schnürer für die Beantwortung all meiner AUTOSAR-Spezialfragen.

Ebenso bin ich dankbar, dass ich Teil der großen Gemeinschaft aus Doktorandinnen und Doktoranden vom Institut für Technik der Informationsverarbeitung (ITIV), dem Forschungszentrum Informatik (FZI) und weiteren Industriebetrieben sein durfte. Die zahllosen Gespräche und Vorträge auf all den Seminaren und Klausuren waren teils lehrreich, teils motivierend – stets eine Bereicherung.

Ein ganz besonderer Dank geht auch an meine Familie, insbesondere an meine Frau Lena Brühl: Danke für all die Geduld, das Verständnis, die Aufmunterung und Unterstützung.

Viernheim, im April 2025 *Andreas Vetter*

Inhaltsverzeichnis

Anhang

1 Einleitung

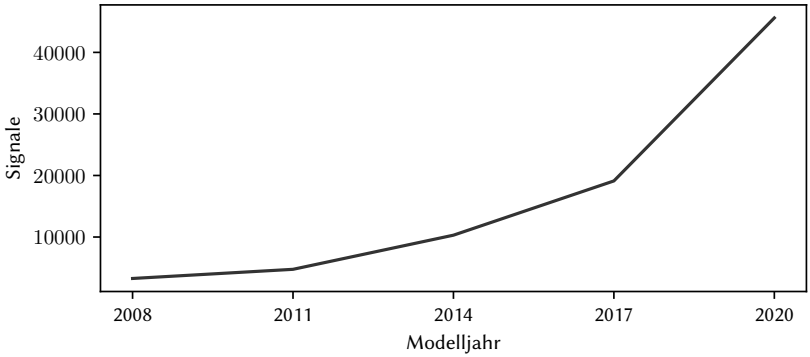

Abbildung 1.1: Zunahme der Signale über die Zeit in einer Oberklassenbaureihe

Die vier Megatrends autonomes Fahren, Elektrifizierung, Sharing[1] und Konnektivität[2] verändern aktuell die Automobilindustrie [Eis19]. Dadurch wächst die Komplexität in der Fahrzeugelektronik und Software. Ein Indikator dafür ist der exponentielle Zuwachs an, innerhalb eines Fahrzeuges übertragenen, Einzelinformationen, den sogenannten Signalen (vgl. Abb. 1.1[3]).

Sender und Empfänger der Signale sind die im Fahrzeug verbauten Steuergeräte. Bosch führte 1967 mit der D-Jetronic die erste elektronische Motorsteuerung ein [Bau67][Sch05a]. Inzwischen werden bis zu 150 elektronische Steuergeräte bzw. Electronic Control Units (ECUs) in aktuellen Premiumfahrzeugen

[1] Unter Sharing versteht man Mobilitätskonzepte, die auf der (kurzfristigen) Vermietung von Fahrzeugen oder dem Anbieten von Fahrdienstleistungen basieren. Dadurch gewinnen Flottenbetreiber und geschäftliche Kunden an Relevanz. So reduzierte sich der Anteil privater Fahrzeuganmeldungen in Deutschland von 51 % im Jahr 2000 auf 36 % im Jahr 2018 [Web19].

[2] Der Megatrend Konnektivität beinhaltet die Vernetzung von Fahrzeugen untereinander, mit Infrastruktur, mit einem Backend des Fahrzeugherstellers und anderen Internetdiensten sowie den Smartphones der Insassen.

[3] Erstellt auf Basis einer eigenen Auswertung der Fahrzeugnetzwerkdatenbank der Mercedes-Benz AG.

verbaut [Bur20][Mot23] und übernehmen außer der klassischen Motorregelung auch die Funktionen von Assistenzsystemen wie dem Abstandstempomaten oder dem aktiven Spurhalteassistenten, von Bedienelementen wie der Lenkradfernbedienung oder den Türbedieneinheiten, von Sensoren wie vom einfachen Türschließsensor bis zur Stereokamera in der Windschutzscheibe, und von Aktoren wie der Außenspiegelverstellung oder dem Außenlicht.

1.1 Motivation

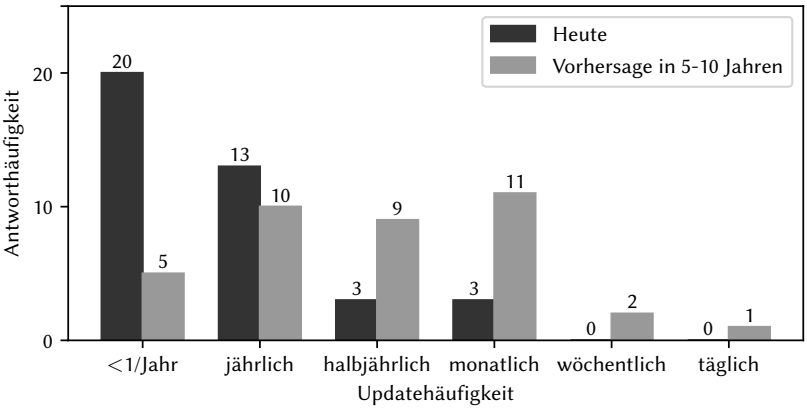

Abbildung 1.2: Updatefrequenz im Jahr 2018 und Erwartungen für die Zukunft [Gui18].

Zusätzlich zu der wachsenden Menge an Elektronik und Software werden die Innovationszyklen kürzer. Guissouma et al. führten 2018 eine Umfrage unter 51 Experten der Automobilindustrie durch, in der nach der aktuellen Updatehäufigkeit, sowie einer Vorhersage für die kommenden 5-10 Jahre gefragt wurde. Wie in Abbildung 1.2 zu sehen ist, wurden die meisten, im Feld befindlichen Fahrzeuge jährlich oder seltener aktualisiert, in fünf bis zehn Jahren werden jährliche bis monatliche Updates erwartet [Gui18]. In bereits ausgelieferten Fahrzeugen sollen neue Funktionen durch nachträgliche Softwareverkäufe (analog zum „App Store“ von Apple bzw. dem „Play Store“ von Google)

integriert werden können [Mül17]. Diese Bedingungen stellen eine Herausforderung für die bestehenden Entwicklungsprozesse dar und machen neue Methoden und Prozesse erforderlich [Kat14][Tra18].

Funktionen, die über unterschiedliche Teile eines Fahrzeugs verteilt sind, werden durch den Austausch von Informationen zwischen den verschiedenen Steuergeräten ermöglicht. Beispielsweise kann ein Türsteuergerät, das vom Regensensor in der Windschutzscheibe eine entsprechende Botschaft erhält, automatisch das Fenster schließen. Andere Funktionen können durch den Datenaustausch über einen Datenbus, d. h. ein von mehreren Teilnehmern gemeinsam genutztes Übertragungsmedium, günstiger umgesetzt werden. Ein Antiblockierystem (ABS)[1] Steuergerät, welches sich in der Motorhaube neben dem Bremszylinder befindet, ist auf die Information der Raddrehzahlsensoren, welche sich bei den Rädern befinden, angewiesen um ein Blockieren der Räder zu erkennen. Anstatt vier einzelne Datenleitungen zu den vier Rädern zu legen, können alle vier Drehzahlsensoren an ein solches Medium angeschlossen werden [Gal09]. Auf die dadurch auf dem Bus vorliegende Information können weitere Steuergeräte wie ein Elektronisches Stabilitätsprogramm (ESP) oder eine passive Reifendrucküberwachung zugreifen [Per02]. So können zusätzliche Leitungen und Gewicht eingespart werden, was zu einer reduzierten CO_2-Emission führt [Sch14].

Um das Netzwerk deterministisch zu halten und das notwendige Echtzeitverhalten[2] zu gewährleisten, wird die gesamte Kommunikation im Voraus geplant. D. h. es wird für jede Botschaft festgelegt, von welchem Startpunkt, über welche Route (ggf. Weiterleitung über mehrere Busse), sie zu welchen Zielpunkten gelangt. Das erste Serienfahrzeug mit CAN Bus, der zunächst

[1] „Das ABS verhindert, innerhalb seiner Systemgrenzen, bei einer Vollbremsung oder bei glatter Fahrbahn das Blockieren der Räder und erhält die Lenkfähigkeit des Fahrzeugs." [Pis21, S.1113]

[2] Unter Echtzeitanforderungen versteht man das Vorhandensein von festen Obergrenzen für zeitliche Abläufe. Für die anspruchsvollsten Anwendungen im Automobil darf die Übertragung der Information nicht länger als 2,5ms dauern [Ste08]. Bei einem geteilten Kommunikationsmedium ist für die Einhaltung solcher Obergrenzen, außer der Übertragungsgeschwindigkeit, die Vermeidung von Konflikten beim Zugriff auf das Netzwerk von Bedeutung [Çel14].

drei Steuergeräte verband, erschien 1991 [Sch05a]. Schon damals musste geplant werden, welche Botschaften man verschickt, welche IDs diese Botschaften bekommen, und welche Steuergeräte die Botschaften jeweils senden und empfangen. Dazu konnte das Netzwerk in einer einfachen Tabelle dargestellt werden. Heute enthält ein Fahrzeug bis zu 150 ECUs [Bur20][Mot23], sodass ein Datenbanksystem, ein mehrstufiger Prozess und ein ganzes Team erforderlich sind, um alle Signale zu verwalten [Sch16a][Vet23a].

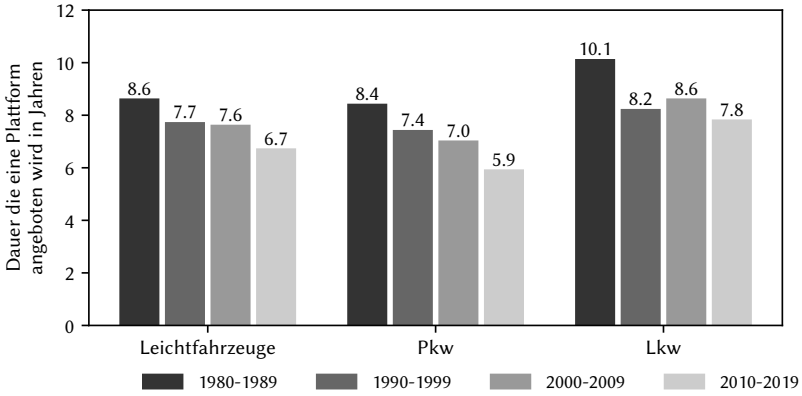

Abbildung 1.3: In den vergangen zehn Jahren hat sich die Zeit die ein PKW Modell angeboten wird um ca. ein Jahr verkürzt [Cen17].

Seit den Neunzigerjahren haben sich außerdem die Entwicklungszyklen verkürzt. Während damals eine Pkw-Generation im Schnitt für 7,4 Jahre angeboten wurde, sind es heute noch 5,9 Jahre, bis ein Nachfolger bereitstehen muss (s. Abb. 1.3). Mit der Entwicklung autonomer Fahrfunktionen, alternativer Antriebstechnologien, zunehmender Vernetzung mit Smartphones und Internet sowie dem stetigen Zuwachs an Komfortfunktionen ist zu erwarten, dass der Aufwand in der Netzwerkentwicklung weiter ansteigen wird [Has16]. Für den Anteil der Elektronik an den Gesamtkosten eines Fahrzeugs von heute 35 % wird in den nächsten zehn Jahren ein Anstieg auf 50 % prognostiziert (s. Abb. 1.4). Besonders herausfordernd ist auch, der Reichtum an unterschiedlichen Varianten und Versionen. So wurden z. B. 2017 in Deutschland

84.000 VW Golf in 58.000 unterschiedliche Konfigurationen verkauft [Dol18]. In Form von optionalen Assistenzsystemen (wie z. B. eine Abstandsautomatik, Spurhalteassistent oder automatisiertes Einparken) und Ausstattungsvarianten (z. B. einfaches Radio *oder* Infotainmentsystem[1] mit integrierter Navigationsfunktion) betrifft diese Variabilität auch die Netzwerkentwicklung. Ein Indikator für die Komplexität ist hierbei die Anzahl möglicher Kabelbaumkonfigurationen für ein Fahrzeug. Sie wird in der Literatur mit 10^{10} [Dom20] bis 10^{27} [Tha18] möglichen Kombinationen angegeben.

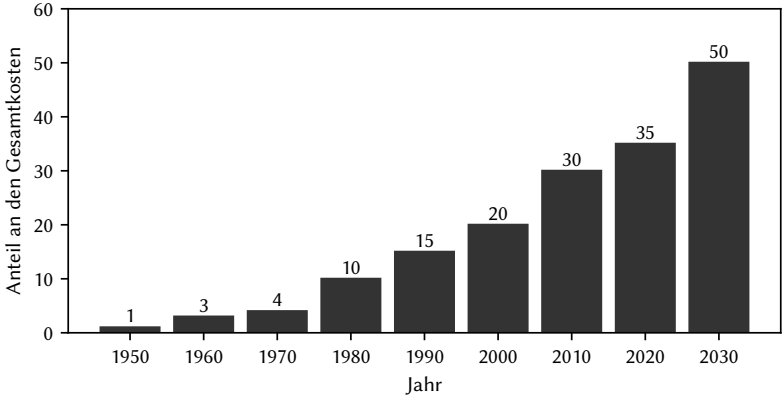

Abbildung 1.4: Entwicklung des Kostenanteils der Fahrzeugelektronik [Vai17]

Um den Aufwand in der Netzwerkentwicklung zu reduzieren und neue Anwendungen schneller in Fahrzeuge integrieren zu können, findet in der Automobilindustrie ein Umstieg von der bestehenden, Signal-basierten Kommunikation auf eine sogenannte Service-orientierte Architektur (SOA) statt (die beiden Kommunikationsmechanismen werden in Abschnitt 2.4.6 u. 2.4.7 näher erläutert). Dabei wird nicht mehr von Signalen mit festen Sendern und Empfängern ausgegangen, sondern es gibt Steuergeräte, welche Dienste anbieten (Service Provider) und andere Steuergeräte, welche diese Dienste in

[1] „Infotainment“ ist ein Kofferwort aus „Information“ (z. B. Navigation u. Verkehrsmeldungen) und „Entertainment“ (engl. f. „Unterhaltung“, z. B. Radio u. Musikstreamingdienste).

Anspruch nehmen (Clients). Bei den Diensten kann es sich z. B. um die Bereitstellung von Informationen oder das Ausführen von Funktionen (z. B. „Fenster öffnen“) handeln. Die Verbindung zwischen Clients und Servern wird über ein IP[1]-basiertes Netzwerk dynamisch während des Systemstarts oder der Laufzeit hergestellt. Eine statische Planung aller Verbindungen ist somit nicht mehr notwendig. Bei der Fahrzeugentwicklung muss jedoch weiterhin sichergestellt werden, dass für jeden Client alle erforderlichen Services in einer kompatiblen Form vorhanden sind.

Mit der Entwicklung des Scalable service-Oriented MiddlewarE over IP (SOME/IP) Protokolls wurde die Grundlage für Service-orientierte Kommunikation im Fahrzeug gelegt [Wec14][AUT23g]. Verschiedene Fahrzeughersteller arbeiten an Service-orientierten Architekturen [Tra17][Sch17]. Softwarezulieferer bieten geeignete Betriebssysteme, Middleware-Lösungen und Designtools an [Hel17]. Während die grundsätzliche Anwendbarkeit von SOA-Ansätzen im automotive Bereich nachgewiesen ist [Ste08][Kug17][Kam19], fehlt es noch an für SOA-optimierten Prozessen und Methoden für die Serienentwicklung.

1.2 Ziele und Forschungsfragen

Die existierenden Prozesse und Methoden für Signal-basierte Kommunikation wurden seit den ersten Fahrzeugen mit vernetzten Steuergeräten sukzessive weiterentwickelt. Sie gehen von einer im Voraus festgelegten (und entsprechend fix geplanten) Kommunikation aus. Eine wesentliche Methode zur Entwicklung von Fahrzeugnetzwerken ist die Weiterentwicklung eines bestehenden Netzwerks und die Wiederverwendung von bereits existierenden Komponenten [Siv19]. Bestenfalls kann ein gesamtes Steuergerät unverändert in einer neuen Fahrzeugbaureihe weiterverwendet werden. Die Wiederverwendung von Steuergeräten ist aus wirtschaftlicher Perspektive sinnvoll (vgl. Abschnitt 2.1.2). Hierzu ist es erforderlich, dass die ECU der alten Baureihe kompatibel zum Netzwerk der neuen Baureihe ist.

[1] Internet Protocol (IP), vgl. Abs. 3.1.5.

Steuergeräte, deren Funktion für die Sicherheit des Fahrzeugs, oder dessen Einhaltung von Umweltschutzvorschriften eine Rolle spielt, erfordern eine Zertifizierung (vgl. Abschnitt 2.6.1), die mit zeitlichen und finanziellen Aufwänden verbunden ist. Entsprechend werden für derartige Steuergeräte, entgegen des eingangs beschriebenen Trends, Softwareänderungen vermieden.

Aufgrund der detailliert im Voraus geplanten Kommunikation (vgl. Abschnitt 2.5) und der Abhängigkeit von Steuergerätesoftware und Kommunikation (vgl. Abschnitt 2.6.4) führt die Anforderung nach schnellen Entwicklungszyklen für die Kunden-sichtbare Software und für das Fahrzeug als Ganzes einerseits, und die Anforderung von beständiger Software, für einen Teil der Steuergeräte andererseits, zu einem Zielkonflikt. Daraus ergeben sich die folgenden Forschungsfragen (FF):

FF1 Wie kann der Widerspruch zwischen der Wiederverwendung von bestehenden Steuergeräten und der Forderung nach schnellen Innovationszyklen und den damit einhergehenden Veränderungen am Gesamtsystem sinnvoll aufgelöst werden?

FF2 Wovon hängt die Kompatibilität zwischen Steuergeräten und Fahrzeugnetzwerken bzw. zwischen Clients und Servern ab, und wie kann diese bewertet werden?

FF3 Wie kann ein gemeinsamer, praxistauglicher Prozess zur Bewertung und Optimierung der Kompatibilität sowohl in Signal-basierten als auch in Service-orientierten Netzwerken aussehen?

1.3 Aufbau der Dissertation

Nachdem in diesem Kapitel die Motivation und die Forschungsfragen vorgestellt wurden, geht Kapitel 2 auf die Grundlagen der Fahrzeugnetzwerkentwicklung ein. In Kapitel 3 wird der aktuelle Stand von Wissenschaft und Technik der Fahrzeugvernetzung untersucht und speziell auf die bestehenden Mechanismen zu Kompatibilität und Änderungsverwaltung eingegangen. In Kapitel 4 wird mit der „hierarchischen Versionierung“ ein neues Verfahren entwickelt, welches in der Lage ist, die genannten Herausforderungen zu bewältigen. In Kapitel 5 wird ein Modell zur Abschätzung der benötigten Bandbreite aufgebaut. Mit den Daten eines vergangenen Entwicklungsprozesses und einer darauf aufbauenden Simulation wird das Modell überprüft und untersucht, ob das neue Verfahren anwendbar ist und welche Resultate erreicht werden können. Dabei wird auf die verschiedenen Parameter der neuen Methodik eingegangen. Kapitel 6 bewertet die Ergebnisse aus Kapitel 5 und geht auf weitere für die Anwendbarkeit relevante Faktoren ein. Mögliche Einschränkungen und Variationen werden diskutiert. Die Antworten auf die Forschungsfragen und die Schlussfolgerungen der Dissertation werden in Kapitel 7 zusammengefasst. Die anstehenden Schritte zur Einführung des Verfahrens in der Praxis und weitere Forschungsfelder werden aufgezeigt.

2 Einführung in die Fahrzeugnetzwerkentwicklung

Die Entwicklung eines Fahrzeugnetzwerks ist ein mehrschrittiger Prozess, in dem unterschiedliche Interessen zu berücksichtigen sind. Dieses Kapitel führt in den Prozess, und die anfallenden Artefakte ein, um die aktuellen Herausforderungen im Detail zu beschreiben. Die Anforderungen der Entwicklung von Netzwerken im Pkw-Bereich werden ausgeführt und von den Anforderungen in anderen Branchen abgegrenzt. Die verschiedenen Prozessschritte und Artefakte werden anhand von Beispielen dargestellt.

2.1 Rahmenbedingungen

Einige der Rahmenbedingungen ergeben sich durch die technischen Eigenschaften von Kraftfahrzeugen. Weitere Bedingungen kommen durch die Geschäftsmodelle für Pkw sowie die Organisationsstrukturen, innerhalb derer ein Fahrzeug entwickelt wird, zustande.

2.1.1 technische Rahmenbedingungen

Automobile werden von arktischen Regionen nördlich des Polarkreises bis in die afrikanischen Wüsten überall eingesetzt. Dabei sind die Steuergeräte je nach Position im Fahrzeug auch der Abwärme und den Vibrationen des Motors sowie den Einflüssen der im Auto verwendeten Betriebsmittel ausgesetzt [Rei15, S. 254].

Tabelle 2.1: Die Umwelteinflüsse, denen Fahrzeugelektronik standhalten muss [Joh04].

Temperatur	Innenraum	-40 ℃ bis +85 ℃
	Motorraum	-40 ℃ bis +125 ℃
	Motoroberfläche	-40 ℃ bis +150 ℃
	In Abgas- und Verbrennungsbereichen	-40 ℃ bis +200-600 ℃
Mechanische Erschütterungen	Während der Montage (Falltest)	3000 g
	Am Fahrzeug	50 g bis 500 g
Mechanische Vibration		15 g, 100 Hz bis 2 kHz
Elektromagnetische Impulse		100 V/m bis 200 V/m
Kontakt zu	allgemein	Feuchtigkeit, Sprühsalz
	in manchen Anwendungen	Kraftstoff, Öl, Bremsflüssigkeit, Getriebeöl, Ethandiol, Abgase

Eine Übersicht über die verschiedenen Umwelteinflüsse und deren Intensität ist in Tabelle 2.1 gegeben. Schwankungen der Versorgungsspannung (vom Kaltstart bei schwacher Batterie bis zur maximalen Ladespannung der Lichtmaschine) sowie Spannungsspitzen müssen von den Steuergeräten toleriert werden [Rei15, S. 254].

Von diesen allgemeinen Randbedingungen abgesehen gibt es auch noch solche, die sich auf die Vernetzung innerhalb des Fahrzeugs auswirken. Aufgrund des nicht vernachlässigbaren Platzbedarfs müssen Steuergeräte bei der Rohbauplanung berücksichtigt werden. Der hierfür benötigte Platz eines Steuergeräts wird als Verbauraum bezeichnet. Entsprechend können Anzahl, Größe und Position der Steuergeräte nach der Rohbauplanung nur eingeschränkt geändert werden. Von Bedeutung für die Platzierung der Geräte sind Faktoren wie die räumliche Nähe zum Einsatzbereich des Geräts (z. B. Motorsteuergerät in der Nähe des Motors), Umgebungsbedingungen wie die Temperatur bzw. Möglichkeiten zum Wärmeabtransport, oder die Anbindung an die Stromversorgung und geeignete Datenbusse [Str12, S. 25ff]. Dies führt dazu, dass die Topologie, d. h. die Anzahl und Verbindungsstruktur der Steuergeräte für ein

Fahrzeug (vgl. Abs. 2.4), bereits zu einem relativ frühen Zeitpunkt (Abschluss der Rohbauplanung) weitestgehend festgelegt werden.

Bei der Auslegung des Kabelbaums ist das Gewicht ein wesentlicher Faktor. Es kann 60 – 80 kg betragen [Ruf15, S. 11][Han22] und trägt so über die Materialmenge zu den Produktionskosten sowie dem Kraftstoffverbrauch bei. Folglich müssen die Kabellängen (d. h. Abstand/Position der Steuergeräte) sowie der Kabelquerschnitt (elektrischer Energieverbrauch der Steuergeräte) optimiert werden. Allein die Optimierung des Energiebordnetzes ist ein eigenständiges Forschungsfeld [Ruf15][Bra18]. Dies führt ebenfalls zu Einschränkungen bei der Entwicklung des Datennetzes innerhalb des Fahrzeugs.

2.1.2 wirtschaftliche Rahmenbedingungen

Fahrzeughersteller produzieren jährlich zwischen 200.000 und 600.000 Fahrzeuge ihrer meistverkauften Baureihen (s. Tab. 2.2).

Tabelle 2.2: 2022 produzierte Stückzahlen der Spitzenmodelle verschiedener Hersteller

Modell	Stückzahl	Quelle
VW Tiguan	604.536	[Vol23, S. 25]
VW Polo	448.043	[Vol23, S. 25]
VW Passat	447.246	[Vol23, S. 25]
Mercedes GLC	369.000	[Mer23, S. 52]
Mercedes C-Klasse	303.000	[Mer23, S. 52]
Mercedes E-Klasse	302.700	[Mer23, S. 52]
BMW 3er/4er	478.932	[Bay23, S. 70]
BMW 5er/6er	315.590	[Bay23, S. 70]
BMW X3	400.898	[Bay23, S. 70]
Audi Q5	319.162	[Vol23, S. 33]
Audi Q3	239.340	[Vol23, S. 33]
Audi A3	234.395	[Vol23, S. 33]

In Verbindung mit einem Zeitraum von ca. 6 Jahren in denen ein Modell[1] angeboten wird (vgl.,Abb. 1.3) ergeben sich Gesamtstückzahlen in Millionenhöhe. Bei einer Million Fahrzeuge ergibt sich aus jedem eingesparten Cent pro Fahrzeug eine Einsparung von 10.000 € für den Hersteller. Dementsprechend ist es wirtschaftlich sinnvoll, für Einsparungen im Centbereich nennenswerte Entwicklungsaufwände zu betreiben.

Tabelle 2.3: 2022 produzierte Stückzahlen anderer Transportmittel. Es handelt sich hierbei jeweils um die Gesamtstückzahlen über alle Baureihen eines Herstellers.

Produkt	**Stückzahl**	**Quelle**
Kommerzielle Boeing Flugzeuge	480	[Boe23, S. 29]
Kommerzielle Airbus Flugzeuge	661	[Air23, S.15]
Airbus Hubschrauber	344	[Air23, S.16]
Scania Lkw	82.827	[Vol23, S.37]
Scania Busse	5.315	[Vol23, S.37]
MAN Lkw	62.009	[Vol23, S.37]
MAN Busse	4.675	[Vol23, S.37]
Daimler Busse	24.041	[Dai23, S. 3]
Mercedes-Benz Lkw	166.369	[Dai23, S. 3]
BMW Motorräder	215.932	[Bay23, S. 11]

Diese besondere Economy of Scales unterscheidet die Pkw-Entwicklung von anderen Transportmitteln wie Flugzeugen, Bussen oder Motorrädern (vgl. Tab. 2.3). Dort haben, aufgrund der deutlich geringeren Stückzahlen, die Entwicklungskosten eine größere Bedeutung. Die Gesamtstückzahlen bei Lkw liegen in vergleichbaren Größenordnungen wie einzelne Pkw-Baureihen, allerdings teilen sich diese Stückzahlen auf Produktpaletten auf, die vom Kleinlaster bis zum Vierzigtonner reichen.

[1] In der Regel findet nach etwa der Hälfte der Zeit eine sogenannte Modellpflege (engl.: „Facelift“) statt. Dabei wird das Äußere eines Fahrzeuges geringfügig angepasst (z. B. neues Scheinwerfer- oder Kühlerdesign) und neue Funktionen oder Assistenzsysteme (z. B. ein sprechendes Infotainment System oder ein aktiver Spurhalteassistent) werden integriert. Dies dient dazu, das Kaufinteresse der Kunden aufrechtzuerhalten und um wettbewerbsfähig zu neueren Baureihen (anderer Hersteller) zu bleiben.

Ein spezielles Konzept der Automobilindustrie ist die sogenannte Verblockung. Als solche bezeichnet man es, wenn ein Steuergerät aus einer Baureihe zur Verwendung in einer weiteren Baureihe eingeplant wird. Dies ist für die Netzwerkentwicklung von besonderer Bedeutung, da verblockte Steuergeräte zu mehreren Fahrzeugnetzwerken kompatibel gehalten werden müssen. Ein Vorteil dieser Methode ist die Einsparung von Entwicklungskosten, da für eine neue Baureihe nicht Steuergeräte neu entwickelt werden müssen. Insbesondere für Baureihen mit geringen Absatzzahlen ist dies wichtig. Ein weiterer Vorteil der Verblockung ist, dass hierdurch bei den Zulieferern größere Stückzahlen bestellt und entsprechende Mengenrabatte in Anspruch genommen werden können. Man unterscheidet zwischen „Vorwärtsverblockung“ (wenn bei einem Steuergerät geplant wird, es auch in folgenden Baureihen einzusetzen) und „Rückwärtsverblockung“ (wenn ein neu entwickeltes Steuergerät auch in bestehende Baureihen integriert wird, z. B. um einzelne neue Funktionen in bereits in Serienproduktion befindlichen Fahrzeugen anbieten zu können).

2.1.3 organisatorische Rahmenbedingungen

Kraftfahrzeuge werden von einem Hersteller bzw. Original Equipment Manufacturer (OEM) gebaut. Dieser verbaut dabei Komponenten, welche von anderen Firmen produziert wurden. Die Unternehmen, die den OEM beliefern, werden als Tier 1, Tier 2, ... 1 („tier“ engl. für „Reihe“/„Rang“/„Ebene“) bezeichnet. Diese Zulieferer verwenden ihrerseits die Produkte anderer Firmen, welche dann als Tier 2 bezeichnet werden. Entsprechend setzt sich die Kette mit Tier 3 etc. fort. Diese Struktur ist für die Entwicklung von Fahrzeugnetzwerken von Bedeutung, da mehrere Ebenen dieser Hierarchie unmittelbar involviert sind. Beispielsweise verbaut ein OEM Steuergeräte von verschiedenen Tier 1, welche wiederum die Software verschiedener Tier 2 zukaufen (vgl. Abs. 2.7.6). Damit das Netzwerk und die darauf aufbauenden Anwendungen reibungslos funktionieren, müssen sich alle beteiligten Parteien miteinander abstimmen (z.,B. um eine einheitliche Interpretation der eingesetzten Standards zu gewährleisten).

2.2 Beispielanwendung

Abbildung 2.1: Ein moderner Außenspiegel (Bild: [Mer20b])

Zur besseren Veranschaulichung, der im Folgenden vorgestellten Konzepte, wird zunächst ein Beispielsystem eingeführt. Betrachtet wird ein moderner Außenspiegel (s. Abb. 2.1). Der Spiegel verfügt über die folgenden Funktionen:

Fn1 Der horizontale und vertikale Winkel des Spiegelglases ist verstellbar.

Fn2 Das Spiegelgehäuse wird beim Abschließen des Fahrzeugs automatisch eingeklappt.

Fn3 Das Spiegelglas wird bei niedrigen Außentemperaturen automatisch beheizt.

Fn4 Das Spiegelglas wird bei Dunkelheit und nachfolgendem Verkehr automatisch abgedunkelt.

Fn5 Eine in das Spiegelglas eingelassene LED warnt vor Verkehrsteilnehmern im toten Winkel.

Fn6 Im Spiegelgehäuse befindet sich ein Seitenblinker, der synchron mit den restlichen Seitenblinkern leuchtet.

Tabelle 2.4: Statt für die sechs Funktionen zusätzlich zur Spannungsversorgung acht Steuerleitungen einzusetzen, können alle Informationen über eine digitale Kommunikationsverbindung mit ein oder zwei Drähten übertragen werden.

Funktion	Leitungen	Begründung
-	(2)	Gemeinsame Masseleitung und Stromversorgung.
Fn1	2	Der Winkel kann entlang zweier Achsen verändert werden.
Fn2	1	Es gibt einen zu verändernden Winkel.
Fn3	1	Die Spiegelheizung kann an- und ausgeschaltet werden.
Fn4	1	Die Verdunkelung kann an- und ausgeschaltet werden.
Fn5	2	Die LED kann in zwei Farben (gelb oder rot) leuchten.
Fn6	1	Die Blinkerleuchte kann an- und ausgeschaltet werden.
Summe	**10 (8)**	Mit (ohne) Stromversorgung.

Anhand dieser Funktionen lässt sich der Vorteil digitaler Datenübertragung gegenüber einzelner Steuerleitungen zeigen. In Tabelle 2.4 ist aufgelistet, wie viele einzelne Leitungen nötig wären, um die Funktionen umzusetzen. Bei einer digitalen Informationsübertragung genügen, zusätzlich zur Spannungsversorgung, ein bis zwei Leitungen, um alle Daten zu übertragen (siehe auch Abschnitt 2.3.2).

2.3 Komponenten eines Fahrzeugnetzes

In einem Kraftfahrzeug werden Steuergeräte an gemeinsame Datenleitungen angeschlossen, um Informationen auszutauschen. Gateways verbinden mehrere solcher Teilnetze zu einem Fahrzeugnetzwerk. Im Folgenden wird näher auf diese Komponenten eingegangen.

2.3.1 Steuergeräte

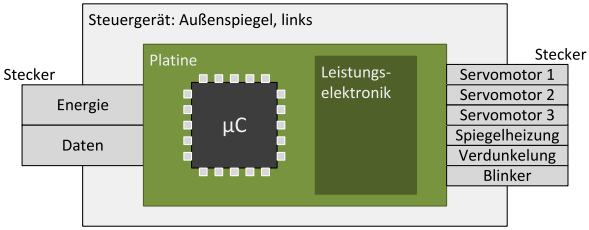

Abbildung 2.2: Schematische Darstellung der Außenspiegel-ECU

Im Beispiel mit dem Außenspiegel werden die Funktionen 1-6 durch eine digitale Datenübertragung gesteuert. Hierzu ist ein Gerät notwendig, welches die digitalen Signale empfängt und daraus die Ansteuerung für die Motoren, Leuchten, Heizung und Verdunkelung ableitet.

Die ECUs bilden die einzelnen Elemente des Fahrzeugnetzwerks. Sie haben Anschlüsse zur elektrischen Energieversorgung sowie zum Datenaustausch. Ferner können sie über Anschlüsse für Sensoren oder Aktoren verfügen oder unmittelbar in solche Sensoren/Aktoren integriert sein. Im Beispiel mit dem Außenspiegel könnte das Steuergerät wie in Abbildung 2.2 aussehen.

Ein Steuergerät besteht in der Regel aus einer Platine mit Spannungsversorgung, einem oder mehreren Transceiver-Bausteinen[1], einem Mikrocontroller, sowie anwendungsspezifischen Komponenten (z. B. Schalttransistoren zur Ansteuerung von Elektromotoren oder Leuchtmitteln oder Auswertungselektronik für angeschlossene Sensoren). Diese Platine wird in ein robustes Gehäuse eingebaut, welches die Elektronik vor den in Abschnitt 2.1.1 dargestellten Umwelteinflüssen schützt. Für die Kontaktierung werden besondere,

[1] Mikrocontroller können ihre Pins in der Regel nur zwischen einer logischen 0 (Spannung auf Massepotenzial) und einer logischen 1 (Spannung auf Versorgungspotential, bei Mikrocontrollern i. d. R. zwischen 1,8V und 5V) hin und her schalten. Mithilfe spezieller, integrierter Schaltkreise, den sogenannten Transceivern (Kofferwort aus Transmitter, d. h. Sender und Receiver, d. h. Empfänger), werden diese logischen Spannungspegel an die jeweilige physische Übertragungsschicht angepasst.

vibrationssichere Stecker verwendet. Die Spanne der eingesetzten Mikrocontroller reicht von sehr klein (Singlecore, CPU < 10 MHz, RAM < 1 kB, ROM < 10 kB) bis sehr groß (Multicore, CPU > 400 MHz, RAM > 1 MB, ROM > 8 MB) [Ban21]. Auf dem Mikrocontroller befindet sich Software, welche sowohl die funktionalen Aufgaben des Geräts umsetzt, als auch die Kommunikation mit anderen Steuergeräten abwickelt. Da die Steuergerätesoftware von besonderer Bedeutung für diese Arbeit ist, wird sie in Abschnitt 2.6 ausführlicher behandelt. ausführlicher behandelt.

Tabelle 2.5: Steuergeräte, die zu den Funktionen des Außenspiegels beitragen.

Name und Beschreibung	**Funktion**
Außenspiegel, links; Empfängt über eine digitale Schnittstelle Informationen zur Positionierung des Spiegels und der Ansteuerung von Heizung, Verdunkelung, LEDs und Blinker.	alle
Türsteuergerät, vorne, links; Wertet Tastendrücke der Türbedieneinheit aus und überträgt diese in digitaler Form an den Außenspiegel. Außerdem sind Tasten für die Sitzverstellung und den Fensterheber vorhanden, sowie die Ansteuerung des Fensterhebers und des Türschlosses.	Fn1 u. Fn2
Außentemperatursensor; Misst die Außentemperatur.	Fn3
Blendsensor, hinten; Erkennt helle Lichtquellen hinter dem Fahrzeug.	Fn4
Toter Winkel Assistent, links; Überwacht per Radar den jeweiligen toten Winkel.	Fn5
Bedienelemente Lenksäule; Hebel für Blinker, Scheibenwischer und Tempomat werden elektronisch erfasst und digital übertragen.	Fn6

Im Beispiel mit dem Außenspiegel befindet sich ein Steuergerät unmittelbar im Spiegelgehäuse. Dieses empfängt Informationen von anderen Geräten im Fahrzeug. Tabelle 2.5 listet die, an den Funktionen des Außenspiegels beteiligten, Steuergeräte auf. Zu beachten ist, dass diese nicht ausschließlich zur Funktion des Außenspiegels beitragen, sondern auch weitere Aufgaben im Fahrzeug erfüllen.

2.3.2 Datenbusse

Die digitale Kommunikation zwischen den Steuergeräten erfolgt über Datenbusse. Dabei handelt es sich um unter mehreren Kommunikationsteilnehmern geteilten Kommunikationsmedien (im einfachsten Fall gemeinsame „Drähte“, an welche alle Partner angeschlossen sind) über welche die Teilnehmer Daten austauschen können. Eine über einen Bus übertragene Botschaft wird auch als „Frame“ bezeichnet. Dabei gibt es unterschiedliche Mechanismen, um sicherzustellen, dass immer nur ein Teilnehmer aktiv auf den Bus (Daten-) zugreift („auf den Bus schreiben“) während die anderen Teilnehmer passiv bleiben und die gesendeten Informationen empfangen („vom Bus lesen“) oder diese ignorieren (z. B. weil vom jeweiligen Sender keine relevanten Informationen erwartet werden). Diese Mechanismen werden als Zugriffsverfahren bzw. Arbitrierung bezeichnet [Fre17].

Single Master

Im einfachsten Fall gibt es pro Bus nur einen einzigen Knoten, der von sich aus Kommunikation initiieren darf. Dieser Knoten wird als „Master“ bezeichnet. Alle weiteren Knoten (die „Slaves“) senden, wenn sie durch den Master aufgefordert wurden. Indem der Master nur einen einzelnen Slave zum Senden auffordert, werden Kollisionen vermieden. Ein Beispiel für eine Single Master Technologie aus dem Bereich der Fahrzeugnetzwerke ist das Local Interconnect Network (LIN).

Netzwerke, in denen mehrere Steuergeräte die Möglichkeit haben, Übertragungen zu initiieren, werden als „Multi Master“ bzw. „Vielfachzugriff“ bezeichnet und erfordern geeignete Zugriffsverfahren, um Kollisionen zu vermeiden.

CSMA

CSMA steht für „Carrier Sense Multiple Access“ frei übersetzt „Mehrfachzugriff mit Überwachung des Mediums“. Dabei sind an einem Bus mehrere Steuergeräte angeschlossen, welche vor dem Senden überprüfen, ob bereits ein anderes Steuergerät sendet („Überwachung des Mediums“). Wird eine laufende Übertragung erkannt, müssen alle anderen Geräte warten, bis der Bus wieder frei ist, um selbst senden zu können. Ein Beispiel für einen Bus mit dem CSMA Zugriffsverfahren ist das Controller Area Network (CAN).

TDMA

TDMA steht für „Time Division Multiple Access“ frei übersetzt „Mehrfachzugriff mit zeitlicher Einteilung“ bzw. „Zeitmultiplex“. Die Kollisionsvermeidung erfolgt hierbei, indem den am Medium angeschlossenen Teilnehmern unterschiedliche Zeitfenster zugeteilt werden, zu denen sie senden dürfen. In der Automobilindustrie kommt das TDMA Verfahren bei FlexRay zum Einsatz.

Verschiedene in der Praxis eingesetzte Technologien werden in Tabelle 2.6 verglichen. Bei Ethernet handelt es sich um eine für den automotive Bereich angepassten Form des Standards (100BASE-T1 anstatt 100BASE-TX)[Don18]. Im Beispiel mit dem Außenspiegel könnte dessen Steuergerät über LIN mit dem Türsteuergerät verbunden sein, da dies die kostengünstigste Technologie ist und die geringe Bandbreite ausreicht.

Tabelle 2.6: Verschiedene Technologien zur Verbindung von Steuergeräten [Zim14][Bor21]

Technologie	Geschwindigkeit	echtzeitfähig	Arbitrierung	Anwendungsbeispiele
Keyword-Line (K-Line)	1,2 kbit/s – 10,4 kbit/s	Nein	Single Master	OBD
Local Interconnect Network (LIN)	2,4 kbit/s – 19,2 kbit/s	Ja	Single Master	Sitzverstellung, Klimaanlage, Außenspiegel
Controller Area Network (CAN)	125 kbit/s – 500 kbit/s	Ja[a]	CSMA	Airbags, Türsteuergeräte
CAN FD (Flexible Datarate)	bis zu 4 Mbit/s	Ja	CSMA	Motorsteuerung, Fahrwerksfunktionen (z. B.: ABS, ESP)
FlexRay	2,5 Mbit/s – 10 Mbit/s	Ja	TDMA	Steer-by-Wire, Brake-by-Wire
MOST	25 Mbit/s – 150 Mbit/s	Ja	proprietär[b]	Audio, Video, Unterhaltung
Ethernet	10 Mbit/s – 10 Gbit/s	Nein[c]	Punkt zu Punkt Verbindungen	Multimedia, Unterhaltung, Firmwareupdates, Diagnose

[a] Details zu den Bedingungen, unter denen CAN echtzeitfähig ist, finden sich in Abschnitt 6.1.1.

[b] Die nicht vollständig öffentliche Spezifikation des MOST-Busses deckt sämtliche Schichten des ISO-Modells (vgl. Abs. 3.1) ab. Ausgehend von der zeitlichen Synchronisierung der Frames und der ringförmigen Topologie [Zim14][Bor21] handelt es sich um eine Kombination aus TDMA und Token-Passing.

[c] Während Ethernet von sich aus nicht echtzeitfähig ist, gibt es Erweiterungen des Standards, die dies ermöglichen sollen. Z. B. unterteilt IEEE802.1Q ein physikalisches Netz in mehrere, unterschiedlich priorisierte Virtual Local Area Networks (VLANs) ein. Der in Entwicklung befindliche Standard IEEE802.1DG ermöglicht eine exakte, zeitliche Synchronisation mehrerer Ethernet-Teilnehmer („Time Sensitive Networking“).

2.3.3 Gateways

Es gibt Steuergeräte, welche über mehr als eine Datenverbindung verfügen. Typisch sind etwa Geräte, die an zwei verschiedene CAN Busse angebunden sind oder Geräte mit einem CAN Bus und einem LIN Master Interface. Solche ECUs sind dazu in der Lage (ausgewählte) Informationen von einem Bus in den anderen zu übertragen. Auf diesem indirekten Weg können Steuergeräte aus unterschiedlichen Bussen miteinander kommunizieren.

Steuergeräte mit diesen Eigenschaften werden als Gateway bezeichnet. Im Rahmen dieser Arbeit wird zwischen zwei verschiedenen Gatewaytypen unterschieden.

Standard Gateways

Hierbei handelt es sich um Gateways, welche Datenpakete nach einem festgelegten Schema übertragen. Dabei wird es sich in aller Regel nur um ein Subset der auf den jeweiligen Bussen vorkommenden Daten handeln (andernfalls wäre ein gemeinsamer Bus sinnvoller). Die Pakete werden unverändert übertragen, es findet allenfalls eine Übersetzung der Paketformate zwischen verschiedenen Busformaten statt.

Die Entscheidung, welche Pakete von welchen Eingangsbussen zu welchen Ausgangsbussen weitergeleitet werden, wird als Routing bezeichnet. Die Veränderung der Routings ist eine Möglichkeit – im Rahmen der gegebenen Bandbreiten- und Rechenleistungskapazitäten – ein Netzwerk mit fixer Topologie durch Softwareanpassungen zu modifizieren.

Im Anwendungsbeispiel könnte das Türsteuergerät über einen CAN Bus mit dem Rest des Fahrzeugs verbunden sein. Über diese Datenleitung erhält es Botschaften, die für den Außenspiegel vorgesehen sind (z. B. das Signal, die Heizung zu aktivieren). Da diese Botschaften unverändert, automatisch an den Außenspiegel weitergeleitet wird, ist das Türsteuergerät (neben anderen Funktionen) ein Standard Gateway.

Smart Gateways

Als Smart Gateways werden solche Gateways bezeichnet, welche Datenpakete nicht unverändert übertragen, sondern eine Verarbeitung durchführen. Dabei kann es sich um ein Umpacken der in den Paketen enthaltenen Informationen zu neuen Datenpaketen handeln. Es können aber auch Verarbeitungsschritte mit den Daten durchgeführt und die Ergebnisse dieser Arbeitsschritte übertragen werden.

Ein Steuergerät verarbeitet die Daten des Radar (Abstand und Geschwindigkeit eines Objekts im Toten Winkel). Hierzu können auch weitere Informationen aus anderen Quellen (z. B. Geschwindigkeit, Lenkwinkel und Blinkerstellung des eigenen Fahrzeugs) verwendet werden. Eine zu übertragende verarbeitete Information ist der Sollzustand für die Totwinkel-LED (aus, gelb leuchtend, rot leuchtend oder rot blinkend). Damit erfüllt das verarbeitende Steuergerät die Bedingungen an ein Smart Gateway.

2.3.4 Systeme im Automobil

Im Kontext der automobilen Elektronik- u. Softwareentwicklung versteht man unter einem (Sub-)System[1] die Gruppierung einer Menge an logischen Komponenten, welche eine Gruppe an Funktionen umsetzen, die zusammen eine geforderte Funktionalität bieten [Mar16][Sta21, S. 8][2].

Ein Beispiel für ein System ist die automatische Spiegelheizung (s. Abb. 2.3). Es besteht aus Komponenten, die sich auf unterschiedlichen ECUs befinden (die eigentliche Heizung als Teil des Spiegelsteuergeräts, ein Taster und eine Status-LED zur manuellen Bedienung im Türsteuergerät, die Steuerungssoftware im Türsteuergerät sowie der Temperatursensor). Zu beachten ist, dass

[1] Das gesamte Fahrzeug kann als System betrachtet werden, welches sich aus mehreren Subsysteme zusammensetzt.

[2] Eine allgemeine, formelle Definition des Systembegriffs erfolgt in Abschnitt 3.2.1.

eine ECU mehrere Komponenten beinhalten kann. Insbesondere die Komponenten verschiedener Systeme (hier enthält das Spiegelsteuergerät z. B. auch Komponenten der Spiegelverstellung und des Totwinkelassistenten).

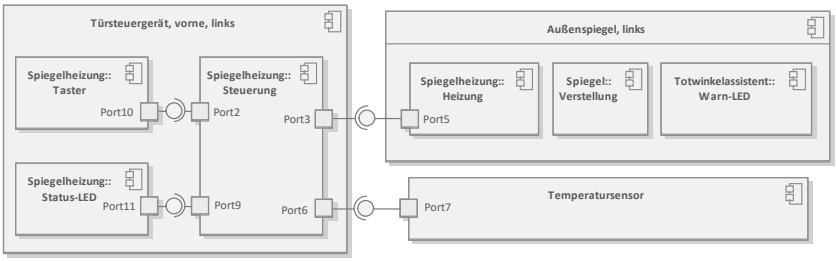

Abbildung 2.3: Komponentendiagramm für das System „Spiegelheizung“

Die Spiegelheizung kann auch Subsystem für ein übergeordnetes System sein. Beispielsweise könnte das Fahrzeug ein System zur Anpassung an beliebige Wetterlagen beinhalten. Hier wäre die Spiegelheizung eine Komponente, die zusammen mit weiteren (z. B. der Scheibenwaschautomatik) die Gesamtfunktionalität bietet.

2.4 E/E-Architektur

Im Bereich der Elektrik/Elektronik-Architektur (E/E-Architektur) gibt es zwei Konzepte, die jeweils als „Architektur“ bezeichnet werden. Sowohl bei Netzwerkstrukturen als auch bei Kommunikationsparadigmen spricht man von Architektur. Im Folgenden wird zur Unterscheidung nur die Gesamtheit aus Topologie und Kommunikationsparadigma als Architektur bezeichnet.

2.4.1 Grundtopologien

Die Anordnung der Steuergeräte sowie die Struktur der Verbindungen zwischen den Steuergeräten wird als Topologie bezeichnet. Ein grobes Blockschaltbild, welches alle Steuergeräte und Verbindungen darstellt, wird als

Topologiebild bezeichnet. Beispiele für Topologiebilder sind die Abbildungen 2.5 – 2.8.

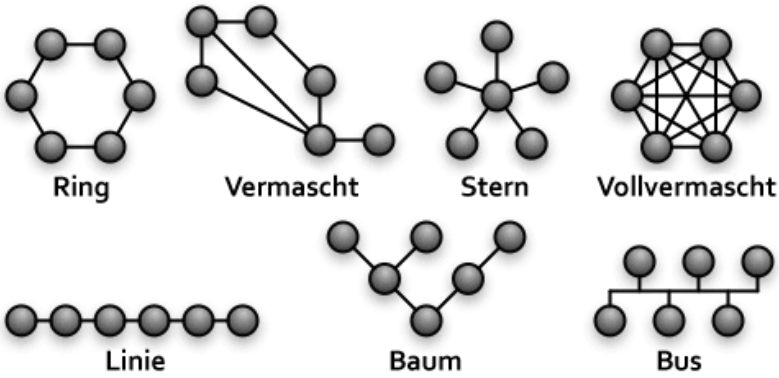

Abbildung 2.4: Verschiedene Netzwerk-Topologien (Bild: [Par13])

Es gibt verschiedene Grundstrukturen (vgl. Abb. 2.4). Welche davon im Fahrzeugnetz vorkommen, hängt von den verwendeten Technologien ab. Mit CAN, LIN und FlexRay werden klassische Bustopologien gebaut. MOST Busse werden i. d. R. als Ringtopologie aufgebaut [Zim14, S. 121]. Ethernet erlaubt nur Punkt-zu-Punkt Verbindungen, sodass hier Sterntopologien typisch sind. Dabei befindet sich in der Sternmitte ein Gerät, welches automatisch Verbindungen zwischen beliebigen Sendern und Empfängern herstellt[1]. Dieses Gerät wird als (Ethernet-)Switch bezeichnet.

Von Maschen spricht man, wenn es zwischen zwei Knoten mehr als einen Pfad gibt. Im Extremfall, wenn jeder Knoten eine direkte Verbindung zu jedem anderen Knoten hat, bezeichnet man das Netzwerk als voll vermascht. Das Vorhandensein zweier möglicher Routen ist eine Redundanz, die aus Kostengründen vermieden wird, sofern sie nicht notwendig ist (z. B. aus Sicherheitsgründen).

[1] Dies ermöglicht eine Kommunikation wie auf einem geteilten Medium. In Topologiebildern werden Ethernet Netzwerke daher vereinfacht als Bustopologie dargestellt.

2.4.2 klassische E/E-Topologie

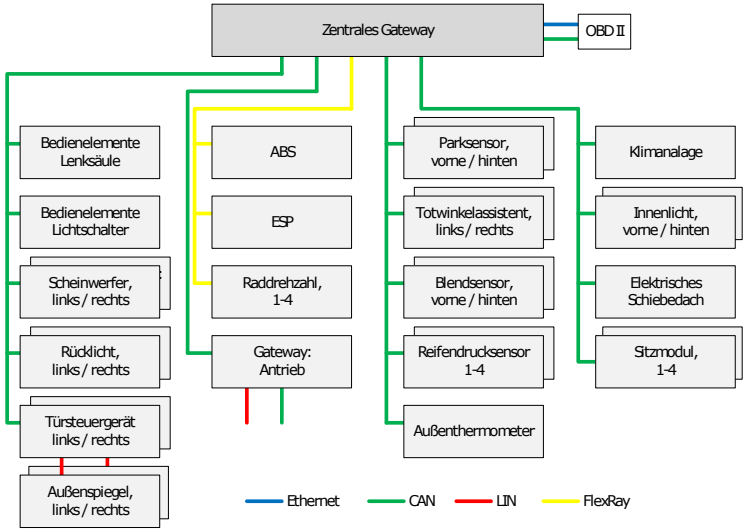

Abbildung 2.5: klassische E/E-Topologie

Klassischerweise werden Steuergeräte ihren Funktionen entsprechend gruppiert und an einen gemeinsamen Bus angeschlossen. Diese einzelnen Busse werden über ein zentrales Gateway verbunden (s. Abb. 2.5). So wird ein Informationsaustausch über Funktionsgruppen hinweg ermöglicht (z. B. Regelung der Musiklautstärke in Abhängigkeit der Fahrzeuggeschwindigkeit: Hier muss Information aus dem Bus des Antriebs an die Unterhaltungselektronik übertragen werden)[Str12]. Dabei wird das zentrale Gateway zum Flaschenhals, da dieses alle Nachrichten auf allen Bussen auswerten und über eine Weiterleitung entscheiden muss [Bru17]. Die klassische Topologie wird mit einem rein Signal-basierten Kommunikationsschema verwendet.

2.4.3 Domänentopologie

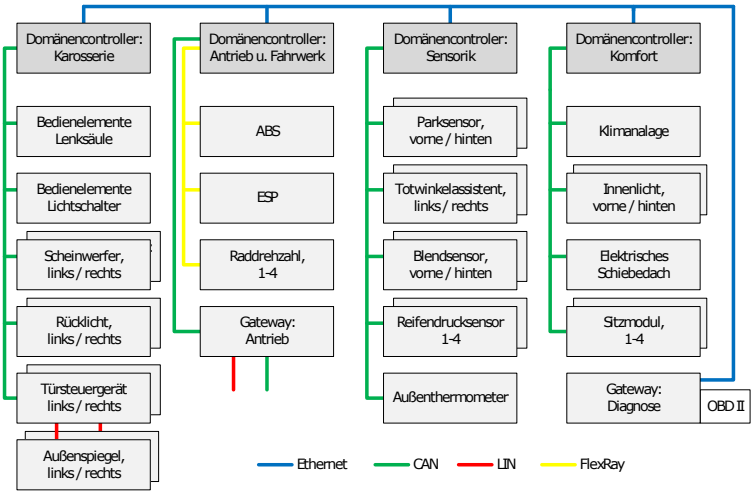

Abbildung 2.6: Domänen basierte E/E-Topologie

In jeder Funktionsgruppe („Domäne“) wird ein Steuergerät, welches die Kommunikation mit dem Rest des Fahrzeugs verwaltet (z. B. die Nachrichten auswählen, welche an den Rest des Fahrzeugs weitergeleitet werden), eingebaut. Dieses Gerät wird als Domänencontroller, Domänengateway oder Domain Control Unit (DCU) bezeichnet. Über einen zentralen Bus (bzw. englisch: Backbone) werden die Domänencontroller miteinander verbunden (s. Abb. 2.6). Für den zentralen Bus wird Ethernet verwendet. Das zentrale Gateway ist ein (für die Nutzung im Fahrzeug geeigneter) Ethernet Router [Han13].

Die Domänentopologie kann rein Signal-basiert verwendet werden. Sie ist für ein hybrides Service- und Signal-orientiertes Kommunikationsschema geeignet, mit Services auf dem Ethernet und Signalen auf den anderen Bussen [Vet21a]. Da die Domänengateways nur die für das Restfahrzeug relevanten Nachrichten weiterleiten, reduziert sich die Menge der Informationen, die am zentralen Gateway eintreffen [Bru17].

Premiumfahrzeuge, deren Netzwerk nach diesem Topologieprinzip aufgebaut ist, befinden sich aktuell in Serienproduktion [Sch20b].

2.4.4 Zonentopologie

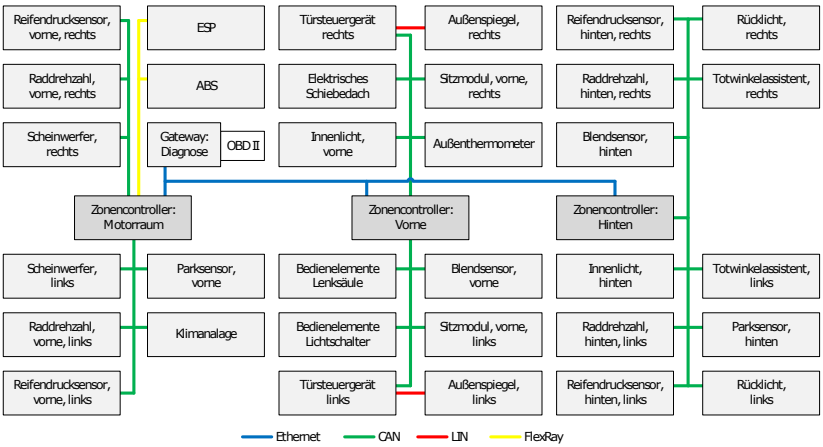

Abbildung 2.7: Zonen basierte E/E-Topologie

Die Steuergeräte müssen nicht zwangsläufig nach Funktionen gruppiert werden. Eine Gruppierung anhand der räumlichen Position innerhalb des Fahrzeugs ist ebenfalls möglich. In diesem Fall spricht man von einer Zonentopologie. Dabei werden die Informationen innerhalb einer Zone von sogenannten Zonencontrollern aggregiert und an zentrale Rechner weitergeleitet (s. Abb. 2.7).

Algorithmen und Funktionen werden Service-orientiert auf dem zentralen Rechner umgesetzt. Da hier alle Informationen vorliegen, sind Änderungen möglich, ohne umfangreiche Updates an mehreren Steuergeräten durchzuführen. Dadurch, dass Komponenten nicht mehr nach funktionaler, sondern räumlicher Nähe verdrahtet werden, ist durch Einsparungen von Leitungen eine Reduktion des Kabelbaumgewichtes um 15 bis 20 kg zu erreichen. Diese Topologie eignet sich für Service-orientierte Kommunikation. [Mau18]

2.4.5 Beispieltopologie

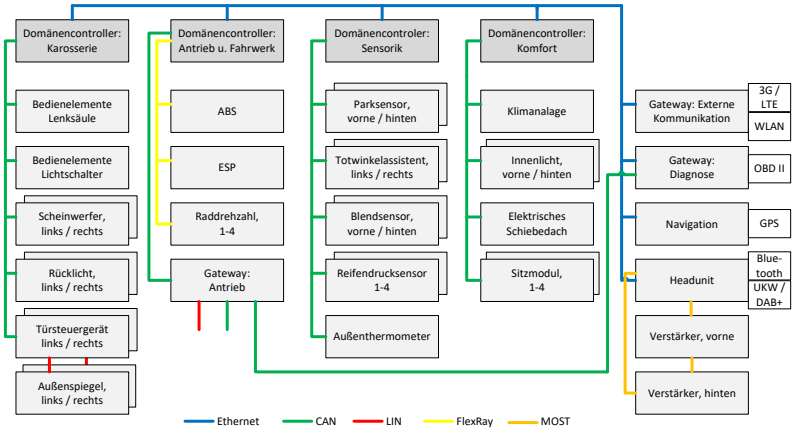

Abbildung 2.8: Topologie für das Beispielszenario

Als Beispiel wird eine Domänenarchitektur mit den Domänen „Karosserie“, „Antrieb und Fahrwerk“, „Sensorik“ und „Komfort“ verwendet (s. Abb. 2.8). Die Domänencontroller sind über ein automotive Ethernet Netzwerk miteinander verbunden. Steuergeräte, die mehr als 10MBit/s Bandbreite benötigen

(z. B. die Headunit[1] für die Übertragung von Multimediainhalten oder das Navigationsgerät für die Aktualisierung von Kartendaten), werden ebenfalls an dieses Netzwerk angeschlossen.

Innerhalb der Domänen wird CAN oder FlexRay für die Vernetzung der Steuergeräte untereinander und mit den Domänencontrollern verwendet. FlexRay wird in der Antriebs- und Fahrwerksdomäne eingesetzt, um für die Sicherheitsfunktionen wie ABS und ESP eine redundante physische Schicht, sowie die fest definierten Timings des TDMA Kommunikationsschemas zur Verfügung zu stellen.

Zum Übertragen von Audiodaten zwischen den Verstärkern und der Headunit wird die Multimedia-spezifische MOST Technologie eingesetzt. Im Topologiebild ist die charakteristische Ringstruktur eingezeichnet.

Innerhalb einer Domäne können Steuergeräte als Gateways für Subnetze dienen. Ein Beispiel ist das Antriebsgateway in der Antriebs- und Fahrwerksdomäne. Zur Anbindung peripherer Steuergeräte werden LIN Busse verwendet. Die Verbindung der Außenspiegel zu den Türsteuergeräten ist ein Beispiel.

Die Beispieltopologie enthält mit der CAN-Verbindung zwischen dem Antriebsgateway und dem Diagnosegateway eine Masche. So wird ein direkter Zugriff auf die Diagnosefunktionen des Antriebes ermöglicht.

2.4.6 Signal-basierte Kommunikation

Die Kommunikation zwischen Steuergeräten kann auf Basis von zwei verschiedenen Konzepten basieren: auf Signalen oder auf Services (s. Abs. 2.4.7). Signale werden zur Systemlaufzeit unabhängig von etwaigen Empfängern versendet. Das Signal wird von Steuergeräten empfangen, für die das während des Entwicklungsprozesses eingeplant wurde (entweder ECUs, welche die Daten selbst verwenden, oder Gateways, die das Signal in andere Netzwerkteile senden). Zur Laufzeit benötigt der Sender keine Informationen von

[1] Unter einer Headunit versteht man das zentrale Steuergerät des Infotainmentsystems [Zim14, S. 135].

den Empfängern. Die Übertragung von einem Sender an eine beliebige Anzahl Empfänger wird auch als „Broadcast“ bezeichnet.

Das Empfänger-unabhängige Versenden eines Signals wird durch die Sender-ECU ausgelöst, es gibt dazu drei Möglichkeiten:

- **sporadisch:** Das Signal wird zu unregelmäßigen, i. d. R. Ereignis-abhängigen Zeitpunkten gesendet. Im Beispiel mit dem Außenspiegel könnte das die Information sein, dass der Taster, zum Ein- bzw. Ausschalten der Heizung, gedrückt wurde. Zwischen zwei Ereignissen sind keine Änderungen zu erwarten.
- **zyklisch:** Das Signal wird in regelmäßigen Abständen, z. B. alle 50ms gesendet. Die Daten des Temperatursensors, mit denen die automatische Entscheidung zur Aktivierung der Spiegelheizung getroffen wird, könnten zyklisch übertragen werden. Die Temperatur bildet einen Werteverlauf, bei dem regelmäßige Änderungen zu erwarten sind.
- **zyklisch+Ereignis-abhängig:** Das Signal wird mit einer etwas höheren Zykluszeit (z. B. 1000ms) periodisch gesendet. Zusätzlich sind, durch Ereignisse ausgelöst, vorzeitige Übertragungen möglich. Durch die regelmäßigen Übertragungen kann sichergestellt werden, dass der Kommunikationspfad funktioniert (bei Ausbleiben der Signale müsste ein entsprechendes Notprogramm, z. B. die Ausgabe einer Warnmeldung, aktiviert werden). Gleichzeitig kann auf Ereignisse schneller als die Zykluszeit reagiert werden. In der Beispielanwendung wäre das ein Übertragungsmodus für den Totwinkelassistenten. Ein sporadisches Signal, das nicht empfangen wird, könnte hier bedeuten, dass sich kein Fahrzeug im toten Winkel befindet oder ein Fahrzeug im toten Winkel liegt, aber der entsprechende Sensor ausgefallen ist. Mit einer zyklisch+Ereignis-abhängigen Übertragung würde der Ausfall des Sensors innerhalb einer Zykluszeit erkannt werden.

Als Beispiel für eine Kommunikation kann der Informationsaustausch zwischen den Komponenten des Systems aus Abschnitt 2.3.4 verwendet werden. Wird hier die aktuelle Außentemperatur als ein Signal übertragen, ergibt sich

eine Sequenz wie in Abbildung 2.9. Der Temperatursensor überträgt (z. B. nach einem festgelegten Intervall) die gemessene Temperatur an die Komponente, die die Heizung steuert. Diese Komponente wertet die Temperatur aus (z. B. in Form eines Vergleichs mit einem Grenzwert und einer Hysteresefunktion) und leitet daraus eine Steuerungsinformation für die Spiegelheizung ab. Dieser Steuerbefehl wird wiederum als Signal („Heizung an/aus“) an die Spiegelheizung übertragen.

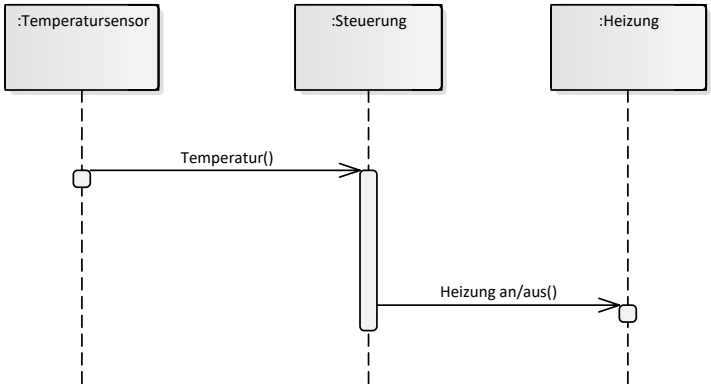

Abbildung 2.9: Sequenzdiagramm für eine Signalübertragung

2.4.7 Service-orientierte Kommunikation

Ein anderes Kommunikationsschema ist die Service-orientierte Kommunikation. Hier werden sowohl Informationen (z. B. „Außentemperatur“) als auch Funktionen (z. B. „Spiegel heizen“) als sogenannte Services angeboten. Diese werden anders als bei der Signal-basierten Kommunikation (vgl.: Der Temperatursensor, der zyklisch ein Temperatursignal sendet) nicht automatisch versendet. Stattdessen wird der Austausch durch einen sogenannten Client angestoßen. Der Client ist die Komponente, die die Information verwendet bzw. die Funktion benötigt. Die Verbindung zwischen Client und Server wird zur Laufzeit hergestellt.

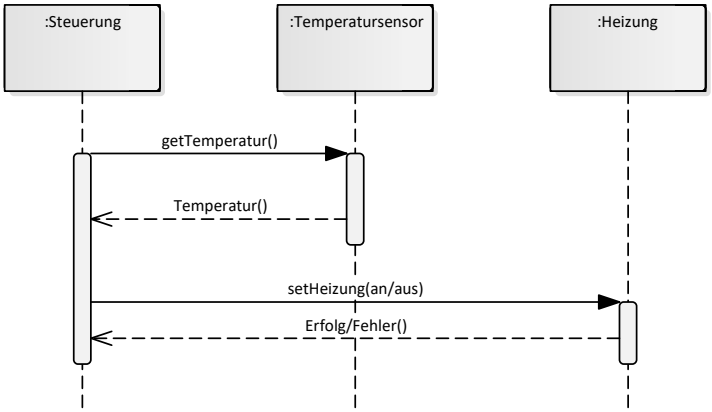

Abbildung 2.10: Sequenzdiagramm für einen Serviceaufruf

Im Fall der Spiegelheizung erfolgt der Informationsaustausch wie in Abbildung 2.10 dargestellt: Die Steuerung ist sowohl für den Service des Temperatursensors als auch für den Service der Spiegelheizung der Client. Zunächst wird beim Temperatursensor die aktuelle Temperatur angefordert. Als Antwort darauf überträgt der Sensor seine Information. Aus dieser wird die Steuerungsfunktion für die Heizung abgeleitet. Anschließend wird der Spiegelheizungsservice aufgerufen, um diese an- oder auszuschalten.

In der Praxis kommen, um die Vorteile beider Kommunikationsparadigmen nutzen zu können, hybride Architekturen vor [Vet21a]. D. h. in Topologien analog zu Abb. 2.8 wird im Ethernet-Teil Service-orientiert und auf den anderen Bussen Signal-basiert kommuniziert.

2.5 K-Matrix

Tabelle 2.7: Eine einfache K-Matrix mit zwei Bussen und einem Gateway.

	ECU1	ECU2	ECU3	Gateway1	ECU4	ECU5
Signal 1	send.	empf.	empf.			
Signal.2	empf.	empf.				
Signal 3	empf.		empf.	weiterl.	send.	empf.
Signal 4		send.	empf.	weiterl.		empf.
Signal 5					send.	empf.
Signal 6					empf.	send.
Bus	1	1	1	1 & 2	2	2

Zur statischen Planung der Kommunikation wird für jedes Signal festgelegt, von welcher ECU es gesendet wird, welche ECUs es empfangen, und ggf. durch welche Gateways es zwischen unterschiedlichen Netzwerkteilen weitergeleitet wird. Mit den Signalen als Zeilen sowie den Steuergeräten und Gateways als Spalten lässt sich die geplante Kommunikation als Matrix darstellen. Diese wird als Kommunikationsmatrix (kurz: K-Matrix) bezeichnet.

Ein Beispiel für eine K-Matrix ist in Tabelle 2.7 dargestellt. Die Signale 1 u. 2 sowie 5 u. 6 werden jeweils nur lokal übertragen, während 3 u. 4 durch das Gateway „GW1“ weitergeleitet werden.

In der Praxis sind zur vollständigen Beschreibung des Datenaustauschs in einem Fahrzeugnetzwerk mehr Informationen als Sender und Empfänger ECUs der Signale erforderlich. Beispiele sind die Identifiers (IDs) der Botschaften oder die Konfiguration von Transportmechanismen, die übergroße Datenpakete fragmentiert übertragen können. Zur vollständigen Beschreibung des Netzwerks wird das Metamodell des AUTOSAR Standards (siehe Abschnitt 2.7) verwendet und in einer Datenbank abgelegt. Auch hierfür ist der Begriff K-Matrix gebräuchlich.

Eine wesentliche Eigenschaft von K-Matrizen ist deren Konsistenz.

Definition 2.1 (K-Matrix Konsistenz)**.** *Eine K-Matrix wird als konsistent bezeichnet, wenn sie keine Widersprüche enthält.*

Beispiele für Widersprüche sind Signale ohne Sender bzw. ohne Empfänger (Voraussetzung für Kommunikation ist ein oder mehrere Sender und ein oder mehrere Empfänger). Zustände, die nicht vorkommen dürfen, bzw. Bedingungen, die zwingend erfüllt sein müssen, werden als Konsistenzregeln bezeichnet. Werkzeuge zur K-Matrix Entwicklung können, anhand solcher Konsistenzregeln, Kommunikationsmatrizen automatisch auf Konsistenz überprüfen.

2.6 Software

Zentraler Bestandteil einer ECU ist ein Mikrocontroller (vgl. Abschnitt 2.3.1). Dessen Prozessor benötigt ausführbaren Programmcode und Daten (z. B. Konfigurationen oder Kennlinienfelder) welche auf dem Steuergerät gespeichert werden. Diese bezeichnet man als „Software“.

Wesentlich für den Betrieb des Fahrzeugnetzwerkes ist, dass auf allen Steuergeräten die Software, welche auch Datenversand und -empfang regelt, reibungslos mit dem Rest des Netzwerks interagiert. Auf ein einzelnes Steuergerät bezogen, lässt sich die Software grob in zwei Einheiten aufteilen: die Funktions- und die Basissoftware.

2.6.1 Zertifizierung

Im Rahmen der EG-Typgenehmigung muss nachgewiesen werden, dass ein Fahrzeug den Gesetzen und Vorschriften der Europäischen Union entspricht. Die Typgenehmigung ist Voraussetzung für die Erteilung einer allgemeinen Betriebserlaubnis (ABE) und die Zulassung der jeweiligen Fahrzeuge [Bun21]. Software auf Komponenten, die von dieser Typgenehmigung erfasst werden

(z. B. ABS, Airbag- oder Motorsteuerung), bezeichnet man als „zertifizierungsrelevant“. Bei Änderungen an solchen Softwarekomponenten müssen Teile der Typgenehmigung erneuert, bzw. der Nachweis erbracht werden, dass sich die relevanten Eigenschaften nicht verändert haben. In der Praxis führt das dazu, dass Änderungen an zertifizierungsrelevanten Steuergeräten vermieden werden.

2.6.2 Funktionssoftware

Die Funktionssoftware setzt die eigentliche Funktionalität des Steuergerätes um, z. B. „wenn Rad blockiert, dann Bremse für X Millisekunden öffnen“. Die Funktionssoftware ist dabei abstrakt gehalten. D. h. es wird nicht konkret die Spannung einzelner Pins eines Mikrocontrollers beeinflusst, sondern es werden abstrakte Funktionen der Basissoftware wie z. B. „Bremse öffnen“ oder „Datenpaket Y auswerten“ aufgerufen. So können die Funktionen eines Steuergerätes unabhängig von der zugrunde liegenden Hardware entwickelt werden.

Die Komponente „Steuerung“ (vgl. Abb. 2.3) aus dem Beispiel mit der Spiegelheizung ist ebenfalls Funktionssoftware.

2.6.3 Basissoftware

Verschiedene Steuergeräte können Mikrocontroller mit unterschiedlichen Prozessorarchitekturen verwenden. Ferner werden Peripheriekomponenten (wie z. B. Kommunikationsschnittstellen) bei Mikrocontrollern verschiedener Hersteller unterschiedlich konfiguriert. Weitere Unterschiede können durch den Aufbau der Platinen entstehen. So kann etwa ein Aktor in einer Revision eines Steuergeräts an einen anderen Pin des Controllers angeschlossen sein. All diese Unterschiede sollen durch die Basissoftware vor der Funktionssoftware verborgen werden. So stellt die Basissoftware einer

ECU, die in Abschnitt 2.6.2 verwendeten abstrakten Funktionen, der Anwendungssoftware zur Verfügung. Dabei wird auf die konkret vorliegende Hardware eingegangen.

Ein Beispiel wäre, dass der Schaltkreis zum Öffnen der Bremse an Pin X des Mikrocontrollers angeschlossen ist, dass der Pin auf einen bestimmten Pegel geschaltet werden muss, um die Bremse zu öffnen, und bei dem vorliegenden Mikrocontroller der Wert Y in die Speicheradresse Z geschrieben werden muss, um den Pin auf den erforderlichen Pegel zu bringen. All diese Informationen werden in eine Funktion „Bremse öffnen“ gekapselt, die durch die Funktionssoftware aufgerufen wird. Finden Änderungen an der Hardware statt, muss nur die Basissoftware angepasst werden, während die Funktionssoftware unverändert weiterverwendet werden kann.

Ferner müssen die verschiedenen Ressourcen (Rechenzeit, Speicher, Zugriff auf Peripheriebausteine etc.) verwaltet und die Ausführungsreihenfolge und -zeitpunkte der verschiedenen Funktionssoftwaremodule festgelegt werden (d. h. ein Scheduling durchführen). Mit diesen Eigenschaften kann die Basissoftware als ein Betriebssystem angesehen werden.

Für eine gemeinsame Basissoftware in der Automobilindustrie existiert der AUTomotive Open System ARchitecture (AUTOSAR) Standard [AUT23d].

2.6.4 Software-Integration

Das Zusammenführen einer oder mehrerer Funktionssoftwarekomponenten mit der Basissoftware wird als Integration bezeichnet. Aus der abstrakten Funktionssoftware und der an Steuergerätehardware und K-Matrix angepassten Basissoftware wird eine Einheit geschaffen, die auf dann auf einem realen Steuergerät ausgeführt werden kann.

2.6.5 (Over-the-Air-) Updates

Wird die bestehende Software in einem Steuergerät durch eine neuere ersetzt, spricht man von einem Update. Bei Fahrzeugen, die bereits ausgeliefert wurden, erfolgt dies über das Mobilfunknetz. Man spricht dann von Over The Air (OTA) Updates bzw. Software Over The Air (SOTA) [Sax17, Gui18].

2.7 AUTOSAR

Im August 2002 fanden sich fünf große Automobilhersteller und -Zulieferer[1] zusammen, um die gemeinsame Entwicklung eines Softwarestandards zu diskutieren. Dies führte zur Gründung der AUTOSAR Entwicklungspartnerschaft im Jahr 2003. Ziel dieser Partnerschaft war es, die Entwicklungskosten für Standardkomponenten wie die Basissoftware zu reduzieren. Außerdem sollten einheitliche Austauschformate geschaffen und eine einfachere Übertragbarkeit von Software zwischen unterschiedlichen Hardwareplattformen ermöglicht werden [Pel05]. Bereits 2009 hatte die Mehrheit der Entwicklungspartner AUTOSAR Software in Serienfahrzeugen enthalten [Für09].

Der AUTOSAR Standard lässt sich gemäß [Dar21] in vier Komponenten unterteilen:

- Eine standardisierte Middleware für Steuergeräte
- Eine standardisierte Steuergerätearchitektur
- Eine standardisierte Entwicklungsmethodik
- Ein standardisiertes Austauschformat

[1] BMW, Bosch, Continental, DaimlerChrysler und Volkswagen

2.7.1 Middleware für Steuergeräte

Nach [Tan07] ist ein verteiltes System „eine Sammlung unabhängiger Computer, die ihren Nutzern als ein einzelnes zusammenhängendes System erscheinen“. Ersetzt man den Begriff „Computer“ durch „Steuergeräte“, beschreibt dies die Verhältnisse in einem Fahrzeugnetzwerk. Um zwischen den heterogenen Geräten eine einheitliche Kommunikation zu ermöglichen, ist eine Zwischenschicht erforderlich, die als Middleware bezeichnet wird [Ber96].

Der AUTOSAR Standard enthält detaillierte funktionale Beschreibungen des Kommunikationsablaufs bzw. bietet den OEMs die Möglichkeit den Kommunikationsablauf zu konfigurieren. Der Standard beinhaltet Protokolle für die Kommunikation auf den verschiedenen Bussen (z. B. Scalable service-Oriented MiddlewarE over IP (SOME/IP) – ein Protokoll zur Service-orientierten Kommunikation über Ethernet [AUT23g]), sowie Routingmechanismen mit denen die Weiterleitung von Botschaften gesteuert werden kann. Damit erfüllt AUTOSAR die Rolle einer Middleware.

2.7.2 Steuergerätearchitektur

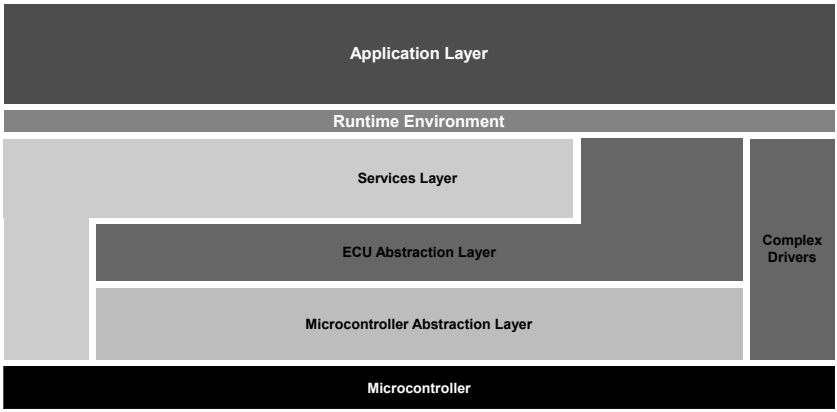

Abbildung 2.11: Die Schichten der AUTOSAR Steuergerätearchitektur (Bild: [AUT23d, S. 22])

Die Referenzarchitektur für AUTOSAR Steuergeräte wird in [AUT23d] vorgestellt. Von unten nach oben betrachtet basiert die Architektur eines AUTOSAR Steuergeräts zunächst auf einem Mikrocontroller (s. Abb. 2.11). Um unabhängig von einer speziellen Mikrocontrollerfamilie oder Version zu sein, gibt es eine Mikrocontrollerabstraktionsschicht. Diese stellt der ECU-Abstraktionsschicht einheitliche Schnittstellen zur Verfügung. Die Mikrocontrollerabstraktionsschicht besteht aus der Treibersoftware für die diversen Hardwarebausteine des Mikrocontrollers.

Die ECU-Abstraktionsschicht greift auf die Treiber der Mikrocontrollerhardware zu und enthält die Treiber für (vom Mikrocontroller unabhängige) Peripheriegeräte. Ein Beispiel hierfür kann ein externer Beschleunigungssensor sein, welcher über eine Schnittstelle an den Mikrocontroller angebunden ist. Der Treiber für den Beschleunigungssensor befindet sich in der ECU-Abstraktionsschicht, greift aber auch auf den in der Mikrocontrollerabstraktionsschicht enthaltenen Schnittstellentreiber zu, um mit dem Sensor kommunizieren zu können. Oberhalb der ECU-Abstraktionsschicht spielt es keine Rolle mehr, ob eine Komponente Teil des Mikrocontrollers oder ein externer Baustein (aber Teil desselben Steuergeräts) ist.

Die Service-Schicht setzt auf der ECU-Abstraktionssicht auf. Sie bietet diverse Standardfunktionen – vergleichbar mit den Funktionen des Betriebssystems auf einem PC – für die Anwendungssoftware an (z. B. Diagnosefunktionen oder das dauerhafte[1] Abspeichern von Informationen).

Die Runtime Environment (RTE) bzw. Laufzeitumgebung stellt das Interface der Basissoftware dar. Sie bildet die Middleware für die Kommunikation der Anwendungssoftwaremodule auf dem Steuergerät. Durch die RTE kann die Anwendungssoftware auf die ECU Abstraktionsschicht bzw. auf die Serviceschicht zugreifen, um zu kommunizieren und die Hardwarefunktionen der ECU anzusteuern.

[1] Dauerhaft abspeichern bedeutet im Kontext von Automobilelektronik, dass Informationen erhalten bleiben, auch wenn das Steuergerät nicht mehr mit Strom versorgt wird. Der Speicher hierfür wird als Nonvolatile Random Access Memory (NVRAM), also nichtflüchtiger, frei adressierbarer Speicher bezeichnet. Technisch handelt es sich dabei in der Regel um Flash- oder EEPROM-Speicher welcher Teil des Mikrocontrollers ist.

Oberhalb der RTE befindet sich das Application Layer (deutsch: Anwendungsschicht), in welchem sich die Funktionssoftware (vgl. Abs. 2.6.2). Diese ist in einzelne Module aufgeteilt, welche als Software Components (SWCs) bezeichnet werden. Die SWCs kommunizieren untereinander über die RTE. Es wird zwischen sogenannten „Atomic SWCs“ und „SWC Compositions“ unterschieden. Eine „SWC Composition“ setzt sich aus „Atomic SWCs“ und anderen „SWC Compositions“ zusammen. Ebenfalls durch die RTE geschieht das Scheduling der verschiedenen Applikationsmodule.

Abbildung 2.11 stellt die besprochenen Schichten dar. Zu sehen ist außerdem, dass Schichten teilweise direkt auf die Hardware oder ECU-Abstraktionsschicht zugreifen. So kann der Betriebssystemanteil der Service-Schicht den Mikrocontroller direkt verwenden und die RTE direkt die ECU-Abstraktionssicht. Von besonderer Bedeutung sind die sogenannten „Complex Drivers“, welche einen Direktzugriff von der RTE auf den Mikrocontroller ermöglichen. Dieses Konzept ermöglicht die Integration bestehender (nicht auf AUTOSAR basierender) Softwareelemente, in ein auf AUTOSAR basierendes Gesamtsystem.

2.7.3 Virtual Functional Bus

Für zwei Applikationen ist es nicht von Bedeutung, ob diese sich auf demselben Steuergerät befinden oder nicht. Dies ist ein besonderes AUTOSAR Konzept. Ihre RTE-Schnittstelle ist einheitlich, während die Kommunikation entweder lokal auf dem Steuergerät umgesetzt oder, eine externe Kommunikation über Busnachrichten durchgeführt wird. Dieses Konzept wird als Virtual Functional Bus (VFB) bezeichnet (siehe Abbildung 2.12). Es vereinfacht das Verschieben von SWCs zwischen Steuergeräten, und kann so dazu beitragen, eine ungleichmäßige Auslastung von Steuergeräten innerhalb eines Fahrzeuges zu reduzieren[1]. Innerhalb der Basissoftware ist das AUTOSAR Communication Module (COM) zuständig, den VFB umzusetzen. COM stellt

[1] Zu beachten ist, dass die gesamte Software eines Steuergerätes statisch integriert wird. D. h. ein Verschieben einzelner Applikationen kann während der Entwicklung geschehen, jedoch nicht dynamisch zur Laufzeit.

der RTE und den darin laufenden SWCs die Signal-basierte Schnittstelle zur Verfügung [AUT23f].

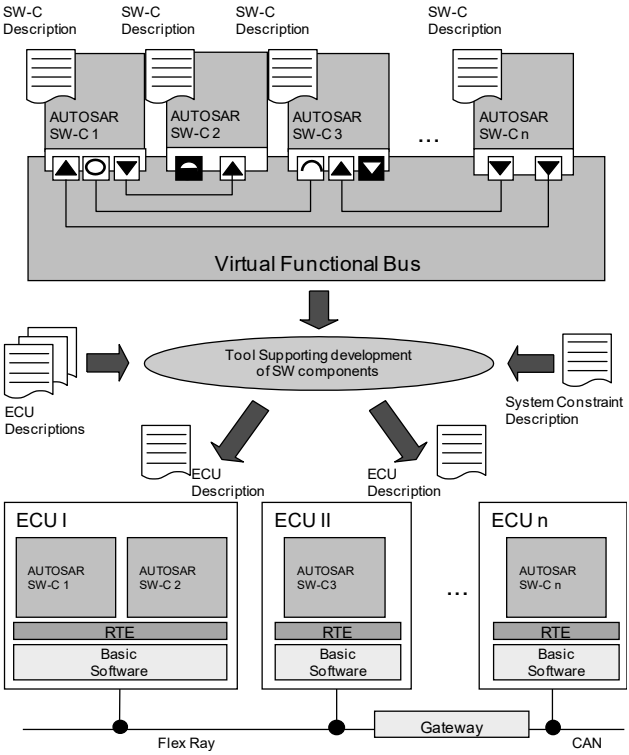

Abbildung 2.12: Unterschiedliche Applikationen – teils auf derselben, teils auf verschiedenen ECUs – über den Virtual Functional Bus verbunden (Bild: [AUT23j, S. 13])

Im Abschnitt 2.3.4 wurde das Beispielsystem „Spiegelheizung“ vorgestellt. Dessen Komponenten befinden sich auf drei verschiedenen Steuergeräten, wobei die Komponente „Steuerung“ ein reines Softwaremodul (d. h. eine SWC, die nicht direkt auf Hardwareressourcen zugreift) ist. Über den VFB ist sie mit der Status LED, dem Taster, dem Temperatursensor und der Heizung verbunden. Dabei stellt die RTE zum Temperatursensor und zur Heizung Verbindungen über physikalische Busse (LIN zur Heizung, und

CAN zum Temperatursensor) her. Für die Verbindungen zur Status-LED und zum Taster werden lediglich Daten, zwischen den verschiedenen Software Komponenten auf dem Türsteuergerät, ausgetauscht. So ließe sich die Steuerungskomponenten problemlos auf das Spiegelsteuergerät verschieben.

2.7.4 Anwendungsdatentypen

Über den VFB werden Informationen digital, d. h. in Form von Bitfolgen, übertragen. Auf Systemebene repräsentieren diese Bitfolgen physikalische Größen, Systemzustände oder andere Informationen. Mittels sogenannter Application Data Type (ADT), deutsch: Anwendungsdatentypen, wird beschrieben, wie durch eine Bitfolge eine physikalische Größe oder eine andere Information darstellt. Der Anwendungsdatentyp ist eine wesentliche Eigenschaft jedes Signals (vgl. Abs. 2.4.6).

Im Beispiel mit dem Außenspiegel, könnte der ADT für die Temperatur z. B. festlegen, dass mittels 8 Bit (256 mögliche Werte) der Temperaturbereich von -30 ℃ bis +95 ℃ in 0,5 K Schritten (250 mögliche Werte) abgebildet wird. Ausgehend von einem Zahlenbereich von 0 bis 255, bleiben damit die Werte 250 bis 255 unbelegt. Diese verbleibenden Werte können für Statuscodes (z. B. 255 entspricht „Sensor defekt“) genutzt werden. Es ist aber auch möglich, den Wertebereich komplett für Messwerte (in diesem Fall z. B. von -30 ℃ bis +98 ℃) zu nutzen. Das Signal für die Heizung kann als Beispiel für zu übertragende Systemzustände genutzt werden. So könnten z. B. die Zustände „nicht Heizen“, „halbe Leistung“ und „volle Leistung“ auf die mit zwei Bit zu kodierenden Zahlen von null bis drei abgebildet werden. Auch hier bleibt eine mögliche Kombination der Bits ungenutzt. [AUT23a, S. 33ff]

2.7.5 Metamodell und Austauschformat

Die Unified Modeling Language (UML) der Object Management Group (OMG) erlaubt es beliebige Systeme grafisch in verschiedenen Diagrammen zu modellieren [Obj17]. UML basiert auf der Meta Object Facility (MOF) der

OMG [Obj15]. Unter Verwendung eines beschränkten Subsets der UML wurde das AUTOSAR Metamodell erstellt. Dieses beschreibt, was in AUTOSAR modellierbar ist. Das AUTOSAR Metamodell lässt sich wie in Tabelle 2.8 dargestellt in fünf Ebenen aufteilen [AUT23b, S. 22ff].

Ein Teil des AUTOSAR Metamodells ist in Abbildung 2.13 dargestellt. In diesem Ausschnitt wird die Verbindung zwischen einem abstrakten Signal – in AUTOSAR als `SystemSignal` bezeichnet – und einem konkreten, physischen Übertragungsmedium – in AUTOSAR als `PhysicalChannel` bezeichnet – gezeigt. Die als „Application Layer“ bezeichnete Ebene in Abb. 2.13 entspricht dem VFB (vgl. Abs. 2.7.3) bzw. der Anwendungsschicht in der Steuergerätearchitektur (vgl. Abs. 2.7.2 bzw. Abb. 2.11). Die abstrakten `SystemSignal` Objekte sind die Ein- und Ausgaben der Hardware-unabhängigen SWCs. In der Ebene „Interaction & Network Layer“ liegen Elemente, die den Umgang der AUTOSAR Basissoftware mit den zu übertragenden Signalen betreffen. Sie entspricht der Serviceschicht in Abb. 2.11. Die Schicht „Data Link Layer“ entspricht der Mikrocontrollerabstraktionsschicht bzw. dem physischen Mikrocontroller und dem Bus[1]. Eine Einordnung der AUTOSAR Protokollschichten in das OSI-Referenzmodell bzw. das TCP/IP-Modell erfolgt in Tabelle 3.2 in Abschnitt 3.1.

[1] Die als Teil der `FrameTriggering` Objekte modellierten Frame-IDs entsprechen den IDs, die bei einer Messung, mit einem geeigneten Messgerät, an einem physischen Fahrzeugnetzwerk abgelesen werden.

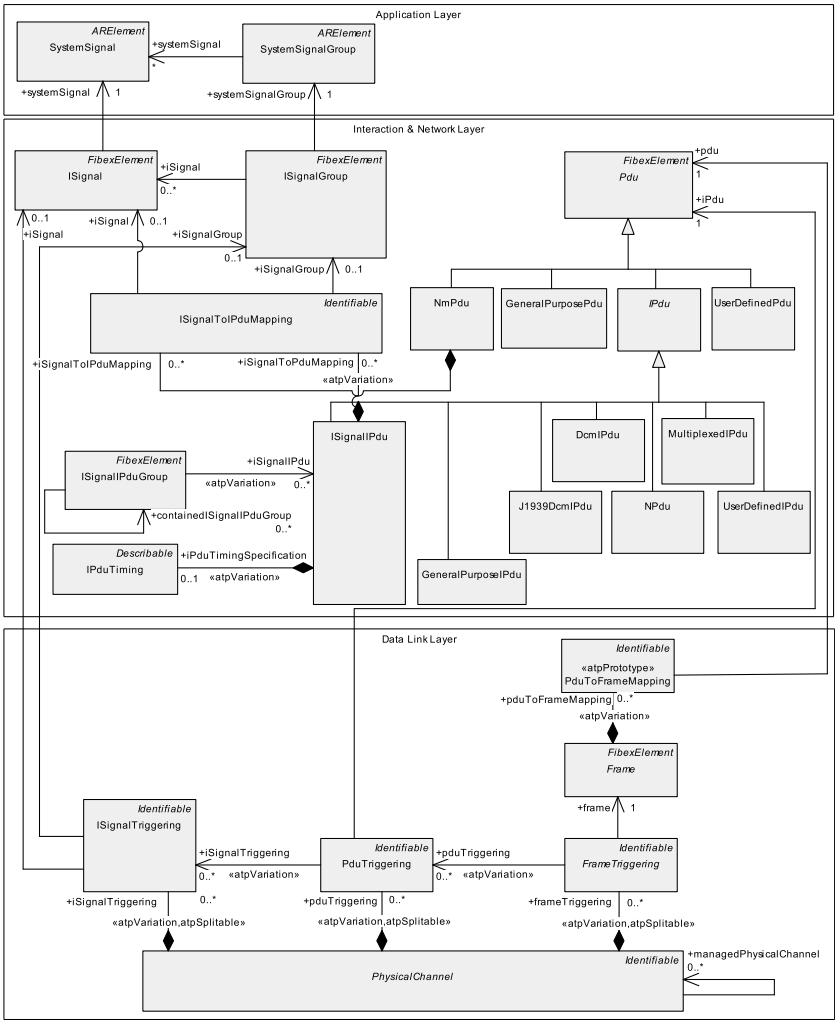

Abbildung 2.13: Überblick über das AUTOSAR Metamodell für Datenübertragung im Fahrzeugnetzwerk (Bild: [AUT23i, S. 295])

Die Teile des Metamodells, welche ausschließlich die Kommunikation beschreiben, entsprechen dem 2002 etablierten „Field Bus EXchange format

(FIBEX)“ Standard [Kra04]. In Abbildung 2.13 sind die entsprechenden Elemente als `FibexElement` markiert.

Zu einem abstrakten `SystemSignal` gibt es ein oder mehrere `ISignal` Objekte (Signale der Interaktionsschicht [AUT23i, S. 320]), jeweils eins pro verwendetem Übertragungspfad. Ein `ISignal` kann entweder alleine verwendet oder mit anderen `ISignal` Objekten zu einer `ISignalGroup` zusammengefasst werden.

Tabelle 2.8: Vergleich der Modellierungsebenen

AUTOSAR Modell	OMG Modell	Beschreibung
M4	MOF Klasse	Das Metamodell, mit dem andere Metamodelle wie UML2 spezifiziert werden[Obj15, S. 41]
M3	UML Klasse	AUTOSAR UML Profil (d. h. das UML Subset für AUTOSAR)
M2	UML Model	Beschreibung wie man in AUTOSAR Steuergeräte beschreibt
M1	UML Model	Beschreibung eines Steuergeräts
M0	User Object	Ein Steuergerät

Über ein `ISignalToIPduMapping` werden `ISignal` bzw. `ISignalGroup` zu einer `IPdu` (eine Protocol Data Unit (PDU) der Interaktionsschicht[1]) hinzugefügt. PDUs repräsentieren die übertragene Information in den unterschiedlichen AUTOSAR-Netzwerkschichten [AUT23i, S. 294]. Die Interaktionsschicht („Interaction Layer“) dient der Verbindung zwischen der Anwendungsschicht („Application Layer“) und den darunter liegenden Schichten. In der Netzwerkschicht („Network Layer“) befinden sich Mechanismen, um beispielsweise PDUs in mehreren Teilstücken über ein Medium zu übertragen, welches keine ausreichend große Datenpakete übertragen kann. Z. B. kann eine 16 Bytes große `IPdu` aufgetrennt und in Form zweier 8 Byte großer `NPdu` Objekte (PDUs der Netzwerkschicht) über ein CAN (maximale Framegröße von 8 Bytes) übertragen werden. Bei der Übertragungsschicht („Data Link Layer“)

[1] Siehe Definition 3.3 für eine allgemeine, AUTOSAR-unabhängige Definition des PDU-Begriffs.

handelt es sich um die konkreten Datenbusse und deren Adressierungs- und Paketierungsmechanismen.

Eine PDU wird über ein `PduToFrameMapping` mit einem `Frame` des jeweiligen Kommunikationsmediums verknüpft. Da der konkrete Aufbau der übertragenen `Frame` Objekte das nach außen sichtbare Verhalten ergibt, ist diese Hierarchie für die vorliegende Arbeit relevant. Das `Frame` Objekt ist über ein `FrameTriggering` mit dem `PhysicalChannel` verbunden. Das `FrameTriggering` beschreibt die Instanzen der Frames auf dem physikalischen Übertragungskanal [AUT23i, S. 418]. Der Begriff „Triggering“ (engl. für „Auslösung“) drückt aus, dass das Objekt den Versand oder Empfang eines Frames auslöst. Analog dazu existieren auch `PduTriggering` und `ISignalTriggering` Objekte.

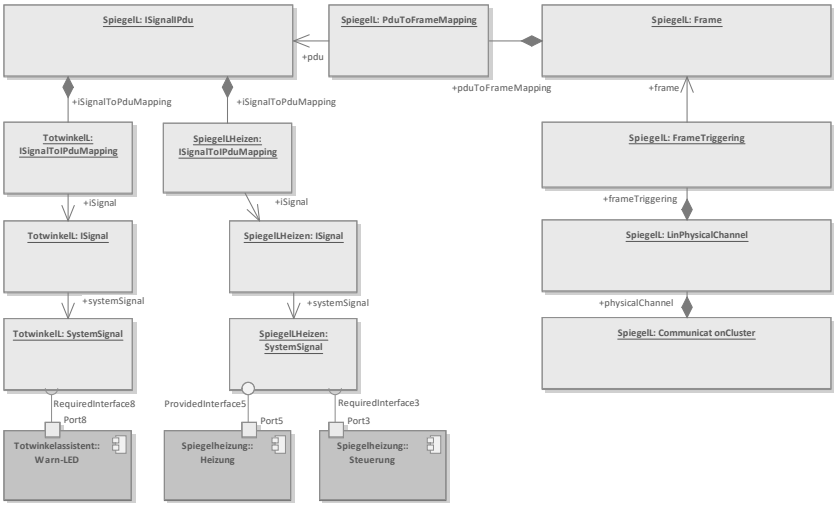

Abbildung 2.14: Objektstruktur einer PDU, die der Außenspiegel empfängt

Zur Veranschaulichung dieses Datenmodells kann wieder der Außenspiegel und dessen Kommunikation herangezogen werden. Das Komponentendiagramm in Abbildung 2.3 beinhaltet zwischen der Steuerung auf dem Türsteuergerät und der Heizung auf dem Spiegelsteuergerät eine Verbindung. Diese kann wie in Abschnitt 2.4.6 erläutert durch ein

`SystemSignal SpiegelLHeizen` („SpiegelL“ für den linken Spiegel) hergestellt werden. Dieses `SystemSignal` wird durch das `ISignal SpiegelLHeizen` umgesetzt und durch das `ISignalToIPduMapping SpiegelLHeizen` in die `ISignalIPdu SpiegelL` integriert. Diese enthält außerdem das `ISignal TotwinkelL`. Das `PduToFrameMapping SpiegelL` verknüpft die `IPdu` mit dem `Frame SpiegelL`. Der `Frame` wiederum ist über das `FrameTriggering SpiegelL` mit dem `LinPhysicalChannel SpiegelL`. Diese Objektstruktur ist in Abb. 2.14 dargestellt, einschließlich der Komponenten aus Abb. 2.3, die die Signale verwenden. Sichtbar ist auch, dass in einer `ISignalIPdu` mehrere, unterschiedlichen Systemen zugehörige, `SystemSignal` enthalten sein können.

Der Austausch zwischen verschiedenen Entwicklungspartnern erfolgt auf Basis eines standardisierten Extensible Markup Language (XML) Schemas. Dieses wird als AUTOSAR XML (ARXML) bezeichnet [Pag06]. ARXML dient dem Speichern des auf Basis des AUTOSAR Metamodells (vgl. M2 in Tab. 2.8 u. Abb. 2.13) erstellten Modells (vgl. M1 in Tab. 2.8 u. Abb. 2.14). An den verschiedenen Übergabepunkten des Entwicklungsprozesses (siehe Abschnitt 2.7.6) werden jeweils ARXML-Dateien verwendet, die sich in Detaillierungsgrad und dem beschriebenen Bereich unterscheiden (z. B. „ein Steuergerät“ oder „ein Bus“) [AUT23a]. All diese Dateien sind jeweils eine Teilmenge des AUTOSAR System Extracts, welcher das gesamte Fahrzeugnetzwerk modelliert.

Zur Veranschaulichung dient das `SystemSignal SpiegelLHeizen` (siehe Listing 2.1). Das gesamte `SystemSignal` Objekt wird als `<SYSTEM-SIGNAL>` Block innerhalb der ARXML-Datei abgebildet. Darin befinden sich der Name in Kurz- und Langform sowie Beschreibungen in unterschiedlichen Sprachen. Es folgt der `<ADMIN-DATA>` Block, welcher in `<DOC-REVISIONS>` und `<SDGS>` (Special Data Group) aufgeteilt ist.

Innerhalb von `<DOC-REVISIONS>` befinden sich Metadaten zur aktuellen Version des Objekts, sowie optional zu Vorgängerversionen. Im dargestellten Beispiel ist nur die aktuelle Version 1.0.0 enthalten. Deren Zustand (`<STATE>`) ist als „released“ (deutsch: veröffentlicht) eingetragen. Außerdem ist das Datum der letzten Änderung notiert. Für weitere Versionen gäbe es mehrere `<DOC-REVISION>` Blöcke innerhalb von `<DOC-REVISIONS>`.

Listing 2.1: Der vollständige ARXML Code zum `SystemSignal SpiegelLHeizen`.

```
<SYSTEM-SIGNAL T="2020-08-31T10:42:41+01:00" UUID="138
    FED6FE20311EAB65B68B5996D7596">
  <SHORT-NAME>SpiegelLHeizen</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="FOR-ALL">SpiegelLHeizen</L-4>
  </LONG-NAME>
  <DESC>
    <L-2 L="DE">Spiegel L Heizen</L-2>
    <L-2 L="EN">Mirror L Heating</L-2>
  </DESC>
  <ADMIN-DATA>
    <DOC-REVISIONS>
      <DOC-REVISION>
        <REVISION-LABEL>1.0.0</REVISION-LABEL>
        <STATE>RELEASED</STATE>
        <DATE>2020-08-31T10:43:04+01:00</DATE>
      </DOC-REVISION>
    </DOC-REVISIONS>
    <SDGS>
      <SDG GID="uuid">
        <SDG-CAPTION>
          <SHORT-NAME>uuid</SHORT-NAME>
          <DESC>
            <L-2 L="EN">debug</L-2>
          </DESC>
        </SDG-CAPTION>
        <SD GID="uuid">Signal:
            EEBEB0A6EB6511EA86E90050560E0A7C</SD>
      </SDG>
    </SDGS>
  </ADMIN-DATA>
  <DYNAMIC-LENGTH>false</DYNAMIC-LENGTH>
  <PHYSICAL-PROPS>
    <DATA-CONSTR-REF DEST="DATA-CONSTR">/
        ApplicationDataTypes/XDIS_66a3n5md1
    </DATA-CONSTR-REF>
  </PHYSICAL-PROPS>
</SYSTEM-SIGNAL>
```

Die Special Data Groups sind ein Mechanismus, um AUTOSAR Objekte mit Zusatzinformationen zu versehen, die nicht Teil des Standards sind. So können herstellerspezifische Funktionalitäten und Prozessschritte unterstützt werden. Im abgebildeten Code gibt es die SDG „UUID“ (Universally Unique Identifier[1]). Diese dient der Fehlersuche (engl. „Debugging, daher die Beschreibung „debug“), indem sie einen eindeutigen Verweis auf das Datenbankobjekt, der Software, mit welcher die ARXML-Datei erstellt wurde, liefert.

Den Special Data Groups folgen die eigentlichen Eigenschaften des jeweiligen Objekts. Im Beispiel mit dem `SystemSignal` wird festgelegt, ob die Länge statisch oder dynamisch ist (durch `<DYNAMIC-LENGTH>`), sowie die physikalischen Eigenschaften (vgl. Abs. 2.7.4) im Block `<PHYSICAL-PROPS>` (kurz für „physical properties“). Im Beispielfall erfolgt ein Verweis auf den eigens angelegten Datentyp `/ApplicationDataTypes/XDIS_66a3n5md1`. Dieser ist ein eigenes Objekt an anderer Stelle in der ARXML-Datei (von der das abgebildete `SystemSignal` Objekt nur einen kleinen Ausschnitt darstellt) definiert.

2.7.6 Entwicklungsmethodik

Die AUTOSAR Entwicklungsmethodik sieht einen Top-Down Entwurf vor (vgl. Abb. 2.15). Dabei wird zunächst auf hoher Abstraktionsebene das Gesamtsystem[2] beschrieben („System Configuration Input“). Dieses besteht aus einer Menge an SWCs, die die Subsysteme repräsentieren, sowie einer Menge an Steuergeräten, welche die Sensoren und Aktoren zur Verfügung stellen und die SWCs ausführen können. Bei der Systemkonfiguration werden die SWCs (und damit die Subsysteme) den ECUs zugeordnet und als „System Configuration Description“ abgelegt. Subsysteme, die über mehrere Steuergeräte verteilt sind, werden dabei durch SWC Compositions abgebildet, deren Komponenten den jeweiligen Steuergeräten zugeordnet sind. Anschließend werden

[1] UUIDs sind 128 Bit lange eindeutige Identifizierungsnummern für digitale Objekte, die nach einem standardisierten Schema automatisch erzeugt werden [Int05b].

[2] In AUTOSAR versteht man unter System das vollständige Fahrzeugnetzwerk. Darin enthaltene Funktionsblöcke, wie die Spiegelheizung aus dem Beispiel, werden als Subsysteme angesehen.

aus der Beschreibung des konfigurierten Systems die Anteile der einzelnen Steuergeräte extrahiert. Dabei entstehen die sogenannten „ECU-Extracts“.

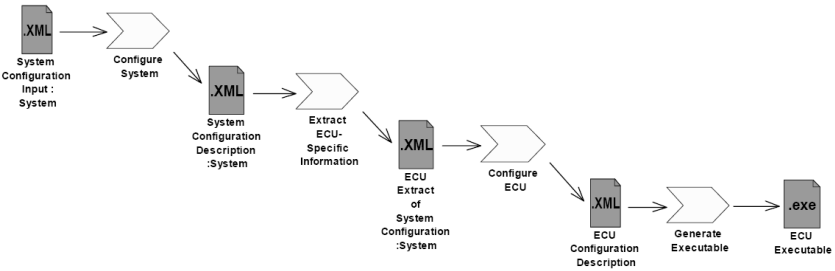

Abbildung 2.15: AUTOSAR Top-Down Entwurf (Bild: [AUT23a, S. 12])

Im ECU-Konfigurationsschritt werden die ECU-Extrakte mit zusätzlichen Informationen angereichert, welche für die Implementierung der Steuergerätesoftware notwendig sind. So z. B. das Scheduling der einzelnen Komponenten oder die Auswahl von Funktionen der Basissoftware. Diese Konfiguration wird als „ECU Configuration Description“ gespeichert. Im letzten Schritt wird aus der vervollständigten ECU-Beschreibung die ausführbare Software für die Steuergeräte generiert.

Der Entwicklungsprozess kann auch in die in Abschnitt 2.1.3 vorgestellten Unternehmensklassen von OEM bis Tier 3 aufgeteilt werden (s. Abb. 2.16). Die Kombination aus den Artefakten „Logical System Design“ u. „Physical System Design“ entspricht in Abb. 2.15 dem „System Configuration Input“, „SWC ECU Model“ und „COM ECU Model“ entsprechen dem „ECU Extract of System Configuration“ in Abb. 2.15. Die Entwicklung von Basissoftware[1], Hardware, Compiler etc., d. h. die Tier 2 u. Tier 3 Ebenen in Abb. 2.16, sind nicht Teil der AUTOSAR Methodik.

[1] AUTOSAR beschreibt die Anforderungen an eine Basissoftware und wie diese konfiguriert werden kann. Wie die Entwicklung der Basissoftware erfolgen soll, ist nicht beschrieben.

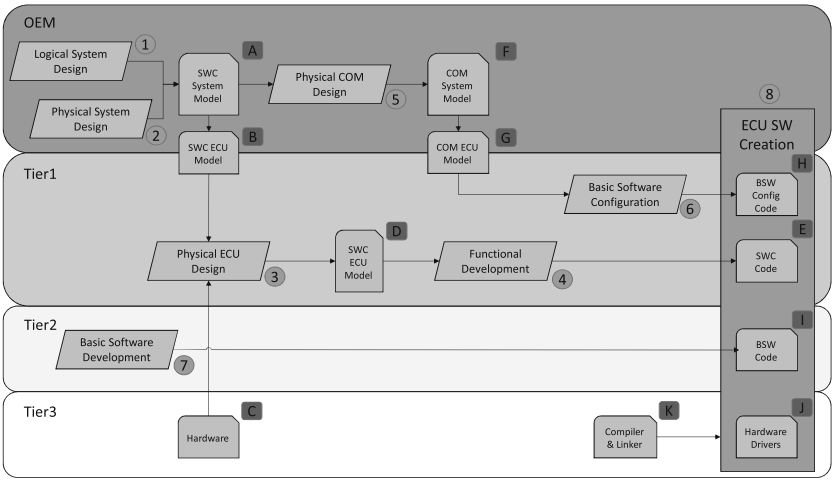

Abbildung 2.16: Arbeitsschritte und Rollen bei der Entwicklung eines Steuergerätes (Bild: [Dar21, S. 103])

Der OEM führt die Systemkonfiguration durch, wobei die Zuordnung von SWCs zu ECUs („SWC System Model“) entsteht. Aus der Aufteilung von Subsystemen auf mehrere Steuergeräte ergibt sich außerdem der Bedarf nach Buskommunikation, welche im „COM System Model“ geplant wird. In Form eines ECU-Extracts reicht der OEM Beschreibungen der Software Komponenten eines Steuergeräts („SWC ECU Model“) und eine Beschreibung sämtlicher Netzwerkkommunikation („COM ECU Model“) an den mit der Entwicklung eines Steuergeräts beauftragten Tier 1 weiter[1]. Der Fokus des OEMs liegt dabei auf der Entwicklung und Modellierung des Gesamtsystems, und der Kommunikation zwischen allen Steuergeräten.

Ein Tier 1 ist verantwortlich ein bestimmtes Steuergerät zu entwickeln. Dabei werden die vom OEM erstellten Modelle, die von einem Tier 2 entwickelte Basissoftware sowie die von Tier 3s hergestellte Hardware verwendet. Dies

[1] Über diese Artefakte hinaus müssen, unabhängig von der AUTOSAR Methodik, Unterlagen wie z. B. ein Lastenheft oder Funktionsmodelle für die Software vom OEM an den Tier 1 übergeben werden.

entspricht dem ECU-Konfigurationsschritt aus Abb. 2.15. Da all diese Komponenten beim Tier 1 zu einem Steuergerät integriert werden, bezeichnet man die Tier 1 auch als Integratoren.

Der Tier 2 entwickelt die Basissoftware, welche auch als AUTOSAR Stack bezeichnet wird. Zu einem Stack gehört jeweils eine Toolchain, mit der die Basissoftware und die funktionale Anwendungssoftware integriert werden, sowie die vom OEM erstellten Software- und Kommunikationsmodelle importiert werden. Der „Generate Executable“ Schritt in Abb. 2.15 wird durch Codegeneratoren der Tier 2 durchgeführt. Dadurch, dass mehrere Steuergeräteprojekte dieselbe Basissoftware verwenden, reduzieren sich die Kosten pro Steuergerät. Da verschiedene OEMs dieselbe Basissoftware verwenden können, teilen sich die Entwicklungskosten des AUTOSAR Stacks unter mehreren OEMs auf.

Tier 3s entwickeln die Hardware, welche für Steuergeräte verwendet wird. Für die AUTOSAR Methodik von Bedeutung sind hierbei die Mikrocontroller. Diese können unterschiedliche Prozessorarchitekturen und Peripheriebausteine aufweisen. Entsprechend fällt es in die Tier 3 Verantwortung, geeignete Treibersoftware zur Verfügung zu stellen.

3 Stand der Wissenschaft und Technik auf dem Gebiet der Fahrzeugnetzwerkentwicklung

3.1 Computernetzwerke und das ISO / OSI-Referenzmodell

In den 1970er-Jahren, mit dem Aufkommen der ersten Computernetzwerke, die die Vorläufer des heutigen Internets darstellen, begann die Standardisierung digitaler Kommunikation [Day83]. Es entstand das Open Systems Interconnection (OSI) Referenzmodell, welches in der ISO-7498 normiert wurde [Int89b][Int89a][Int94][Int97], sowie Standards mit darauf basierenden Kommunikationsprotokollen. Bis heute wird die Kommunikation zwischen verschiedenartigen Computersystemen gemäß dieses Standards beschreiben.

3.1.1 Begriffe

Definition 3.1. *Eine Service Data Unit (SDU) ist der Nutzdatenanteil, der von einer höheren Schicht zur Übertragung an eine niedrigere Schicht weitergegeben wird. [Int94, S. 15]*

Definition 3.2. *Bei der Protocol Control Information (PCI) handelt es sich um zusätzliche Informationen, wie z. B. eine Zieladresse, die zur Steuerung der Datenübertragung erforderlich sind.[Int94, S. 15]*

Definition 3.3. *Die Kombination aus Nutzdaten und Steuerinformationen stellt eine Protocol Data Unit (PDU) dar. Dies kann z. B. Die Kombination aus einer CAN Frame ID und den im Frame enthaltenen Daten sein.[Int94, S. 15]*

Diese Begriffe werden genau so auch in AUTOSAR verwendet [AUT23d, S. 106]. In der Praxis werden dabei nur die PDUs betrachtet, die aber eine Kombination aus SDU und PCI darstellen. Sowohl das OSI-Referenzmodell als auch AUTOSAR beschreiben PDUs in unterschiedlichen Schichten (vgl. insbesondere IPDU und Network Layer Protocol Data Unit (NPDU) in Abschnitt 2.7.5 sowie die weiteren PDU Typen in Abbildung 2.13). Mit Ausnahme dieses Kapitels, sind im Rahmen dieser Arbeit stets AUTOSAR IPDUs gemeint, wenn von PDUs die Rede ist.

3.1.2 Qualitative Eigenschaften einer Datenübertragung

Die Qualität einer Datenverbindung wird im OSI-Referenzmodell als „Quality Of Service“ (QOS) bezeichnet. Sie setzt sich aus den folgenden Parametern zusammen [Int94, S. 27]:

- Erwartete Verzögerung einer Übertragung
- Wahrscheinlichkeit der Verfälschung der übertragenen Daten
- Wahrscheinlichkeit von Verlust oder Duplikaten
- Wahrscheinlichkeit einer fehlerhaften Auslieferung
- Kosten
- Schutz vor unberechtigtem Zugriff
- Priorität
- Im Fall mehrerer Übertragungen:
  - Erwarteter Datendurchsatz
  - Wahrscheinlichkeit einer Auslieferung in falscher Reihenfolge

3.1.3 Einteilung eines Netzwerks in Schichten

Um ein Netzwerk aus verschiedenen Hard- und Software Komponenten sowie unterschiedlichen Übertragungstechniken handhabbar zu machen, beschreibt das OSI-Modell zunächst, wie ein Netzwerk in Schichten bzw. Ebenen zu unterteilen ist, und wie die Interaktion innerhalb und zwischen den Schichten beschrieben wird [Day83].

Eine Ebene baut dabei auf der darunter liegenden Ebene auf und stellt Funktionalität für die darüber liegende Schicht zur Verfügung. Die der übergeordneten Schicht zur Verfügung gestellte Funktionalität wird als Service bezeichnet. Ein Service für eine übergeordnete Schicht entsteht, indem der Service der untergeordneten Schicht um Funktionalität erweitert wird. Zu beachten ist, dass es sich dabei zwar um denselben Begriff, aber ein anderes Konzept wie bei der Service-orientierten Kommunikation in Abschnitt 2.4.7 handelt. Bereits in der ISO7498 wird darauf hingewiesen, dass unterschiedliche Bedeutungen für den Begriff existieren [Int94, S. 42]. Dabei ist die Funktionalität unabhängig von der technischen Umsetzung und insbesondere unabhängig von den darunter liegenden Schichten. So wird die Beschreibung und Entwicklung eines Kommunikationssystems in überschaubare Einheiten aufgeteilt. Vorteilhaft ist auch, dass einzelne Ebenen ausgetauscht werden können, ohne dass die darüber liegenden Ebenen verändert werden müssen, sofern die unterschiedlichen Implementierungen einer Schicht denselben Service anbieten.

Eine Schicht besteht beispielsweise aus einem oder mehreren Elementen, welche zusammenarbeiten, um den Service für die darüber liegende Schicht zur Verfügung zu stellen (s. Abb. 3.1). Die Verbindung der Elemente innerhalb einer Schicht N erfolgt dabei über schichtspezifische N-Protokolle. Die Elemente stellen Service Access Points (SAPs) an den Schichtgrenzen zur Verfügung und greifen auf solche SAPs zu, um Verbindungen zwischen den Schichten herzustellen. Ein Element kann jeweils einen oder mehrere SAPs zur Verfügung stellen (z. B. N-Element 1) und auf einen oder mehrere SAPs zugreifen

(z. B. N-Element 2). Es ist jedoch nicht möglich, einen SAP mit mehreren Elementen derselben Schicht zu verbinden. Stattdessen muss jedes Element seinen eigenen SAP verwenden. Ein Beispiel für ein System, das mehrere Elemente in einer Ebene beinhaltet, ist ein WLAN-Router: Dieser verbindet unterschiedliche Verbindungsmedien (Kabelgebundene Datenübertragung und Funkübertragung) indem die physikalische Schicht mehrere Elemente umfasst. Ein automobiles Beispiel wäre eine Gateway (vgl. Abschnitt 2.3.3), welches unterschiedliche Busse verbindet.

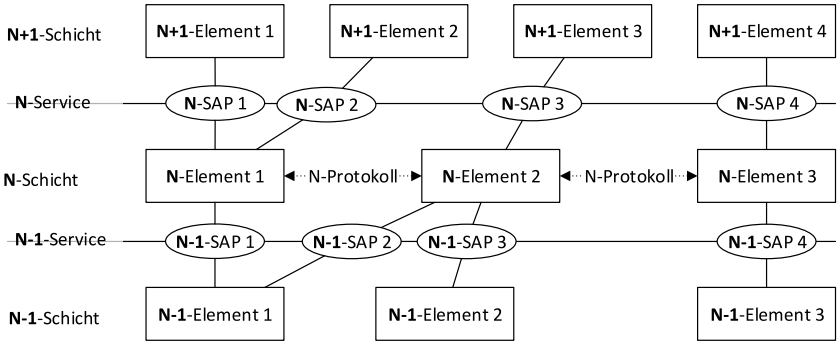

Abbildung 3.1: Struktur der OSI-Schichten [Day83]

3.1.4 Definition von Referenzschichten

Weiterhin definiert das OSI-Modell sieben Referenzschichten, mit denen eine über ein Computernetzwerk verteilte Anwendung beschrieben werden kann [Int94, S. 32-52][Day83].

1 Zuunterst befindet sich die Bitübertragungsschicht. Deren Aufgabe ist es, eine physikalische Verbindung (z. B. elektrisch, optisch, elektromagnetisch etc.) zur Verfügung zu stellen. Über diese Schicht können Datenbits übertragen werden, die Qualität des Kanals ermittelt und Verbindungsfehler erkannt werden. Je nach verwendetem Übertragungsmedium ergeben sich unterschiedliche Funktionalitäten,

die die Bitübertragungsschicht zur Verfügung stellt. Die Schicht endet in Service Access Points zur Datenverbindungsschicht.

2 Die Sicherungsschicht erweitert die Bitübertragungsschicht um Mechanismen wie Kanalcodierung zur Fehlerkorrektur und erweiterte Fehlererkennung. Einzelne Bits werden zu Sequenzen zusammengefasst. Ebenfalls in die Sicherungsschicht fällt die Steuerung und Synchronisierung des Sendens und Empfangen dieser Sequenzen. Gegenüber der Vermittlungsschicht werden Datenverbindung und zugehörige Adressen sowie Qualitäts- und Fehlerinformationen zur Verfügung gestellt.

3 Die Vermittlungsschicht verbindet mehrere einzelne Systeme zu einem Netzwerk. In dieser Ebene erfolgt die Weiterleitung von Daten, die eine Kommunikation zwischen nicht unmittelbar verbundenen Systemen ermöglicht. Die Wahl geeigneter Routen (und ggf. Ausweichrouten) erfolgt in dieser Schicht. Die übergeordnete Schicht benötigt lediglich eine Zieladresse, um eine Übertragung über mehrere Systeme hinweg zu starten. Es werden Informationen bezüglich der Qualität der gesamten Route an die Transportschicht weitergereicht.

4 Die Transportschicht dient zur Verbindung der endgültigen Kommunikationspartner (d. h. sämtliche Weiterleitungen, Medien etc. sind nicht sichtbar). Sie gewährleistet unter Berücksichtigung der von den unteren Ebenen gemeldeten Parametern, dass eine Übertragung in einer definierten Qualität stattfindet. Mechanismen zur Segmentierung und Desegmentierung in dieser Ebene ermöglichen die Übertragung von Botschaften, die größer sind als die maximalen Botschaftsgrößen der unteren Ebenen.

5 Die Sitzungsschicht verwaltet den Aufbau und die Synchronisation von Datenverbindungen. Eine Sitzung (englisch: „Session“) stellt einen gemeinsamen Kontext dar, innerhalb dessen die Darstellungsschichten zweier Systeme Daten austauschen.

6 Die Darstellungsschicht gewährleistet eine einheitliche Präsentation übertragener Daten. Dies kann z. B. durch standardisierte Austauschformate erfolgen.

7 In der obersten Schicht befinden sich die Anwendungen, welche auf dem Datenaustausch aufbauen. Hierbei handelt es sich um die Anwendungen, die ein Benutzer eines netzwerkbasierten Systems zu Gesicht bekommt. In der Automobilindustrie spricht man dabei von den „kundenerlebbaren Funktionen“ (im Gegensatz zu der zugrunde liegenden Basistechnologie, die für die Nutzer unsichtbar ist).

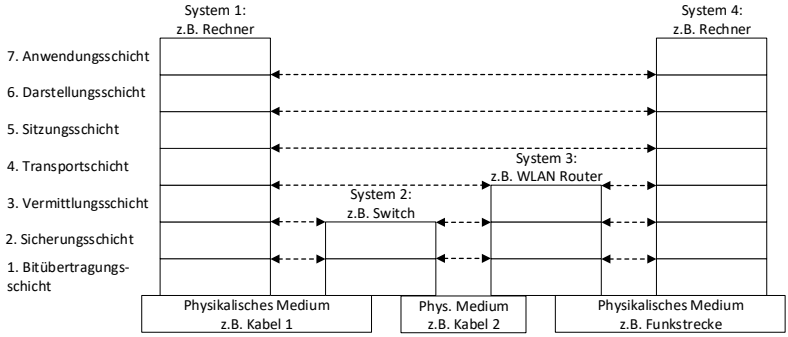

Abbildung 3.2: Verbindung zweier Rechner über unterschiedliche OSI-Schichten. Eigenes Beispiel, basierend auf [Int94, S. 29].

In der Praxis ist die Trennschärfe zwischen den unterschiedlichen Schichten nicht immer gegeben. Insbesondere die Bitübertragungsschicht und die Sicherungsschicht sind nicht eindeutig voneinander zu trennen [Pis93, S. 43].

Nicht jedes Teilsystem eines OSI-Netzwerks weist notwendigerweise alle sieben Schichten auf. Beispielsweise können zwei Rechner über Umwege miteinander verbunden sein (s. Abb. 3.2). Die Aufgabe von System 2 ist dabei lediglich Datenverbindungen zwischen unterschiedlichen Systemen herzustellen, ohne sich inhaltlich mit den Daten zu befassen, sodass lediglich Elemente aus den Schichten eins und zwei erforderlich sind. Bei System 3 handelt es

sich um einen Router, d. h. ein System mit Zugang zu unterschiedlichen Teilnetzwerken. Daher erfolgt hier die Weiterleitung auf der Vermittlungsebene (indem unter Auswertung der PCI die korrekten Start- und Zielteilnetze ermittelt werden).

3.1.5 TCP / IP-Modell bzw. Internetmodell

Parallel zum OSI-Referenzmodell und den zugehörigen Protokollstandards entwickelte sich, unter wechselseitiger Beeinflussung [Pis93, S. xiii], das TCP / IP-Modell und die zugehörigen Standards. Während das OSI-Referenzmodell heute im Wesentlichen zur theoretischen Beschreibung von vernetzten Computersystemen verwendet wird [Mei12, S. 40], bildet das TCP/IP-Modell und die zugehörigen Protokolle die Grundlage für das Internet und ist daher relevant für die Praxis.

Das TCP/IP-Modell basiert auf nur 4 [Hun02] oder 5 [Pis93] Schichten – je nachdem, ob die zugrunde liegende Physik als eigene Schicht gezählt wird [Mei12].

Hardware (Physical Layer)

Hierbei handelt es sich um die Hardware und die physikalischen Prinzipien, die eine Kommunikation ermöglichen. Diese Ebene wird teilweise nicht mitgezählt und ist keiner OSI-Schicht zugeordnet.

Netzzugangsschicht (Link Layer)

Da die Bitübertragung und -absicherung voneinander und von der verwendeten Technologie abhängen, sind sie im TCP/IP-Modell zur Netzzugangsschicht zusammengefasst. Beispiele für entsprechende Standards ist die kabelgebundene Datenübertragung über die Familie der IEEE802.3 Standards („Ethernet“) oder die kabellose Übertragung über die IEEE802.11 Standardfamilie („WLAN“). In der Automobilindustrie wird derzeit IEEE802.3bw [Ins16b]

(„100BaseT1 Ethernet“ oder „Automotive Ethernet“) mit einer Übertragungsrate von bis zu 100MBit/s und IEEE802.3bp [Ins16a] („1000BASE-T1“) mit 1GBit/s, jeweils über ein ungeschirmtes, verdrilltes Adernpaar, als Netzzugangsschicht für bandbreitenintensive Anwendungen verwendet.

Internetschicht (Internet Layer)

Die Internetschicht im TCP/IP-Modell entspricht der Vermittlungsschicht (Network Layer) im OSI-Modell. Auf dieser Ebene befindet sich das namensgebende Internetprotokoll (IP) welches in Version 4 (IPv4) in RFC791 [Int81a] oder Version 6 (IPv6) in RFC8200 [Int17] standardisiert ist. IPv4 ist für die Automobilindustrie von Bedeutung, da die SOME/IP Middleware darauf aufbaut [AUT23g]. Jeder Teilnehmer eines Netzwerks erhält eine eigene, eindeutige IP-Adresse, mit welcher Nachrichten gesendet und empfangen werden können. Die zwischengeschaltete Netzwerkknoten können anhand der IP-Adresse automatisch eine Weiterleitung durchführen.

Transportschicht (Transport Layer)

Die Transportschicht entspricht der vierten OSI-Ebene und wird auch als „Host-to-Host Layer“ (Rechner-zu-Rechner Schicht) bezeichnet, da in dieser Ebene die Verbindung zweier Rechner als gegeben betrachtet wird. Den Austausch zwischen diesen beiden Rechnern zu regeln, ist Aufgabe der Transportschicht. Dazu existiert das „Transmission Control Protocol“ (TCP), welches das Herstellen einer Verbindung, den Datenaustausch selbst, die Verifikation des korrekten Empfangs sowie das Schließen der Verbindung regelt. [Int81b]

Alternativ kann z. B. das „User Datagram Protocol“ (UDP) verwendet werden, um ohne den Aufbau einer Verbindung und eine abschließende Verifikation Daten zu übertragen. Damit können auf Basis von UDP anwendungsspezifische Protokolle implementiert werden. [Int80]

Anwendungsschicht (Application Layer)

Auf der Transportschicht (d. h. auf dem TCP- oder dem UDP-Protokoll) basieren die Netzwerkanwendungen in der sogenannten Anwendungsschicht. Im TCP/IP-Modell entspricht die Anwendungsschicht den Schichten 5 – 7 im OSI-Modell (s. Abb. 3.3).

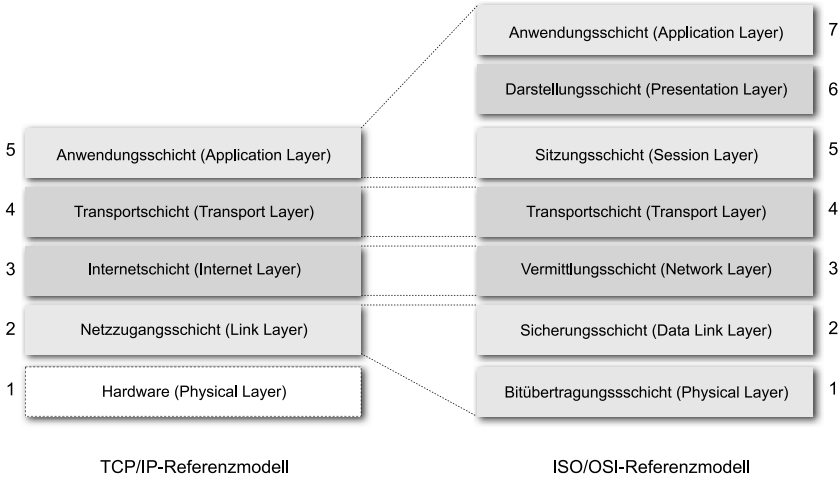

Abbildung 3.3: Vergleich zwischen dem TCP/IP-Modell und dem OSI-Modell (Bild: [Mei12, S. 50])

3.1.6 Einordnung von AUTOSAR in das TCP / IP-Modell und das ISO / OSI-Referenzmodell

Die Schichtenmodelle aus Abschnitt 3.1.4 und Abschnitt 3.1.5 können zur Einordnung der in Abschnitt 2.7 vorgestellten AUTOSAR Basissoftware verwendet werden.

AUTOSAR Classic

Das in Abbildung 2.13 skizzierte Metamodell für die Datenübertragung besteht aus einer Anwendungsschicht („Application Layer“) einer Interaktionsschicht („Interaction Layer“), einer Netzwerkschicht („Network Layer“) und einer Datenübertragungsschicht („Data Link Layer“). Insbesondere bei der Definition der Interaktionsschicht in AUTOSAR [AUT23c, S. 92] wird auf den OSEK Standard (ISO17356-4 [Int05a]) verwiesen. Dabei handelt es sich um einen der Vorläufer von AUTOSAR [Pel05, S. 299]. OSEK verortet die Interaktionsschicht zwischen Anwendungsschicht und Netzwerkschicht. Der Netzwerkschicht werden, sowohl in OSEK als auch in AUTOSAR, die Übertragung von Nachrichten, die Segmentierung und Desegmentierung größerer Botschaften zugeordnet. Sie entspricht damit den OSI Schichten 3 – 4. Die darauf aufbauende Interaktionsschicht entspricht den OSI Schichten 5 – 6.

AUTOSAR Adaptive mit SOME/IP

Die Service-orientierte Kommunikation der AUTOSAR Adaptive Plattform basiert auf der SOME/IP Middleware, welche innerhalb eines IP-basierten Fahrzeugnetzes auf TCP bzw. UDP aufsetzt [AUT23g, S. 67]. Entsprechend erfolgt die Einordnung innerhalb des TCP/IP-Modells in der Anwendungsschicht. Da die Anwendungsschicht des TCP/IP-Modells im OSI Modell in die Sitzungs-, Darstellungs- und Anwendungsschicht unterteilt ist, ist hier eine weitere Unterteilung möglich. Die Spezifikation des SOME/IP Protokolls [AUT23g] beschreibt wie Daten innerhalb von Services in Methoden, Felder und Events gegliedert werden, wie die Daten übertragenen Daten formatiert werden (z. B. dass das IEEE754 Format für Fließkommazahlen verwendet wird [AUT23g, S. 25]), und wie die einzelnen Elemente innerhalb eines Datenpakets angeordnet werden. Kundenerlebbare Funktionen werden nicht spezifiziert. Damit fällt SOME/IP in die Darstellungsschicht des OSI Modells.

Eine Gegenüberstellung der verschiedenen Schichtenmodelle sowie die Einordnung von AUTOSAR Classic und SOME/IP ist in Tabelle 3.2 gegeben.

Tabelle 3.2: Einordnung von AUTOSAR Classic und SOME/IP in die OSI- und TCP/IP-Modelle

OSI-Modell	TCP/IP-Modell	Autosar Classic	SOME/IP
7. Anwendungsschicht (Application Layer)	5. Anwendungsschicht (Application Layer)	**Anwendungen**	**Anwendungen**
6. Darstellungsschicht (Presentation Layer)		**Interaktionsschicht (Interaction Layer)**	**SOME/IP Middleware**
5. Sitzungsschicht (Session Layer)			
4. Transportschicht (Transport Layer)	4. Transportschicht (Transport Layer) z. B. TCP o. UDP	**Netzwerkschicht (Network Layer)**	**TCP oder UDP**
3. Vermittlungsschicht (Network Layer)	3. Internetschicht (Internet Layer) z. B. IPv4 o. IPv6		**IPv4**
2. Sicherungsschicht (Data Link Layer)	2. Netzzungangsschicht (Link Layer) z. B. Ethernet oder WLAN	**Netzzungangsschicht (Data Link Layer) z. B. CAN, LIN, FlexRay**	**IEEE802.3bw oder IEEE802.3bp**
1. Bitübertragungs-schicht (Physical Layer)			
nicht definiert	1. Hardware (Physical Layer)	je nach Netz-zungangsschicht	ein einzelnes, unge-schirmtes, verdrilltes Adernpaar

3.2 Systems Engineering

3.2.1 System

Ein Automobil besteht aus verschiedenen mechanischen, elektrischen, elektronischen und Softwarekomponenten, die zusammenarbeiten, um einen Zweck zu erfüllen. Dabei ist die Funktion, die insgesamt erbracht wird, mehr als die Summe der Einzelfunktionen (d. h. z. B. die Kombination aus Antrieb, Lenkung und Bremse ermöglicht kontrollierbare Fortbewegung, eine Funktion, die keine der Einzelkomponenten erfüllt). Das entspricht der Systemdefinition aus [Bor19, S. 3]:

Definition 3.4 (System)**.** *Ein System ist eine Menge interagierender Elemente, die zusammenarbeiten, um einen Zweck zu erfüllen. Und es ist generell der Fall, dass die von diesem Ensemble gelieferte Funktionalität mehr ist, als die Summe der Funktionalitäten der einzelnen Teile oder Subsysteme. [Bor19, S. 3]*

Vergleichbare Definitionen finden sich u. a. in [Eis08], [Int24b], [Kos20] und [SAE10].

Diese Definition erfüllen aber auch viele der Komponenten eines Fahrzeugs, wie z. B. der Antrieb (bestehend aus Tank, Kraftstoffpumpe, Turbolader, Motor etc.) oder die Musikanlage (bestehend aus Bildschirm, Bedieneinheit, Verstärker, Lautsprecher, Audioquellen etc.). Insbesondere kann die E/E-Architektur (vgl. Abschnitt 2.4) als System gesehen werden, oder eine Funktionalität, die über mehrere Steuergeräte verteilt ist [Sax08, S. 2][Sta21, S. 8].

Eine mögliche Perspektive ist, das Fahrzeug als Gesamtsystem anzusehen und die E/E-Architektur als eines dessen (Sub-)Systeme. Im AUTOSAR Standard wird die gesamte E/E-Architektur als System bezeichnet, während die verschiedenen, von teils mehreren Steuergeräten erbrachten, Funktionalitäten Subsysteme genannt werden [AUT23e, S. 39f]. Im allgemeinen

Sprachgebrauch[1] wird für diese Subsysteme vereinfacht „Systeme“ verwendet [Mar16][Sta21, S. 8], während die gesamte E/E-Architektur zur Unterscheidung als „Gesamtsystem“ betitelt wird. Deshalb wird in diesem Werk, wo nicht anders angegeben, der in Abschnitt 2.3.4 vorgestellte Systembegriff benutzt.

3.2.2 Verifikation, Validierung und Co

In Entwicklungsprozessen, insbesondere bei formellen Vorgehensmodellen, ist die Überprüfung der diversen (Zwischen-)Ergebnisse von zentraler Bedeutung. Immer wieder verwendet werden dabei die folgenden beiden Begriffe:

Definition 3.5 (Verifikation)**.** *Die Überprüfung, ob ein System die spezifizierten Funktionen und Eigenschaften korrekt und genau implementiert. [Kos20, S. 432]*

Entsprechend kann die Verifikation erfolgen, indem man die Produkte mit der Spezifikation vergleicht. Dabei können Implementierungsfehler gefunden werden.

Definition 3.6 (Validierung)**.** *Die Überprüfung, ob ein System die Bedürfnisse der Anwender bzw. Kunden befriedigt. [Kos20, S. 433]*

Ein korrekt implementiertes (d. h. erfolgreich verifiziertes) Produkt erfüllt nicht notwendigerweise die Bedürfnisse der Kunden. Wurden bei der Anforderungsanalyse und dem Erstellen der Spezifikation Fehler gemacht, so werden diese bei der Validierung erkannt.

Drei weitere wichtige Begriffe in diesem Zusammenhang sind:

Definition 3.7 (Messen)**.** *„Messen“ ist das Erfassen von Werten (Messwerten) zu einem definierten Zeitpunkt oder Zeitraum. [Sax08, S. 6]*

[1] Im Bereich der E/E-Entwicklung.

Definition 3.8 (Prüfen). *„Prüfen“ ist das Vergleichen von Messwerten mit erwarteten Werten. [Sax08, S. 6]*

Die Implementierung eines (Sub-)systems wird verifiziert, indem das Einhalten von Parametern der Spezifikation geprüft wird.

Definition 3.9. *In „Erproben“ steckt das „Ausprobieren“ in einer unbestimmten Situation. Im Gegensatz zu „Prüfen“ basiert „Erproben“ eher auf der Analyse des Gesamten als dem Auswerten spezieller Messwerte. [Sax08, S. 6]*

Das Erproben eines (Gesamt-)systems ist in der Lage einen subjektiven Eindruck (z. B. „Fahrgefühl“) zu vermitteln. Dies ist insbesondere zur Validierung sinnvoll.

3.2.3 Komplexität und deren Bewältigung

Das Zusammenspiel der Komponenten innerhalb der Systeme, für sich betrachtet, sowie das Zusammenspiel der zahlreichen, verschiedenartigen Systeme innerhalb des Gesamtsystems „Fahrzeug“, umfasst so viele Faktoren (vgl. z. B. die 45.000 Signale in Abbildung 1.1 oder die Anzahl von bis zu 150 ECUs in einem Fahrzeug [Bur20][Mot23]), dass es sich ohne methodische Analyse dem Verständnis entzieht. Dies ist eine Eigenschaft, die in [SAE10, S. 10] als Komplexität definiert wird:

Definition 3.10 (Komplexität). *Eine Eigenschaft von Funktionen, Systemen oder Gegenständen, welche deren Betrieb, deren Fehlermodi oder Fehlereffekte, ohne Hilfe analytischer Methoden, schwer verständlich macht [SAE10, S. 10].*

Auch in [Bor19] und [Kos20] werden ähnliche Definitionen verwendet, wobei die letztere Quelle moderne Automobile explizit als Beispiel für komplexe Systeme anführt [Kos20, S. 19]. Weiterhin werden Automobile und deren E/E-Architektur explizit in [Sax08], [Rei11], [Zim14], [Sch16b], [Sta21] und [Pis21] sowie in zahllosen weiteren Quellen als komplex bezeichnet und diese Komplexität als Herausforderung angesehen.

Die etablierte Methodik, zur Bewältigung ebendieser Herausforderung, ist das sogenannte „Systems Engineering“ [Bor19][Coo00][Eis08][Kos20]. Der International Council on Systems Engineering (INCOSE) definiert Systems Engineering folgendermaßen:

Definition 3.11 (Systems Engineering)**.** *Systems Engineering ist eine interdisziplinäre und integrative Herangehensweise, um die erfolgreiche Entwicklung, Nutzung, und Außerbetriebnahme von konstruierten Systemen zu ermöglichen. Dabei werden Systemprinzipien und -konzepte, und wissenschaftliche, technologische und Managementmethoden angewendet. [INC21]*

Dabei wird der gesamte Lebenszyklus eines Systems, d. h. die Spanne vom ersten Konzept bis zur finalen Außerbetriebnahme, betrachtet und in unterschiedliche Phasen eingeteilt. Nach [Bor19, S. 4f] sind das z. B.:

- Konzeptentwicklung und -analyse
- Erfassen, Analysieren, Zuordnen und Nachverfolgen von Anforderungen
- Grobe und detaillierte Ausarbeitung
- Integration und Test
- Verifikation und Validierung (vgl. Abschnitt 3.2.2)
- Übergang zu Betrieb und Wartung

Über die reine Produktentwicklung hinaus führt z. B. die ISO/IEC/IEEE 24748-1 [Int24b] Betrieb, Wartung und Außerbetriebnahme als eigene Phasen. Zwischen der Produktion des ersten und letzten Fahrzeugs einer Serie liegen aktuell ca. 6 Jahre [Cen17]. Hinzu kommt, dass zum 1.1.2024 das Durchschnittsalter der in Deutschland angemeldeten Pkw laut Kraftfahrtbundesamt 10,3 Jahre betrug [Abt24]. Innerhalb des Produktionszeitraumes müssen durch Aktualisierungen einzelner Komponenten die Attraktivität erhalten und Kaufanreize geschaffen werden. Auch nach dem Ende einer Serienproduktion müssen die Fahrzeuge gewartet und mit Ersatzteilen versorgt

werden. Entsprechend sind für Automobilhersteller auch die späteren Phasen des Lebenszyklus von Bedeutung und müssen im Entwicklungsprozess berücksichtigt werden.

Die verschiedenen Phasen des Lebenszyklus werden in sogenannten Vorgehensmodellen angeordnet. Verbreitet sind z. B. das lineare Durchlaufen der Phasen, das Wasserfallmodell, in dem einfache Rücksprünge möglich sind, das Spiralmodell, welches durch ein Wiederkehren der Phasen einen iterativen Prozess abbildet, oder das V-Modell (s. Abs. 3.2.4), welches die Systementwicklung auf unterschiedlichen Abstraktionsniveaus betrachtet und auf jedem Niveau eigene Feedbackmechanismen aufweist [Kos20, S. 16].

Aufgrund der Abhängigkeiten zwischen Mechanik, Elektrik/Elektronik und Software (vgl. z. B. die in Abschnitt 2.1.1 beschriebene Problematik mit dem Verbauraum für Steuergeräte) erfolgt die Entwicklung des Gesamtsystems Automobil gemäß dem V-Modell [Kat15, S. 615][Tri15].

3.2.4 V-Modell

Das V-Modell nach VDI/VDE2206 [Ver21] (s. Abb. 3.4) unterteilt einen Entwicklungsprozess grob in drei Stränge (Modellierung und Analyse in Blau, Kernaufgaben der Systementwicklung in Orange, und Anforderungsentwicklung in Gelb) und sechs Meilensteine (1. Geschäftsmodell, 2. Spezifikation, 3. Architektur, 4. Implementierung, 5. Integration und 6. Übergabe). Die V-Form ergibt sich aus den übergeordneten drei Bereichen:

- Den linken Schenkel, in dem die Anforderungen von einer hohen, abstrakten Ebene immer weiter verfeinert und aufgegliedert werden. Hier stehen ganz oben die kundenerlebbaren Funktionen als Anforderungen und ganz unten die vollständigen Spezifikationen aller Komponenten, die zum Erfüllen der Anforderungen beitragen.
- In der Mitte befindet sich der Bereich, in dem die einzelnen Komponenten anhand der Spezifikationen implementiert werden.

- Im rechten Schenkel werden von unten nach oben die einzelnen Komponenten zu Subsystemen und schließlich zum Gesamtsystem integriert.

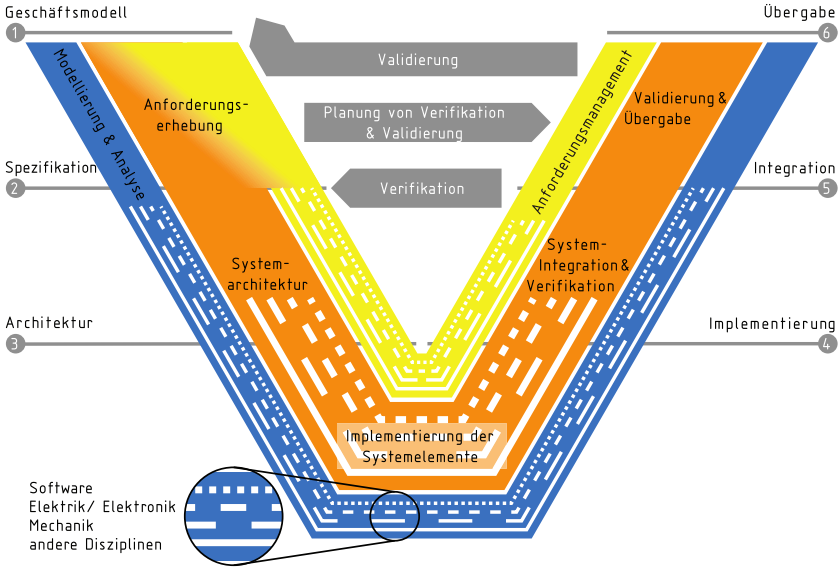

Abbildung 3.4: Das V-Modell nach VDI/VDE2206 (Bild: [Ver21, S. 22])

Dabei ist die fortschreitende Zeit von links nach rechts und die zunehmende Abstraktionshöhe von unten nach oben dargestellt. Zu jedem der drei Meilensteine auf der linken Seite gehören eindeutig überprüfbare Kriterien, anhand derer, bei Erreichen des zugehörigen Meilensteins auf der linken Seite, der Projektfortschritt überprüft werden kann (vgl. Abschnitt 3.2.2).

Außer dem Standard VDI/VDE2206 gibt es noch das V-Modell 97 [Drö00], das V-Modell XT [Ver24] sowie diverse allgemeine Beschreibungen des Konzeptes [Bor19, S. 69][SAE10, S. 24]. Das V-Modell XT erweitert das klassische V-Modell um einen Balken nach rechts, mit dem die Abschnitte des Produktlebenszyklus nach Abschluss der Entwicklung abgedeckt werden.

In der Forschung gibt es verschiedene Vorschläge zur Verbesserung des V-Modells [Deu13][Liu16].

Studien in den Jahren 2006 und 2013 ergaben, dass die verschiedenen V-Modelle zusammen, eines der am häufigsten verwendeten Vorgehensmodelle ist [Eng14]. Insbesondere für reine Softwaresysteme sind aber auch weniger formelle, agile Entwicklungsmethoden beliebt, die schneller zu validierbaren Prototypen führen. In [Ebe20] wurde gezeigt, dass agile Methoden grundsätzlich auch im automotive Bereich eingesetzt werden können. Zu beachten ist, dass innerhalb eines nach dem V-Modell strukturierten Gesamtprojekts, Teilprojekte (z. B. die Implementierung eines spezifizierten Subsystems) nach anderen Vorgehensmodellen (z. B. agil) umgesetzt werden können.

Die Elektrik/Elektronik Architektur eines Automobils kann als eine Komponente des Gesamtsystems betrachtet werden (parallel zu Komponenten wie Karosserie, Fahrwerk, und Antriebsstrang[1]). Mit den Anforderungen an diese Komponente kann die E/E-Architektur entwickelt und später verifiziert werden. Entsprechend ist die E/E-Entwicklung zwischen der (Gesamt-) Systemspezifikation und der Integration anzusiedeln. Die Implementierung spezifizierter Komponenten, angesiedelt im unteren Teil des „V“, erfolgt in der Regel durch Zulieferer (vgl. Abschnitt 2.1.3). In der E/E-Entwicklung sind das die Steuergeräte, welche durch den OEM spezifiziert und dann durch einen Tier1 geliefert werden. Anschließend werden die von unterschiedlichen Zulieferern bereitgestellten Steuergeräte beim OEM integriert (rechter Schenkel) und das resultierende System verifiziert und validiert[Zim14, S. 415ff]. Die Darstellung der AUTOSAR Entwicklungsmethodik in Abschnitt 2.7.6 und Abbildung 2.15 entspricht dem linken Schenkel des V-Modells.

[1] Letztlich bestehen zwischen all diesen Subsystemen Abhängigkeiten, weshalb das methodische Vorgehen und einhalten der Spezifikationen wichtig ist: Nur wenn die, im linken Schenkel des V-Modells definierten, Schnittstellen vorhanden sind und sich korrekt verhalten, kann im rechten Schenkel die Integration gelingen und ein funktionales Gesamtsystem entstehen.

3.2.5 Verifikation und Validierung in der Praxis

Unten in der rechten Hälfte des V-Modells werden die einzelnen Softwarekomponenten verifiziert. Die nächsthöhere Stufe ist das Testen einer kompletten ECU (Hardware mit Software). Hierzu werden die Sensor- und Aktor- und Busanschlüsse des Steuergeräts mit einem Prüfstand verbunden, der die reale Umgebung des Steuergeräts nachbildet. Da die simulierten Sensorsignale teilweise von der gemessenen Ansteuerung der Aktoren abhängt, schließt der Prüfstand mit dem Steuergerät einen Regelkreis (engl.: „control loop“). Man bezeichnet dieses Verfahren daher als Hardware in the Loop (HIL oder ECU-HIL). [Sax08][Hak15]

Auf mittlerer Höhe im linken Schenkel des V-Modells (vgl. Abb. 3.4) stehen „Systemintegration & -Verifikation“. Um ein einzelnes System zu verifizieren, ohne ein ganzes Fahrzeug aufzubauen, kann ein erweiterter HIL Aufbau verwendet werden. Dabei werden die am System beteiligten Steuergeräte über ihre jeweiligen Busse miteinander und mit dem Prüfstand verbunden. Die Sensor- und Aktoranschlüsse werden ebenfalls mit dem Prüfstand verbunden, sodass dieser sowohl die physikalische Umgebung des Systems, als auch den Rest des Fahrzeugnetzwerks[1] simulieren kann. Einen solchen Aufbau bezeichnet man als System-HIL [Hak15] oder Cluster-HIL [Ott18].

[1] Auch als „Restbussimulation“ bezeichnet, da der Rest des Fahrzeugnetzwerks, z. B. allgemein relevante Informationen wie Zündstatus oder Geschwindigkeit, simuliert wird.

Abbildung 3.5: Erprobungsträger mit der typischen Tarnung des Fahrzeugdesigns durch aufgeklebte Folien, Kunststoffverschalung, abgeklebte Scheinwerfer sowie dem Fehlen aller Markenzeichen (Bild: [Mer20a])

Die höchste Integrationsstufe sind Prototypenfahrzeuge, auch Erprobungsträger genannt, die sämtliche Steuergeräte beinhalten (s. Abb. 3.5). Diese Ebene dient der Validierung des Gesamtsystems. Mit den Prototypenfahrzeugen werden Testfahrten in extremen Klimazonen und auf speziellen Teststrecken (z. B. Schotterpiste), aber auch auf regulären öffentlichen Straßen durchgeführt [Ung09]. So werden einerseits die Langlebigkeit überprüft und mögliche Schwachstellen ermittelt, andererseits wird so das Zusammenspiel der gesamten E/E-Architektur mit realen Komponenten getestet [Wei13]. Eine Herausforderung dabei ist es, dass zum Aufbau eines Erprobungsträgers ein vollständiges Set an ECU Prototypen vorhanden sein muss. Verzögerungen bei einzelnen Steuergeräteprojekten können den Aufbau von Erprobungsträgern blockieren.

3.2.6 Modellbasierte Entwicklung

Ein weiterer Baustein zur Bewältigung von Komplexität ist die sogenannte modellbasierte Entwicklung. Dabei wird die Spezifikation nicht in Form eines

beschreibenden Textdokuments erstellt, sondern als ein formelles Modell angelegt. So werden Mehrdeutigkeiten vermieden, die sich in informellen Textdokumenten ergeben können. Dies kann z. B. in Form von SySML/UML oder Matlab Simulink geschehen, wobei sich je nach Modellierungswerkzeug unterschiedliche Möglichkeiten in der weiteren Verwendung des Modells ergeben [Zim14, S. 432f][Sta21, S. 155].

So bietet ein SySML/UML Modell die Möglichkeit ein System zu beschreiben und Systemelemente mit Anforderungen zu verknüpfen. Abhängig von den verwendeten Diagrammen und dem Umfang der Modellierung, ist die Generierung einer leeren Codestruktur, die Generierung einzelner ausführbarer Funktionen, bis hin zur vollständigen Modellierung eines Gesamtsystems möglich. Matlab Simulink ist weniger auf die Beschreibung eines Gesamtsystems ausgelegt, als auf das Modellieren mathematischer und physikalischer Zusammenhänge. Aus einem solchen Modell kann ausführbarer Code generiert werden [Sta21, S. 155]. Je nach Modellierung ist es auch möglich, das Modell zur Verifikation einer fertiggestellten Software zu verwenden [Sch05b, S. 141].

Modellbasierte Entwicklung ermöglicht auch, eine Funktion unabhängig von einer ausführenden Hardware zu entwickeln. So wird vermieden, die gleiche Anwendung bei einem Wechsel der Hardware oder des Zulieferers neu zu entwickeln. Ausführbare Modelle ermöglichen es durch Simulation Funktionen zu validieren, bevor entsprechende Hardware bereitsteht. Mittels Rapid-Prototyping-Plattformen können Funktionsmodelle sogar in Fahrzeugen erprobt werden [Str12, S. 69ff].

Auch die Entwicklung von Fahrzeugnetzen mit AUTOSAR ist modellbasierte Entwicklung. Als Modellierungssprache dient dabei das AUTOSAR Metamodel (vgl. Abschnitt 2.7.5). Das resultierende Systemmodell kann zu verschiedenen Zwecken verwendet werden:

- Das Systemmodell wird nach Hardwarekomponenten aufgeteilt exportiert (in Form der ECU-Extracts, vgl. Abschnitt 2.7.6) und zur automatisierten Erstellung der Basissoftware für die Steuergeräte verwendet [Heb09].
- Die Modelle einzelner Teilnetzwerke können exportiert werden, um auf damit Hard- oder Software in the Loop (HIL/Software in the Loop (SIL)) Tests zu konfigurieren [Zim14, S. 427f].
- Mit den „AUTOSAR Acceptance Tests“ kann anhand des Modells verifiziert werden, ob ein vorliegendes Steuergerät gemäß dem AUTOSAR Standard kommuniziert [Rüp15].
- Auf Basis des Systemmodells können Teilaufgaben der Netzwerkentwicklung (z. B. Konsistenzprüfung von unterschiedlichen Signalzykluszeiten) automatisiert werden [Sch16a].

3.3 Schichtenmodell der E/E-Architektur

So wie sich die Kommunikation (vgl. Abschnitt 3.1) und die Basissoftware bzw. der Aufbau einzelner Steuergeräte (vgl. Abschnitt 2.7.2) in Schichten unterteilen lässt, kann auch die gesamte E/E-Architektur aus der Perspektive eines Schichtenmodells betrachtet werden. In [Str12, S. 15ff] werden die folgenden vier Schichten (s. Abb. 3.6) beschrieben.

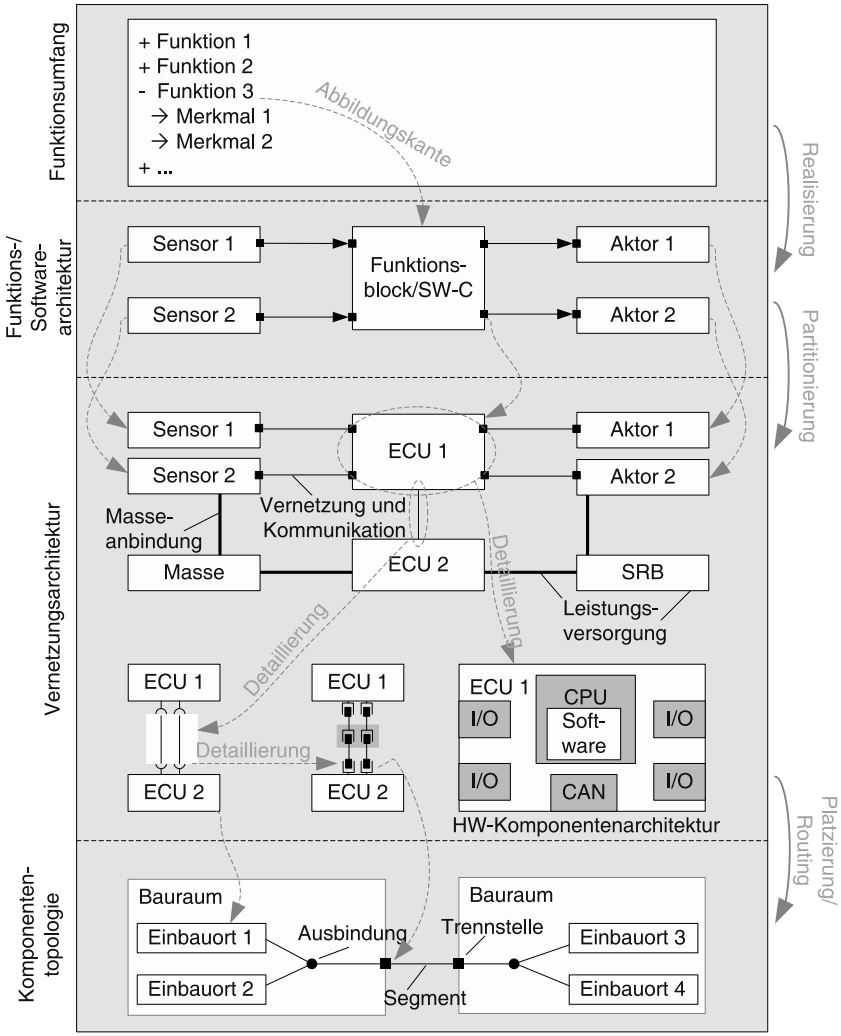

Abbildung 3.6: Die vier Systemebenen einer E/E-Architektur (Bild: [Str12, S. 16])

3.3.1 Funkionsumfang

Der Funktionsumfang ist eine Liste mit Funktionen, die durch die E/E-Architektur bereitgestellt werden. Diese können in Unterfunktionen aufgeteilt werden und Merkmale zugewiesen bekommen. Das Erstellen des Funktionsumfangs entspricht der Anforderungserhebung im linken, oberen Bereich des V-Modells in Abbildung 3.4. Im rechten oberen Bereich wird im Rahmen der Validierung überprüft, ob all diese Funktionen zweckgemäß umgesetzt sind.

3.3.2 Funktions- / Softwarearchitektur

Der Funktionsumfang wird in einzelne, Technologie-unabhängige Funktionsblöcke (Sensor, Aktor oder Softwarefunktion) aufgeteilt. Diese Funktionsblöcke entsprechen in AUTOSAR den Atomic SWCs und SWC Compositions (vgl. Abschnitt 2.7.2). Das Zuordnen von Systemfunktionen zu SWCs entspricht innerhalb der AUTOSAR Entwicklungsmethodik (vgl. Abschnitt 2.7.6) dem Systemkonfigurationsschritt. Im V-Modell befindet sich der Schritt zwischen Spezifikation und Architektur.

3.3.3 Vernetzungsarchitektur

Die nächste Ebene ist die Vernetzungsarchitektur. Hier werden Steuergeräte einschließlich ihrer Hardware-Umfänge (anzuschließende Sensoren und Aktoren) definiert. Weiterhin werden die Verbindungen unter den Steuergeräten, sowie deren Energieversorgung beschrieben. Die Technologie-unabhängigen Funktionsblöcke der Funktions- / Softwarearchitektur werden nun auf konkrete Steuergeräte, Sensoren und Aktoren abgebildet. Dieser Schritt entspricht dem Extrahieren der ECU spezifischen Informationen aus der Systemkonfiguration in AUTOSAR.

Die Vernetzungsarchitektur kann weiter detailliert werden, indem der Aufbau der Steuergeräte (Auswahl des Mikrocontrollers, der Transceiver-Bausteine,

evtl. vorhandene Hardwareinterfaces zur Sensorauswertung oder Aktoransteuerung, sowie die verwendete Software; vgl. Abschnitt 2.3.1) verfeinert wird. In [Str12, S. 17] wird dies als „Hardware-Komponentenarchitektur“ bezeichnet. Aus der AUTOSAR Perspektive ist hier auch der Begriff „Steuergerätearchitektur“ anwendbar. Von den im Steuergerät verbauten Komponenten hängt unmittelbar die Konfiguration des „AUTOSAR Microcontroller Abstraction Layers“ sowie des „AUTOSAR ECU Abstraction Layers“ ab (vgl. Abschnitt 2.7.2). In der AUTOSAR Methodik ist dieser Schritt die ECU-Konfiguration (Die vollständige ECU-Konfigurationsbeschreibung im ARXML-Format wird als Eingabe für den AUTOSAR-Codegenerator eines Stackherstellers verwendet).

3.3.4 Komponententopologie

Die unterste Schicht bildet die Komponententopologie. Hier werden die Steuergeräte, unter Berücksichtigung ihrer geometrischen Maße, geeigneten Verbauräumen innerhalb des Fahrzeugs zugeordnet. Aufbauend auf der räumlichen Position der ECUs im Fahrzeug sowie den Verbindungen der Vernetzungsarchitektur, wird außerdem der Kabelbaum einschließlich aller Abzweige, Verbindungsstellen und Stecker geplant.

3.4 Kompatibilität

Im gängigen Sprachgebrauch bezeichnet man zwei Dinge als kompatibel, wenn diese „zusammenpassen“. Im SAE EIA-IS-632 Standard für Systems Engineering wird Kompatibilität folgendermaßen definiert:

Definition 3.12 (Kompatibilität nach SAE EIA-IS-632)**.** *Die Fähigkeit von zwei oder mehr Elementen, Komponenten von Geräten oder Materialien, ohne gegenseitige Störung in demselben System oder derselben Umgebung zu existieren oder zu funktionieren. [SAE16, S. 37]*

Der reine Ausschluss von gegenseitiger Störung ist für diese Arbeit jedoch zu allgemein. Eine notwendige, zusätzliche Eigenschaft ist, dass die Komponenten (im Fall dieser Arbeit: Steuergeräte) innerhalb des Gesamtsystems (im Fall dieser Arbeit: Das Fahrzeug, genauer gesagt dessen E/E-Architektur) erfolgreich zusammenarbeiten, um verteilte Systeme zu bilden.

Die Frage nach Kompatibilität stellt sich besonders an den Stellen, an denen verschiedene Teilsysteme zusammengeführt werden:

Definition 3.13 (Schnittstelle, allgemein)**.** *Schnittstellen (engl.: „Interfaces“) sind Orte, an denen irgend zwei Systemteile zusammen geschaltet werden können. Art und Ausführung sind zum Teil sehr verschieden; auch Umfang oder Bedeutung der Systemteile unterscheiden sich. Verschieden sind ebenfalls die Informationsdarstellung und die über die Schnittstellen geleiteten Signale. [Sch94]*

3.4.1 Kompatibilität auf unterschiedlichen Ebenen des Schichtenmodells

Beschreibt man ein Fahrzeugnetzwerk anhand des OSI-Referenzmodells oder des TCP/IP-Modells (vgl. Abschnitt 3.1), so kann die auch die Kompatibilität auf den unterschiedlichen Ebenen betrachtet werden:

1 Hardware: Steuergerät und Fahrzeug sind kompatibel, wenn die Stecker von Kabelbaum und Steuergerät ineinander gesteckt werden können.

2 Netzzugangsschicht: Steuergerät und Fahrzeug sind kompatibel, wenn dieselbe Bustechnologie (z. B. CAN in einer definierten Geschwindigkeit) verwendet wird.

3 Internet- und Transportschicht: Kompatibilität bedeutet, dass eine gemeinsame Middleware (z. B. AUTOSAR oder SOME/IP) vorliegt, welche Botschaften adressieren, übertragen und weiterleiten (mittels Gateways auch über Bus- und Technologiegrenzen hinweg, vgl. Internet) kann.

4 Anwendungsschicht: Steuergerät und Fahrzeug sind kompatibel, wenn die Kommunikation die Anforderungen der Softwarekomponenten (SWCs, vgl. Abschnitt 2.7.2) befriedigt.

Im Folgenden wird die Kompatibilität der Punkte 1 und 2 vorausgesetzt. Technik, welche die Übertragung und Weiterleitung von Botschaften zwischen Steuergeräten ermöglicht, ist in der Literatur hinreichend beschrieben [Rei11][Str12][Zim14][Sch16b][Sta21][Pis21]. Die entsprechende Technologie wird in der Regel zu Beginn eines Fahrzeugentwicklungsprozesses festgelegt und dann nicht mehr verändert (die nachträgliche Änderung einer Bustechnologie hätte Hardwareänderungen an sämtlichen damit verbundenen Steuergeräten zur Folge). Im Weiteren wird die Anwendungsebene fokussiert. Die Kompatibilität hängt hier von den Softwarekomponenten und der Konfiguration der Middleware ab, welche über den gesamten Entwicklungszeitraum bis zum Start der Serienproduktion und teilweise selbst nach der Auslieferung an die Endkunden noch verändert werden [Aut18][Sch20a][Men20][Mai20][Neu20].

3.4.2 Definitionen

Kampmann et al. definieren in [Kam19] den Begriff „Kompatibilität“ formell, im Kontext Service-orientierter Fahrzeugnetzwerke. Der dabei verwendete Servicebegriff bezieht sich auf die Anwendungsschicht (vgl. Abschnitt 3.1.5) und nicht auf die Kommunikation im Sinne von Abschnitt 2.4.7.

Definition 3.14 (Service). *Sei $\mathbb{S}$ die Menge fester Services, mit $\mathbb{S} = \{S_1, ..., S_\sigma\}$, $|\mathbb{S}| = \sigma \in \mathbb{N}^{>0}$. Jeder Service $S_i = (R_i, G_i) \in \mathbb{S}$, $1 \leq i \leq \sigma$ wird durch eine Menge fester Anforderungen R_i und einer Menge fester Garantien G_i definiert. Die Anzahl an Garantien für einen Service S_i ist $|G_i| = \kappa(i) \in \mathbb{N}$. Analog dazu ist für den Service S_i die Anzahl an Anforderungen $|R_i| = \mu(i) \in \mathbb{N}$. Entweder gilt $G_i = \emptyset$ oder $G_i = \{g_i^1, ..., g_i^{\kappa(i)}\}$ und entweder gilt $R_i = \emptyset$ oder $R_i = \{r_i^1, ..., r_i^{\mu(i)}\}$ für jedes $S_i \in \mathbb{S}$. [Kam19]*

Das heißt es gibt eine Menge an Services und jeder Service kann Anforderungen besitzen und Garantien geben (wobei die Sinnhaftigkeit eines Services, der keinerlei Garantien bietet, fraglich ist). Um die Eigenschaften der Anforderungen und Garantien zu beschreiben, werden sogenannte „Informationstypen“ definiert:

Definition 3.15 (Informationstypen)**.** *Wir definieren die Menge von Informationstypen* $\mathbb{T} = \{\tau_1, ..., \tau_\pi\}, \pi \in \mathbb{N}^{>0}$. *Jedes* $\tau_i = (I_i, D_i, Q_i, P_i) \in \mathbb{T}$ *hat einen festen Bezeichner* I_i *und besteht aus Datentypendefinitionen für Daten* D_i*, Qualität* Q_i *und Parameter* P_i*. Für* $1 \leq i \leq \pi$ *mit* $\tau_i \in \mathbb{T}$*, werden die Mengen aller möglichen,* D_i*,* Q_i *und* P_i *zuweisbaren Werte als* $\mathcal{V}^D_{\tau_i}$*,* $\mathcal{V}^Q_{\tau_i}$ *und* $\mathcal{V}^P_{\tau_i}$ *notiert. [Kam19]*

Dabei werden nicht nur die Daten D_i alleine, sondern auch deren Qualität Q_i (z. B. die Messunsicherheit eines Signals) und weitere Parameter P_i (z. B. der Wertebereich, den eine Datenquelle abdecken kann) angegeben. Man kann Q_i und P_i jeweils als Metadaten bezeichnen, wobei Q_i zur Laufzeit veränderlich und P_i statisch ist. Der Bezeichner I_i gibt an, um was für eine Information es sich handelt (z. B.: „Fahrzeuggeschwindigkeit“).

Um jeder Anforderung und jeder Garantie einen Informationstyp zuzuordnen, werden Typenfunktionen definiert:

Definition 3.16 (Typenfunktion)**.** *Jedes nicht-leere Anforderungs- oder Garantieelement wird durch* $\mathcal{T}_G$ *auf ein Element in der Menge der Informationstypen abgebildet, mit* $\mathcal{T}_G : \{(i,j) | S_i \in \mathbb{S}, g_i^j \in G_i\} \rightarrow \mathbb{T}$ *und* $\mathcal{T}_R$ *mit* $\mathcal{T}_R : \{(i,j) | S_i \in \mathbb{S}, r_i^j \in R_i\} \rightarrow \mathbb{T}$. *[Kam19]*

Unter diesen Voraussetzungen lässt sich Kompatibilität über die Befriedigung von Anforderungen durch Garantien definieren:

Definition 3.17 (Kompatibilität einer Garantie)**.** *Wir bezeichnen eine Garantie* $g_i^m \in G_i$ *des Service* $S_i \in \mathbb{S}$ *als kompatibel, wenn* $\mathcal{T}_G(i,m) = \mathcal{T}_R(j,n)$. *[Kam19]*

In [Kam19] werden Services beschrieben, um Service-orientierte Kommunikation (vgl. Abschnitt 2.4.7) abzubilden. Entsprechend wird davon ausgegangen, dass die Verbindung zwischen Anforderungen und Garantien zur Laufzeit erfolgt. Es wird explizit nicht vorausgesetzt, dass eine Service-orientierte Kommunikation durch einen Client initiiert werden muss, daher werden in den Definitionen auch nicht die Begriffe „Sender“ und „Empfänger“ verwendet. Ignoriert man den Verbindungszeitpunkt, so kann mit dem Modell auch Signal-basierte Kommunikation beschrieben werden. In Bezug auf das AUTOSAR Metamodell (vgl. 2.7.5) können mit $S_i \in \mathbb{S}$ anstatt Services die Softwarekomponenten (SWCs) beschrieben werden.

Die Indifferenz zwischen Service-orientierter und Signal-basierter Kommunikation wird durch die Definition der Verbindung noch deutlicher, da der Zeitpunkt, zu welchem die Middleware konfiguriert wird (zum Entwicklungszeitpunkt, oder zur Systemlaufzeit), nicht festgelegt wird:

Definition 3.18 (Service Verbindung)**.** *Eine Anforderung $r_i^m \in R_i, m \leq \mu(i)$ eines Services S_i wird als verbunden, mit einer Garantie $g_j^n \in G_j, n \leq \kappa(j)$ eines Services S_j, bezeichnet, wenn die zugrunde liegende Middleware so konfiguriert ist, dass Nachrichten in Bezug auf g_j^n von S_i empfangen werden. [Kam19]*

Auch das Zusammenfügen von Atomic SWCs zu SWC Compositions (vgl. Abschnitt 2.7.2) kann in dieser Notation beschrieben werden:

Definition 3.19 (Service-Composition)**.** *$\mathbb{C}$ sei die Menge aller Service-Compositions mit $\mathbb{C} = \{C_1, ..., C_\lambda\}, \lambda \in \mathbb{N}^{>0}$. Eine Service-Composition $C_i \in \mathbb{C}$ ist ein gerichteter Graph $C_i = (V_i, E_i)$. Die Knoten $V_i \subseteq \mathbb{S}$ sind eine Teilmenge der Menge der Services $\mathbb{S}$. Die Kanten $e_j \in E_i$ mit $j \in \mathbb{N}$ sind Verbindungen zwischen Anforderungen und Garantien der Services in V_i mit identischen Informationstypen:*
$E_i = \{(r_i^m, g_j^n) | S_i, S_j \in V_i \subseteq \mathbb{S}, r_i^m \in R_i, g_j^n \in G_j, \mathcal{T}_G(g_j^n) = \mathcal{T}_R(r_i^m)\}$.
[Kam19]

Auf dem Modell von Kampmann et al. aufbauend, können die folgenden Spezialfälle definiert werden:

Definition 3.20 (Atomic SWC). *Eine Service-Composition C_i wird als unteilbare Softwarekomponente („Atomic SWC“) bezeichnet, wenn es darin genau einen Knoten gibt:* $C_i = (V_i, E_i) : |V_i| = 1$.

Definition 3.21 (SWC Composition). *Eine Service-Composition C_i wird als zusammengesetzte Softwarekomponente („SWC-Composition“) bezeichnet, wenn es darin mehr als einen Knoten gibt:* $C_i = (V_i, E_i) : |V_i| > 1$.

Definition 3.22 (Autarkie). *Eine Service-Composition C_i wird als autark bezeichnet, wenn alle darin enthaltenen Anforderungen erüllt, d. h. Teil einer Kante, sind:* $\forall r_k^l \in R_k, (R_k, G_k) = S_k, \forall S_k \in V_i, C_i = (V_i, E_i) : \exists e_j \in E_i$.

Da der Graph C_i nicht zwingend zusammenhängend ist, kann damit auch ein gesamtes Fahrzeug `Fzg`, oder ein einzelnes Steuergerät `ECU` beschrieben werden. In diesem Fall werden die Symbole C_{Fzg} oder C_{ECU} verwendet.

Um weitere Aussagen zur Kompatibilität zwischen C_{Fzg} und C_{ECU} zu treffen, wird zunächst die folgende Funktion definiert:

Definition 3.23 (Kompatibilitätsfunktion). *Die Kompatibilität einer Service-Composition C_i zu einer weiteren Service-Composition C_j wird durch die Kompatibilitätsfunktion $\mathcal{K}$ bestimmt, welche die Menge der Anforderungen in C_i angibt, für welche weder in C_i noch in C_j kompatible Garantien bestehen:*
$\mathcal{K}(C_i, C_j) : \{r_i^m \in R_i | \nexists g_{i,j}^n \in G_i \cup G_j : \mathcal{T}_R(i, m) = \mathcal{T}_G((i, j), n)\}$

Zu beachten ist, dass $\mathcal{K}(C_i, C_j) \neq \mathcal{K}(C_j, C_i)$ für $i \neq j$. Damit können weitere Zustände definiert werden:

Definition 3.24 (Vollständige Kompatibilität). *Ein Steuergerät C_{ECU} heißt vollständig kompatibel zu einem Gesamtsystem C_{Fzg} wenn* $\mathcal{K}(C_{\text{Fzg}}, C_{\text{ECU}}) = \emptyset$.

Definition 3.25 (Eingeschränkte Kompatibilität). *Eine SWC C_{ECU} heißt eingeschränkt kompatibel zu einem Gesamtsystem C_{Fzg} wenn*
$0 < |\mathcal{K}(C_{\text{Fzg}}, C_{\text{ECU}})| < |R_{\text{Fzg}}|$.

Praktisch reicht die Spanne eingeschränkter Kompatibilität vom Fehlen einer geringfügigen, optionalen Funktionalität, wobei alle wesentlichen Funktionen vorhanden sind, sodass eine Erprobung nahezu uneingeschränkt möglich ist, bis zum Wegfall essenzieller Funktionen, die eine Erprobung unmöglich machen.

Ist ein Steuergerät zur Erfüllung seiner Aufgaben auf externe Ressourcen angewiesen, können die Garantien des Steuergeräts nur eingehalten werden, wenn dessen Anforderungen durch kompatible Garantien im Fahrzeug abgedeckt sind.

Definition 3.26 (Inkompatibilität)**.** *Eine SWC C_{ECU} heißt inkompatibel zu einem Gesamtsystem C_{Fzg} wenn $\mathcal{K}(C_{\text{Fzg}}, C_{\text{ECU}}) = R_{\text{Fzg}}$.*

Inkompatibilität führt in der Praxis zu einem Verhalten, als wäre ein entsprechendes Steuergerät nicht vorhanden. Diese Situation muss unterschieden werden, von Fällen in denen Kompatibilität vorzuliegen *scheint*:

Definition 3.27 (Scheinkompatibilität)**.** *Gegeben seien zwei unterschiedliche Garantien $g_i^a \neq g_i^b$, $g_i^a \in G_i$, $g_i^b \in G_i$, des Service $S_i \in \mathbb{S}$, sodass $\exists r_j^n \in R_i | \mathcal{T}_G(i,a) = \mathcal{T}_R(j,n)$ d. h. es gibt eine Anforderung r_j^n zu der die Garantie g_i^a kompatibel ist. Situationen, in denen die Anforderung r_j^n fälschlich mit g_i^b verbunden wird, werden als Scheinkompatibilität bezeichnet, da die Anforderung scheinbar befriedigt ist, die Garantie aber tatsächlich nicht kompatibel ist.*

Ein praktisches Beispiel dafür wäre eine Situation, in der die Signale innerhalb eines CAN-Frames vertauscht wurden. Ein Steuergerät, das den Frame empfängt, sieht weiterhin einen Frame mit der erwarteten ID und der korrekten Anzahl Bits. Der Frame *scheint* kompatibel zu sein. Die übertragenen Bits werden jedoch gemäß der ursprünglichen Position interpretiert, sodass ein fehlerhaftes Systemverhalten auftritt.

Definition 3.28 (Bedingte Kompatibilität)**.** *Die bedingte Kompatibilitätsfunktion $\mathcal{K}_b$ gibt die Menge an Anforderungen an, die durch eine SWC C_j erfüllt sein müssen, damit eine SWC C_i die Garantien $G_x \subset G_i$ bereitstellen kann.*

Dies ist primär dann von Bedeutung, wenn z. B. ein Steuergerät C_{ECU} aus mehreren SWCs $C_{1...n}$ besteht, die nicht miteinander verbunden sind. In dem Fall kann eine SWC, welche über unerfüllte Anforderungen verfügt, ihre Garantien nicht bieten, während andere SWCs, deren Anforderungen erfüllt sind, ihre Garantien bieten können. Im Fall des eingangs beschriebenen Beispiel mit dem Außenspiegel (vgl. Abschnitt 2.2) kann z. B. der Fahrer nicht über die LED auf Fahrzeuge im toten Winkel hingewiesen werden, wenn das Fahrzeug die entsprechende Information nicht bereitstellt. Ein Ausbleiben des Totwinkelsignals muss jedoch nicht die Möglichkeit zur elektronischen Verstellung des Spiegelglases beeinträchtigen.

Betrachtet man die zeitliche Entwicklung einer Komponente, stellt sich die Frage, ob eine ältere SWC C_x durch eine neuere SWC C_y ersetzt werden kann. Diese Eigenschaft bezeichnet man als Rückwärtskompatibilität:

Definition 3.29 (Rückwärtskompatibilität)**.** *Eine SWC C_y mit den Garantien G_y heißt rückwärtskompatibel zu der SWC C_x mit den Garantien G_x, wenn $G_x \subseteq G_y$ **und** wenn für keine der Garantien Scheinkompatibilität gemäß Def. 3.27 vorliegt.*

D. h. das System kann alle bisherigen Anforderungen weiterhin erfüllen (d. h. „rückwärts“), während weitere Funktionen hinzukommen, und durch die Änderungen oder neuen Elemente keine Kommunikationsfehler auftreten.

Die allgemeine Definition einer Schnittstelle in Def. 3.13 kann durch dieses Modell formalisiert werden:

Definition 3.30 (Schnittstelle, formell)**.** *Die Menge aller Anforderungen $r_i^m \in R_i$ und Garantien $g_i^n \in G_i$ einer Service-Composition C_i wird als dessen Schnittstelle bezeichnet.*

Diese Definition kann auf ein ganzes Steuergerät erweitert werden:

Definition 3.31 (ECU-Schnittstelle)**.** *Die Menge aller Anforderungen $r_i^m \in R_i$ und Garantien $g_i^n \in G_i$ aller Service-Compositions C_i eines Steuergeräts wird als dessen Schnittstelle bezeichnet.*

3.4.3 Kompatibilität in AUTOSAR

Signal-orientierte Kommunikation

Die Signal-orientierte Kommunikation basiert auf in PDUs gruppierten Signalen. Mögliche Änderungen an einem Steuergerät sind das Hinzufügen, Verändern oder Entfernen von PDUs, das Hinzufügen, Verändern oder Entfernen von Signalen zu PDUs, das Verändern von nichtfunktionalen Parametern, sowie Änderungen an der Funktionssoftware.

Die AUTOSAR Middleware betreffen davon die Änderungen an den PDUs und Signalen. Der COM-Stack, d. h. die Komponente der Basissoftware, welche die unteren Netzwerkschichten auf den Virtual Functional Bus (vgl. Abschnitt 2.7.3) abbildet, eines empfangenden Steuergeräts kopiert eine empfangene PDU in einen Puffer der erwarteten Länge. Wurden mehr Bytes übertragen als erwartet, so werden die überschüssigen Bytes verworfen. Dies ermöglicht nachträgliche Erweiterungen an einer PDU, ohne alte Empfangssteuergeräte anpassen zu müssen. Ebenfalls möglich ist es auf der Empfängerseite Signale aus einer PDU zu entfernen, ohne den Sender anzupassen. Andere Modifikationen wären nicht rückwärtskompatibel.

Die Signale innerhalb einer PDU werden in der festgelegten Reihenfolge seriell über den Bus übertragen. Es gibt innerhalb der PDU außer der Reihenfolge keine Markierungen, die die Signale identifizieren. Daher können Elemente nur am Ende einer PDU hinzugefügt oder entfernt werden. Sonst würde sich die Reihenfolge der Signale innerhalb einer PDU verändern und Daten missinterpretiert werden.

Wird ein Steuergerät in veränderter Form, aber mit derselben Netzwerkkonfiguration (d. h. gleiche PDUs, Signale, Interfaces etc.) neu integriert, ist es mit dem Rest des Fahrzeugs kompatibel, ohne dass andere Geräte neu integriert werden müssen. In dieser Hinsicht ist AUTOSAR Kompatibilität mit Java Binärkompatibilität vergleichbar.

Service-orientierte Kommunikation über SOME/IP

Bei der SOME/IP Kommunikation (vgl. Abschnitt 3.1.6) hängt die Kompatibilität von diversen optionalen Funktionen der AUTOSAR Middleware ab. Je nach Konfiguration sind Hinzufügen, Entfernen, Erweitern und Verschieben (d. h. Veränderung der Übertragungsreihenfolge) von Datenelementen kompatible Änderungen. Eine Übersicht, welche Änderungen unter welchen Bedingungen kompatibel sind, befindet sich in Anhang A.

3.5 Versionen und Versionsverwaltung

Definition 3.32 (Version)**.** *Eine Version v repräsentiert einen Zustand eines sich entwickelnden Elements e. v wird durch ein Tupel v = (pr, vr) charakterisiert, wobei pr einen Zustand im Produktraum und vr einen Punkt im Versionsraum bezeichnen. [Con98]*

Der Versionsraum besteht aus zwei orthogonalen Dimensionen [Wed94][Wit22]:

Definition 3.33 (Revision)**.** *Sequentielle Versionen, die sich entlang der Zeitdimension entwickeln, werden Revisionen genannt. [Wes01]*

Definition 3.34 (Variante)**.** *Parallele oder alternative Versionen, die zu einem gegebenen Zeitpunkt koexistieren, werden Varianten genannt. [Wes01]*

Der Raum aus Varianten und Revisionen füllt sich mit Punkten, indem zunächst eine initiale Version angelegt wird (z. B. „V00“). Durch Änderungen an dieser Version entsteht eine neue Revision (z. B. „V01“). Auch diese kann verändert werden, um eine neue Revision zu schaffen (z. B. „V02“). So ergibt sich eine Kette aus aufeinanderfolgenden Revisionen. Werden von einer Revision ausgehend verschiedene neue Versionen geschaffen (z. B. von „V01“ ausgehend „VA02“ und „VB02“), handelt es sich um Varianten. Zu einem gegebenen Zeitpunkt können mehrere unterschiedliche Varianten aktuell sein. Varianten

können parallel weiterentwickelt werden (d. h. zu jeder der Varianten „VA2“ und „VB2“ können neue Revisionen geschaffen werden). Die Entwicklung von Revisionen wird auch als vertikale Entwicklung bezeichnet, während die Entwicklung von Varianten als horizontale Entwicklung bezeichnet wird.

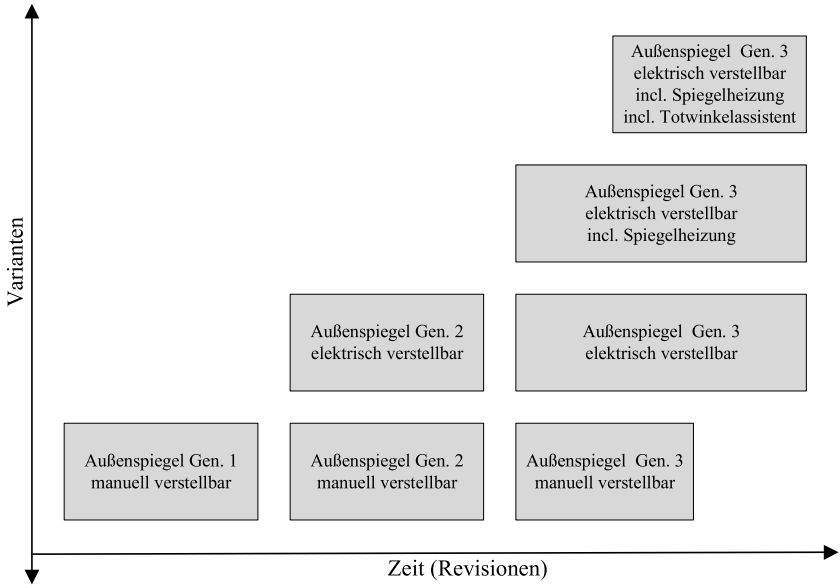

Abbildung 3.7: Varianten vs. Revisionen

Anhand der technischen Entwicklung von Außenspiegeln werden die zwei Dimensionen „Variante“ und „Revision“ in Abb. 3.7 veranschaulicht. Ursprünglich gab es ausschließlich manuell verstellbare Außenspiegel. Mit einer neuen Fahrzeuggeneration mussten diese angepasst werden (z. B. um mechanisch u. optisch zur neuen Fahrzeugform zu passen), sodass es zu einer neuen Revision des Außenspiegels kam. Parallel dazu wurden elektrisch verstellbare Spiegel eingeführt, welche optional anstatt der manuell verstellbaren verbaut werden können. Diese stellen eine Variante dar. Mit der Zeit kann die Zahl der Varianten zunehmen (vgl. Abschnitt 3.6), zuvor

bestehende Varianten können aber auch eingestellt werden (im Beispiel: die manuell verstellbaren Spiegel).

Als Element kommen sämtliche Arbeitsprodukte und Entwicklungseinheiten infrage [Mar11]. So z. B. Dateien und Ordner in einem Dateisystem (im Bereich der Softwareentwicklung vor allem Programmcode, allgemein aber auch Dokumente wie z. B. Spezifikationen, Testberichte oder die vorliegende Dissertation), Objekte in einer Datenbank[1], Entwicklungsstände ganzer Hard- oder Softwareprojekte, oder Schnittstellen in Hard- oder Software.

Definition 3.35 (Konfiguration)**.** *Eine Konfiguration ist eine versionierbares Element, welches eine Menge anderer versionierbarer Elemente referenziert. Durch eine Version wird das Element an sich verwaltet. Dagegen werden durch eine Konfiguration nur die Beziehungen zu den in der Konfiguration enthaltenen Elementversionen und nicht die Elementversionen an sich verwaltet. [Sch16b]*

Projekte, insbesondere eine E/E-Netzwerkentwicklung, bestehen i. d. R. aus mehr als einem einzelnen Element. Versionen können sowohl die Zustände der Gesamtheit, von Teilmengen des Gesamtsystems, oder von Einzelelementen (z. B. der Entwicklungsstand einer Fahrzeugbaureihe, eines Steuergeräts oder eines einzelnen Softwaremoduls auf einem Steuergerät), bezeichnen. Als Vereinfachung wird für ein System nur eine Gesamtversion angegeben, während die Verwaltung der Subsysteme und derer Versionen als Konfigurationsmanagement bezeichnet wird [Mar11].

3.5.1 Versionskontrollsysteme

Weil Systeme schrittweise entwickelt und inkrementell verbessert werden, sind Methoden notwendig, um die Versionen der einzelnen Elemente bis hin

[1] Dies ist insofern relevant, als die Modellierung auf Basis des AUTOSAR Metamodells (vgl. Abschnitt 2.7.5) mittels datenbankbasierten Werkzeugen erfolgt. Beim Export solcher Objekte in eine ARXML-Datei, können diese mit Versionsnummern (für jedes Objekt individuell) versehen werden.

zum Gesamtsystem zu verwalten. Werkzeuge zur Verwaltung von Softwareversionen, sogenannte Versionskontrollsysteme (englisch: „Version Control System“, VCS), gibt es seit den 1970er Jahren [Roc75]. Bereits das „Source Code Control System“ (SCCS) war in der Lage, die Veränderungen und den Zugriff auf ein Softwareprojekt zu verwalten. Damit war es möglich, jederzeit zu ermitteln, wann eine Änderung durchgeführt wurde, durch wen dies geschah und was ggf. als Begründung für die Änderung hinterlegt wurde.

Aktuelle Werkzeuge zur Verwaltung von Softwareständen sind zum Beispiel „Apache Subversion“ (SVN) mit einem zentralisierten Quellcodearchiv (Repository) [Pil08] oder die Open-Source-Projekte Mercurial [Arn20] und Git [Cha14], jeweils mit einer (über die Rechner aller beteiligten Personen) verteilten Struktur. Ein wesentliches Merkmal der modernen Tools ist, dass diese kollaboratives Arbeiten an einer gemeinsamen Codebasis ermöglichen. Einer Studie von 2016 zufolge ist Git das am weitesten verbreitete VCS Werkzeug, mit SVN auf Platz zwei, gefolgt von Mercurial [Rho16].

Im Bereich der Fahrzeugentwicklung wird für die Verwaltung von Software in der Regel Git verwendet. Für andere Artefakte, wie z. B. die K-Matrix oder die Dokumentation der Anforderungen werden spezialisierte Datenbankprogramme wie z. B. „Preevision“ oder „XDIS“ eingesetzt, die neben ihrer Kernfunktionalität ebenfalls Funktionen zur Versionskontrolle aufweisen [Sch16a][Hel17][Vet23a].

Sobald ein Benutzer den aktuellen Zustand seines Projektverzeichnisses in das Repository „eincheckt“ entsteht eine neue Revision. Die Revisionsnummer ist bei SVN eine fortlaufende Nummer. Bei Git ist sie ein Hashwert basierend auf dem aktuellen Zustand, und der Historie, die zu diesem Zustand geführt hat. Jeder Revision kann ein beliebiger Bezeichner („Tag“) zugeordnet werden. Dieser kann zur expliziten Versionierung verwendet werden (z. B. kann eine Revision „72dc9f05...“ das Tag „1.0.2_final“ bekommen).

Diese Mechanismen zur expliziten Versionierung müssen von den Benutzern mit sinnvollen Werten befüllt werden. Nicht alle Revisionen müssen ein explizites Tag erhalten. Es genügt die Revisionen zu markieren, welche eine Version enthalten, die anderweitig (z. B. für einen Test oder eine Auslieferung) weiterverwendet werden. Innerhalb eines Entwicklungsprojektes können Zwischenstände anhand ihrer Revisionsnummer eindeutig referenziert werden.

3.5.2 Änderungen

Neue Revisionen eines Systems entstehen durch Änderungen. Diese lassen sich in drei Kategorien einteilen:

Definition 3.36 (Fehlerkorrektur („Bug Fix“))**.** *Als Fehlerkorrektur bezeichnet man eine interne Änderung, die ein falsches Verhalten verbessert. [Pre13]*

Bei einer reinen Fehlerkorrektur kommen weder neue Funktionen hinzu, noch fallen alte weg.

Definition 3.37 (Geringfügige Änderung)**.** *Eine Änderung ist geringfügig, wenn sie neue Funktionalität einbringt oder die Entfernung bestehender Funktionen ankündigt, dabei aber vollständig rückwärtskompatibel ist. [Pre13]*

Definition 3.38 (Inkompatible Änderung)**.** *Eine Änderung, die nicht rückwärtskompatibel ist, wird als inkompatibel bezeichnet. [Pre13]*

D. h. eine inkompatible Änderung, zeichnet sich durch das Wegfallen, bisher bestehender Garantien, aus.

3.5.3 Semantische Versionierung

Prinzipiell kann die Vergabe der Versionsbezeichnung nach einem beliebigen Schema erfolgen. Beispielsweise erhalten die Veröffentlichungen der Ubuntu Linux Distribution eine Nummer bestehend aus Jahres- und Monatszahl

des Veröffentlichungszeitpunktes sowie eine Alliteration bestehend aus einer Eigenschaft und einem Tier, die mit jeder neuen Veröffentlichung einen Buchstaben weitergezählt werden (z. B. „23.10 Mantic Minotaur“ und „24.04 Noble Numbat“) [Hei24]. Andere Schemata können z. B. eine einfache Zahl, die mit jeder Veröffentlichung inkrementiert wird, oder eine Kombination aus zwei Zahlen, bei denen, je nach (subjektiver) Größe einer Änderung, die erste oder die zweite Zahl inkrementiert wird, sein.

Idealerweise erfolgt die Versionierung nicht nach einem willkürlichen Schema, sondern bringt Informationen über die zu erwartenden Unterschiede zwischen zwei Versionen mit sich.

Als ein formelles Schema zur Vergabe von Versionsnummern wurde 2010 von Tom Preston-Werner die sogenannte „Semantische Versionierung“ (englisch: „Semantic Versioning“, s. Abb. 3.8) vorgeschlagen [Pre13]. Dabei besteht eine Versionsnummer aus drei durch Punkte getrennte Stellen. Diese werden als „Major-Version“ (dt.: „Hauptversion“), „Minor-Version“ (dt.: „Unterversion“), und „Patch-Version“ (dt.: „Korrekturversion“) bezeichnet. Dabei wird „Patch“ inkrementiert, wenn eine Fehlerkorrektur (vgl. Def. 3.36) stattgefunden hat. Geringfügige Änderungen (vgl. Def. 3.37) inkrementieren die „Minor“ Stelle. Inkompatible Änderungen (vgl. Def. 3.38), welche zu einer Verringerung der Kompatibilität zu anderen Komponenten führen, z. B. weil bestehende Schnittstellen ersetzt werden, inkrementieren die „Major“ Stelle. Wird die Hauptversion um eins erhöht, werden Minor- und Patch-Version auf null zurückgesetzt. Wird die Unterversion erhöht, wird die Korrekturversion auf null reduziert. Den drei semantischen Stellen kann mit einem Bindestrich ein zusätzliches Etikett („Label“) angehängt werden, welches zusätzliche Informationen, ohne festgelegte Form beinhalten kann.

$$\overset{Major}{\overbrace{1}} . \overset{Minor}{\overbrace{0}} . \overset{Patch}{\overbrace{2}} - \overset{Label}{\overbrace{beta2}}$$

Abbildung 3.8: Beispiel einer semantischen Versionsnummer

Durch „Semantic Versioning“ sollen Änderungen, welche eine Anpassung anderer Komponenten eines gemeinsamen Systems erforderlich machen, an der Versionsnummer abgelesen werden können. Eine Studie zeigte jedoch, dass in der Praxis das Schema nicht konsequent eingesetzt wird und auch in einem Drittel der untersuchten Minor-Versionen inkompatible Änderungen enthalten [Rae14].

3.5.4 Releases

Definition 3.39 (Release)**.** *Der englische Begriff „Release“ (deutsch: „Veröffentlichung“, „Freigabe“, „Abgabe“) bezeichnet die Veröffentlichung (oder Weitergabe an ein anderes Glied in der Lieferkette) zu entwickelnder Elemente in einer bestimmten Version (vgl. Definition 3.32).*

Der Prozess, der zu neuen Releases führt, wird als „Release Management“ bezeichnet [Bis07]. Es wird in zwei Kategorien unterteilt. Werden neue Versionen veröffentlicht, sobald sich wesentliche Funktionen geändert haben oder hinzugekommen sind, unabhängig vom jeweiligen Zeitpunkt, spricht man von „Feature Based Releases“. Erfolgt die Veröffentlichung neuer Versionen in einem festen, zeitlichen Schema, spricht man von „Time Based Releases“ [Mic15]. Feature Based Releases werden vor allem bei kleineren Projekten verwendet, während mit zunehmender Größe und Komplexität auf zeitbasierte Veröffentlichungsschemata gesetzt wird [Tei17].

Für Fahrzeuge erfolgen Releases in der Entwicklungsphase viertel- bis halbjährlich zu festgelegten Zeitpunkten [Sax17][Gui18][Mar19]. Je nach Häufigkeit sind unterschiedliche Änderungsumfänge sowie Verifikations- u. Validierungsaufwände (vgl. Abs. 3.2.2) für jedes einzelne Release möglich [Mar19]. Alternativ können Teilumfänge, z. B. jedes zweite oder vierte Release umgesetzt werden. Bei in Serienproduktion befindlichen Fahrzeugen erfolgen halbjährliche Releases der Gesamtsoftware (auch als Grundlage für OTA-Updates, vgl. Abschnitt 2.6.5), wobei in Zukunft kürzere Release-Zyklen erwartet werden [Sax17][Gui18].

K-Matrix-Releases

Veränderungen an einer Kommunikationsmatrix (vgl. Abschnitt 2.5) werden in Form erneuerter ARXML-Dateien (vgl Abschnitt 2.7.5) veröffentlicht. Man spricht dabei von K-Matrix-Releases. Bei K-Matrizen werden Time Based Releases verwendet.

Definition 3.40 (K-Matrix-Release-Menge)**.** *K sei die Menge aller K-Matrix-Releases, mit $\mathbb{K} = \{K_0, K_1, ..., K_{\varphi-1}\}$ und $|\mathbb{K}| = \varphi \in \mathbb{N}$.*

Die Indizes bilden die zeitliche Reihenfolge ab, entsprechen aber keinem festen zeitlichen Abstand (vgl. Abschnitt 5.1.3).

Release-konformes-Flashen

Ein K-Matrix-Release hat zur Folge, dass in Steuergeräteprojekten die Basissoftware (vgl. Abschnitt 2.6.3), insbesondere die Komponenten, die die Netzwerkschnittstellen betreffen, an die veränderte Kommunikation angepasst werden muss. Um die Kompatibilität der Steuergeräte untereinander sicherzustellen, werden sämtliche Steuergeräte, die gemeinsam betrieben werden sollen, mit Softwareversionen geflasht[1], die auf demselben K-Matrix-Release basieren. Dieses Vorgehen bezeichnet man als „Release-konformes-Flashen“.

Basierend auf einem K-Matrix-Release, kann es für die einzelnen Steuergeräte mehr als ein Software-Release geben. Diese unterschiedlichen Software-Releases können ein verändertes Verhalten des Steuergerätes beinhalten, müssen aber jeweils dieselbe, durch das K-Matrix-Release beschriebene, Schnittstelle aufweisen.

[1] Unter „Flashen“ versteht man das Beschreiben von Steuergeräten mit Software. Der Begriff basiert auf dem in Mikrocontrollern verwendeten nicht flüchtigen Flash-Speicher, in dem die Programme abgelegt werden.

3.6 Varianten

Fahrzeuge werden hochindividuell gefertigt. Im Jahr 2017 wurden für den Verkauf in Deutschland 84.000 VW Golf gefertigt – darunter befanden sich 58.000 unterschiedliche Varianten [Dol18]. Eine eigene Untersuchung des VW-Golf Konfigurators [Vol21, Stand 22.1.2021] ergibt folgende Liste:

- 19 verschiedene Antriebsstränge
  - 8 Varianten mit Ottomotor (Benzin)
  - 5 Varianten mit Dieselmotor
  - 3 Varianten mit Mildhybrid (Benzin)
  - 2 Varianten mit Plug-in-Hybrid (Benzin)
  - 1 Variante mit Ottomotor (Methan)
- 16 verschiedene Felgen
- 12 verschiedene Lackierungen
- 4 verschiedene Innenraumfarben
- Ca.[1] 50 verschiedene optionale Sonderausstattungen

So ergeben sich theoretisch bis zu $19 \cdot 16 \cdot 12 \cdot 4 \cdot 2^{50} = 1{,}64 \cdot 10^{19}$ mögliche Konfigurationen. Diese Zahl wird sowohl durch technische Einschränkungen (z. B. die Einparkautomatik nur in Kombination mit den Ultraschallabstandssensoren) als auch durch Design- bzw. Markenregeln (z. B. Elemente der Sportoptik erst ab einer Mindestmotorisierung) eingeschränkt. Für die 2020 erschienene Mercedes-Benz S-Klasse werden „theoretisch mehr als $1{,}6 \cdot 10^{103}$ unterschiedliche Endproduktvarianten beziehungsweise technische Bauteilanordnungen“ angegeben, aber ebenfalls die Einschränkung gemacht, dass real

[1] „Circa“, da nicht ganz eindeutig ist, was noch als Teil des Fahrzeugs zählt und was Zubehörteile sind, die der Fahrzeughersteller anbietet. Würde man Aschenbecher, Feuerlöscher, die optionalen Schutzfolien für Heckklappe und Seitenschweller etc. mitzählen, fiele die Zahl noch größer aus.

nicht alle dieser Kombinationen baubar sind [Pet20]. Mittels spezieller Software werden, unter Berücksichtigung der erforderlichen Bauräume für die einzelnen Komponenten und weiterer Faktoren, Plausibilitätsprüfungen für individuelle Konfigurationen durchgeführt.

Für die Netzwerkentwicklung ist diese Varianz nur zum Teil von Bedeutung. Lack, Felgen oder die Farbe der Sitzbezüge spielen für die Elektronik keine Rolle. Ausstattungsvarianten, die das Netzwerk betreffen, sind z. B. die Headunit, welche typischerweise in einer einfachen, mittleren und hochwertigen Ausführung bestellt werden kann, oder Zusatzfunktionen wie eine Lenkradheizung. Bei diesen Teilen sind entweder ein Alternativverbau (z. B. gibt es ein Interface für eine Headunit, an das jede der möglichen Varianten angeschlossen werden kann) oder ein optionaler Einbau (z. B. kann die Lenkradheizung verbaut werden, oder nicht) möglich.

Im Beispiel mit der Headunit bedeutet das, dass in der K-Matrix (vgl. Abs. 2.5) drei Headunits eingetragen sind. Mehrere Varianten eines Gerätes senden und empfangen überlappende Sets an Signalen. Beispielsweise sind alle drei Headunitvarianten Empfänger für ein Signal zur Regelung der Lautstärke eingetragen. Eine höherwertige Headunit wird zusätzlich das Signal einer Rückfahrkamera empfangen, das die Einstiegsvariante nicht empfängt. Das 150%-Modell enthält außerdem sämtliche optionalen Elemente. Reale Fahrzeuge werden durch Teilmengen dieses 150%-Modells beschrieben.

In der Anwendungsentwicklung, müssen die möglichen Kombinationen aus Sendern u. Empfängern, sowie das Verhalten bei fehlenden Sendern oder Empfängern, berücksichtigt werden. Dies ist nicht Gegenstand dieser Arbeit.

Aus der Perspektive der Netzwerkentwicklung stellt ein Signal, welches zwar gesendet, aber nicht empfangen wird, kein Problem dar, da andere Verbindungen dadurch nicht beeinträchtigt werden. Ein vorhandener Empfänger, für den nie ein Signal gesendet wird, verursacht auch keine Störungen in der Kommunikation. Dadurch kann der Variantenreichtum mittels einer 150%-K-Matrix abgebildet werden, anstatt dass für jede (real vorkommende) Konfiguration eine eigene K-Matrix modelliert werden muss.

Auf diese Art werden Varianten durch die zentrale Vernetzung ermöglicht. Eine Verwaltung oder ein Konfigurationsmanagement findet nicht statt. Beispielsweise kann ein System aus zwei Komponenten bestehen, von denen die eine ein Signal sendet und die andere es empfängt. Wird in einem Fahrzeug nur eine der beiden Komponenten verbaut, so wird das System nicht funktionieren. Aus Sicht der Vernetzung stellt dieser Fall keinen Konflikt dar. Entweder wird ein Signal gesendet, für das es keinen Empfänger gibt oder ein Empfänger existiert, für den kein Signal gesendet wird. Andere Steuergeräte oder Systeme werden hierdurch nicht beeinträchtigt. Die Sicherstellung das in ein reelles Fahrzeug nur mit einer sinnvollen Kombination[1] aus Steuergeräten aufgebaut wird, erfolgt außerhalb der zentralen Vernetzung. Hierfür sind die Bauteil- und Systemverantwortlichen zuständig.

3.7 Buslastberechnung

In der Literatur werden Methoden zur Buslastberechnung genannt [Zim14, Nem19], die eine Abschätzung ermöglichen, aber nicht bitgenau ausformuliert sind, und zwischen den unterschiedlichen CRC-Größen[2] von CAN-FD [Har12] nicht unterscheiden. Im Folgenden wird zur Buslastberechnung das in der Literatur vorgestellte Prinzip verwendet, jedoch mit den exakten Zahlen aus dem CAN- bzw. CAN-FD-Standard [Int24a]. Die mathematische Modellierung der Kommunikationsmatrix und der für die Buslastberechnung notwendigen Eigenschaften erfolgt eigenständig, als Erweiterung des aus [Kam19] übernommenen Modells zur Beschreibung von Services und deren Kompatibilität (vgl. Abs. 3.4.2).

[1] Z. B. anhand von Regeln wie „nur eine Headunit“ oder „Langstreckenradar ausschließlich in Kombination mit Tempomathebel“ zu überprüfen.

[2] Cyclic Redundancy Check (CRC) oder zyklische Redundanzprüfung ist ein Verfahren, um mittels zusätzlicher Prüfbits die Integrität der übertragenen (oder gespeicherten) Botschaft sicherzustellen.

Die gesamte Buslast setzt sich aus der sogenannten dynamischen und der statischen Last zusammen. Die dynamische Last ergibt sich aus den Event-abhängigen Frames, die statische Last wird durch die zyklisch gesendeten Botschaften verursacht (vgl. Abs. 2.4.6). Die statische Buslast ist anhand von in der K-Matrix vorliegenden Informationen berechenbar.

Voraussetzung für eine Berechnung der dynamischen Last wären detaillierte Kenntnisse über die Event-abhängigen Systeme, aus denen sich die Häufigkeit, mit der Events auftreten, ableiten lassen. Um ohne diese, teilweise unvorhersehbaren[1] Eigenschaften zu kennen, die Buslast sinnvoll berechnen zu können, wird für dynamische Botschaften eine „Minimum Update Time“ (MUT) festgelegt [Dai17] und in der K-Matrix eingetragen. Sporadische Nachrichten dürfen nicht häufiger als die MUT gesendet werden. Anhand dieser maximalen Sendehäufigkeit lässt sich die im schlechtesten Fall verursachte Buslast, analog zur statischen Last, errechnen.

Definition 3.41 (Bus-Menge). *$\mathbb{B}$ sei die Menge aller Busse, mit* $\mathbb{B} = \{B_1, ..., B_\psi\}, |\mathbb{B}| = \psi \in \mathbb{N}$.

Definition 3.42 (Bus-Release). *Jeder Bus $B_j \in \mathbb{B}$ besteht aus einer Menge Bus-Releases $B_{i,j}$ für die gilt: $B_j = \{B_{i,j} | i \in [0,\varphi - 1]\}$.*

Jedes Bus-Release lässt sich einem K-Matrix-Release zuordnen:
$\forall B_{i,j} \exists K_k | i = k$.

Definition 3.43 (PDU-Menge). *$\mathbb{W}$ sei die Menge aller PDUs, mit* $\mathbb{W} = \{W_1, ..., W_\eta\}, |\mathbb{W}| = \eta \in \mathbb{N}$.

Definition 3.44 (PDU-Release). *Jede PDU $W_j \in \mathbb{W}$ besteht aus einer Menge PDU-Releases $W_{i,j}$ für die gilt: $W_j = \{W_{i,j} | i \in [0,\varphi - 1]\}$.*

[1] Z. B. lässt sich das Auftreten von durch Benutzereingaben, wie z. B. Knopfdrücke, verursachten Botschaften nicht vorhersehen.

Jedes PDU-Release lässt sich einem K-Matrix-Release zuordnen: $\forall W_{i,j} \exists K_k | i = k$. Jedes Bus-Release $B_{i,j}$ besteht aus einer Menge PDU-Releases: $B_{i,j} = \{W_{i,j} | j \in [1, \eta]\}$.

Innerhalb eines K-Matrix-Releases K_i kann jedes PDU-Release $W_{i,j}$ einer Menge $B^{W(i,j)}$ von Bussen zugeordnet werden, wobei gilt:

$$\forall (i,j) \exists B^{W(i,j)} = \{B_{i,k} | W_{i,j} \in B_{i,j}\} \quad \text{mit } 0 \leq |B^{W(i,j)}| \leq \psi \tag{3.1}$$

Im Fall von PDUs, die über Gateways zwischen mehreren Bussen weitergeleitet werden, gilt $|B^{W(i,j)}| > 1$.

Busse, die keine PDUs enthalten, d. h. für die gilt $B_{i,j} = \emptyset$, sind zum Release K_i nicht Teil des Fahrzeugs. In der Praxis bedeutet das üblicherweise, dass ein Bus nachträglich eingeführt wurde, um zusätzliche Anforderungen umzusetzen, oder ein Bus nachträglich entfernt werden konnte, weil dieser z. B. durch Optimierungen obsolet wurde.

Definition 3.45 (Signalmenge)**.** *$\mathbb{Y}$ sei die Menge aller Signale, mit* $Y = \{Y_1, ..., Y_\theta\}, |\mathbb{Y}| = \theta \in \mathbb{N}$.

Definition 3.46 (Signal-Release)**.** *Jedes Signal $Y_j \in \mathbb{Y}$ besteht aus einer Menge Signal-Releases* $Y_j = \{Y_{i,j} | i \in [0, \varphi - 1]\}$.

Analog zum PDU-Release lässt sich jedes Signal-Release einem K-Matrix-Release zuordnen: $\forall Y_{i,j} \exists K_k | i = k$. Jedes PDU-Release $W_{i,l}$ besteht aus einer Menge Signal-Releases $W_{i,l} = \{Y_{i,j} | j \in [1, \theta]\}$.

Innerhalb eines K-Matrix-Release K_i kann jedes Signal-Release $Y_{i,j}$ einer Menge $W^{Y(i,j)}$ von PDUs zugeordnet werden, wobei gilt:

$$\forall (i,j) \exists W^{Y(i,j)} = \{W_{i,k} | Y_{i,j} \in W_{i,k}\} \quad \text{mit } 0 \leq |W^{Y(i,j)}| \leq \eta \tag{3.2}$$

Im Sonderfall $W^{Y(i,j)} = \emptyset$, d. h. einem Signal $Y_{i,j}$ welches im Release K_i keiner PDU zugeordnet ist, existiert dieses Signal zum jeweiligen Release nicht im Fahrzeug. PDUs, die keine Signale enthalten, d. h. für die gilt $W_{i,j} = \emptyset$,

sind zum Release K_i nicht Teil des Fahrzeugs. Die praktische Häufigkeit dieser Szenarien (hinzukommende/wegfallende PDUs/Signale) wird in Abschnitt 5.2 untersucht.

Anhand dieses Modells lassen sich zwei Beispiele für Verletzungen der K-Matrix Konsistenz (vgl. Definition 2.1) darstellen:

- Ein Signal $Y_{i,j}$, welches einer PDU $W_{i,k}$ zugeordnet ist, d. h. $Y_{i,j} \in W_{i,k}$, sollte Teil des Fahrzeugs sein. Ist die PDU $W_{i,k}$ im Release K_i keinem Bus $B_{i,l}$ zugeordnet, d. h. $B^{W(i,k)} = \varnothing$, sollte sie nicht im Fahrzeug vorkommen.
- Eine leere PDU $W_{i,k} = \varnothing$ ist einem Bus $B_{i,l}$ zugeordnet, d. h. $W_{i,k} \in B_{i,l}$. Die PDU sollte als leere PDU nicht Teil des Fahrzeugs sein, ist aber einem Bus des Fahrzeugs zugeordnet.

Das sind jeweils Widersprüche, anhand derer Modellierungsfehler automatisiert erkannt werden können.

Definition 3.47 (Signaltypfunktion)**.** *Jedes Signal-Release $Y_{i,j}$ wird durch $\mathcal{T}_Y$ auf ein Element in der Menge der Informationstypen (vgl. Definition 3.15) abgebildet, mit $\mathcal{T}_Y : Y_{i,j} \rightarrow \mathbb{T}$.*

Diese Definition ist angelehnt an die Typenfunktion gem. Definition 3.16. Anhand der Informationstypen kann überprüft werden, ein Signal den Anforderungen bzw. Garantien eines Systems entspricht und Kompatibilität gem. Definition 3.17 vorliegt.

Definition 3.48 (Bitfunktion)**.** *Aus einem Informationstyp $\tau_i \in \mathbb{T}$ lässt sich durch $\mathcal{B}$ die Anzahl der zur Übertragung einer Information erforderlichen Datenbits d ermitteln, mit $\mathcal{B} : \tau_i \rightarrow d \in \mathbb{N}$.*

Definition 3.49 (Signalgröße)**.** *Sei $s_{i,j}$ die Signalgröße, als Bezeichnung für die Anzahl der Datenbits des Signals Nr. j im Release Nr. i.*

$$s_{i,j} = \mathcal{B}(\mathcal{T}_Y(i,j)) \tag{3.3}$$

Definition 3.50 (Zykluszeitfunktion). *Aus einem Informationstyp $\tau_i \in \mathbb{T}$ lässt sich durch $\mathcal{Z}$ die Zykluszeit z in Sekunden, innerhalb der eine Information übertragen werden muss, ermitteln mit $\mathcal{Z} : \tau_i \rightarrow z \in \mathbb{R}^{>0}$.*

Aus den Bit- und Zykluszeitfunktionen werden die zur Bestimmung der Buslast relevanten Eigenschaften eines Signals ermittelt. Die Zykluszeiten ergeben sich aus den Anforderungen der Systeme, die die in den PDUs enthaltenen Daten verarbeiten. Sie werden im Rahmen der Systementwicklung festgelegt und in der K-Matrix-Datenbank gespeichert.

Definition 3.51 (Zykluszeit). *$z_{i,j}$ sei die Dauer des Zyklus, in dem die PDU Nr. j in Release Nr. i gesendet wird.*

Um die Anforderungen aller enthaltenen Signale zu erfüllen, entspricht die Zykluszeit einer PDU der Zykluszeit des Signals, mit der höchsten Sendehäufigkeit.

$$z_{i,j} = min(\{\mathcal{Z}(\mathcal{T}_Y(i,k)) | Y_{i,k} \in W_{i,j}\}) \tag{3.4}$$

Definition 3.52 (Datenbits). *$d_{i,j}$ sei die Anzahl der Datenbits einer PDU Nr. j in einem Release Nr. i.*

Die Anzahl der Datenbits $d_{i,j}$ einer PDU $W_{i,j}$ ergibt sich aus der Summe der Bits aller Signale $Y_{i,k} \in W_{i,j}$:

$$d_{i,j} = \sum_{\forall Y_{i,k} \in W_{i,j}} s_{i,k} \tag{3.5}$$

Definition 3.53 (Maximale PDU-Größe). *$d_{max.}(j)$ sei die für PDUs maximal mögliche Anzahl Datenbits auf einem Bus Nr. j, mit*

$$d_{max.}(j) = \begin{cases} 64 & CAN \\ 512 & CAN\text{-}FD \end{cases}$$

Definition 3.54 (Paddingbits). *Sei $p_{i,j,k}$ die Anzahl der Paddingbits („Padding" engl. für „Füllung"), die benötigt werden, um die Datenbits einer PDU $W_{i,j}$ aufzufüllen, sodass sie den Anforderungen eines Bus-Release $B_{i,k}$ entspricht. ISO-11989 sieht vor, das Bitmuster CC_{Hex} als Padding zu verwenden. [Int24a]*

Gem. ISO-11989 gilt für die Summe aus Datenbits $d_{i,j}$ und Paddingbits $p_{i,j,k}$ einer PDU $W_{i,j}$ auf einem Bus $B_{i,k}$ [Int24a]:

$$d_{i,j} + p_{i,j,k} \in \begin{cases} \{8, 16, 24, 32, 40, 48, 56, 64\} & \text{CAN} \\ \{8, 16, 24, 32, 40, 48, 56, 64, & \text{CAN-FD} \\ 96, 128, 160, 192, 256, 384, 512\} & \end{cases} \tag{3.6}$$

Die größeren Schritte zwischen den CAN-FD Nutzdatengrößen sind so gewählt, um den „Data Length Code" (DLC) auf vier Bits (d. h. 16 Stufen) beschränken zu können. Die Anzahl der Datenbits einer PDU sind unabhängig vom jeweiligen Bus (eine PDU beinhaltet im gesamten Netzwerk dieselben Informationen).

Definition 3.55 (Adressbits). *Sei $a_{i,j,k}$ die Anzahl der Adressbits der PDU Nr. j in Release Nr. i, auf Bus Nr. k.*

Es gilt:

$$a_{i,j,k} = \begin{cases} 11 & \text{Basis ID} \\ 29 & \text{Erweiterte ID} \end{cases} \tag{3.7}$$

Die Anzahl der Adressbits ist Bus-abhängig (in unterschiedlichen Teilnetzen kann für dieselbe PDU eine andere Adresse verwendet werden).

Definition 3.56 (Stuffingfaktor). *Sei b der Faktor, um den die Anzahl Bits durch das Bitstuffing ansteigt.*

CAN und CAN-FD verwenden die Non-Return-to-Zero (NRZ) Kodierungsmethode, um den maximalen Abstand zwischen zwei Signalflanken zu begrenzen und so eine Bitsynchronisierung zwischen den Steuergeräten zu ermöglichen. Nach fünf aufeinanderfolgenden Bits desselben Pegels wird ein Stuffingbit mit invertiertem Pegel eingefügt [Int24a]. Abhängig von den zu übertragenden Daten werden im schlechtesten Fall nach jedem fünften Bit ein zusätzliches Bit eingefügt. Im besten Fall werden keine Bits eingefügt.

Mittels Heuristiken kann der Stuffingfaktor geschätzt und durch geeignete Transformation der Nutzdaten[1] die Auftretenswahrscheinlichkeit von Stuffingbits verringert werden [Nol01][Ala20].

Definition 3.57 (Zu übertragende Bits)**.** *Sei $n_{i,j,k}$ die Anzahl der für PDU Nr. j auf Bus Nr. k in Release Nr. i zu übertragenden Bits.*

Definition 3.58 (In erhöhter Geschwindigkeit zu übertragende Bits)**.** *Sei $n'_{i,j,k}$ die Anzahl der für PDU Nr. j auf Bus Nr. k in Release Nr. i in erhöhter Geschwindigkeit zu übertragenden Bits.*

Die Berechnung der Anzahl, der für eine PDU zu übertragenden Bits, erfolgt gemäß ISO-11989 [Int24a]. Es wird zwischen sechs Fällen unterschieden. Bei CAN-FD muss die Gesamtanzahl in Bits, die in der Grundgeschwindigkeit zu übertragen sind, sowie Bits, die in der erhöhten Geschwindigkeit zu übertragen sind, unterschieden werden.

[1] Z. B. kann auf jedes Byte die XOR-Funktion mit 55_{Hex} oder 01010101_{Bin} angewandt, d. h. jedes zweite Bit invertiert werden. Dadurch sinkt die Wahrscheinlichkeit Stuffingbits einfügen zu müssen, in Szenarien in denen kleine Ganzzahlen (<= 7) oder Wahrheitswerte jew. als eigenes Byte übertragen werden.

A) CAN mit Basis ID

B) CAN mit erweiterter ID

C) CAN-FD mit Basis ID und bis zu 16 Datenbytes

D) CAN-FD mit Basis ID und mehr als 16 Datenbytes

E) CAN-FD mit erweiterter ID und bis zu 16 Datenbytes

F) CAN-FD mit erweiterter ID und mehr als 16 Datenbytes

$$n_{i,j,k} = \begin{cases} (34 + d_{i,j}) \cdot b + 13 & \text{Fall A} \\ (54 + d_{i,j}) \cdot b + 13 & \text{Fall B} \\ 16 \cdot b + 13 & \text{Fall C \& D} \\ 35 \cdot b + 13 & \text{Fall E \& F} \end{cases} \tag{3.8}$$

$$n'_{i,j,k} = \begin{cases} 0 & \text{Fall A \& B} \\ (6 + d_{i,j}) \cdot b + 28 & \text{Fall C \& E} \\ (6 + d_{i,j}) \cdot b + 30 & \text{Fall D \& F} \end{cases} \tag{3.9}$$

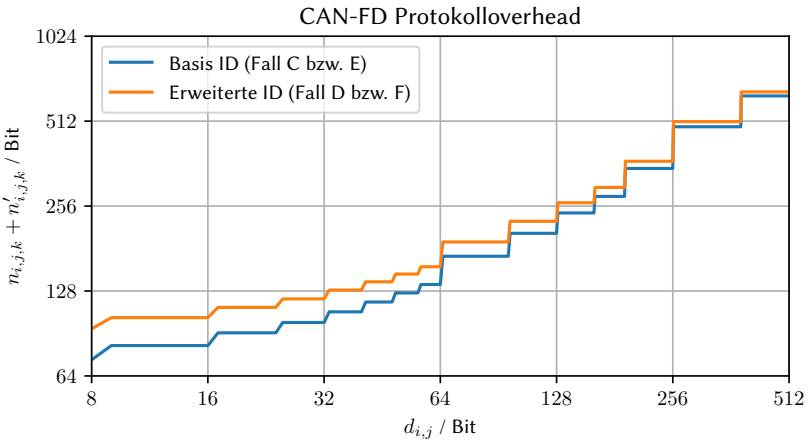

Abbildung 3.9: Die Anzahl der zu sendenden Bits ist größer als die der Nutzdaten $d_{i,j}$.

Fasst man die Gleichungen 3.8 u. 3.9 zusammen, zur Gesamtanzahl der in einem CAN-FD-Frame zu übertragenden Bits, entsteht eine Funktion mit mehreren Stufen (vgl. Abb. 3.9). Diese kommen durch die Limitierung der erlaubten Frame-Größen bei CAN-FD, sowie durch die unterschiedlichen CRC-Funktionen zustande.

Definition 3.59 (Baudrate). *$R_{i,j}$ sei die Baudrate des Busses Nr. j im Release Nr. i und $R'_{i,j}$ die erhöhte Baudrate auf CAN-FD Bussen.*

Definition 3.60 (Übertragungsdauer). *$t_{i,j,k}$ sei die Zeit, die es dauert, die PDU Nr. j in Release Nr. i über den Bus Nr. k zu übertragen.*

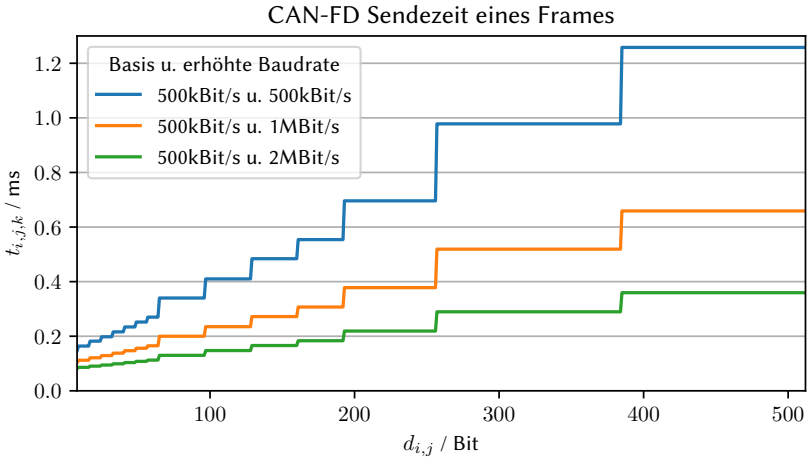

Abbildung 3.10: Benötigte Zeit, um die Nutzdaten $d_{i,j}$ über den Bus $B_{i,k}$ zu übertragen.

Die Übertragungsdauer ergibt sich aus der Anzahl der zu sendenden Bits und der Baudrate des Busses. Bei CAN-FD muss zwischen den Bits unterschieden werden, die in der Grundgeschwindigkeit übertragen werden, und denen, für die die erhöhte Baudrate angewendet wird. Höhere Baudraten für die Übertragung der Nutzdaten, bei gleicher Grundbaudrate, führen vorwiegend bei

großen Nutzdatenmengen zu einer deutlichen Verkürzung der Sendezeit (vgl. Abb. 3.10).

$$t_{i,j,k} = \frac{n_{i,j,k}}{R_{i,k}} + \frac{n'_{i,j,k}}{R'_{i,k}} \tag{3.10}$$

Definition 3.61 (Lastanteil). *$L_{i,j,k}$ sei der absolute Anteil der PDU Nr. j an der Gesamtauslastung des Busses Nr. k in Release Nr. i.*

$$L_{i,j,k} = \frac{t_{i,j,k}}{z_{i,j}} \tag{3.11}$$

Definition 3.62 (Buslast). *$L_{i,j}$ sei die Auslastung des Busses Nr. j in Release Nr. i.*

$$L_{i,j} = \sum_{\forall P_{i,k} \in B_{i,j}} L_{i,k,j} \tag{3.12}$$

4 Hierarchische Versionierung[1]

4.1 Konflikt zwischen Schnelllebigkeit und Beständigkeit

Die erste Forschungsfrage in Abschnitt 1.2 zielt auf den (scheinbaren) Widerspruch ab, der sich aus dem Verblocken von Steuergeräten und der Vermeidung von Softwareänderungen in zertifizierungsrelevanten (vgl. Abs. 2.6.1) Komponenten einerseits (im Folgenden: „beständige ECUs“ oder „beständige Komponenten“), und der Forderung nach schnellen Innovationszyklen andererseits (im Folgenden: „schnelllebige ECUs“ oder „schnelllebige Komponenten“) ergibt. Für diesen Konflikt existieren zwei Lösungsansätze:

4.1.1 Einheitliche Entwicklungszyklen

Die Kompromisslösung besteht in der Definition eines einheitlichen Entwicklungszyklus. Die schnelllebigen Steuergeräte müssen für die Dauer des Zyklus ihre Schnittstellen unverändert beibehalten, während bei den beständigeren ECUs mehr Softwareänderungen erfolgen, als es im Rahmen der jeweiligen Projekte erforderlich wäre.

Der historischen Entwicklung folgend, war es zunächst am naheliegendsten, die Schnittstellen der überschaubaren Anzahl Steuergeräte auf Basis einer zentralen K-Matrix – und einheitlichen, konsistenten K-Matrix Ständen (vgl. Abschnitt 2.5) – zu entwickeln. In Form des Release-konformen Flashens (vgl. Abschnitt 3.5.4) stellt dies den Stand der Technik dar. Die einheitlichen Aktualisierungszeitpunkte für alle Steuergeräte dienen als Synchronisierung für die Entwicklung der Fahrzeugsoftware. Inzwischen

[1] Teile, des in diesem Kapitel vorgestellten Konzepts, wurden durch den Autor bereits im Dezember 2020, im Rahmen der 27th International Conference on Systems Engineering, vorgestellt [Vet21b].

ist jedoch nicht mehr zu rechtfertigen, dass eine Motorsteuerung mit der gleichen Häufigkeit aktualisiert werden muss, wie Unterhaltungselektronik, oder die Entwicklung von autonomen Fahrfunktionen durch elektrische Fensterheber ausgebremst wird.

4.1.2 Unterschiedliche Entwicklungszyklen

Die Alternative zu fahrzeugweit einheitlichen Änderungszeitpunkten für die Kommunikationsschnittstellen – und somit der Ansatz zur Beantwortung der Forschungsfrage **FF1** (vgl. Abs. 1.2) – sind unterschiedliche, an die jeweiligen Anforderungen angepasste Entwicklungszyklen.

Bei der Unterteilung des Fahrzeugs kommen Steuergeräte, Busse oder Systeme (vgl. Abschnitt 2.3) infrage. Eine Unterteilung gemäß der Busse entspräche einer Gruppierung von ECUs, auf Basis derer Positionen im Topologiebild (vgl. Abs. 2.4.1). Die Sinnhaftigkeit hängt also davon ab, ob die Topologie auf räumlicher oder funktionaler Nähe der Steuergeräte basiert. Eine Unterteilung in Systeme bietet die Trennung in die verschiedenen Domänen, die praktische Umsetzung wird aber dort eingeschränkt, wo die Softwarekomponenten verschiedener Systeme auf einer gemeinsamen ECU integriert werden (vgl. Abschnitt 2.6.4). In der vorliegenden Untersuchung wird das Gesamtfahrzeug in einzelne ECUs unterteilt. Auf einer Teilung in einzelne Steuergeräte aufbauend, kann später, unter Berücksichtigung von Topologie und Systempartitionierung, eine Gruppierung mehrerer ECUs zu Bussen oder Systemen erfolgen. Die Unterteilung in einzelne Steuergeräte entspricht außerdem der organisatorischen Struktur, in der die Fahrzeugkomponenten entwickelt und produziert werden (vgl. Abschnitte 2.1.3 u. 3.2.4).

Werden ECUs in unterschiedlichen Zyklen entwickelt, bedeutet dies für das Gesamtprojekt, dass Verifikations- und Validierungsschritte mit Steuergeräten durchgeführt werden, die auf unterschiedlichen Software- und K-Matrix-Releases basieren (vgl. Abschnitt 3.5.4). Um sinnvolle Cluster-HIL Tests und den Aufbau von Erprobungsträgern (vgl. Abschnitt 3.2.5) zu ermöglichen, müssen die ECUs vollständig oder zumindest eingeschränkt

kompatibel (vgl. Definitionen 3.24 u. 3.25) zum Fahrzeug respektive dem zu testenden System sein.

4.2 Kompatibilität eines Steuergeräts zu einem Fahrzeug oder einem System

Der Fokus dieser Arbeit liegt auf der Kompatibilität der Anwendungsschicht (gemäß TCP/IP-Modell) bzw. den OSI-Schichten 5 – 7. Kompatibilität der unteren Schichten wird vorausgesetzt (vgl. Abschnitt 3.4.1). Zuoberst, in der 7. OSI Schicht, liegen die Anwendungen. Sowohl bei AUTOSAR Classic, als auch bei AUTOSAR Adaptive, bestehen diese aus ineinander verschachtelten SWC Compositions. Abschnitt 3.4.2 definiert Kompatibilität für SWC Compositions und einen formellen Schnittstellenbegriff. Die Definitionen basieren jeweils auf den Mengen von Anforderungen und Garantien von Signalen bzw. Services, wobei die Menge aller Garantien und Anforderungen, die Schnittstelle eines Steuergeräts darstellen.

Die darunter liegenden OSI Schichten fünf und sechs werden in AUTOSAR Classic durch die Interaktionsschicht abgebildet, in AUTOSAR Adaptive durch die SOME/IP Middleware (vgl. Tab. 3.2). Diese definieren, wie die Signale und Services auf den zugrunde liegenden Bussen bzw. dem TCP/IP Stack dargestellt werden. Sowohl die Interaktionsschicht als auch SOME/IP bieten die Möglichkeit nachträglicher, kompatibler Erweiterungen.

4.2.1 Kompatibilitätsmetrik

Ob ein Steuergerät vollständig oder eingeschränkt kompatibel zu einem System oder Fahrzeug ist, hängt davon ab, ob alle erforderlichen Signale bzw. Services verfügbar sind. Dies gilt sowohl für die vom Steuergerät, als auch für die vom System bzw. Fahrzeug geforderten Informationsübertragungen. Im Falle vollständiger Kompatibilität sind alle Systemfunktionen vorhanden – bei eingeschränkter Kompatibilität kann zumindest der Teil der Funktionen,

deren Anforderungen erfüllt sind, mittels Cluster-HIL und Erprobungsfahrten getestet werden.

Aus dem Verhältnis aller Funktionen, und den tatsächlich verfügbaren Funktionen ergibt sich der folgende Wertebereich:

- Vollständig kompatibel
- Eingeschränkt kompatibel
  - Obere Grenze: Alle, bis auf eine einzelne Funktion, sind vorhanden.
  - Untere Grenze: Nur eine einzelne Funktion ist vorhanden.
- Vollständig inkompatibel / scheinkompatibel

Inkompatibilität und Scheinkompatibilität werden dabei auf eine Stufe gestellt, da beide eine Erprobung unmöglich machen.

Theoretisch erfolgt die Bewertung, indem die von einer Version eines Steuergeräts geforderten und gebotenen Garantien (d. h. dessen Interface) mit dem Interface des Fahrzeugs abgeglichen wird. Diese Metrik bezieht sich auf die Funktionsfähigkeit des (Gesamt-)Systems. Es ist daher nicht sinnvoll der Kompatibilität eine Richtung zuzuschreiben[1].

Zur Beantwortung der zweiten Forschungsfrage **FF2** (vgl. Abs. 1.2) stellt diese theoretische Metrik einen Ansatz dar. Die manuelle Ermittlung, vorhandener und fehlender Systemfunktionen, ist in Anbetracht der Menge an Steuergeräten und Signalen (vgl. Abschnitt 3.2.3) jedoch nicht praktikabel. Für eine „alltagstaugliche“ Bewertung sowie eine Optimierung der Kompatibilität sind weitere Maßnahmen erforderlich.

[1] D. h. es ist nicht sinnvoll zwischen „Steuergerät ist kompatibel zu Fahrzeug“ und „Fahrzeug ist kompatibel zu Steuergerät“ zu unterscheiden. Für die Kompatibilitätsfunktion (vgl. Def. 3.23) gilt $K(C_i, C_j) \neq K(C_j, C_i)$ für $i \neq j$, d. h. die Richtung ist relevant. Das liegt daran, dass hier die Anforderungen der einzelnen SWCs SWC_i und SWC_j betrachtet werden. Ob der gesamte oder nur ein eingeschränkter Funktionsumfang zur Verfügung steht, ist dagegen eine Eigenschaft des Systems, das aus diesen SWCs besteht.

In einem iterativen Entwicklungsprozess, und speziell in Szenarien, in denen verschiedene Komponenten unterschiedlich häufig geändert werden, stellt sich die Frage, wie die Änderung eines Elements dessen Kompatibilität zum Rest des Systems beeinflusst. In der Softwareindustrie werden bei semantischer Versionierung Änderungen, je nach deren Kompatibilität, in die Kategorien „**Fehlerkorrektur**“, „Geringfügige Änderung“ und „**Inkompatible Änderung**“ unterteilt (vgl. Def. 3.36 – 3.38). Im Rahmen dieser Arbeit wird die Kategorie „Geringfügige Änderung“ als „**Kompatible Änderung**“ bezeichnet, um zu unterstreichen, dass durch die jeweiligen Änderungen ausschließlich neue Funktionen hinzukommen, jedoch keine alten wegfallen.

4.2.2 ECU-Interface als Schlüsselstelle

Das in Abschnitt 4.1 formulierte Ziel unterschiedlicher Entwicklungszyklen, führt zu Erprobungsfahrzeugen bzw. Cluster-HIL Aufbauten, mit Steuergeräten, die auf verschiedenen K-Matrix-Releases basieren. Eine Schnittstelle kann aus mehreren Services/PDUs bestehen, deren Sender/Empfänger in unterschiedlichen ECUs liegen können (vgl. Definition 3.31). Befindet sich ein Teil der Services/PDUs auf den neueren ECUs und ein anderer Teil auf den älteren ECUs, kann dem Interface des Fahrzeugs keine eindeutige Version zugewiesen werden. Die Version des Interface eines Steuergeräts zu einem Fahrzeug basiert auf der Version dieses einen Steuergeräts, und ist somit eindeutig. Abbildung 4.1 illustriert das, anhand der in Abschnitt 2.2 vorgestellten Beispielanwendung und deren System „Spiegelheizung“ (vgl. Abs. 2.3.4). ECU-seitig ist die eindeutige Versionsnummer des Türsteuergerätes, Fahrzeug-seitig basiert die Schnittstelle sowohl auf dem Temperatursensor als auch dem linken Außenspiegel, welche jeweils eine eigene, möglicherweise voneinander abweichende Version aufweisen.

Um die Kompatibilität eines Steuergeräts zu einem Versuchsaufbau (Erprobungsträger oder Cluster-HIL) bewerten zu können, wird jedem Aufbau ein K-Matrix-Release zugeordnet. Die Interface-Version des Versuchsaufbaus wird

definiert, als die Version, die das ECU-Interface in dem gewählten K-Matrix-Release aufweist.

Die Menge der Anforderungen und Garantien, die einer Schnittstelle (vgl. Definition 3.31) entsprechen, werden in der K-Matrix vollständig beschrieben (vgl. Abschnitt 2.5) und technisch durch dem AUTOSAR COM-Stack (vgl. Abschnitt 2.7.3) realisiert. In der Praxis wird dazu der ECU-Extract, d. h. der das Steuergerät betreffende Teil der K-Matrix, in Form einer ARXML-Datei (vgl. Abschnitt 2.7.5) als Eingabe für einen Code-Generator verwendet, der einen Steuergeräte-spezifischen COM-Stack erzeugt(vgl. Abschnitt 2.7.6).

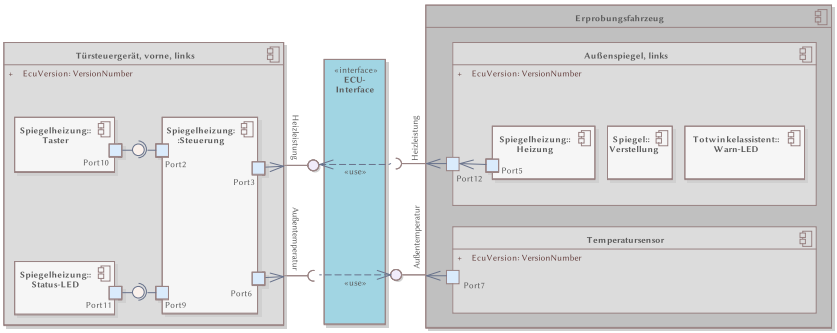

Abbildung 4.1: Auf der Seite des Türsteuergeräts gibt es eine eindeutige `EcuVersion`. Auf der Seite des Erprobungsfahrzeuges basiert das Interface auf den zwei `EcuVersion`en des linken Außenspiegels und des Temperatursensors. Falls der Außenspiegel und der Temperatursensor in unterschiedlichen Zyklen entwickelt werden, können sich die Versionen unterscheiden.

Die eindeutige Versionierbarkeit des ECU-Interfaces, sowie dessen Auswirkung auf die Kompatibilität zwischen Fahrzeug und Steuergerät, machen es zur Schlüsselstelle, an der die Entwicklung eines Prozesses ansetzen muss: Hier treffen die unterschiedlichen Entwicklungsgeschwindigkeiten und die resultierenden Softwarestände aufeinander. Die organisatorische Struktur, in der verschiedene ECUs von unterschiedlichen Zulieferern entwickelt werden (vgl. Abschnitt 2.1.3), spricht auch für eine Fokussierung auf die Steuergeräteschnittstellen. Ein möglicher Nachteil dieser Perspektive ist, dass die jeweiligen Systeme (z. B. „Spiegelheizung“) in den Hintergrund geraten.

4.3 Anforderungen

Die dritte Forschungsfrage **FF3** (vgl. Abs. 1.2) fordert einen einheitlichen Prozess sowohl für Signal-basierte als auch Service-orientierte Netzwerke. Dieser bietet idealerweise auch die Möglichkeit ECUs in unterschiedlichen Zyklen zu entwickeln und eine „alltagstaugliche“ Methodik zur Ermittlung der Kompatibilität zwischen verschiedenen Komponenten.

Aus den vorangegangenen Abschnitten ergeben sich die folgenden Anforderungen an einen solchen Prozess:

A1 Die Entwicklung von Steuergeräten, insbesondere die Änderung deren Schnittstellen, in unterschiedlichen Zyklen muss möglich sein.

A2 Die Kompatibilität einer ECU, basierend auf einem K-Matrix-Release, zu einem Fahrzeug bzw. Cluster-HIL Aufbau, basierend auf einem oder mehreren anderen K-Matrix-Releases, muss unmittelbar erkennbar sein.

A3 Die Kompatibilität von Steuergeräten auf Basis unterschiedlicher K-Matrix-Releases soll optimiert werden.

A4 Es darf nicht zu einer Erhöhung der Fertigungs- und Materialkosten (FMK) pro Fahrzeug kommen.

A5 Sowohl Signal-basierte als auch Service-orientierte Kommunikation müssen unterstützt werden.

A6 Die Verfügbarkeit von Schnittstellen muss planbar sein.

Anforderung **A1** ergibt sich unmittelbar aus der ersten Forschungsfrage **FF1** und dem in Abschnitt 4.1.2 vorgestellten Ansatz zu deren Lösung. **A2** bezieht sich einerseits auf **FF2** und bildet gleichzeitig die Grundlage für **A3**, welche die Forderung nach Optimierung von Kompatibilität in **FF3** abbildet.

Die in Abschnitt 2.1.2 vorgestellten wirtschaftlichen Rahmenbedingungen finden sich in **A4** wieder. In Anbetracht der hohen Stückzahlen bei Serienfahrzeugen, muss ein Entwicklungsprozess entweder neutral gegenüber den

Stückkosten sein, oder die Entwicklungskosten so stark reduzieren, dass die erhöhten Produktionskosten ausgeglichen werden.

A5 sorgt dafür, dass der zu entwickelnde Prozess zukunftssicher ist. Die in Abschnitt 2.1.3 beschriebene Organisationsstruktur, einzelne Steuergeräte bei unterschiedlichen Zulieferern entwickeln zu lassen, macht eine verlässliche Planbarkeit der Steuergeräteschnittstellen erforderlich und begründet **A6**.

4.4 Neues Konzept

4.4.1 Hierarchische Struktur

Änderungen an der Anwendungssoftware (d. h. in der AUTOSAR-Anwendungsschicht respektive OSI-Schicht 7, vgl. Abb. 2.13 und Tabelle 3.2), die für die Kommunikation relevant sind, verändern, löschen oder erstellen neue `SystemSignal`s. Diese werden in AUTOSAR Classic durch `iSignal`s, welche in `iPDU`s gruppiert werden, in der Interaktionsebene (OSI Layer 5&6) realisiert. Die Menge aller zu sendenden und empfangenden PDUs eines Steuergeräts kann als ECU-Interface zusammengefasst werden.

In AUTOSAR Adaptive wirken sich Änderungen in der Anwendungsebene (OSI-Layer 7) auf die Kommunikation aus, indem Serviceinterfaces verändert, gelöscht oder erstellt werden. In der SOME/IP Middleware (OSI Layer 5&6) bestehen diese aus Methoden, welche wiederum aus Parametern und Rückgabewerten bestehen, sowie aus Eventgruppen, die aus unterschiedlichen Events bestehen. Die Steuergeräteschnittstelle ist die Menge aller angebotenen und konsumierten Services.

Beiden Welten gemein ist, die hierarchische Struktur, in der zu übertragende Einzelinformationen zu einem ECU-Interface zusammengefasst werden können (s. Abb. 4.2). Diese Hierarchie weist zwei Eigenschaften auf, die in den folgenden zwei Abschnitten vorgestellt werden.

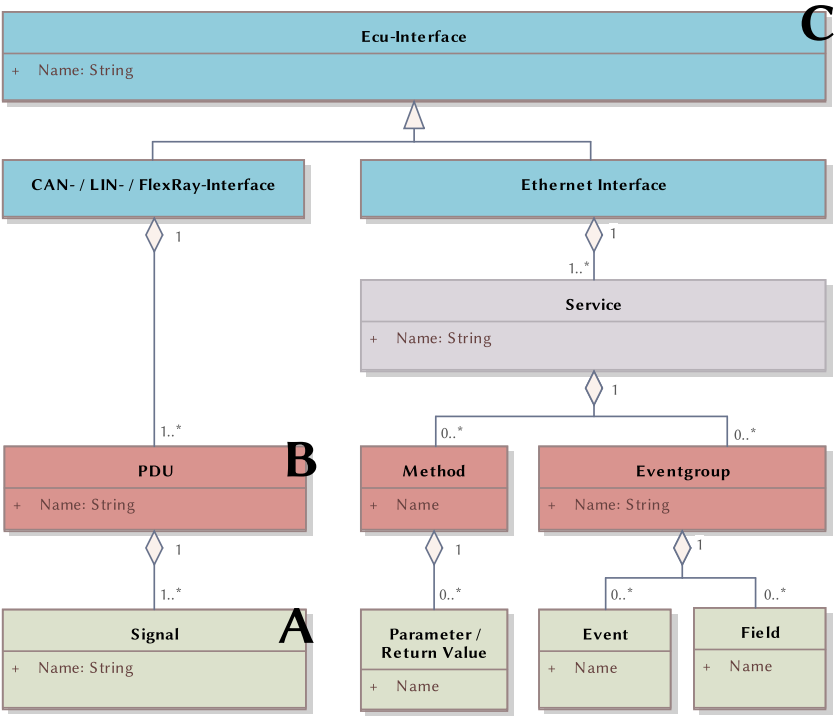

Abbildung 4.2: Hierarchie innerhalb der Interaktionsschicht (AUTOSAR Classic, links) respektive der SOME/IP Middleware (AUTOSAR Adaptive, rechts).

4.4.2 Ebenen übergreifende Kompatibilität bei Änderungen

Abschnitt 3.4.3 erläutert, wie auf unterschiedlichen Ebenen dieser Hierarchie Änderungen vorgenommen werden können, die für die übergeordnete Ebene (eingeschränkt) kompatibel sind. Im Folgenden werden am Beispiel von AUTOSAR Classic (d. h. die Hierarchie des CAN-Interface in Abbildung 4.2) für jede Hierarchieebene Änderungen gezeigt, und den drei Kategorien „**Fehlerkorrektur**“, „**Kompatible Änderung**“ und „**Inkompatible Änderung**“ (vgl. Abschnitt 4.2.1) zugeordnet.

Signal (‚A' in Abb. 4.2)

- **Fehlerkorrektur:** In diese Kategorie fallen Änderungen, welche die Schnittstelle nicht beeinflussen. Wird eine Anwendungssoftware, die zuvor falsche Werte erzeugt hatte, korrigiert, sodass über dieselbe Schnittstelle (d. h. ein Signal mit selbem Anwendungsdatentyp, vgl. Abs. 2.7.4), korrekte Werte übertragen werden, fällt die Änderung in diese Kategorie. Änderungen an den Metadaten der K-Matrix Datenbank, beispielsweise an Beschreibungstexten, können ebenfalls in diese Kategorie fallen. Wird z. B. bei einem Signal die Beschreibung von „Horizontaler Winke**k** des linken Außenspiegels" in „Horizontaler Winke**l** des linken Außenspiegels" geändert, ist dies eine vollständig kompatible Fehlerkorrektur. Voraussetzung für die Kompatibilität einer Metadatenänderung ist, dass dem Signal durch die Änderung keine neue Bedeutung zugewiesen wird (s. u.).
- **Kompatible Änderung:** Auf Signalebene sind die Änderungen kompatibel, bei denen sich der Anwendungsdatentyp in einer Form ändert, sodass zuvor undefinierte Werte definiert werden (vgl. Abs. 2.7.4).
- **Inkompatible Änderung:** In diese Kategorie fallen Änderungen des Anwendungsdatentyps, bei denen bereits definierte Werte umdefiniert werden (z. B. Änderung von Faktor oder Offset, mit denen der physikalische Wert aus dem übertragenen Wert berechnet wird). Änderungen, welche die Signallänge (Anzahl Bits) beeinflussen, sind ebenfalls inkompatibel.

  Hinzu kommen Metadatenänderungen, die die Bedeutung eines Signals verändern. Diese sind scheinkompatibel (vgl. Def. 3.27), da sich die Schnittstelle nicht verändert. Durch die geänderte Bedeutung kommt es zu unterschiedlichen Interpretationen bei Sender und Empfänger, und dadurch zu einem fehlerhaften Systemverhalten. Ein Beispiel hierfür wäre die Änderung von „**Horizontaler** Winkel des linken Außenspiegels" zu „**Vertikaler** Winkel des linken Außenspiegels".

PDU (‚B' in Abb. 4.2)

- **Fehlerkorrektur:** Änderungen, die die Schnittstelle nicht beeinflussen, sind auch auf der PDU-Ebene Korrekturen der Metadaten. Z. B. Veränderungen an der Beschreibung, die deren Bedeutung nicht umkehren.

  Wird an einem, in der PDU enthaltenen, Signal eine Fehlerkorrektur durchgeführt, entspricht dies der (geringfügigen) Änderung eines der Attribute der PDU.

- **Richtungsabhängige Fälle:** Auf der PDU-Ebene ist zur Entscheidung, ob eine Änderung kompatibel ist, in vielen Fälle von Bedeutung, ob die Änderung auf der Seite des Senders oder des Empfängers erfolgt.

  Ein Beispiel dafür ist das Hinzufügen bzw. Entfernen von Signalen aus einer PDU. Wird auf der Seite des Senders ein zusätzliches Signal hinzugefügt, so ist diese Änderung kompatibel (sofern nicht die Reihenfolge der bestehenden Signale verändert wird). Empfänger, die das neue Signal nicht kennen, können es ignorieren. Wird dagegen ausschließlich auf der Seite des Empfängers ein neues Signal hinzugefügt, kommt es zu Fehlern: Die empfangene PDU wird als zu kurz verworfen, oder vorhandene Paddingbits werden fälschlich als Signal interpretiert, sodass es zu einem Fehlverhalten des Systems kommt. Hinzufügen von Signalen auf der Seite des Senders *und* einem oder mehreren Empfängern ist ebenfalls kompatibel. In diesem Fall muss gewährleistet sein, dass der neue Empfänger nur in Kombination mit dem neuen Sender eingesetzt wird.

  Umgekehrt ist es beim Entfernen eines Signals (ohne die Reihenfolge der verbleibenden Signale zu verändern): Ignoriert ein Empfänger nach einer Änderung ein zuvor genutztes Signal, führt dies nicht zu Kommunikationsfehlern. Entfernt ein Sender ein Signal, verlieren alle Empfänger, die zuvor dieses Signal verwendet hatten, ihre Funktion.

  Wird der Fokus von einzelnen, zu sendenden oder zu empfangenden PDUs auf das Gesamtsystem mit allen PDUs verschoben, gilt folgende

Vereinfachung: Das Entfernen eines Signals aus einer PDU ist nur bei zu empfangenden PDUs kompatibel. Um die Rückwärtskompatibilität zu gewährleisten, muss mindestens im sendenden Steuergerät das Signal Teil der PDU bleiben. Aus der Gesamtsystemperspektive ist es daher nicht kompatibel, Signale zu entfernen. Zu einem Sender (und damit zum Gesamtsystem) ein Signal hinzuzufügen, ist kompatibel.

Ein Metadatum mit vergleichbarer Eigenschaft ist das Zeitverhalten von periodischen Signalen und PDUs[1]: Sender seitig ist eine Frequenzerhöhung kompatibel (ggf. mit gleichzeitiger Erhöhung der Frequenz bei einzelnen Empfängern), Empfänger seitig eine Reduktion der Frequenz.

- **Kompatible Änderung:** Für das Gesamtsystem sind Änderungen an PDUs kompatibel, bei denen Signale hinzugefügt werden oder die Sendefrequenz erhöht wird.

  Hinzu kommen kompatible Änderungen an Signalen gemäß Abschnitt 4.4.2: PDUs, die kompatibel geänderte Signale enthalten, gelten selbst als kompatibel geändert.

- **Inkompatible Änderung:** Inkompatibel für das Gesamtsystem sind Änderungen, bei denen Signale entfernt, oder die Sendefrequenz verringert wird.

  Änderungen an der Signalreihenfolge oder Metadaten wie der Bytereihenfolge[2] oder der PDU-ID sind inkompatibel, da sie zu Fehlinterpretationen auf der Empfängerseite führen.

  Wird in einer PDU ein Signal inkompatibel verändert, gilt auch die PDU als inkompatibel geändert.

[1] Bei PDUs, die periodisch zu sendende Signale enthalten, entspricht die Sendefrequenz der, des Signals mit der kürzesten Periodendauer.

[2] Fahrzeugnetzwerke sind verteilte, eingebettete Systeme, die aus sehr unterschiedlichen Mikrocontrollern und Mikroprozessoren bestehen (vgl. Abs. 2.3.1). Um Interpretationsfehler zu vermeiden, muss explizit festgelegt werden, ob bei aus mehreren Bytes bestehenden Zahlenwerten, zuerst das kleinstwertige Byte („little-endian“ oder „Intel-Format“) oder das höchstwertige Byte („big-endian“ oder „Motorola-Format“) übertragen wird.

Interface (‚C' in Abb. 4.2)

- **Fehlerkorrektur:** Ähnlich wie bei Signalen und PDUs belaufen sich auch auf der Interface-Ebene die geringfügigen Änderungen auf Korrekturen an den Metadaten, die sich nicht auf die Bedeutung der hinterlegten Informationen auswirken.

  Wird an einer PDU, die über ein Interface übertragen wird, eine Fehlerkorrektur durchgeführt, gilt auch das Interface als geringfügig verändert. Da eine Fehlerkorrektur an einem Signal zu einer als geringfügig verändert bewerteten PDU führt, zählt dies ebenfalls als Fehlerkorrektur am Interface.

- **Richtungsabhängige Fälle:** Analog zum Hinzufügen und Entfernen von Signalen in PDUs, gibt es eine Richtungsabhängigkeit beim Hinzufügen und Entfernen von PDUs an Interfaces: Werden neue, zu sendende PDUs (mit neuer ID) hinzugefügt, ist dies rückwärtskompatibel, da Empfänger, die die neue PDU nicht kennen, diese ignorieren. Das Hinzufügen neuer zu empfangender PDUs ist nur kompatibel, wenn zeitgleich ein Sender hinzugefügt wird, oder das System auch bei Ausbleiben der jeweiligen PDU sinnvoll funktioniert (d. h. wenn die PDU optional ist). Das Entfernen von PDUs ist auf der Seite des Empfängers kompatibel, während es auf der Seite des Senders dazu führt, dass auf dieser PDU basierende Funktionen in anderen Steuergeräten ausfallen.

- **Kompatible Änderung:** Aus der Perspektive des Gesamtsystems ist das Hinzukommen neuer PDUs (mit neuen IDs) zu einem Interface kompatibel. Diese neuen PDUs können aus neuen Signalen, aber auch ganz oder teilweise aus existierenden Signalen anderer PDUs bestehen.

  Wurde eine PDU eines Interfaces kompatibel geändert, so wird auch das Interface als kompatibel geändert betrachtet. Auf diesem Wege wirken sich auch kompatible Änderungen an Signalen auf die Bewertung der Interfaces aus.

- **Inkompatible Änderung:** Für das Gesamtsystem inkompatibel sind Änderungen, bei denen PDUs wegfallen.

  Überträgt ein Interface eine PDU, die inkompatibel geändert wurde, entspricht das einer inkompatiblen Änderung des Interfaces.

  Bustechnologie und Übertragungsrate sind ebenfalls dem Interface zuzuordnende Eigenschaften, deren Änderung das Interface inkompatibel machen würde. Dabei handelt es sich aber um Änderungen an den OSI-Schichten 1 – 4, deren Kompatibilität in dieser Arbeit vorausgesetzt werden (d. h. derartige Änderungen werden ausgeschlossen).

Die Bewertung der Interface-Ebene entscheidet, ob ein Steuergerät in einem Erprobungsträger oder einem Cluster-HIL Aufbau eingesetzt werden kann. Anforderung **A2** ist durch eine Auswertung der Kompatibilität der Interface-Ebene zu erfüllen, während **A3** die Optimierung auf dieser Ebene fordert. Zunächst scheint es dabei so, als würden sich alle Signal- und PDU-Änderungen, insbesondere die inkompatiblen, auf die Schnittstelle auswirken.

Eine vergleichbare Auflistung ließe sich auch für die Hierarchie des Ethernet-Interfaces, und somit für Service-orientierte Kommunikation, aufstellen.

4.4.3 Änderungen können „kompatibel gemacht“ werden

Wenn eine einzige, inkompatible Signaländerung die Kompatibilität eines gesamten Interfaces negiert, sind individuelle Entwicklungszyklen nicht sinnvoll umsetzbar. Auf allen Ebenen bestehen Möglichkeiten, die Elemente rückwärtskompatibel zu erweitern. Daraus lässt sich ein Mechanismus konstruieren, der aus einer inkompatiblen Änderung eine kompatible Änderung macht:

Definition 4.1 (Änderungskopie (Copychange))**.** *Wird von einem zu ändernden Element eine Kopie angelegt, und die Veränderungen an der Kopie vorgenommen, während das Original unverändert bleibt, heißt dieser Schritt „Änderungskopie“ oder engl.: „Copychange“.*

Der Nachteil dieses Mechanismus ist, dass durch jede Kopie eines Elements, die zu übertragende Bandbreite steigt. Dies kann zu einem Konflikt mit Anforderung **A4** führen. Änderungskopien sind nicht immer auf allen Hierarchieebenen möglich. So ergibt sich z. B. durch CAN bzw. CAN-FD eine Begrenzung von maximal 8 Bytes oder 64 Bytes pro PDU. Sind diese Bytes ausgeschöpft, können keine weiteren Signale an das Ende des Frames angehängt werden. In einem solchen Fall kann das Interface trotzdem kompatibel gehalten werden, in dem die Änderungskopie auf PDU-Ebene durchgeführt wird (vgl. Abschnitt 4.5.4).

4.4.4 Versionierung

Die Hierarchieelemente können auf ihren jeweiligen Ebenen unabhängig voneinander verändert werden, und unterschiedliche Änderungen beeinflussen die Kompatibilität in anderem Ausmaß. Ein hierfür geeignetes Versionierungsschema, ist die in Abschnitt 3.5.3 beschriebene, semantische Versionierung. (rückwärts-)kompatible Änderungen erhöhen die Minor-Version, inkompatible Änderungen, oder solche, die die Kompatibilität reduzieren, erhöhen die Major-Version.

Eine Verwendung der Patch-Version ist für die reine Netzwerkentwicklung nicht sinnvoll, da diese nur angepasst wird, wenn eine Fehlerkorrektur, aber keine Änderung der Schnittstelle erfolgt. Sobald sich eine Schnittstelle ändert, wird mindestens die Minor-Version erhöht. Für die Steuergeräteentwicklung, kann es dennoch sinnvoll sein, Patch-Versionen anzugeben. Auf einem K-Matrix-Stand aufbauend, kann mehr als ein Softwarerelease erfolgen, sodass Fehlerkorrekturen ohne Schnittstellenänderungen vorkommen.

4.4.5 Regeln zur Änderung

Auf den Eigenschaften aufbauend, die in den Abschnitten 4.4.1 – 4.4.4 dargestellt wurden, wird im Folgenden ein Regelset entwickelt. Dieses stellt einen

Vorschlag für einen Prozess dar, der die Anforderungen aus Abschnitt 4.3 erfüllt. Zentrales Element dieses Prozesses ist – wie in der automobilen Kommunikationsentwicklung allgemein – die K-Matrix sowie die Regeln, nach denen diese verändert wird. Bezüglich der bereits in die Entwicklungspraxis übernommenen Teile siehe Abschnitt 7.2.

Zeitpunkte

Um Anforderung **A1** entsprechend die Entwicklung in unterschiedlichen Zyklen zu ermöglichen, wird der etablierte einheitliche Release-Zyklus aufgeteilt in einen „Hauptzyklus“ und einen oder mehrere „Zwischenzyklen“.

Regel 1. *Es gibt einen gemeinsamen Hauptzyklus mit einheitlichen Release-Zeitpunkten („Haupt-Releases“) für alle Steuergeräte. Dieser entspricht der Release-Häufigkeit der beständigsten ECUs (d. h. der kleinste, gemeinsame Nenner bezüglich der Release-Frequenz). Hinzu kommen frei wählbare Zwischenzyklen, innerhalb derer „Zwischen-Releases“ für die schnelllebigeren Steuergeräte stattfinden.*

Der Hauptzyklus muss dabei nicht notwendigerweise dem etablierten, einheitlichen Zyklus entsprechen, sondern kann eine geringere Frequenz aufweisen. Dadurch wird beständigeren Steuergeräten ermöglicht, auf für sie unnötige Releases zu verzichten. Ein Zyklus umfasst die Zeit ab einem K-Matrix-Release bis zum folgenden K-Matrix-Release, welches den Zyklus abschließt.

Die K-Matrix-Entwicklung erfolgt, indem zu den Release-Zeitpunkten veränderte K-Matrix-Stände veröffentlicht werden. Die Änderungen können, je nach ihren Abschnitt 4.4.2 entsprechenden Eigenschaften, in unterschiedlichen Releases vorkommen.

In Abschnitt 3.5.4 wird K als die Menge aller K-Matrix-Releases (d. h. aller Haupt- u. Zwischen-Releases gemäß Regel 1) definiert. K kann in die folgenden Untermengen aufgeteilt werden:

Definition 4.2 (Haupt-Release-Menge). *K^H sei die Menge aller K-Matrix Haupt-Releases, mit $K^H \subset \mathbb{K}$.*

Definition 4.3 (Zwischen-Release-Mengen). *K^A, K^B, etc. seien die Mengen aller K-Matrix-Releases in den Zwischenzyklen A, B etc., mit $K^H \subset K^A \subset \mathbb{K}$ und $K^H \subset K^B \subset \mathbb{K}$.*

Allen Zwischen-Release-Mengen gemein ist, dass sie die Haupt-Releases beinhalten. Diese dienen als Synchronisationspunkte zwischen sämtlichen Release-Zyklen. Es gilt:

$$K_i^H = K_{h \cdot i} \tag{4.1}$$

mit:

$$h = \frac{|K|}{|K^H|} \tag{4.2}$$

Analoges gilt für K_i^A und a sowie K_i^B und b.

Weitere Überschneidungen (z. B. $K^B \subset K^A$) sind möglich, aber nicht notwendig. Ggf. könnten sie zur Synchronisation zweier Zwischenzyklen verwendet werden.

Unterschiedliche Steuergeräte können innerhalb dieser Zyklen entwickelt werden. Sinnvolle Kommunikation setzt das gleichzeitige Vorhandensein von Sender und Empfängern voraus. Eine neu eingeführte Funktion kann nur getestet werden, wenn alle beteiligten ECUs die entsprechenden Schnittstellen bereitstellen. Dies macht eine Gruppierung der Steuergeräte, z. B. in Busse oder Systeme (vgl. Abs. 4.1.2), sinnvoll.

Die Dauer der Zwischenzyklen ist frei wählbar (vgl. Regel 1). Sinnvoll ist es, diese durch einen ganzzahligen Teiler aus der Dauer des Hauptzyklus abzuleiten. Andernfalls fällt der Abstand, zwischen dem letzten Zwischen-Release und dem folgenden Haupt-Release, geringer aus als der, zwischen den anderen Zwischen-Releases.

Abbildung 4.3 stellt die Release-Zyklen eines beispielhaften Entwicklungsprozesses dar. Der Hauptzyklus beträgt 6 Monate (d. h. langsamer als die in der Praxis üblichen 3 Monate, vgl. Abschnitt 3.5.4). Es gibt die Zwischenzyklen **A** (Zyklusdauer 1 Monat) und **B** (Zyklusdauer 3 Monate). Die Releases der Zwischen-Release-Menge K^A werden vereinfachend als „A-Releases“, die von K^B als „B-Releases“ und die von K^H als „Haupt-Releases“ bezeichnet. Es wird eine Kurzschreibweise verwendet, beginnend mit 'R' (für „Release“) gefolgt von der Nummer des letzten Haupt-Releases (z. B. R0, R0 ...). Ggf. folgt, verbunden durch einen Bindestrich, der Buchstabe des Zwischenzyklus sowie die Nummer des aktuellen Zwischen-Releases (z. B. -A0, -A1 ...).

Abbildung 4.3: Beispiel für einen Entwicklungsprozess mit einem Haupt-Release-Zyklus von sechs Monaten und Zwischenzyklen mit einem (A) und drei (B) Monaten.

Regel 2. *Änderungen an Elementen der Kommunikationshierarchie sind nur zu bestimmten Zeitpunkten möglich:*

- *Kompatible Änderungen können zu Haupt- und Zwischen-Releases erfolgen.*
- *Inkompatible Änderungen müssen im Rahmen von Haupt-Releases erfolgen.*

Die Notwendigkeit von Haupt-Releases als Synchronisationspunkten wird durch Regel 2 ausgedrückt: Nur an diesen Punkten, ist das Entfernen von Elementen möglich. Gemäß Abs. 4.4.2 ist es nicht rückwärtskompatibel, Elemente aus dem Gesamtsystem zu entfernen. Daher müssen Releases, in denen Elemente entfernt werden, auf allen Steuergeräten zum selben Zeitpunkt ausgerollt werden. Kompatible Erweiterungen der Schnittstellen sind zu allen Releases möglich. Die Bindung an Haupt- und Zwischen-Releases ist nötig, um Gruppen von Steuergeräten (innerhalb der Zwischenzyklen) synchron zu entwickeln. Fehlerkorrekturen, die sich nicht auf die Schnittstellen auswirken, dürfen zu beliebigen Zeitpunkten stattfinden.

Einen Sonderfall stellen Fehlerkorrekturen (engl.: „Bug Fixes“) dar: Handelt es sich um Fehler an der K-Matrix, d. h. Fehler an Schnittstellenbeschreibungen mehrerer[1] Steuergeräte, müssen diese ebenfalls als kompatibel oder inkompatibel bewertet werden und zu einem entsprechenden K-Matrix-Release erfolgen. Handelt es sich um Fehler an der Software einzelner Steuergeräte (die u. U. die übertragenen Werte einzelner Signale, nicht aber die Schnittstelle betreffen), können die Korrekturen jederzeit erfolgen, da keine Synchronisation mit anderen ECUs erforderlich ist.

Versionsnummern und Änderungskopien

Anforderung **A2** verlangt die unmittelbare Ablesbarkeit der Kompatibilität einer ECU zu einem Versuchsaufbau (vgl. Abschnitt 4.2.2).

Regel 3. *Jedes Element der Hierarchie erhält eine eigene, semantische Versionsnummer. Die Versionsnummern werden den Änderungskategorien (vgl. Abschnitte 3.5.3 und 4.4.2) entsprechend inkrementiert.*

Soll überprüft werden, ob ein Steuergerät zu einem Aufbau kompatibel ist, muss lediglich die semantische Version, des Interface am Versuchsaufbau, mit der semantischen Version, am Interface der ECU, verglichen werden

[1] Da jedem Signal ein Sender und mindestens ein Empfänger gehört, betreffen K-Matrix Fehler immer mindestens zwei ECUs.

(vgl. Abb. 4.1). Bei übereinstimmender Hauptversion ist das Steuergerät zumindest eingeschränkt kompatibel. Stimmen Haupt- und Unterversion überein, liegt vollständige Kompatibilität vor. Ist die Unterversion des Steuergeräts größer, als die im K-Matrix-Release des Cluster-HIL Aufbaus oder Erprobungsträgers geforderte, bedeutet das, dass sämtliche Funktionen des Aufbaus vorhanden sind.

Kernelement des vorgestellten Prozesses sind die Änderungskopien. Ein Konstrukt, mit dem inkompatible Änderungen, die zu einem Zeitpunkt notwendig werden, zu dem sie nicht erlaubt sind (Regel 2), kompatibel gemacht werden.

Regel 4. *Ist zu einem Zwischen-Release eine inkompatible Änderung erforderlich, wird durch eine Änderungskopie die übergeordnete Hierarchieebene kompatibel gehalten.*

Dabei bleibt das Ursprungselement (und dessen Versionsnummer) unverändert – daher die Rückwärtskompatibilität – während die (inkompatible) Kopie eine neue Hauptversion erhält (Regel 5). Ab diesem Zeitpunkt existieren von dem veränderten Element zeitgleich zwei Versionen (unterscheidbar an den unterschiedlichen Hauptversionsnummern). In dieser Phase können die beständigeren Steuergeräte auf die ältere Version zurückgreifen, während die ECUs, für die die neuere Version eingeführt wurde, diese bereits nutzen können.

Regel 5. *Bei einer Änderungskopie behält das Ursprungselement seine Versionsnummer. Die Kopie erhält eine um eins erhöhte Major-Version, während Minor- und Patch-Version auf null gesetzt werden. Beim übergeordneten Element wird die Minor-Version um eins inkrementiert und die Patch-Version auf null gesetzt.*

Nach der Durchführung einer Änderungskopie wird durch die unveränderte Versionsnummer am kopierten Objekt gezeigt, dass dieses nicht modifiziert wurde. Die Kopie (welche Regel 4 entsprechend durchgeführt wurde, um eine inkompatible Änderung zu ermöglichen) wird durch eine neue Major-Version

als inkompatibel gekennzeichnet. Minor- und Patch-Version werden zurückgesetzt, da diese sich auf vergangene, kompatible Änderungen beziehen, zu denen mit der inkompatiblen Änderung keine Verbindung mehr besteht.

Durch die Umsetzung der Änderungskopie, wurde die Funktionalität des, in der Hierarchie übergeordneten, Objektes kompatibel erweitert, daher wird hier die Minor-Version inkrementiert. Dadurch wird der erweiterte Umfang und die Rückwärtskompatibilität kenntlich gemacht.

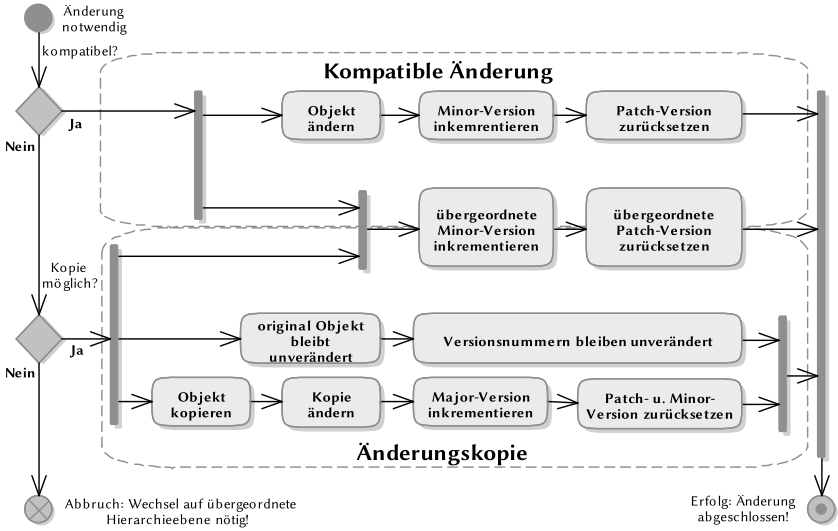

Abbildung 4.4: Ablauf zur Änderung eines Objekts.

In Abhängigkeit von der Kompatibilität der Änderung wird entweder unmittelbar das Element verändert, oder eine Kopie erstellt und diese verändert (s. Abb. 4.4). Die Rückwärtskompatibilität des übergeordneten Objektes ist unabhängig davon, ob eine unmittelbar kompatible Änderung oder eine Änderungskopie erfolgen, gegeben. Deshalb haben beide Pfade einen gemeinsamen Unterpfad, zur Anpassung der übergeordneten Minor-Version.

Es besteht die Möglichkeit, dass eine Änderungskopie nötig, aber nicht möglich ist. In diesem Fall muss auf die nächsthöhere Hierarchieebene ausgewichen werden (vgl. Abschnitt 4.5.4).

Regel 6. *Änderungskopien sind auf der niedrigsten Hierarchieebene durchzuführen. Ist dies nicht möglich, wird die jeweils nächsthöhere Ebene herangezogen.*

Bei hierarchischen Strukturen mit mehreren Ebenen (vgl. Abb. 4.2) bieten sich mehrere Möglichkeiten, um Änderungskopien durchzuführen. Muss z. B. ein Signal inkompatibel verändert werden, kann diese Inkompatibilität durch eine Änderungskopie des Signals innerhalb der PDU oder eine Änderungskopie der PDU innerhalb des Interfaces ausgeglichen werden. Sofern beide Möglichkeiten gegeben sind (und nicht etwa eine Änderungskopie des Signals durch die maximale Größe der PDU ausgeschlossen ist) fordert Regel 6 dass die Operation auf der niedrigsten möglichen Ebene stattfindet. Der Grund hierfür ist die durch **A4** notwendige Sparsamkeit in Bezug auf die benötigte Bandbreite: Je höher die Ebene, auf der die Kopie durchgeführt wird, desto größer ist die dadurch eingeführte Redundanz (vgl. Abschnitt 4.5.4).

Redundanzen begrenzen

Regeln 1 – 5 erfüllen die Anforderungen **A1**-**A3**. Das Erstellen von Änderungskopien und der damit verbundene Mehrbedarf an Übertragungsbandbreite stehen im Widerspruch zu **A4** – der Begrenzung der Fertigungs- und Materialkosten. Regel 6 führt zu einer Minimierung der entstehenden Redundanzen.

Die folgenden Regeln dienen dazu, die durch Änderungskopien eingeführten Redundanzen zeitlich zu begrenzen (und die Koexistenz von „zu vielen“[1] Kopien desselben Ursprungsobjektes zu verhindern). Dazu müssen alte Versionen der kopierten Kommunikationselemente systematisch entfernt werden. Es ergibt sich ein Zielkonflikt aus **A4** (sparen von Ressourcen), in deren Sinne

[1] Konkrete Werte werden in Kapitel 5 ermittelt.

Elemente möglichst früh entfernt werden und **A6** (Planbarkeit) in deren Sinne Interfaces möglichst lange rückwärtskompatibel sein müssen.

Definition 4.4. *Der Zeitraum vom ersten Release, in dem ein Kommunikationselement verfügbar ist, bis zu dessen letzten Release, wird als „Gesamtlebensdauer" bezeichnet.*

Die Gesamtlebensdauer spielt für die vorliegenden Untersuchungen eine untergeordnete Rolle, da im Zeitraum vor der ersten Änderungskopie keine vermeidbare Redundanz vorliegt.

Um Planbarkeit bieten zu können, wird eine Dauer garantiert, die ein Kommunikationselement verfügbar sein wird, sobald eine Änderungskopie erfolgt ist. Diese Dauer entspricht dem minimal möglichen Wert für die Gesamtlebensdauer.

Definition 4.5. *Der Zeitraum, den ein Kommunikationselement garantiert weiter zur Verfügung steht, nachdem es durch eine Änderungskopie obsolet wurde, wird als Mindestlebensdauer bezeichnet.*

Steht zum Einführungszeitpunkt eines Kommunikationselementes fest, wann es wieder entfernt werden wird, entspricht die Mindestlebensdauer der Gesamtlebensdauer. Solange kein Zeitpunkt für die Entfernung feststeht, lässt sich zur zu erwartenden Gesamtlebensdauer keine Aussage treffen (außer, dass das Element noch mindestens den Zeitraum der Mindestlebensdauer über bestehen wird).

Definition 4.6. *Die verbleibende Zeit, die ein obsolet gewordenes Kommunikationselement noch verfügbar ist, wird als Restlebensdauer bezeichnet.*

Zum Zeitpunkt einer Änderungskopie ist die Restlebensdauer gleich der Mindestlebensdauer. Im Verlauf verringert sich die Restlebensdauer stetig, bis zur Entfernung des betroffenen Kommunikationselements. Bevor eine Änderungskopie durchgeführt wurde, ist die Restlebensdauer unbekannt (größer oder gleich der Mindestlebensdauer).

Regel 7. *Jedes Element erhält eine Lebensdauerzahl (angegeben als Anzahl von Haupt-Releases), sowie zwei Markierungen, die angeben, ob sich die Lebensdauerzahl verringert („Lebensdauermarkierung“) oder ob eine inkompatible Änderung bevorsteht („Änderungsmarkierung“).*

Ist die Lebensdauermarkierung nicht gesetzt, gibt die Lebensdauerzahl die Mindestlebensdauer an. Andernfalls zeigt sie die Restlebensdauer an.

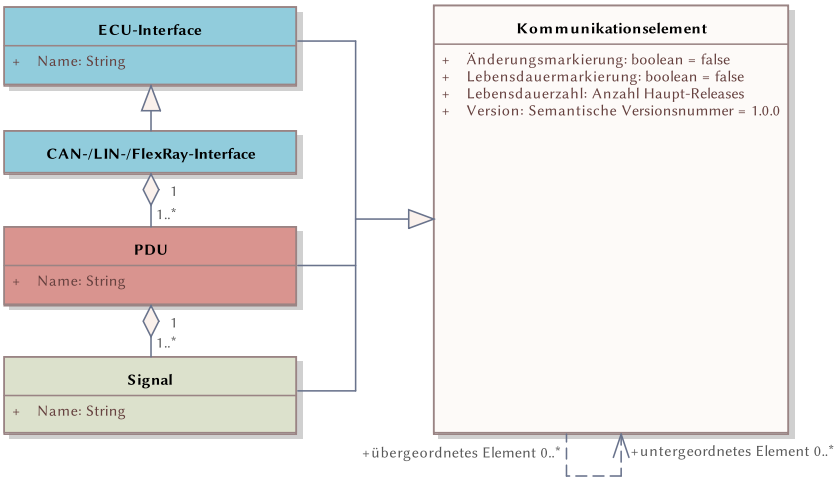

Abbildung 4.5: Die Objekte aller Hierarchieebenen sind Kommunikationselemente mit denselben Attributen.

Grundsätzlich darf die Mindestlebensdauer frei gewählt werden. Durch die Hierarchie ergibt sich aber eine Randbedingung: Da Elemente mit Ablauf ihrer Restlebensdauer entfernt werden, muss die Mindestlebensdauer eines Elements gleich oder größer als die Mindestlebensdauer des übergeordneten Elements sein. Andernfalls könnten sich Situationen ergeben, in denen ein Element, entgegen der gegebenen Garantie, vorzeitig inkompatibel wird, weil eines der untergeordneten Elemente wegfällt.

Zusammen mit der semantischen Versionsnummer gibt es vier Attribute, die allen Kommunikationselementen gemein sind. In einem Klassendiagramm

wie Abb. 4.2 ergibt sich eine gemeinsame Oberklasse wie in Abb. 4.5 gezeigt. (Die Oberklasse gilt gleichermaßen für Services, Events, Methoden, etc., welche hier zur besseren Übersicht ausgelassen wurden.)

Regel 8. *Bei neuen Interfaces (oder bei neuen Hauptversionen bestehender Interfaces) darf die Mindestlebensdauer frei gewählt werden. Bei Elementen der unteren Hierarchieebenen muss die Mindestlebensdauer mindestens so groß wie im übergeordneten Element gewählt werden.*

Bei der Durchführung einer Änderungskopie sind bezüglich der Mindestlebensdauer drei Dinge zu berücksichtigen:

- Für die neue Kopie ist eine neue Mindestlebensdauer zu vergeben. Da ein neues Kommunikationselement entsteht, zu welchem zum Entstehungszeitpunkt keine Abhängigkeiten bestehen (d. h. es gibt keine Steuergeräteprojekte, welche das Objekt mit einer Mindestlebensdauer eingeplant haben), darf eine neue Mindestlebensdauer frei gewählt werden.
- Für das alte Objekt wird aus der Mindestlebensdauer die Restlebensdauer. Dies wird angezeigt, indem das Objekt mit einer „Lebensdauermarkierung“ versehen wird. Für die Empfänger eines Kommunikationselements ist das Setzen der Lebensdauermarkierung verbunden mit der Warnung, dass das Objekt wegfallen wird, und innerhalb der verbleibenden Restlebensdauer auf ein neues Objekt gewechselt werden muss (i. d. R. wird es sich um die neue Kopie handeln, aber auch andere Quellen, die eine äquivalente Information bieten, oder der Verzicht auf das Kommunikationselement sind möglich).
- Ist die Änderungsmarkierung gesetzt, bedeutet das, dass nach Ablauf der Restlebensdauer das Kommunikationselement inkompatibel geändert wird (i. d. R. durch den Wegfall eines untergeordneten Objekts).

Regel 9. *Bei einer Änderungskopie wird beim Ursprungselement die Lebensdauermarkierung gesetzt. Für die Kopie gilt wie für neue Objekte die Regel 8. Bei den übergeordneten Hierarchieebenen, welche dieselbe Mindestlebensdauer aufweisen, werden die Änderungsmarkierungen gesetzt.*

Falls die übergeordneten Hierarchieebenen kleinere Mindestlebensdauern aufweisen, wird die Änderungsmarkierung nicht gesetzt. Größere Mindestlebensdauern können die übergeordneten Elemente wegen Regel 8 nicht aufweisen.

Sind Lebensdauer- oder Änderungsmarkierung gesetzt, beginnt die Restlebensdauer sich zu verringern. Da nur zu den Haupt-Releases inkompatible Änderungen möglich sind, ist [Anzahl Haupt-Releases] die Einheit für die Lebensdauern (vgl. R. 7) und entsprechend sind neue Haupt-Releases die Zeitpunkte, zu denen die Lebensdauerzahlen der markierten Elemente verringert werden.

Regel 10. *Mit jedem Haupt-Release verringert sich die Restlebensdauer aller Elemente, deren Lebensdauermarkierung oder Änderungsmarkierung gesetzt ist, um eins. Bei den übergeordneten Hierarchieebenen, welche dieselbe Mindestlebensdauer aufweisen, werden die Änderungsmarkierungen gesetzt.*

Objekte, deren Restlebensdauer abgelaufen ist, sind in zwei Kategorien zu unterteilen: Die, bei denen ausschließlich die Änderungsmarkierung gesetzt ist, und die, bei denen die Lebensdauermarkierung (und ggf. zusätzlich die Änderungsmarkierung) gesetzt ist.

Ist ausschließlich die Änderungsmarkierung gesetzt, so bedeutet dies, dass ein untergeordnetes Kommunikationselement wegfällt oder inkompatibel geändert wird. Damit gilt automatisch auch das Objekt selbst als inkompatibel geändert (vgl. Abschnitt 4.4.2). Entsprechend muss die Hauptversion erhöht werden.

Regel 11. *Weist ein Element zum Zeitpunkt des Haupt-Releases eine gesetzte Änderungsmarkierung, keine Lebensdauermarkierung, und eine Restlebensdauer von null auf, erhält das Element eine neue Hauptversion, Unter- und Patch-Version werden auf null gesetzt.*

Ist die Lebensdauermarkierung gesetzt, und die Restlebensdauer läuft ab, ist das Kommunikationselement zu entfernen. Es ist zu einem früheren Zeitpunkt durch eine Änderungskopie redundant geworden, und nach Ablauf der Lebensdauer ist davon auszugehen, dass alle Kommunikationspartner ausreichend (d. h. eine Mindestlebensdauer lang) Zeit hatten, auf die neuere Version zu wechseln.

Regel 12. *Weist ein Element zum Zeitpunkt des Haupt-Releases eine gesetzte Lebensdauermarkierung und eine Restlebensdauer von null auf, wird das Element gelöscht.*

Damit hängt die Balance zwischen **A4** und **A6** von den jeweils für die Mindestlebensdauern gewählten Werten, sowie der Häufigkeit, mit der inkompatible Änderungen auftreten (und durch Änderungskopien ausgeglichen werden müssen), ab.

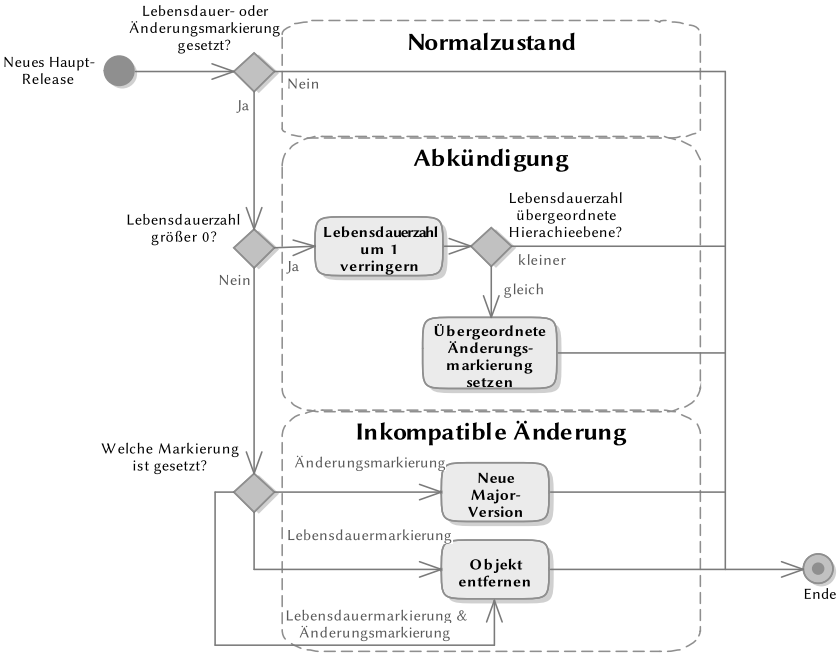

Abbildung 4.6: Ablauf zur Auswertung der Lebensdauerzahlen und Markierungen

Die Auswertung nach den Regeln 7 – 12 erfolgt jeweils vor dem Erscheinen eines neuen Haupt-Releases (zu dem es möglich ist, Objekte inkompatibel zu ändern oder zu entfernen, s. Abb. 4.6). Dabei kann ein Kommunikationselement in drei verschiedene Kategorien unterteilt werden:

- Im Normalzustand ist weder die Lebensdauer- noch die Änderungsmarkierung gesetzt. Das Objekt kann verwendet und für neue Funktionen eingeplant werden.
- Liegen Markierungen vor, bei einer Restlebensdauer größer als Null, kann das Kommunikationselement zunächst weiter verwendet werden. Da das Objekt durch die Markierungen als abgekündigt gilt, dürfen neu zu entwickelnde Funktionen nicht darauf zugreifen (und stattdessen etwa das, durch die Änderungskopie entstandene,

Nachfolgeobjekt verwenden). Die Restlebensdauer abgekündigter Kommunikationselemente verringert sich mit jedem Haupt-Release.

- Weisen Objekte eine der Markierungen und eine Restlebensdauer von null auf, erfolgt eine (hinreichend lange angekündigte – im Sinne der ursprünglich gewählten Mindestlebensdauer) inkompatible Änderung. Entweder durch das Entfernen von Unterobjekten, angezeigt durch das Erhöhen der Haupt-Version, oder durch das Entfernen des Kommunikationselements selbst.

4.4.6 Migration

Um das vorgestellte Konzept, in einem realen Prozess einzusetzen, ist eine Migrationsstrategie erforderlich. Die Umstellung kann vollständig zu einem vorgegebenen Stichtag erfolgen, aber auch ein schrittweiser Wechsel ist möglich. Letzteres profitiert davon, dass bestehende Entwicklungsprozesse als hierarchische Versionierung ohne Zwischenzyklen und ohne Mindestlebensdauern betrachtet werden können.

In der Praxis bietet sich entweder die vollständige Umstellung zum Start der Entwicklung einer neuen Baureihe, oder der schrittweise Übergang in einer laufenden Entwicklungsphase an. Ein vollständiger Wechsel während einer laufenden Entwicklungsphase ist theoretisch ebenfalls möglich. Eine abrupte Änderung des K-Matrix-Entwicklungsprozesses bringt jedoch das Risiko eines verspäteten K-Matrix-Release, und damit einer Verzögerung der gesamten Fahrzeugentwicklung, mit sich. Bei einem schrittweisen Übergang sind die Risiken jedes einzelnen Schritts besser abschätzbar und praktisch auftretende Probleme können einfacher kurzfristig behoben werden.

Bei der schrittweisen Umstellung sind (nicht notwendigerweise in dieser Reihenfolge) die folgenden Einzelschritte nötig:

- Die in Abb. 4.5 gelisteten Attribute werden in die zur Entwicklung verwendeten Werkzeuge und die K-Matrix-Datenbank (vgl. Abschnitt 2.5) eingebaut. Mit diesem Schritt stehen die semantischen Versionsnummern zur Anzeige von kompatiblen und inkompatiblen Änderungen bereit. Änderungskopien können durchgeführt werden. Die Mindestlebensdauern aller Kommunikationselemente sind jeweils auf 0 festgelegt (d. h. es sind keine Kompatibilitätsgarantien gegeben, äquivalent zu dem Zustand vor der Einführung hierarchischer Versionierung) und Änderungs- und Lebensdauermarkierungen sind nicht gesetzt. Inkompatible Änderungen sind zu jedem Release möglich.
- Der Wechsel von einem einheitlichen zu mehreren unterschiedlichen Release-Zyklen kann in Unterschritte geteilt werden:
  - Soll der zukünftige Hauptzyklus eine größere Dauer aufweisen, als der aktuelle Einheitszyklus (z. B. sechs statt drei Monate), werden die Einheitszyklen als erster Zwischenzyklus zusätzlich zu dem Hauptzyklus festgelegt (vgl. Haupt-Releases und B-Releases in Abb. 4.3). Dies setzt voraus, dass die Zyklusdauer der Hauptzyklen ein ganzzahliges Vielfaches der Einheitszyklusdauer ist. Alle Steuergeräte sind automatisch diesem ersten Zwischenzyklus zugeordnet (der ursprünglichen Release-Häufigkeit entsprechend).
  - Weitere Zwischenzyklen innerhalb des Hauptzyklus werden festgelegt (z. B. die A-Releases in Abb. 4.3).
  - Die Steuergeräte werden, unter Berücksichtigung der individuellen Anforderungen (vgl. Abschnitt 4.1.2), den unterschiedlichen Zyklen zugeordnet. Die Verschiebung einzelner Steuergeräte aus dem ersten Zwischenzyklus in den Haupt- oder einen anderen Zwischenzyklus kann zu unterschiedlichen Zeitpunkten erfolgen.

- Um zur praktischen Verwendung von Mindestlebensdauern überzugehen, erhalten neue Kommunikationselemente Werte größer Null. Eine nachträgliche Erhöhung der Mindestlebensdauer für bestehende Kommunikationselemente ist möglich, sofern die durch Regel 8 vorgegebenen Bedingungen eingehalten werden. Sobald die ersten dieser Objekte kopiert wurden, kommen deren Lebensdauer- und Änderungsmarkierungen zum Einsatz.

4.5 Anwendungsbeispiel

Anhand des Außenspiegels (Abschnitt 2.2) und einer exemplarischen Erweiterung dessen Funktionsumfang wird die hierarchische Versionierung illustriert.

4.5.1 Entwicklungszyklen

Für den Entwicklungsprozess wird angenommen, dass es einen Hauptzyklus von sechs Monaten und zwei Zwischenzyklen mit Dauern von einem und drei Monaten gibt (vgl. Abb. 4.3). Die Steuergeräte werden jeweils einem dieser drei Zyklen zugeordnet (s. Tab. 4.1).

Tabelle 4.1: Die Steuergeräte werden den Release-Zyklen zugeordnet.

Steuergerät	**Häufigkeit**	**vgl. Abb. 4.3**
Außenspiegel, links	vierteljährlich	B-Releases
Türsteuergerät, vorn, links	vierteljährlich	B-Releases
Außentemperatursensor	(keine Änderungen)	–
Blendsensor, hinten	halbjährlich	Haupt-Releases
Toter Winkel Assistent, links	monatlich	A-Releases
Domänencontroller Karosserie	vierteljährlich	B-Releases

Im Beispielprozess hat ein Fahrzeughersteller einen Hauptzyklus von sechs Monaten. Im Karosseriebereich ist, um schnellere Innovationen zu ermöglichen, eine größere Geschwindigkeit vorgesehen, es wird daher ein Zyklus von drei Monaten verwendet (darunter fallen der Außenspiegel, die Türbedieneinheit und der Domänencontroller). Aufgrund intensiver Arbeit an autonomer Fahrfunktionen befinden sich die Fahrerassistenzsysteme in einem schnellen Entwicklungszyklus von einem Monat (darunter fällt der Totwinkelassistent).

Einen Sonderfall stellt der Außentemperatursensor dar. Dieser wurde aus Vorgängerbaureihen unverändert übernommen („Verblockt“), um durch den Skaleneffekt einen niedrigeren Stückpreis zu erreichen. Hier sind alle Änderungen ausgeschlossen.

4.5.2 Struktur der Kommunikationselemente

Die Hierarchie aus Abb. 4.2 kann dargestellt werden, indem jedes Kommunikationselement als eine Box gezeichnet wird, welche die Boxen der untergeordneten Kommunikationselemente enthalten (s. Abb. 4.7). Diese Boxen werden mit den Namen der Kommunikationselemente, sowie den für hierarchische Versionierung notwendigen Attributen (s. Abb. 4.5) versehen[1].

Der Außenspiegel verfügt zum Beginn des Entwicklungsprozesses über die Funktionen **Fn1** – **Fn6** (vgl. Abs. 2.2). Die für die sechs Funktionen notwendigen Informationen werden in acht Signalen, die sich auf drei PDUs aufteilen, übertragen. Initial liegen alle Elemente in der Version 1.0.0 vor, und weder Lebensdauer- noch Änderungsmarkierungen sind gesetzt. Die Mindestlebensdauern betragen 0 (es gibt keine anderen Steuergeräteprojekte, die eine langfristige Planung auf den Signalen des Außenspiegels aufbauen).

Solange keine Änderungen an den bestehenden Signalen oder die Einführung neuer Signale notwendig werden, bleiben alle Elemente und Attribute unverändert. Ggf. über einen Zeitraum von mehreren (Zwischen-)Releases.

[1] Aus Platzgründen werden, für das jeweilige Beispiel nicht relevante, Objekte teilweise ohne Attribute dargestellt oder ausgelassen.

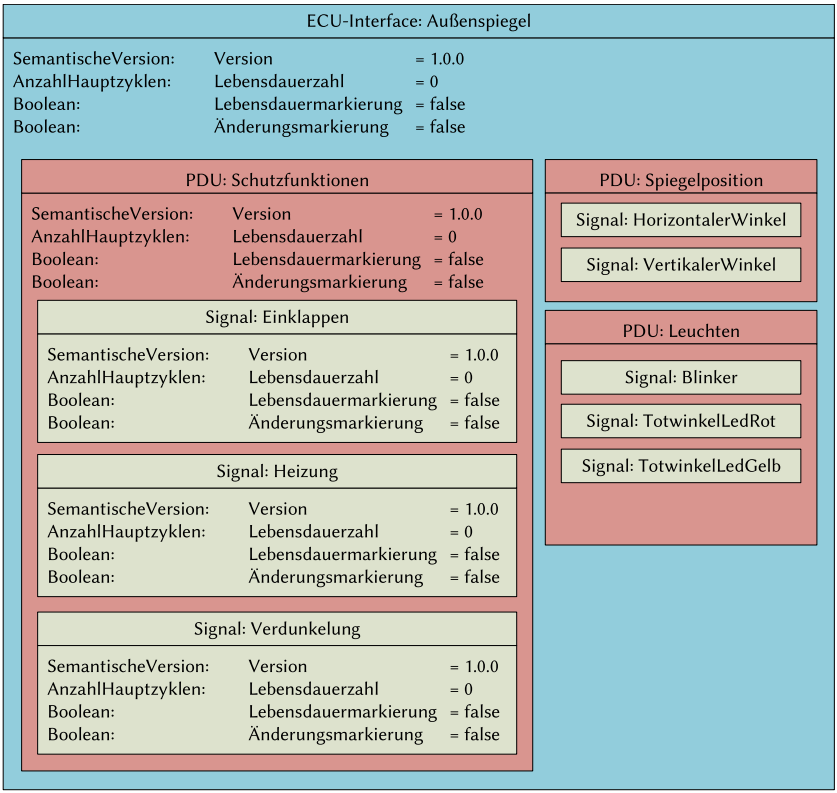

Abbildung 4.7: Die Kommunikationselemente des Außenspiegels.

4.5.3 Eine Änderung wird nötig

Im Rahmen verschiedener Erprobungen (vgl. Abschnitt 3.2.5 u. Abb. 3.5) fällt auf, dass der Außenspiegel, temperaturabhängig, beim Einklappen unerwünschte Knarzgeräusche erzeugt. Weitere Untersuchungen ergeben, dass sich diese Geräusche vermeiden lassen, indem die Geschwindigkeit, mit der der Spiegel eingeklappt wird, an die Temperatur angepasst wird. Deshalb muss der Wertebereich des „Einklappen“-Signals erweitert werden. Bisher

genügten die drei Werte „einklappen“, „ausklappen“ und „stopp“. Diese werden um vier unterschiedliche Geschwindigkeitsstufen erweitert.

Im Laufe mehrerer Releases (vgl. Abb. 4.3) entwickelt sich das Außenspiegel-Interface (vgl. Abb. 4.7) von der ursprünglichen Version 1.0.0 über die kompatible Unter- bzw. Minor-Version 1.1.0 zur inkompatiblen Haupt- bzw. Major-Version 2.0.0 (vgl. Abschnitt 3.5.3) wobei Änderungen an den jeweiligen Attributen gelb hervorgehoben werden (s. Abb. 4.8).

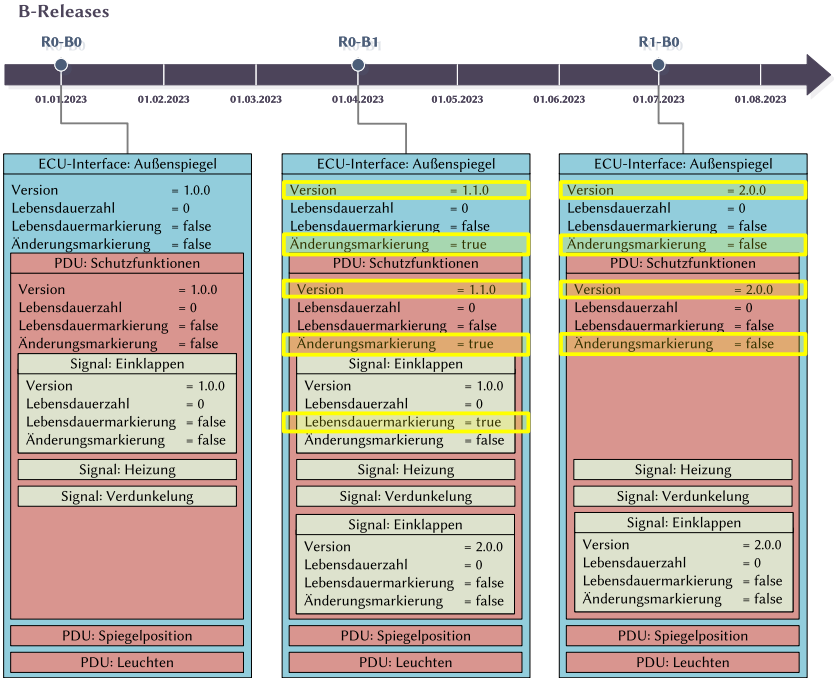

Abbildung 4.8: Entwicklung des Außenspiegel-Interfaces über mehrere Releases

R0-B1

Die Änderung soll zum Zwischen-Release R0-B1 (01.04.2023) integriert werden. Da die Erweiterung des Wertebereichs des Signals eine inkompatible Änderung wäre (vgl. Abschnitt 4.4.2), muss eine Änderungskopie durchgeführt werden. Die Kopie des Signals erhält die Version 2.0.0, PDU und Interface sind durch die Änderungskopie kompatibel verändert, entsprechend wird die Unterversion jeweils um eins erhöht. Beim Original Signal wird die Lebensdauermarkierung gesetzt, und da bei der PDU und Interface die Lebensdauerzahlen bereits 0 betragen, werden hier jeweils die Änderungsmarkierungen gesetzt.

R1-B0

In der „Schutzfunktionen"-PDU der Version 1.1.0 liegt das „Einklappen"-Signal doppelt vor, die benötigte Bandbreite ist erhöht. Da die Lebensdauerzahl 0 beträgt, wird die Version 1.0.0 des Signals zum nächsten Haupt-Release am 01.07.2022 entfernt, die benötigte Bandbreite verringert sich. Das Entfernen des alten Signals ist eine inkompatible Änderung. Die Versionen von PDU und Interface werden jeweils auf 2.0.0 erhöht (die Änderungsmarkierungen werden zurückgesetzt).

4.5.4 Die Änderungskopie misslingt

Die Änderungskopie eines Signals kann fehlschlagen. Ein Beispiel dafür könnte eine PDU sein, die mit ihren Signalen eine Größe erreicht, die so nahe an der Obergrenze des jeweiligen Mediums (z. B. 8 Byte für CAN oder 64 Byte für CAN-FD) liegt, dass kein Platz für die Kopie eines Signals ist. Wie häufig das in der Praxis zu erwarten ist, wird in Abschnitt 5.3, insbesondere in Abb. 5.7 und 5.8, untersucht.

Schlägt die Änderungskopie des Signals fehl, muss auf die nächsthöhere Hierarchieebene, die PDU, ausgewichen werden. Ein alternativer Ablauf zu Abb. 4.8 ist in Abb. 4.9 gegeben.

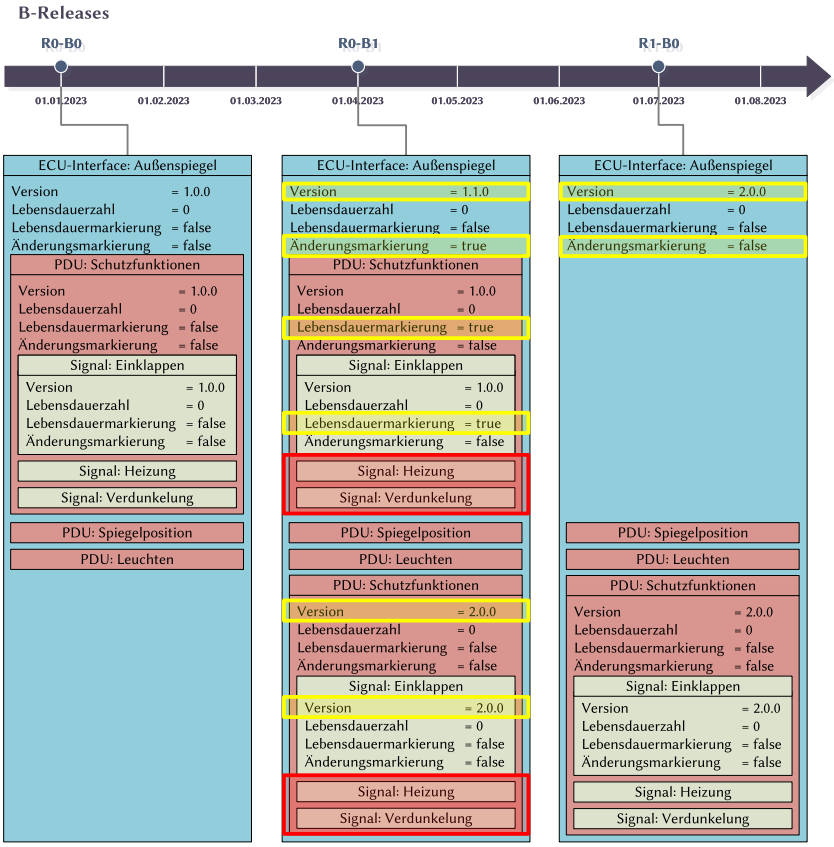

Abbildung 4.9: Die Änderungskopie wird nicht mit dem Signal, sondern mit der PDU durchgeführt.

R0-B1

In dieser Variante erhalten sowohl das „Einklappen“-Signal als auch die „Schutzfunktionen“-PDU jeweils in der Version 1.0 eine Lebensdauermarkierung, sodass sie im Release R1-B0 entfernt werden können. Auf der Hierarchieebene des Interfaces stellt auch die Änderungskopie der PDU eine kompatible Änderung dar, sodass hier in Release R0-B1 wieder die Version

1.1.0 verwendet werden darf (und durch eine Änderungsmarkierung auf eine bevorstehende inkompatible Änderung des Interfaces hingewiesen wird).

Die in Abb. 4.9 Rot markierten Signale kommen durch die Änderungskopie der PDU doppelt vor, obwohl sie nicht verändert wurden. Das stellt eine zusätzliche Redundanz dar, die – wo möglich – durch Regel 6 vermieden wird.

Auch das Ausweichen auf die PDU-Ebene kann fehlschlagen. Voraussetzung für die Änderungskopie einer PDU ist, dass auf dem jeweiligen Bus noch ausreichend Übertragungskapazität (vgl. Abschnitte 3.7 und 6.1.1) zur Verfügung steht[1]. In diesem Fall bestünde keine weitere Ausweichmöglichkeit auf eine höhere Hierarchieebene. Es bleiben zwei Handlungsalternativen, welche beide die Anforderungen verletzen würden:

- Die Änderung ist zum gegebenen Release nicht möglich, **A1** wird nicht eingehalten. Stattdessen muss sie auf ein späteres Release, zu dem inkompatibel geändert werden darf, verschoben werden. Um diese inkompatible Änderung anzukündigen, können entsprechende Lebensdauer- und Änderungsmarkierungen gesetzt werden.
- Eine inkompatible Änderung erfolgt sofort. Die durch die Mindestlebensdauern gegebenen Kompatibilitätsgarantien und damit **A6** werden verletzt.

Keine sinnvolle Alternative wäre, die Bandbreite des Busses zu erhöhen, da dies mit **A4** (Kosten) und **A6** (Planbarkeit von Schnittstellen, ein veränderter Physical-Layer ist inkompatibel, vgl. Abschnitt 3.4.1) gleich zwei Anforderungen verletzen würde.

R1-B0

Zum folgenden Haupt-Release R1-B0, wird die veraltete (d. h. mit einer Lebensdauerzahl von null und einer Lebensdauermarkierung versehene) PDU

[1] Theoretisch kann mangelnde Übertragungskapazität auch die Änderungskopie auf Signalebene verhindern, da durch ein zusätzliches Signal in einer PDU, deren Übertragungsdauer und damit deren Buslastanteil steigt.

„Schutzfunktionen“ in der Version 1.0 entfernt, die Version 2.0 verbleibt. Diese inkompatible Änderung am Interface wird durch das Inkrementieren der Major-Version (und das Zurücksetzen von Minor- u. Patch-Version) angezeigt. Der Endzustand ist damit derselbe, wie in Abschnitt 4.5.3.

4.5.5 Anwendung von Mindestlebensdauern u. Änderungsmarkierungen

Bei Signalen oder PDUs mit mehr als einem Empfänger[1] können Mindestlebensdauern eingetragen sein. Beispielsweise sei für das „Einklappen“-Signal eine Mindestlebensdauer in Höhe von zwei Haupt-Releases (d. h. ein Jahr, bei der in diesem Beispiel verwendeten Hauptzyklusdauer) und für die „Schutzfunktionen“-PDU eine Mindestlebensdauer von einem Hauptzyklus (d. h. sechs Monate) vorgesehen. Abgesehen von den unterschiedlichen Lebensdauern, besteht in Abb. 4.10 in Release R0-B0 die gleiche Ausgangslage wie in Abb. 4.8 und 4.9.

R0-B1

Die zuvor demonstrierte Änderung kann trotz dieser Mindestlebensdauern genauso schnell, zum nächsten Zwischen-Release, durchgeführt werden. Der Unterschied zu dem Szenario ohne Mindestlebensdauern ist, dass die Änderungsmarkierungen in PDU und Interface nicht sofort gesetzt werden. Regeln 9 u. 10 nach werden die Änderungsmarkierungen übergeordneter Hierarchieebenen gesetzt, sobald deren Lebensdauerzahl gleich der Lebensdauerzahl des untergeordneten Objekts ist.

[1] Hier ist der Außenspiegel mit seiner direkten Verbindung zum Türsteuergerät kein ideales Beispiel, und wird nur zur einheitlichen Darstellung verwendet. Mindestlebensdauern sind speziell für Signale mit mehreren Empfängern sinnvoll, sodass, wenn einzelne Empfänger eine Veränderung des Signals erforderlich machen, bekannt ist, wie lange die alte Version weiter vorgehalten werden muss.

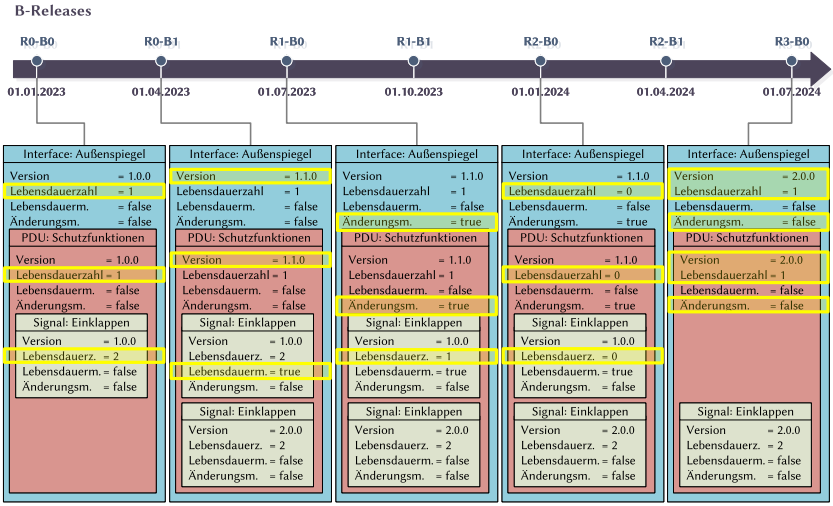

Abbildung 4.10: Anwendung von Mindestlebensdauern

R1-B0

Zu diesem Haupt-Release verringert sich die Lebensdauerzahl des Signals auf den Wert 1. Dieser entspricht den Lebensdauerzahlen von PDU und Interface, sodass dort jeweils die Lebensdauermarkierungen gesetzt werden (vgl. Pfad „Abkündigung“ in Abb. 4.6). Ab diesem Punkt sinken die Restlebensdauern von Signal, PDU und Interface gleichermaßen.

R2-B0

Bei Erreichen dieses Haupt-Releases werden, wegen der gesetzten Lebensdauermarkierung, die Lebensdauerzahl des Signals, und, wegen der gesetzten Änderungsmarkierung, die Lebensdauerzahlen von PDU und Interface um eins verringert (vgl. Pfad „Abkündigung“ in Abb. 4.6).

R3-B0

Bei Erreichen dieses Haupt-Release haben die Lebensdauerzahlen von Signal, PDU, u. Interface den Wert Null. Wegen der gesetzten Änderungsmarkierungen werden an diesem Punkt wie in Abschnitte 4.5.3 und 4.5.4 die Versionsnummern von PDU und Interface angepasst und wegen der gesetzten Lebensdauermarkierung die veraltete Version des „Einklappen"-Signals entfernt (vgl. Pfad „Inkompatible Änderung" in Abb. 4.6).

Durch die Mindestlebensdauer wurde die Geschwindigkeit, mit der die Änderung in das Fahrzeug eingebracht werden konnte, nicht verringert. Der Zeitraum, in dem mehrere Versionen des gleichen Signals vorhanden waren, wurde vergrößert (von drei Monaten auf ein Jahr und drei Monate, vgl. Abb. 4.8 und 4.10).

4.5.6 Testaufbauten mit veraltetem Außenspiegel

Im Mai 2023, den Zwischen-Releases R0-B1 bzw. R0-A4 folgend, soll ein Erprobungsträger aufgebaut werden. Leider liegen nicht von allen Steuergeräten aktuelle Versionen vor. Der Lieferant für den Außenspiegel ist im Verzug. Um die Erprobung nicht zu gefährden, sollen die Versuchsfahrzeuge mit der Vorgängerversion des Spiegels (basierend auf Haupt-Release R0-B0) aufgebaut werden. Zur Prüfung, ob dies praktikabel ist, wird die Versionsnummer des ECU-Interface des vorhandenen Außenspiegels (1.0.0) mit der Versionsnummer verglichen, die das Interface im aktuellen K-Matrix-Release (Stand 01.05.2023) hätte (1.1.0). Die Hauptversionen stimmen überein (jew. 1), die Unterversion ist geringer als sie es sein sollte (0 statt 1). Ohne PDU- und Signallisten abgeglichen zu haben, lässt sich so die Aussage treffen, dass die Erprobung möglich ist, jedoch einzelne Funktionen fehlen werden (in diesem Fall: die Antiknarzfunktion des Spiegels).

4.6 Zusammenfassung

Hierarchische Versionierung erlaubt kompatible Zwischen-Releases, bei der K-Matrix-Entwicklung. Voraussetzung sind hierarchische Kommunikationsstrukturen, wie sie bei AUTOSAR Classic respektive Service-orientierter Kommunikation auf Basis von AUTOSAR Adaptive, vorkommen. Dadurch wird die Entwicklung schnelllebigerer Steuergeräte in kürzeren Zyklen ermöglicht.

Durch die Mindestlebensdauern, die als Teil des Prozesses vergeben werden können, ist eine verlässliche Planung der Nutzungsdauer von Schnittstellen, für alle Netzwerkteilnehmer, möglich. Am Ende der jeweiligen Restlebensdauer werden Elemente geordnet aus dem System entfernt, um den Bandbreitenverbrauch zu minimieren.

Rückwärtskompatible Zwischen-Releases ermöglichen Testaufbauten, bei denen nicht alle ECUs auf demselben K-Matrix-Stand basieren.

5 Buslastzuwachs[1]

Der wesentliche Nachteil, und der die Anwendbarkeit beschränkende Faktor, bei dem vorgestellten Verfahren der hierarchischen Versionierung, sowie bei einfacher (nicht-hierarchischer) Rückwärtskompatibilität, ist der entstehende Anstieg der Buslast. Um das Verfahren zu bewerten, muss der Lastzuwachs, in Abhängigkeit aller relevanten Parameter, untersucht werden.

Das Grundprinzip, dass die vorhandene Buslast durch Änderungskopien ansteigt, gilt für alle Bustechnologien gleichermaßen. Im Folgenden werden am Beispiel von CAN und CAN-FD Formeln aufgestellt, mit denen sich die Auswirkung von hierarchischer Versionierung und deren Parametern auf die Buslast auswirkt.

5.1 Reale Prozessdaten

Um in den folgenden Abschnitten Vereinfachungen zu treffen, und Parameter zur Modellbildung zu ermitteln, wurden die in einem realen Entwicklungsprozess angefallenen Daten ausgewertet. Als Datenbasis diente die K-Matrix-Datenbank der Mercedes-Benz AG über die Dauer einer gesamten Fahrzeugentwicklung.

5.1.1 Netzwerkentwicklung in der Praxis

Bei Mercedes-Benz werden E/E-Architektur Plattformen (sogenannte „Standard Architekturen“, kurz: „STAR“) entwickelt, die als Grundlage für jeweils mehrere Fahrzeugbaureihen dienen. Die Entwicklung einer neuen Plattform wird durch das High End Modell „S-Klasse“ angeführt (man spricht auch von

[1] Teile, der in diesem Kapitel vorgestellten Untersuchung, wurden durch den Autor bereits im Februar 2023, im Rahmen des 8th International Congress on Information and Communication Technology, vorgestellt [Vet23b].

der „Lead-Baureihe“), nach deren Serienstart weitere, Modelle, mit geringerem Funktionsumfang, auf der Plattform aufsetzen. Technisch besteht eine E/E-Plattform aus einem Set an Steuergeräten, einer Topologie, über die diese verbunden sind, und einer 150%-K-Matrix (vgl. Abschnitte 2.5 und 3.6), welche die Kommunikation der Steuergeräte innerhalb der Topologie beschreibt.

Zu Beginn der Entwicklung einer Plattform werden durch Architekturexperten eine initiale Topologie und die Menge der einzusetzenden ECUs festgelegt. Anschließend werden die Steuergeräte einzelnen Abteilungen zugewiesen und sogenannte Bauteilverantwortliche (BTV) definiert, die die Entwicklung der ECUs ausschreiben und die Lieferanten betreuen. In der K-Matrix Datenbank werden zunächst die verblockten Steuergeräte aus der Vorgängerplattform (vgl. Abs. 2.1.2) eingetragen. Bei neu zu entwickelnden ECUs werden Kopien von den Vorgängerversionen angelegt und für neu hinzukommende Steuergeräte werden leere Einträge erstellt.

Das Erstellen und Verändern der K-Matrix erfolgt durch eine zentrale, für das Netzwerk des Fahrzeugs verantwortliche, Abteilung. Diese nimmt von den BTVs Änderungsanträge entgegen, stimmt diese im Rahmen eines Prozesses mit den weiteren, durch Änderung betroffenen BTVs ab, und modellieren, bei allgemeiner Zustimmung, die Kommunikation. Im Konfliktfall existiert ein Eskalationsprozess, über den die Entscheidung an übergeordnete Managementebenen weitergereicht werden. Ein typisches Beispiel für eine solche Situation ist, wenn zur Umsetzung einer zusätzlichen Funktion in einem Steuergerät, Änderungen am Netzwerk nötig sind, die eine (mit Zusatzkosten verbundene) Softwareaktualisierung eines anderen Steuergerätes erforderlich machen.

Die K-Matrix wird in einheitlichen Zyklen von drei Monaten weiterentwickelt. In der Änderungsphase werden Anträge entgegengenommen und umgesetzt. Währenddessen kann sich die K-Matrix, bedingt durch die sequenzielle Bearbeitung der Änderungsanträge, in einem inkonsistenten Zustand (vgl. Def. 2.1) befinden. Ab einem definierten Zeitpunkt werden keine neuen Anträge mehr entgegengenommen, die vorhandenen umgesetzt, und die Netzwerkdatenbank in einen konsistenten Zustand gebracht. Um diesen sicherzustellen, wird automatisch die Einhaltung von über 3000 Konsistenzregeln

überprüft. Aus der konsistenten K-Matrix werden für alle ECUs die jeweils relevanten Teilmengen des Kommunikationsmodells als ARXML-Dateien (vgl. Abschnitt 2.7.5) exportiert und an die BTVs und Lieferanten weitergegeben. Diese Releases werden nach Jahr und Kalenderwoche, also z. B. „2020-17“ benannt. Releases erfolgen in den Kalenderwochen 5, 17, 29 und 42. Dabei werden die jeweils ersten Releases als „a-Release“[1] oder „2020-17a“ bezeichnet, worauf jeweils mindestens ein weiteres Release innerhalb des laufenden Zyklus als b- (oder c- ...) -Release folgt, welches Fehlerkorrekturen und Änderungen an einzelnen Bussen beinhaltet. Die a-Releases sind die Haupt-Releases in einem einheitlichen Release-Zyklus, die b- u. c-Releases sind Fehlerkorrekturen an den Haupt-Releases – keine eigenständigen Release-Zyklen im Sinne der Definitionen 4.2 und 4.3.

Die Releases erfolgen jeweils für die gesamte 150%-K-Matrix, auch wenn der Serienstart einzelner, auf der Vernetzungsplattform aufbauender Baureihen bereits erfolgt ist. In diesen Fällen können die K-Matrix-Releases als Grundlage für OTA-Updates (vgl. Abschnitt 2.6.5) verwendet werden. Zwingend erforderlich ist, dass die Software sämtlicher ECUs auf demselben K-Matrix-Release basiert (vgl. Abschnitt 3.5.4). Dass aus jedem K-Matrix-Release Software-Updates für im Feld befindliche Fahrzeuge abgeleitet werden, ist nicht notwendig.

Basierend auf den ARXML-Dateien erfolgt bei den Lieferanten („Tier 1“, vgl. Abschnitt 2.1.3) die Entwicklung der Steuergeräte und deren Software. Die (von „Tier 2“s zur Verfügung gestellte) AUTOSAR Basissoftware wird durch Codegeneratoren konfiguriert und an Mikrocontroller, zusätzliche Softwarekomponenten[2] und Kommunikationsmodell (beschrieben durch die ARXML-Datei) angepasst. Aus den, mit den Prototypen gewonnenen, Erkenntnissen (vgl. Abs. 3.2.5) ergeben sich inkrementelle Verbesserungen, die mittels neuer Änderungsanträge, in das folgende K-Matrix-Release eingebracht werden.

[1] Nicht zu verwechseln mit den A- und B-Release-Zyklen aus Abschnitt 4.4.5 und Abb. 4.3. Zur Unterscheidung werden einmal Groß- und einmal Kleinbuchstaben verwendet.

[2] Z. B.: Softwaremodule dritter Softwarehersteller oder Treibersoftware der Hardwarelieferanten.

5.1.2 Datenbasis

Betrachtet wird die Neuentwicklung der STAR3, beginnend mit dem Release 2017-05 bis einschließlich 2023-42. In diesen Zeitraum fallen die Entwicklung und Serienstarts der S-Klasse (2020) sowie des EQS[1] (2021), der C-Klasse (2021), des EQE (2022) und der E-Klasse (2023).

Der Datensatz umfasst zum Ende des Betrachtungszeitraums rund:

- 61 K-Matrix-Releases über 7 Jahre
- 50.000 Signale in 7.500 PDUs
- 500 Steuergeräte mit 800 Schnittstellen
- 100 Busse

Es handelt sich hierbei um die 150%-K-Matrix (vgl. Abschnitte 2.5 und 3.6) zu den genannten Baureihen und allen zugehörigen Ausstattungsoptionen sowie Ländervarianten. In individuellen Fahrzeugen fallen die Zahlen geringer aus. Die folgenden Auswertungen beziehen sich auf die CAN- u. CAN-FD Busse.

In der Datenbank sind die verschiedenen K-Matrix-Elemente (Signale, PDUs, Busse etc.) als Zeilen in entsprechenden Tabellen abgelegt. Bei jeder Veränderung eines Objekts wird eine neue Kopie davon in der Datenbank abgelegt, die mit ihren Vorgängerversionen verknüpft ist. Zusätzlich sind die Zwischenstände der Objekte jeweils mit einem oder mehreren K-Matrix-Releases verknüpft. Dadurch ist für jedes Release der exakte Zustand nachvollziehbar.

Nicht in der Datenbank enthalten sind Informationen zur Kompatibilität zwischen zwei Versionen eines Objekts. In den folgenden Auswertungen werden im Sinne einer Worst-Case-Abschätzung alle in der Datenbank abgelegten Änderungen als inkompatibel angenommen.

[1] Beim EQS handelt es sich um eine eigenständige Baureihe, die das batterieelektrische Pendant zur S-Klasse bildet. Die parallele Entwicklung zweier Oberklassebaureihen, ist ein Beispiel für den mit der Verkehrswende verbundenen Anstieg an Entwicklungsaufwänden. Der getarnte Erprobungsträger in Abb. 3.5 ist ein Prototyp dieser Baureihe.

5.1.3 Zeitachse

Aus 61 K-Matrix-Ständen über 7 Jahre ergibt sich, dass im Schnitt 1,2 b- u. c-Releases auf ein a-Release folgen. Bei den folgenden Diagrammen bildet die x-Achse einen Zeitstrahl, auf welchem die a-Stände mit Jahr und Kalenderwoche und die b- u. c-Stände ohne Beschriftung eingetragen sind. Die Abstände zwischen den Markierungen sind proportional zur vergangenen Zeit.

5.2 Laständerungen

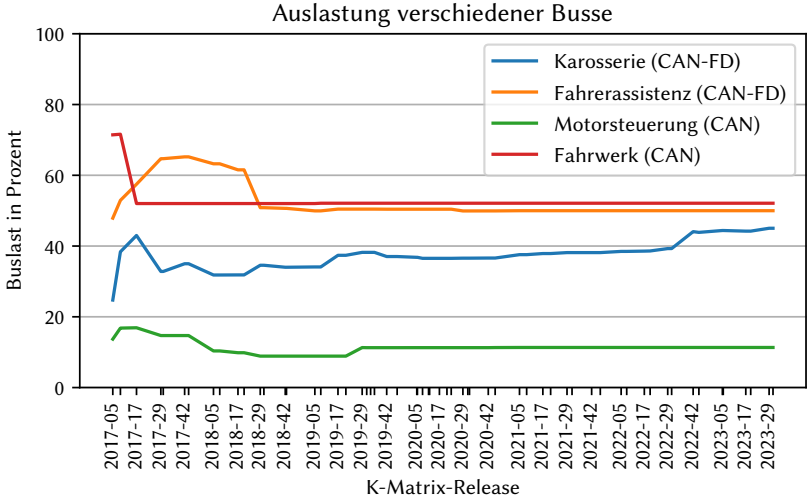

Abbildung 5.1: Buslasten im zeitlichen Verlauf

Definition 5.1 (Buslaständerung). *$\Delta L_{i,j}$ sei die Änderung der Auslastung (vgl. Definition 3.62) zwischen den Busreleases $B_{i-1,j}$ und $B_{i,j}$.*

$$\Delta L_{i,j} = L_{i,j} - L_{i-1,j} \tag{5.1}$$

Zwischen zwei K-Matrix-Releases K_{i-1} und K_i können verschiedene Einflüsse zu einer Veränderung der Auslastung $L_{i,j}$ eines Busses $B_{i,j}$ führen. Ursächlich sind Systeme, die zusätzliche Signale benötigen, bestehende Signale verändern (z. B. die Auflösung, den Messbereich oder die Zykluszeit) oder nicht mehr benötigen. Durch neu hinzukommende oder wegfallende Systeme kann sich der Bedarf an zu übertragenden Signalen auch verändern.

Abbildung 5.1 zeigt, wie sich die Auslastung vier verschiedener Busse im Verlauf des Entwicklungsprozesses verändern. Zu Beginn steht jeweils ein Anstieg, bedingt durch die schrittweise Implementierung der für die neuen Steuergeräte und Systeme notwendigen PDUs und Signale. Es folgt ein leichter Abfall, der durch das Aussortieren überflüssig gewordener Signale der Vorgängerplattform. Im späteren Verlauf der Entwicklung, mit zunehmendem Reifegrad, finden weniger Änderungen statt.

Definition 5.2 (Buslastanteilsänderung). *Sei $\Delta L_{i,j,k}$ die Änderung des absoluten Anteils der PDU Nr. j an der Auslastung des Busses Nr. k zwischen Release Nr. i-1 und i.*

$$\Delta L_{i,j,k} = L_{i,j,k} - L_{i-1,j,k} \tag{5.2}$$

$$\Delta L_{i,k} = \sum_{\forall W_j \in (B_{i-1,k} \bigcup B_{i,k})} \Delta L_{i,j,k} \tag{5.3}$$

Werden Gl. 3.10 u. Gl. 3.11 in Gl. 5.2 eingesetzt, ergibt sich die Änderung des Lastanteils $\Delta L_{i,j,k}$ einer PDU $P_{i,k}$ auf Bus $B_{i,j}$ in Abhängigkeit von den zu übertragenden Bits n, den Baudraten R und R' und der Zykluszeit z:

$$\Delta L_{i,j,k} = \frac{1}{z_{i,j}} \cdot \left(\frac{n_{i,j,k}}{R_{i,k}} + \frac{n'_{i,j,k}}{R'_{i,k}} \right) - \frac{1}{z_{i-1,j}} \cdot \left(\frac{n_{i-1,j,k}}{R_{i-1,k}} + \frac{n'_{i-1,j,k}}{R'_{i-1,k}} \right) \tag{5.4}$$

Die gesamte Buslast setzt sich zusammen aus den Buslastanteilen aller PDUs (vgl. Gl. 3.12). Anhand der Buslastanteile lassen sich Änderungen in vier Kategorien einteilen.

1 Änderungen, der Größe einer PDU mit entsprechendem Einfluss auf den Buslastanteil der PDU.

2 Änderungen, der Zykluszeit einer PDU, mit entsprechendem Einfluss auf den Buslastanteil der PDU.

3 Änderungen, durch die die Anzahl der PDUs (und damit der zu berücksichtigenden Buslastanteile) verändert wird.

4 Änderungen der Baudrate(n)

5.2.1 Änderung von PDU-Größen

Die Abstufung der durch CAN bzw. CAN-FD erlaubten Nutzdatengrößen bewirkt, dass Änderungen an $d_{i,j}$, die keinen Stufenwechsel zur Folge haben, sich nicht auf die Anzahl der zu übertragenden Bits ($n_{i,j,k} + n'_{i,j,k}$) auswirken (vgl. Abb. 3.9). Umgekehrt kann, an den Stufengrenzen, eine Änderung von $d_{i,j}$ um nur ein Bit, den Sprung in die nächst größere oder kleinere Grenze, d. h. +/- 140 Bit, bewirken.

Ändert sich bei einer PDU nur die Anzahl der zu übertragenden Bits, bei gleichbleibenden Zykluszeiten und Baudraten, vereinfacht sich Gl. 5.4:

$$\Delta L_{i,j,k} = \frac{1}{z_{i,j}} \cdot \left(\frac{n_{i,j,k} - n_{i-1,j,k}}{R_{i,k}} + \frac{n'_{i,j,k} - n'_{i-1,j,k}}{R'_{i,k}} \right) \Bigg|_{\substack{z_{i,j}=z_{i-1,j} \\ R_{i,k}=R_{i-1,k} \\ R'_{i,k}=R'_{i-1,k}}} \tag{5.5}$$

Aus den Gleichungen 3.8 u. 3.9 ergibt sich, dass auf klassischen CAN-Bussen $n'_{i,j,k} = 0$ gilt. Gl. 5.5 vereinfacht sich:

$$\Delta L_{i,j,k} = \frac{1}{z_{i,j}} \cdot \left(\frac{n_{i,j,k} - n_{i-1,j,k}}{R_{i,k}} \right) \Bigg|_{\substack{z_{i,j}=z_{i-1,j} \\ R_{i,k}=R_{i-1,k} \\ R'_{i,k}=R'_{i-1,k} \\ n'_{i,j,k}=n'_{i-1,j,k}=0}} \tag{5.6}$$

Im Falle von CAN-FD gilt $n_{i,j,k} = n_{i-1,j,k}$, da nur $n'_{i,j,k}$ von $d_{i,j}$ abhängt:

$$\Delta L_{i,j,k} = \frac{1}{z_{i,j}} \cdot \left(\frac{n'_{i,j,k} - n'_{i-1,j,k}}{R'_{i,k}} \right) \Bigg|_{\substack{z_{i,j}=z_{i-1,j} \\ R_{i,k}=R_{i-1,k} \\ R'_{i,k}=R'_{i-1,k} \\ n_{i,j,k}=n_{i-1,j,k}}} \tag{5.7}$$

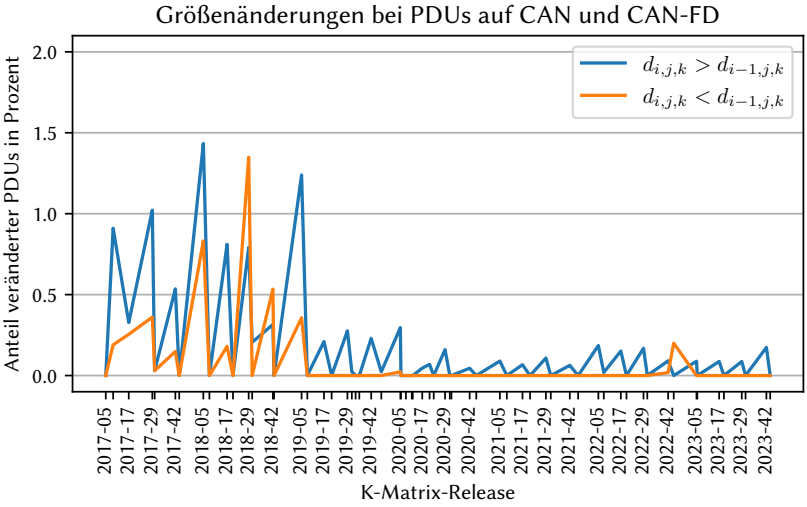

Abbildung 5.2: Häufigkeit von Größenänderungen bei PDUs

Anhand der K-Matrix-Datenbank wurde überprüft, wie häufig Änderungen der PDU-Größe in der Praxis vorkommen. Für jedes K-Matrix-Release wurde überprüft, bei wie vielen der PDUs die Menge der Nutzdatenbits $d_{k,j,x}$ größer oder kleiner als die im Vorgängerrelease geworden ist. Deren Anteil an der (sich verändernden, s. Abs. 5.2.3) Gesamtzahl der PDUs, im Verlauf des Entwicklungsprozesses, ist in Abbildung 5.2 dargestellt.

Zu keinem Zeitpunkt übersteigt der Anteil der größer oder kleiner werdenden PDUs die 1,5%-Marke. Ein Abfall der Änderungen mit zunehmendem Reifegrad der Vernetzungsplattform ist zu sehen. Der Anteil der wachsenden PDUs in Abb. 5.2 überwiegt in 58 von 61 Releases den der schrumpfenden (d. h.

durch die Änderung der PDU-Größen ergibt sich tendenziell ein Zuwachs der Buslast). Dem Diagramm ist zu entnehmen, dass der überwiegende Teil der Änderungen jeweils zu den a-Releases erfolgt.

5.2.2 Änderungen der Zykluszeit

Die Zykluszeit einer PDU ergibt sich aus dem Signal mit der niedrigsten Zykluszeit. Änderungen dieser Dauern an anderen Signalen wirken sich nicht auf den Lastanteil aus, falls sich keine neue geringste Zykluszeit ergibt. Die aus einer geänderten Sendehäufigkeit resultierende Buslaständerung kann durch Einsetzen von Gl. 3.11 in Gl. 5.2 berechnet werden:

$$\Delta L_{i,j,k} = \frac{t_{i,j,k}}{z_{i,j}} - \frac{t_{i-1,j,k}}{z_{i-1,j}} \tag{5.8}$$

Unter der Annahme, dass die Datenmenge und die Baudrate unverändert bleiben, verändert sich gemäß Gl. 3.10 die Sendezeit $t_{k,j,x}$ nicht.

$$\Delta L_{i,j,k} = \frac{t_{i-1,j,k}}{z_{i,j}} - \frac{t_{i-1,j,k}}{z_{i-1,j}} \Big| t_{i,j,k} = t_{i-1,j,k} \tag{5.9}$$

$$\Delta L_{i,j,k} = \frac{t_{i-1,j,k}}{z_{i-1,j}} \cdot \left(1 - \frac{z_{i-1,j,k}}{z_{i,j,k}}\right) \Big| t_{i,j,k} = t_{i-1,j,k} \tag{5.10}$$

$$\Delta L_{i,j,k} = L_{i-1,j,k} \cdot \left(1 - \frac{z_{i-1,j}}{z_{i,j}}\right) \Big| t_{i,j,k} = t_{i-1,j,k} \tag{5.11}$$

Falls es zu einer Änderung der Zykluszeit der PDU kommt, entspricht (bei unveränderter Datenmenge und Baudrate) das Verhältnis der Zykluszeiten dem Verhältnis der Buslastanteile. Das Delta der Buslastanteile errechnet sich gemäß Formel 5.11.

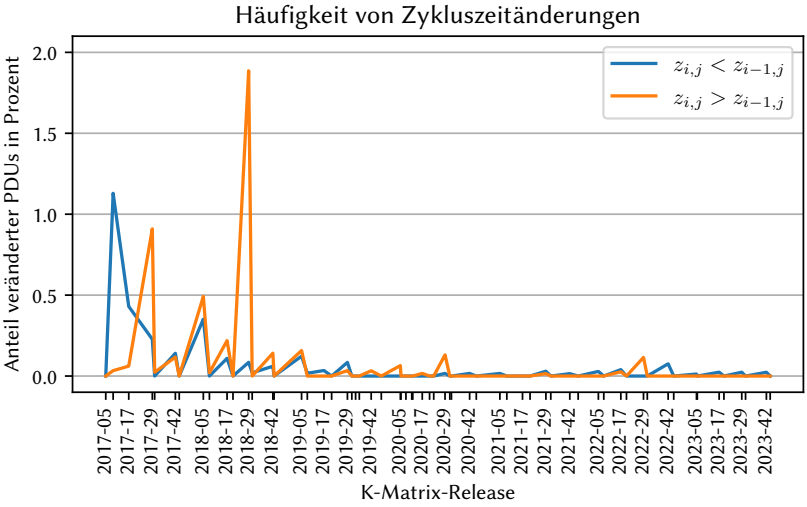

Abbildung 5.3: Änderungen der Zykluszeiten

Die Häufigkeit von Zykluszeitänderungen in der Praxis wurde mittels der K-Matrix-Daten ausgewertet. Analog zu den PDU-Größenänderungen sind die Änderungen der Sendefrequenz prozentual zur Gesamtanzahl der PDUs in Abbildung 5.3 über die K-Matrix-Releases aufgetragen.

Die Häufigkeit von Zykluszeitänderungen liegt, wie die von Änderungen an PDU-Größen, im Bereich von unter 2 % und zeigt ebenfalls einen Abfall mit wachsendem Reifegrad. Anders als bei den Änderungen der PDU-Größen ergibt sich bei denen der Zykluszeit tendenziell eine Abnahme der Buslast. In den meisten Releases liegen mehr Zunahmen als Abnahmen der Zykluszeit vor. D. h. im Mittel verringert sich die Sendefrequenz, welche proportional zur Auslastung des Busses ist.

5.2.3 Änderungen der PDU-Menge

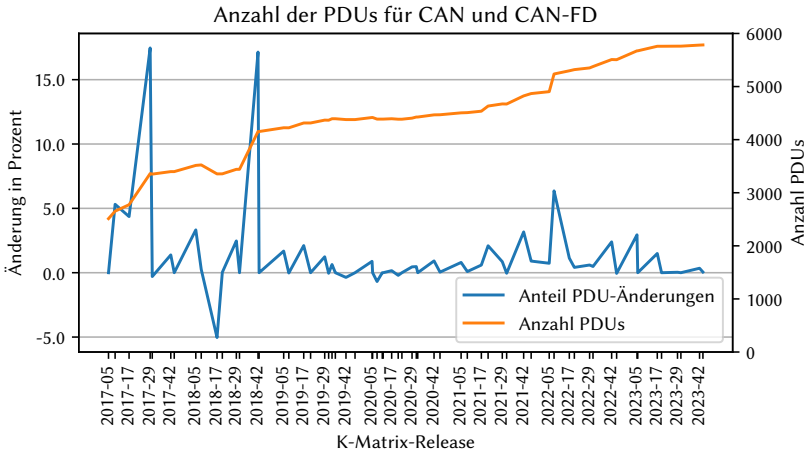

Abbildung 5.4: Änderungen der PDU-Menge

Die Einbindung neuer Systeme oder umfangreichere (d. h. mehr als einzelne Signale umfassende) Änderungen an bestehen Systemen kann zu neuen PDUs bzw. zum Wegfall bestehender führen. Übersteigen alte PDUs die Maximalgrenzen von 8 oder 64 Byte Nutzdaten für CAN bzw. CAN-FD, wird es notwendig diese aufzuteilen, sodass die Anzahl der PDUs zunimmt.

Für neue oder weggefallene PDUs gilt:

$$\Delta L_{i,j,k} = \begin{cases} L_{i,j,k} - 0 = L_{i,j,k} & \text{für neue PDUs} \\ 0 - L_{i-1,j,k} = -L_{i-1,j,k} & \text{für weggefallene PDUs} \end{cases} \tag{5.12}$$

D. h. der gesamte Lastanteil einer solchen PDU fällt der Laständerung zu.

Die sich ändernde Anzahl der PDUs wird der K-Matrix-Datenbank entnommen und über deren Releases aufgetragen. Um eine Vergleichbarkeit mit den Änderungen von PDU-Größe und Zykluszeit herzustellen, wird zusätzlich die

Änderung (gegenüber dem vorigen K-Matrix-Release) prozentual zu der Gesamtanzahl im vorliegenden Release eingetragen (s. Abb. 5.4).

Verglichen mit den beiden vorigen Änderungskategorien trägt die Änderung der PDU-Menge mehr zur Änderung der Buslast bei. Auch die Schwankungen der PDU-Anzahl nehmen mit zunehmendem Reifegrad ab. Ab dem Release 2021-17 ist ein erneuter Anstieg zu verzeichnen, der durch beginnende Arbeiten an weiteren, auf der STAR3 Plattform aufbauenden Baureihen ausgelöst wird.

5.2.4 Änderungen der Baudrate

Änderungen der Baudrate wirken sich jeweils auf alle PDUs gleichermaßen aus. Für CAN Busse gilt:

$$L_{i,k} = L_{i-1,k} \cdot \frac{R_{i,k}}{R_{i-1,k}} \tag{5.13}$$

Für CAN-FD Busse gilt dies nur, falls R und R' um den gleichen Faktor geändert werden. Ist das nicht der Fall, führen die, für potenziell jeden Frame unterschiedlichen, Verhältnisse zwischen in Grund- und erhöhter Geschwindigkeit übertragenen Bits, zu unterschiedlichen Auswirkungen auf die Übertragungsdauer $t_{i,j,k}$. Die unterschiedlichen Übertragungsdauern wirken sich abhängig von den Zykluszeiten der Frames unterschiedlich stark aus. Die Buslast muss jeweils mittels Gl. 3.10 – 3.12 neu errechnet werden.

Bei insgesamt 44 CAN- u. CAN-FD Netzwerken über 61 Releases gibt es in der Datenbank 7 Fälle in denen die Grundgeschwindigkeit R und 4 Fälle in denen die erhöhte Baudrate R' vergrößert wurden, alle innerhalb der ersten drei Haupt-Releases. Fälle, in denen die Baudraten verringert wurden, kamen nicht vor. Im Folgenden werden die Baudraten als über den Entwicklungsprozess konstant angenommen.

5.3 Einflussfaktoren bei hierarchischer Versionierung

Definition 5.3 (Grundlast)**.** *Die Auslastung eines Busses über den Entwicklungsverlauf, in einem Prozess ohne hierarchische Versionierung oder Rückwärtskompatibilität, ist die Grundlast.*

Die reguläre Buslast kann sich während der Fahrzeugentwicklung, bedingt durch die inkrementellen Verbesserungen am Gesamtsystem, mit jedem K-Matrix-Release verändern (vgl. Abb. 5.1). Es ändern sich nicht notwendigerweise alle Busse in allen Releases.

Der Anstieg über diese Grundlast hinaus ist der Preis für die zusätzliche Kompatibilität zwischen den K-Matrix-Releases.

Definition 5.4 (Zusatzlast)**.** $\Delta L^{+}_{i,j}$ *sei die zusätzliche Last auf Bus Nr. j im K-Matrix-Release Nr. i.*

Definition 5.5. *Für die Mindestlebensdauer (vgl. Definition 4.5) wird das Formelzeichen δ verwendet. Die Mindestlebensdauer wird bemessen als Anzahl von K-Matrix-Releases.*

Definition 5.6 (Mindestlebensdauerfunktionen)**.** $\mathcal{L}$ *sei die Mindestlebensdauerfunktion mit* $\mathcal{L}_W : P_{i,j} \rightarrow \delta \in \mathbb{N}$ *bzw.* $\mathcal{L}_Y : Y_{i,j} \rightarrow \delta \in \mathbb{N}$.

Die Zusatzlast in einem K-Matrix-Release K_i hängt von den Releases K_{i-1} bis $K_{i-\delta}$ ab. Die Mindestlebensdauern δ unterschiedlicher PDUs können voneinander abweichen (vgl. Regel 8), sodass die Zusatzlast PDU-weise berechnet werden muss.

Definition 5.7 (Zusatzlastanteil)**.** $\Delta L^{+}_{i,j,k}$ *sei der durch die Kompatibilitätsmechanismen eingeführte zusätzliche Lastanteil der PDU* $W_{i,j,k}$ *auf Bus Nr. k im K-Matrix-Release Nr. i.*

Definition 5.8 (Zusatzsignale). *Y_i^+ sei die Menge der Signale, die im Release Nr. i durch Änderungskopien entstehen.*

Kommen im Rahmen der Fahrzeugentwicklung neue Signale hinzu, fallen sie nicht unter diese Definition. Sie werden der veränderlichen Grundlast zugeordnet (vgl. Def. 5.3 u. Abb. 5.1).

Die Anzahl der Datenbits einer mit solchen Zusatzsignalen erweiterten PDU erhält man durch eine Erweiterung von Gleichung 3.5:

$$d_{i,j}^+ = \sum_{\forall Y_{i,k} \in W_{i,j}} s_{i,k} + \sum_{\forall Y_{i,k}^+ \in W_{i,j}} s_{i,k} = d_{i,j} + \sum_{\forall Y_{i,k}^+ \in W_{i,j}} s_{i,k} \tag{5.14}$$

Durch Einsetzen von $d_{i,j}^+$ in Gl. 3.8 bzw. 3.9 erhält man die Anzahlen der für die vergrößerte PDU Nr. j auf Bus Nr. k insgesamt zu übertragenden Bits $n_{i,j,k}^+$ bzw. $n_{i,j,k}^{+'}$. Die Gleichungen 5.6 u. 5.7 können damit angepasst werden, um den Zusatzlastanteil einer PDU zu errechnen. Auf CAN:

$$\Delta L_{i,j,k}^+ = \frac{1}{z_{i,j}} \cdot \left(\frac{n_{i,j,k}^+ - n_{i,j,k}}{R_{i,k}} \right) \Bigg|_{n'_{i,j,k} = n_{i,j,k}^{+'} = 0} \tag{5.15}$$

Und auf CAN-FD:

$$\Delta L_{i,j,k}^+ = \frac{1}{z_{i,j}} \cdot \left(\frac{n_{i,j,k}^{+'} - n'_{i,j,k}}{R'_{i,k}} \right) \Bigg|_{n_{i,j,k} = n_{i,j,k}^+} \tag{5.16}$$

Definition 5.9 (Zusatz-PDUs). *$W_i^+ \in W_i$ sei die Menge der PDUs, die im Release Nr. i durch Kompatibilitätsmechanismen entstehen.*

Bei einer durch die Versionierung eingeführten zusätzlichen PDU, entspricht deren Lastanteil (vgl. Gl. 3.11) dem Zusatzlastanteil.

$$\Delta L_{i,j,k}^+ = L_{i,j,k} | W_{i,j,k} \in W_i^+ \tag{5.17}$$

Die einzelnen Zusatzlastanteile summieren sich zur Zusatzlast.

$$\Delta L^{+}_{i,j} = \sum_{\forall W_{i,j,k} \in W_{i,j}} \Delta L^{+}_{i,j,k} \tag{5.18}$$

Definition 5.10 (Gesamtlast). *$L^{+}_{i,j}$ sei die Gesamtauslastung des Busses Nr. j in Release Nr. i, bestehend aus der Grundlast und der Zusatzlast.*

$$L^{+}_{i,j} = L_{i,j} + \sum_{\forall k \in \{i-1,\ldots,i-\delta\}} \Delta L^{+}_{k,j} \tag{5.19}$$

5.3.1 Parameter

Aus den Regeln zur hierarchischen Versionierung (vgl. Abs. 4.4.5) ergeben sich verschiedene Einflussfaktoren auf die Zusatzlast. Das Erstellen von Änderungskopien (Regel 4) führt zu zusätzlichen Elementen in der Kommunikation. Nach Regel 6 geschieht dies, falls möglich, auf Signal-, sonst auf PDU-Ebene. Falls keine hierarchische Versionierung, sondern einfache Rückwärtskompatibilität angewandt wird, wird nur auf PDU-Ebene erweitert. Entscheidend sind der Füllstand der PDU $d_{i,k}$ (oder im Umkehrschluss die Anzahl der verwendeten Paddingbits $p_{i,k,l}$), die maximale Größe $d_{\max.}(l)$ für Frames auf dem Bus Nr. l (64 Bit für CAN, 512 Bit für CAN-FD) und die Anzahl Bits $s^{+}_{i,j}$, die für die Änderungskopie des Signals $Y^{+}_{i,j}$ übertragen werden müssen.

$$s^{+}_{i,j} <= p_{i,k,l} \Rightarrow \Delta L^{+}_{i,k,l} = 0 \left| \begin{matrix} Y_{i,j} \in W_{i,k} \\ W_{i,k} \in B_{i,l} \end{matrix} \right. \tag{5.20}$$

Kommen durch die Änderungskopie eines Signals maximal so viele Bits hinzu, wie vorher Paddingbits (vgl. Definition 3.54) im Frame enthalten waren, ändert sich die Anzahl der zu übertragenden Bits nicht, und die Buslast bleibt

unverändert. Fällt das Signal größer aus, aber unterhalb des Frame-Limits, steigt der Buslastanteil der PDU durch die Änderungskopie an.

$$p_{i,k,l} < s^{+}_{i,j} <= d_{\text{max.}}(l) - (n_{i,k,l} + n'_{i,k,l}) \Rightarrow \Delta L^{+}_{i,k,l} > 0 \Big|^{Y_{i,j} \in W_{i,k}}_{W_{i,k} \in B_{i,l}} \quad (5.21)$$

Die exakte Größe von $\Delta L^{+}_{i,k,l}$ muss analog zu Gl. 5.6 bzw. 5.7 errechnet werden. Durch die Stufen in Gl. 3.8 bzw. 3.9 kann der Anstieg sprunghaft erfolgen.

Würde die Datenmenge, nach der Änderungskopie des Signals, die maximale Frame-Größe übersteigen, muss die Änderung in der übergeordneten Hierarchieebene erfolgen (vgl. Abschnitt 4.5.4).

$$d_{\text{max.}}(l) - (n_{i,k,l} + n'_{i,k,l}) < s^{+}_{i,j} \Rightarrow \Delta L^{+}_{i,k,l} >= L_{i-1,k,l} \Big|^{Y_{i,j} \in W_{i,k}}_{W_{i,k} \in B_{i,l}} \quad (5.22)$$

Durch das Kopieren der PDU entsteht eine Zusatzlast, die dem Lastanteil der ursprünglichen PDU entspricht. Bei einer Vergrößerung des veränderten Signals kann die, durch die Änderungskopie erzeugte, PDU größer als die ursprüngliche ausfallen, sodass die Zusatzlast größer ausfällt.

Der Vergleich erfolgt in Gleichungen (5.20) bis (5.22) jeweils mit der Anzahl der veränderten Signalbits.

Definition 5.11 (Signaländerungsrate). *$\alpha_{i,j}$ sei die Änderungsrate auf dem Bus Nr. j im Release Nr. i. α gibt an, wie viel Prozent der Bits sich gegenüber dem Vorgängerrelease i-1 verändert haben.*

$$\alpha_{i,j} = \frac{\sum\limits_{\forall Y_k \in B_{i,j} \bigcup B_{i-1,j}} (s'_{i,k} + s^{+}_{i,k} + s^{-}_{i-1,k})}{\sum\limits_{\forall S_y \in B_{k,j}} s_{k,y}} \Bigg|^{s^{+}_{i,k} \to Y_{i-1,k} \notin B_{i-1,j}}_{s^{-}_{i,k} \to Y_{i+1,k} \notin B_{i+1,j}}_{s'_{i,k} \to Y_{i,k} \neq Y_{i-1,k}} \quad (5.23)$$

Unter der Annahme, dass sich die Fälle aus Gl. 5.20 und 5.21 die Waage halten, liegt in diesen Fällen die Zusatzlast jeweils in der Größenordnung der

Änderungsrate $\alpha_{i,j}$ (d. h. die Stufen aus Abb. 3.9 gleichen sich im Mittel aus). Im Fall von Gleichung (5.22) beträgt die Zusatzlast den Lastanteil einer PDU – ausgehend vom selben, veränderten Signal. Der entscheidende Unterschied ist der ursprüngliche Füllstand der jeweiligen PDU.

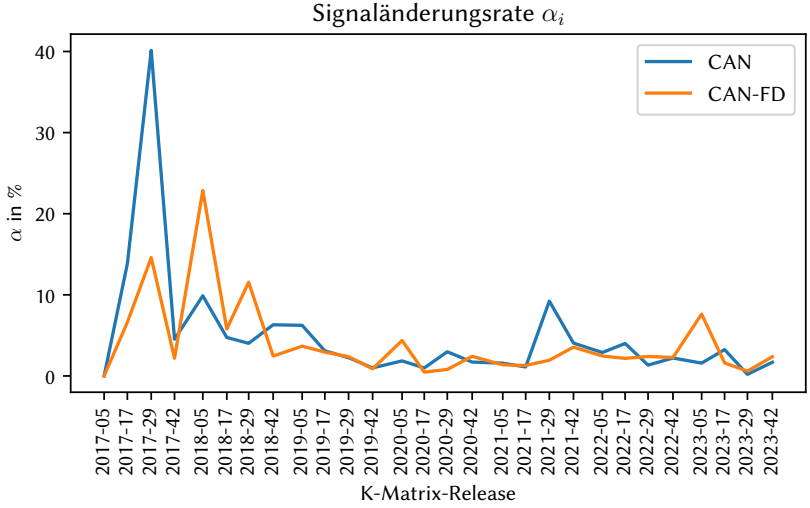

Abbildung 5.5: Signaländerungsrate α_i im zeitlichen Verlauf

Formel 5.23 kann auf die in der K-Matrix-Datenbank enthaltenen Signale u. PDUs angewandt werden, um die realen Änderungsraten zu ermitteln. Dabei werden alle veränderten Signale mit verändertem Datenbankobjekt herangezogen (vgl. Abs. 5.1.2). Es resultieren die Graphen für CAN und CAN-FD in Abbildung 5.5, wobei der Übersicht halber ein Mittelwert über alle Busse (der jeweiligen Technologie) gebildet und die b- und c-Releases ausgeblendet wurden. Da in diesen K-Matrix-Releases nur geringfügige Korrekturen durchgeführt werden, sind die Werte jeweils kleiner als zu den a-Releases.

Der in Gleichung 5.22 beschriebene dritte Fall, indem eine komplette PDU kopiert werden muss, muss für alle PDUs mit verändertem Inhalt betrachtet werden. Die Menge der veränderten PDUs ist nicht aus $\alpha_{i,j}$ ableitbar. Hohe

Werte für $\alpha_{i,j}$ sind gleichermaßen durch wenige PDUs mit vielen veränderten Signalen, wie durch viele PDUs mit jeweils nur einzelnen geänderten Signalen erreichbar. Stattdessen wird die Quote, wie viele der in einem gegebenen K-Matrix-Release i vom Vorgänger-Release i-1 abweichen, als ein weiterer Parameter definiert.

Definition 5.12 (PDU-Änderungsquote). *$\beta_{i,j}$ sei das Verhältnis zwischen der Anzahl der veränderten PDUs sowie aller PDUs auf einem Bus Nr. j eines K-Matrix-Releases Nr. i.*

$$\beta_{i,j} = \frac{|B'_{i,j}|}{|B_{i,j}|} \Big|_{B'_{i,j}=\{W_{i,k} | W_{i,k} \in B_{i,j} \wedge (W_{i,k} \neq W_{i-1,k} \vee \nexists W_{i-1,k})\}} \tag{5.24}$$

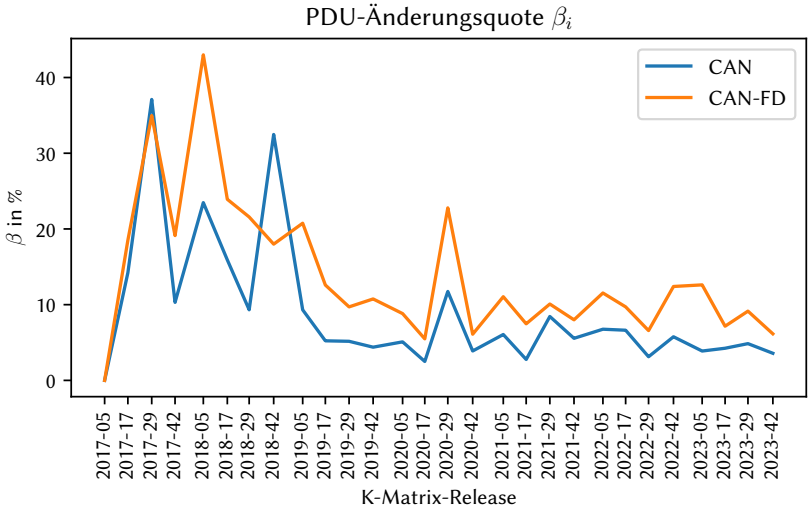

Abbildung 5.6: PDU-Änderungsquote β_i im zeitlichen Verlauf

Aus der vorliegenden K-Matrix-Datenbank ergibt sich (ohne b- u. c-Releases, jew. alle Busse einer Technologie gemittelt) ein Verlauf wie in Abbildung 5.6 dargestellt. Im Vergleich mit Abb. 5.5 weisen beide denselben Trend (weniger Änderungen zum Ende der Entwicklungsphase) auf, aber in einzelnen Releases (z. B. R2017-29 o. R2018-42) voneinander ab.

Bei jeder der veränderten PDUs kann eine vollständige Kopie notwendig sein. Gemäß Gl. 5.22 hängt dies vom Füllstand der jeweiligen PDU ab.

Definition 5.13 (PDU-Füllfaktor)**.** *$\gamma_{i,j,k}$ sei der Füllfaktor einer PDU k auf Bus j in Release i. $\gamma_{i,j}$ ist der mittlere PDU-Füllfaktor über alle PDUs auf Bus j in Release i.*

$$\gamma_{i,j,k} = \frac{d_{i,k}}{d_{\max.}(j)} \tag{5.25}$$

$$\gamma_{i,j} = \frac{1}{|B_{i,j}|} \sum_{\forall W_{i,k} \in B_{i,j}} \gamma_{i,j,k} \tag{5.26}$$

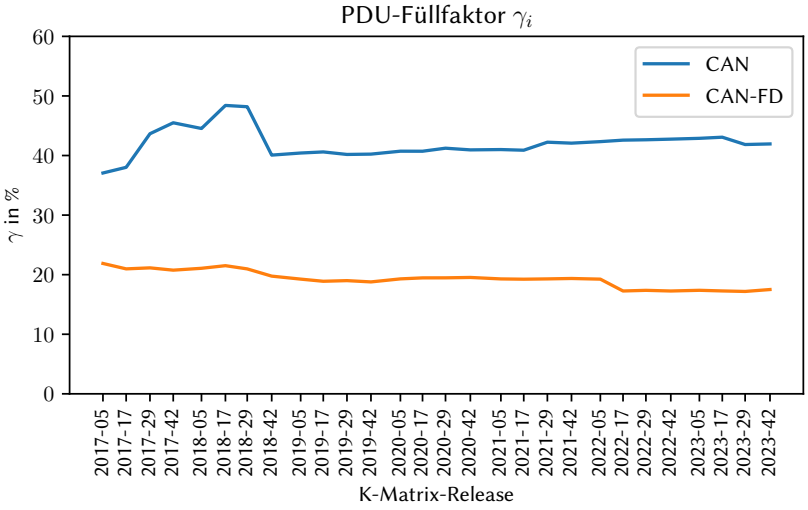

Abbildung 5.7: PDU-Füllfaktor im zeitlichen Verlauf

Bei einem PDU-Füllfaktor von 100 % sind alle PDUs vollständig gefüllt. Änderungskopien müssen auf PDU-Ebene erfolgen, für die Zusatzlast gilt Gl. 5.22. Liegt der Faktor nahe null, werden Gl. 5.20 bzw. 5.21 überwiegen. $\gamma_{i,j}$ kann gemäß Formel 5.26 aus der K-Matrix-Datenbank errechnet werden und ist (gemittelt über alle Busse) in Abb. 5.7 aufgetragen. Auffällig ist, dass die CAN-FD-PDUs einen um die Hälfte kleineren Füllfaktor aufweisen.

Theoretisch ist ein hoher Füllfaktor vorteilhaft, da pro Datenbit anteilig weniger Overhead entsteht (vgl. Abb. 3.9). In der Praxis kann dieser Vorteil nur genutzt werden, wenn es große (d. h. in Summe möglichst nah am Maximum von 64 oder 512 Bit) Gruppen von Signalen gibt, die sich Sender, Empfänger und Zykluszeit teilen. Werden Signale mit unterschiedlichen Zykluszeiten in einer PDU gruppiert, müssen sie gemeinsam gesendet werden. Der Teil mit der größeren Zykluszeit wird dadurch häufiger gesendet wie nötig. So wird ein unnötiger Anstieg der Bandbreite verursacht, der den Gewinn durch den verringerten Overhead durch die Gruppierung aufhebt. Werden Signale für unterschiedliche Empfänger gruppiert, kann dies je nach Netzwerktopologie

und durch Gateways durchgeführte Weiterleitungen zur Folge haben, dass Signale auf Bussen übertragen werden, auf denen diese nicht benötigt werden. Dies stellt ebenfalls einen unnötigen Bandbreitenzuwachs dar, der gegen übermäßiges Gruppieren von Signalen, zugunsten des Overheads, spricht.

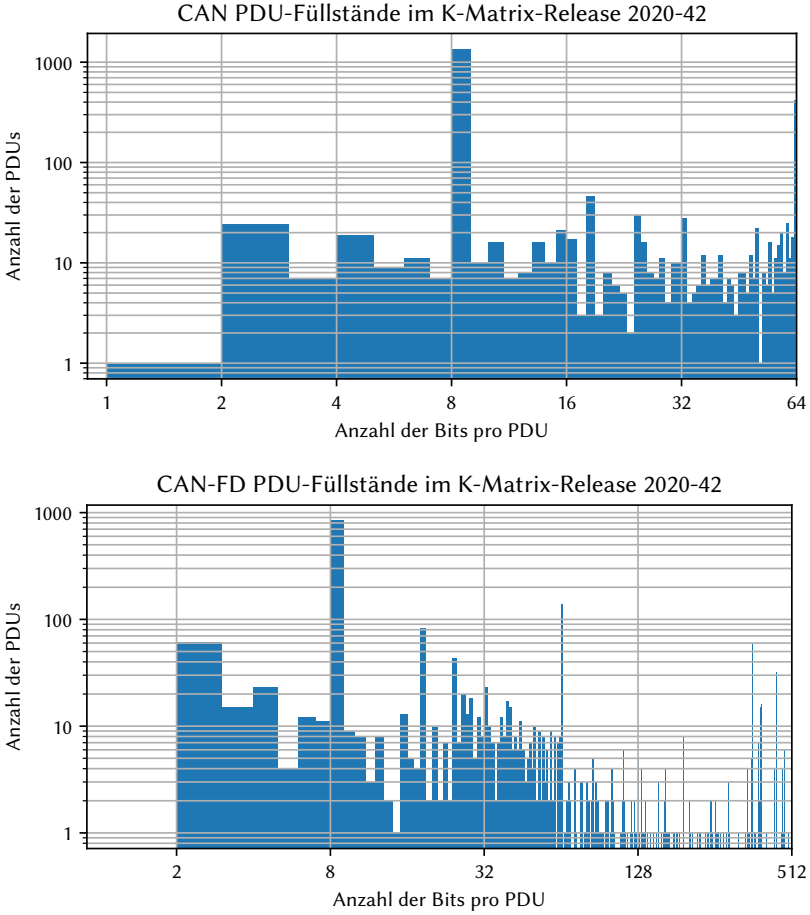

Abbildung 5.8: Häufigkeit unterschiedlicher PDU-Füllstände für CAN und CAN-FD

Wählt man eines der K-Matrix-Releases in der Reifephase, z. B. R2020-42, aus, und trägt die Häufigkeit unterschiedlicher PDU-Füllstände als Histogramme auf, erhält man Abb. 5.8. Bei beiden Bustypen kommen beliebige Größen an Nutzdatenmengen vor, wobei Zweierpotenzen häufiger auftreten als andere Werte. Jeweils am häufigsten kommen 8 und am zweithäufigsten 64 Datenbits vor. Dem CAN-FD Diagramm ist zu entnehmen, dass Frames mit mehr als 64 Datenbits seltener vorkommen als solche mit weniger. Durch die Häufigkeit der kleineren Frames ergibt sich der in Abb. 5.7 sichtbare Unterschied zwischen CAN und CAN-FD.

Die Faktoren $\alpha_{i,j}$, $\beta_{i,j}$ u. $\gamma_{i,j}$ beeinflussen die in einem K-Matrix-Release K_i auf dem Bus B_j durch die Versionierung eingeführte zusätzliche Buslast. Durch die Regeln 7, 10 u. 12 ergibt sich, dass die erzeugte Zusatzlast nach Ablauf einer definierten Zeitspanne mit Erreichen eines Haupt-Releases wieder entfernt wird.

Definition 5.14 (Mittlere Mindestlebensdauer)**.** *$\delta_{i,j}$ sei die mittlere Mindestlebensdauer der PDUs in Release Nr. i auf Bus Nr. j. Die mittlere Mindestlebensdauer der PDUs auf Bus Nr. j über alle Releases ist δ_j.*

$$\delta_{i,j} = \frac{1}{|B_{i,j}|} \sum_{\forall W_{i,k} \in B_{i,j}} \mathcal{L}_W(W_{i,k}) \tag{5.27}$$

$$\delta_j = \frac{1}{|\mathbb{K}|} \sum_{\forall K_i \in \mathbb{K}} \delta_{i,j} \tag{5.28}$$

Die mittlere Mindestlebensdauer dient zur vereinfachten Abschätzung der Zusatzlast, wenn für unterschiedliche PDUs unterschiedliche Mindestlebensdauern verwendet werden.

5.3.2 Abschätzung

Die vorgestellten Parameter $\alpha_{i,j}$, $\beta_{i,j}$ und $\gamma_{i,j}$ sind jeweils gegebene Eigenschaften des Netzwerks, genauer gesagt dessen Entwicklungsprozesses. δ_j und $\Delta L^+_{i,j}$ sind die Zielgrößen, die gegeneinander abgewogen werden müssen.

Ausgehend von Gl. 5.18 und den Gl. 5.20 u. 5.21 sind pro K-Matrix-Release ein zusätzlicher Anstieg der Buslast in der Größenordnung der Signaländerungsrate $\alpha_{k,j}$ zu erwarten:

$$\frac{L^+_{i,j}}{L_{i,j}} = 1 + \delta_j \cdot \alpha_{i,j} \tag{5.29}$$

$$\frac{L_{i,j} + \Delta L^+_{i,j}}{L_{i,j}} = 1 + \delta_j \cdot \alpha_{i,j} \tag{5.30}$$

$$1 + \frac{\Delta L^+_{i,j}}{L_{i,j}} = 1 + \delta_j \cdot \alpha_{i,j} \tag{5.31}$$

$$\frac{\Delta L^+_{i,j}}{L_{i,j}} = \delta_j \cdot \alpha_{i,j} \tag{5.32}$$

$$\Delta L^+_{i,j} = \delta_j \cdot \alpha_{i,j} \cdot L_{i,j} \tag{5.33}$$

Diese erste Näherung bildet die Möglichkeit, dass PDUs bei Erreichen von $d_{max.}$ vollständig kopiert werden müssen, nicht ab. Die Wahrscheinlichkeit, dass eine veränderte PDU kopiert werden muss, ergibt sich aus dem PDU-Füllfaktor $\gamma_{i,j}$. (Bei leeren PDUs, d. h. $\gamma = 0\%$, ist eine Erweiterung immer möglich. Bei vollen PDUs, d. h. $\gamma = 100\%$, ist eine Kopie zwingend erforderlich.) Nach Gl. 5.22 kann bei einer zu kopierenden PDU der Zusatzlastanteil

größer sein als der Lastanteil der ursprünglichen PDU. Dies wäre der Fall, wenn das veränderte Signal größer als das ursprüngliche Signal wäre. Da diese Komponente der Zusatzlast durch $\alpha_{i,j}$ berücksichtigt wird, wird bei den zu kopierenden PDUs angenommen, dass der Zusatzlastanteil dem Lastanteil der ursprünglichen PDU entspricht.

Gl. 5.29 wird erweitert zu:

$$\frac{L^{+}_{i,j}}{L_{i,j}} = 1 + \delta_j \cdot \alpha_{i,j} + \delta_j \cdot \beta_{i,j} \cdot \gamma_{i,j} \tag{5.34}$$

Oder mit Umformungen analog zu Gl. 5.33:

$$\Delta L^{+}_{i,j} = \delta_j \cdot (\alpha_{i,j} + \beta_{i,j} \cdot \gamma_{i,j}) \cdot L_{i,j} \tag{5.35}$$

Setzt man statt hierarchischer Versionierung auf einfache Rückwärtskompatibilität, werden Änderungskopien nur auf einer, der höchsten, Hierarchieebene (in diesem Fall der PDU-Ebene) durchgeführt. Dies kann dargestellt werden, indem man $\gamma = 100\%$ setzt. Gl. 5.35 vereinfacht sich dadurch zu:

$$\Delta L^{+}_{i,j} = \delta_j \cdot (\alpha_{i,j} + \beta_{i,j}) \cdot L_{i,j} \tag{5.36}$$

Für alle $\gamma < 100\%$ ist $\Delta L^{+}_{i,j}$ bei einfacher Rückwärtskompatibilität größer als bei hierarchischer Versionierung.

Diese Formeln ermöglichen eine Abschätzung der Zusatzlast bei einer gewählten mittleren Mindestlebensdauer. Hat man sich für die Anwendung hierarchischer Versionierung entschieden, kann Gl. 5.35 umgestellt werden, um bei gegebenen Netzwerkeigenschaften und einem gewählten Maximalwert für die Gesamtlast $L_{i,j}$, den größtmöglichen Wert für die Mindestlebensdauer δ_j zu bestimmen:

$$\frac{\Delta L^{+}_{i,j}}{(\alpha_{i,j} + \beta_{i,j} \cdot \gamma_{i,j}) \cdot L_{i,j}} = \delta_j \tag{5.37}$$

$$\frac{L^{+}_{i,j,max.} - L_{i,j}}{(\alpha_{i,j} + \beta_{i,j} \cdot \gamma_{i,j}) \cdot L_{i,j}} = \delta_j \tag{5.38}$$

$$\frac{L^{+}_{i,j,max.}}{(\alpha_{i,j} + \beta_{i,j} \cdot \gamma_{i,j}) \cdot L_{i,j}} - 1 = \delta_j \tag{5.39}$$

Tabelle 5.1: Anstieg der Buslast bei einfacher Rückwärtskompatibilität für unterschiedliche α, β u. δ.

α_k	β_k	δ	$\frac{\Delta L^{+}_{k}}{L_k}$	δ	$\frac{\Delta L^{+}_{k}}{L_k}$	δ	$\frac{\Delta L^{+}_{k}}{L_k}$
5 %	5 %	1	10 %	2	20 %	3	30 %
15 %	5 %	1	20 %	2	40 %	3	60 %
40 %	5 %	1	45 %	2	90 %	3	135 %
5 %	15 %	1	20 %	2	40 %	3	60 %
15 %	15 %	1	30 %	2	60 %	3	90 %
40 %	15 %	1	55 %	2	110 %	3	165 %
5 %	40 %	1	45 %	2	90 %	3	135 %
15 %	40 %	1	55 %	2	110 %	3	165 %
40 %	40 %	1	80 %	2	160 %	3	240 %

Der Anstieg einer gegebenen Buslast bei unterschiedlichen Werten für α, β, γ u. δ, jeweils in Wertebereichen wie in Abb. 5.5, 5.6 und 5.7, ist in Tab. 5.1 für einfache Rückwärtskompatibilität und Tab. 5.2 für hierarchische Versionierung gemäß der Formeln 5.35 u. 5.36 berechnet.

Diese Ergebnisse werden zusammen mit denen aus Abschnitt 5.4 in Abschnitt 6.1 diskutiert.

Tabelle 5.2: Anstieg der Buslast bei hierarchischer Versionierung für unterschiedliche α, β, γ u. δ.

α_k	β_k	γ	δ	$\frac{\Delta L_k^+}{L_k}$	δ	$\frac{\Delta L_k^+}{L_k}$	δ	$\frac{\Delta L_k^+}{L_k}$
5 %	5 %	20 %	1	6 %	2	12 %	3	18 %
15 %	5 %	20 %	1	16 %	2	32 %	3	48 %
40 %	5 %	20 %	1	41 %	2	82 %	3	123 %
5 %	15 %	20 %	1	8 %	2	16 %	3	24 %
15 %	15 %	20 %	1	18 %	2	36 %	3	54 %
40 %	15 %	20 %	1	43 %	2	86 %	3	129 %
5 %	40 %	20 %	1	13 %	2	26 %	3	39 %
15 %	40 %	20 %	1	23 %	2	46 %	3	69 %
40 %	40 %	20 %	1	48 %	2	96 %	3	144 %
5 %	5 %	30 %	1	7 %	2	13 %	3	20 %
15 %	5 %	30 %	1	17 %	2	33 %	3	50 %
40 %	5 %	30 %	1	42 %	2	83 %	3	125 %
5 %	15 %	30 %	1	10 %	2	19 %	3	29 %
15 %	15 %	30 %	1	20 %	2	39 %	3	59 %
40 %	15 %	30 %	1	45 %	2	89 %	3	134 %
5 %	40 %	30 %	1	17 %	2	34 %	3	51 %
15 %	40 %	30 %	1	27 %	2	54 %	3	81 %
40 %	40 %	30 %	1	52 %	2	104 %	3	156 %
5 %	5 %	40 %	1	7 %	2	14 %	3	21 %
15 %	5 %	40 %	1	17 %	2	34 %	3	51 %
40 %	5 %	40 %	1	42 %	2	84 %	3	126 %
5 %	15 %	40 %	1	11 %	2	22 %	3	33 %
15 %	15 %	40 %	1	21 %	2	42 %	3	63 %
40 %	15 %	40 %	1	46 %	2	92 %	3	138 %
5 %	40 %	40 %	1	21 %	2	42 %	3	63 %
15 %	40 %	40 %	1	31 %	2	62 %	3	93 %
40 %	40 %	40 %	1	56 %	2	112 %	3	168 %

5.4 Simulation

Die auf einigen Annahmen u. Vereinfachungen (z. B. dass Gleichungen (5.20) und (5.21) sich die Waage halten, die Vernachlässigung der Zykluszeitänderungen, oder die Mittelwertbildung über alle Busse) basierenden Schätzfunktion weist eine große Spannweite der möglichen Buslastzunahme auf (vgl. Tabellen 5.1 und 5.2). Einen realen Entwicklungsprozess über mehrere Jahre unter Anwendung von hierarchischer Versionierung durchzuführen, ist aus Zeit- und Kostengründen nicht umsetzbar. Ein Vergleich der Auswirkung unterschiedlicher Werte für δ und zwischen hierarchischer Versionierung und einfacher Rückwärtskompatibilität wäre nicht möglich. Eine teilweise Anwendung beschränkt auf einzelne Busse oder Steuergeräte wird durch die fahrzeugweite Gültigkeit der K-Matrix verhindert.

Stattdessen wurde auf Basis der historischen Prozessdaten (vgl. Abs. 5.1) ein Simulationsprogramm erstellt. Dieses wendet die Regeln der hierarchischen Versionierung auf die Releases der realen K-Matrix-Datenbank an, und erzeugt eine alternative K-Matrix, bei der ein Teil der Releases kompatibel ist. Die einzige vorgenommene Vereinfachung ist, eine einheitliche Mindestlebensdauer δ für alle Elemente anzunehmen. Über diese Dauer werden alle Elemente kompatibel gehalten. Zum ersten Release nach deren Ablauf werden alle durch Änderungskopien duplizierte Elemente entfernt, sodass dieses Release der Grundlast entspricht. Von dieser Vereinfachung abgesehen werden die Lastanteile jeder einzelnen PDU bitgenau gemäß Gl. 3.3 – 3.12 berechnet, wobei sich verändernde Zykluszeiten und Busgeschwindigkeiten berücksichtigt werden, und für jede PDU individuell entschieden wird, ob Änderungskopien auf Signalebene möglich sind, oder die gesamte PDU kopiert werden muss.

Eine ausführliche Beschreibung der Entwicklung der Simulationssoftware und den dabei aufgetretenen Herausforderungen befindet sich in Anhang B.

Die Simulation wird mit drei Parametern konfiguriert. Der eine Parameter ist die Mindestlebensdauer (vgl. Definitionen 4.5 und 5.5), über welche die Elemente kompatibel gehalten werden. Wie in Abs. 5.3.2 gezeigt, ist mit größer

werdendem δ eine höhere Zusatzlast zu erwarten. Der zweite Parameter ist ein Offset, ab dem die Kompatibilitätsmechanismen aktiviert werden. Durch die Durchführung mit unterschiedlichen Offsets sollen mögliche Abweichungen durch einzelne, änderungsstarke Releases erkannt werden.

Definition 5.15 (Offset)**.** *ϵ sei der Offset als Anzahl Releases, ab dem ersten Release, ab dem Kompatibilitätsmechanismen aktiviert werden.*

Der dritte Parameter gibt die Anzahl der verwendeten Hierarchieebenen an. Wird für die Kompatibilitätsmechanismen eine Ebene verwendet, entspricht dies einfacher Rückwärtskompatibilität. Zwei Hierarchieebenen entsprechen hierarchischer Versionierung.

Definition 5.16. *ζ sei die Hierarchietiefe, d. h. die Anzahl der für Kompatibilitätsmechanismen verwendeten Hierarchieebenen.*

Unabhängig von Offset und Mindestlebensdauer werden b- und c-Releases jeweils kompatibel zu den a-Releases gehalten. Die Mindestlebensdauer δ wird nicht als Zeitdauer angegeben, sondern als Anzahl von a-Releases (vgl. Definition 5.5). (Der exakte Erscheinungstermin eines a-Release hängt von den jeweiligen Wochen- u. ggf. Feiertagen ab, die Dauer zwischen zweier solcher Releases unterliegt daher einer geringen Unschärfe.) Die schwankende Anzahl der b-, c- ... -Releases hat, abgesehen vom Lastzuwachs durch die so eingebrachten Änderungen, keinen Einfluss auf die Mindestlebensdauer.

Die Auswertung wurde für sämtliche CAN- und CAN-FD-Netzwerke der STAR3-Plattform vorgenommen. Der Übersicht halber werden hier jeweils die vier in Abb. 5.1 gezeigten Busse abgebildet. In allen Diagrammen ist jeweils die Grundlast in Schwarz eingetragen. Die von Mercedes-Benz intern festgelegte Obergrenze von 55 %[1] statischer Auslastung auf CAN- und CAN-FD-Bussen ist jeweils in Rot eingetragen.

[1] Auf das Zustandekommen des Grenzwerts und dessen Übertragbarkeit auf unterschiedliche Fahrzeughersteller wird in Abs. 6.1.1 eingegangen.

5.4.1 Variation des Offsets

Für den Offset sind, je nach gewählter Mindestlebensdauer δ, unterschiedliche Werte wählbar. Bei einer Mindestlebensdauer von $\delta = 1$ folgt jeweils ein kompatibles auf ein inkompatibles a-Release. Offsets von 0 und 1 sind sinnvoll. Ein Offset von 2 (4, 6 ...) hätte jeweils denselben Effekt wie 0 und 3 (5, 7 ...) entsprächen 1. Bei einer Mindestlebensdauer von $\delta = 3$ folgen einem inkompatiblen a-Release drei kompatible, sodass Offset-Werte von 0 bis 3 möglich sind.

Die Abbildungen 5.9 – 5.12 stellen die Grundlast sowie die Simulationsergebnisse für eine Mindestlebensdauer $\delta = 1$ und die beiden möglichen Offset-Werte dar.

Betrachtet man die Grundlast, fällt auf, dass diese in Abb. 5.9 und 5.10 die Grenze von 55 % überschreitet. Im Kontext des gesamten Fahrzeugentwicklungsprozesses bzw. des V-Modells (vgl. Abs. 3.2.4) fallen diese Übertretungen in den mittleren Bereich des V-Modells, indem die Einzelkomponenten implementiert, aber noch nicht zu (Teil-)Systemen integriert werden. Die Auslastung eines Busses wird ab dem Punkt relevant, ab dem die ersten (Teil-)Systeme, z. B. im Rahmen von Cluster-HIL Aufbauten (vgl. Abs. 3.2.5), getestet werden.

Ab R2018-29 wird der Grenzwert überall eingehalten. Im Vergleich mit Abb. 5.3 und 5.4 fällt auf, dass die Reduktion der PDUs in R2018-17 und die verhältnismäßig hohe Zahl der Zykluszeitvergrößerungen in R2018-29 mit der Verringerung der Grundlast im Fahrerassistenzbus zusammenfallen. Der Abfall ist, in unterschiedlicher Ausprägung, auf einigen Bussen zu beobachten, auf anderen (u. a. den drei hier abgebildeten) nicht.

Die vier vorgestellten Busse weisen ab R2018-29 Grundlasten in unterschiedlichen Bereichen (Fahrerassistenz u. Fahrwerk: $\approx$ 50%; Karosserie: $\approx$ 40%; Motorsteuerung: $\approx$ 10%) auf. Entsprechend bieten sich unterschiedliche Spielräume für Erhöhungen der Buslast durch Kompatibilitätsmechanismen. Im CAN-Bus der Motorsteuerung würde bei einer Vervierfachung der Last der Grenzwert noch eingehalten. Fahrwerk und Fahrerassistenz liegen nur ca. 10 % unterhalb der Grenze.

Betrachtet man die Kurven für zwei Offset-Werte, sieht man, dass diese entweder mit der Grundlast übereinstimmen oder sie übertreffen. Das deckt sich mit der Erwartung, durch die Kompatibilitätsverfahren eine zusätzliche Last (unterschiedlichen Umfangs) zu erzeugen.

Im Vergleich zwischen den unterschiedlichen Offset-Werten fällt auf allen Bussen die Zusatzlast bei einem Offset von 1 geringer aus als bei 0. Die Ursache liegt darin, dass der in Abs. 4.1 beschriebene Konflikt zwischen Schnelllebigkeit und Beständigkeit aktuell so gelöst wird, dass einige der beständigeren Steuergeräte ihre Schnittstellen nicht im Rahmen des vierteljährlichen Zyklus, sondern nur halbjährlich (zu den KW05 bzw. KW29 a-Releases) ändern. In den Releases dazwischen dürfen in den Schnittstellen der betroffenen ECUs eingesetzte Elemente nicht verändert werden. In den Kurven für α (Abb. 5.5) und β (Abb. 5.6) ist dies an geringeren Werten für die betreffenden Releases zu erkennen. Gemäß Gl. 5.35 führen höhere Werte für α und β zu größeren Zusatzlasten.

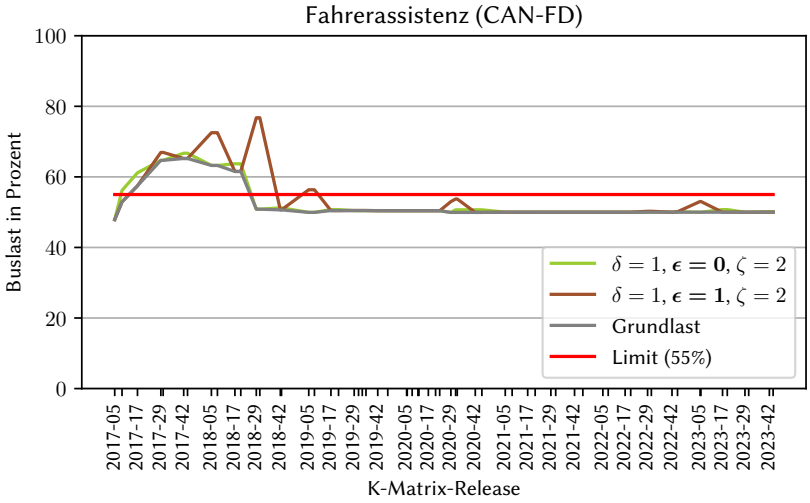

Abbildung 5.9: Unterschiedliche Offsets

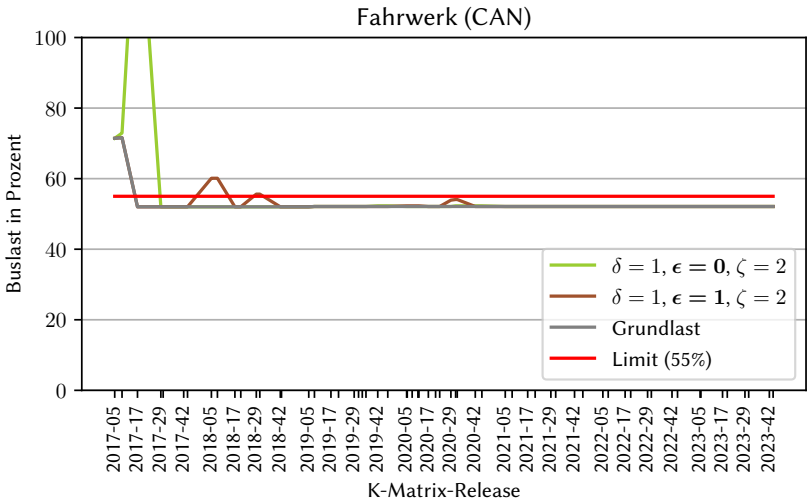

Abbildung 5.10: Unterschiedliche Offsets

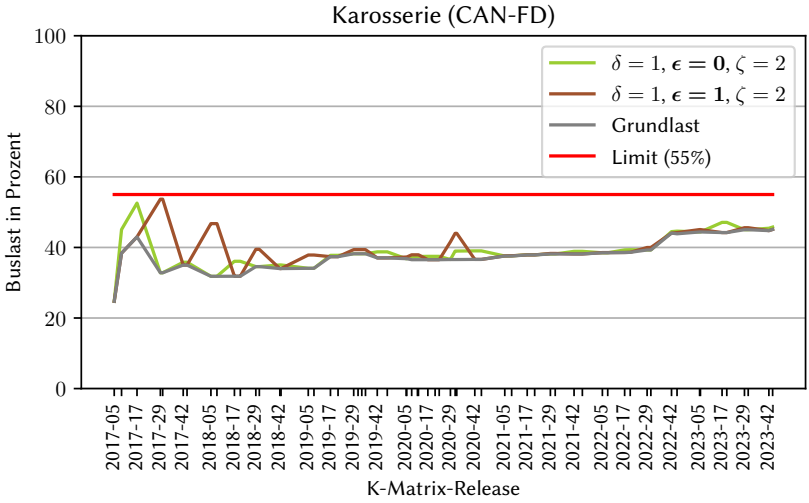

Abbildung 5.11: Unterschiedliche Offsets

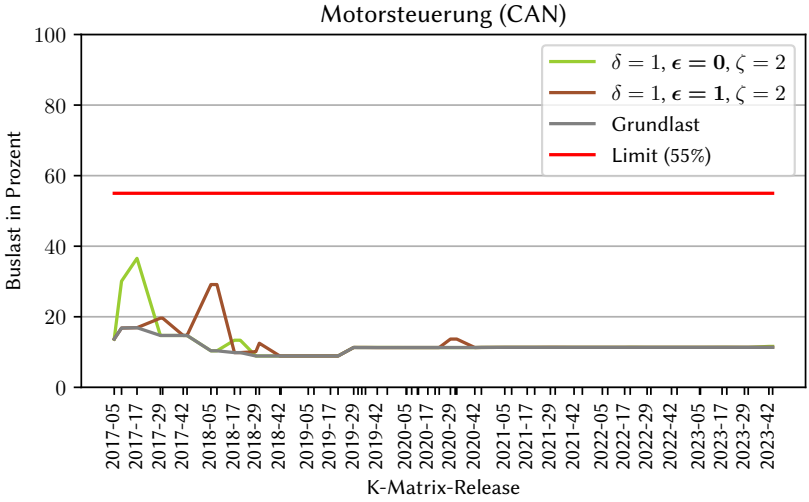

Abbildung 5.12: Unterschiedliche Offsets

5.4.2 Variation der Mindestlebensdauer

Für die folgenden Simulationen wird konstant ein Offset von 1 verwendet, um die Mindestlebensdauern der Objekte an den realen Entwicklungszyklen zu orientieren. Bei einer Mindestlebensdauer von $\delta = 1$ wird jeweils Kompatibilität zwischen zwei a-Releases hergestellt. Die Steuergeräte aus einem Zeitfenster von 6 Monaten werden kombinierbar. Werden nicht ein, sondern drei Releases kompatibel gemacht (d. h. $\delta = 3$), erhöht sich diese Spanne auf 1 Jahr. Die resultierenden Buslasten sind in Abb. 5.13 – 5.16 aufgetragen.

Den Wert von δ auf zwei zu setzen, wäre nicht sinnvoll. Mit einer Gesamtdauer von 9 Monaten würden die halbjährlichen Zyklen der beständigeren Steuergeräte jeweils verbunden (d. h. die größeren Werte für α und β würden in Kauf genommen). D. h. der resultierende Buslastanstieg wäre ähnlich hoch wie bei $\delta = 3$, bei einer geringeren Mindestlebensdauer.

Die hohen Werte für α und β in der ersten Hälfte des Entwicklungsprozesses wirken sich bei $\delta = 3$ auf die Zusatzlast aus und führen im Fall der Netzwerke, deren Grundlast sich nahe dem Grenzwert befindet, für zusätzliche Überschreitungen. Bei Karosserie und Motorsteuerung werden die Grenzwerte durchgängig eingehalten.

Das Beispiel des Fahrwerks CAN-Bus zeigt, dass eine hohe Grundlast nicht zwingend die Anwendung von Änderungskopien infrage stellt. Bei hinreichend wenigen Änderungen kann die Kompatibilität über ein volles Jahr ohne Überschreitung des Buslastlimits hergestellt werden.

In der ersten Hälfte der Entwicklungsphase werden die Grenzwerte für $\delta = 3$ überschritten, während sie (wo nicht bereits die Grundlast über dem Limit liegt) für $\delta = 1$ eingehalten werden. Zusammen mit der im vorigen Abschnitt durchgeführten Einordnung in den Gesamtfahrzeugentwicklungsprozess lässt sich das Konzept dahin gehend erweitern, dass in unterschiedlichen Entwicklungsphasen unterschiedliche Werte für δ vorgegeben werden. Dieses Vorgehen wird in Abs. 6.1.3 weiter ausgeführt.

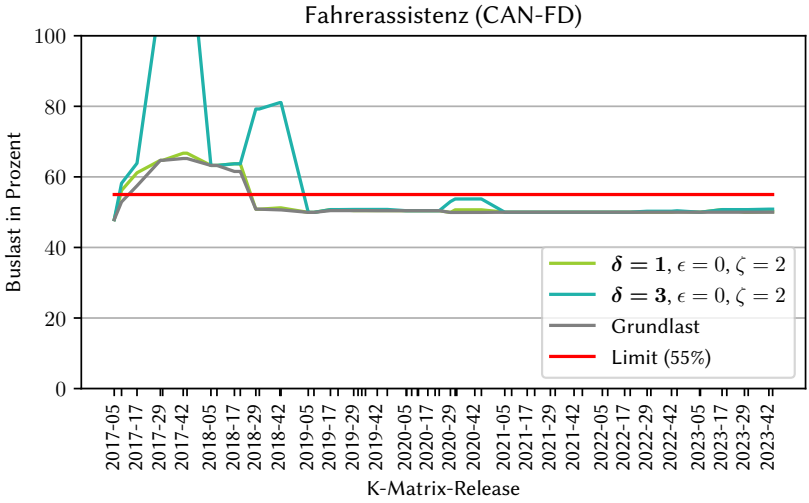

Abbildung 5.13: Unterschiedliche Mindestlebensdauern

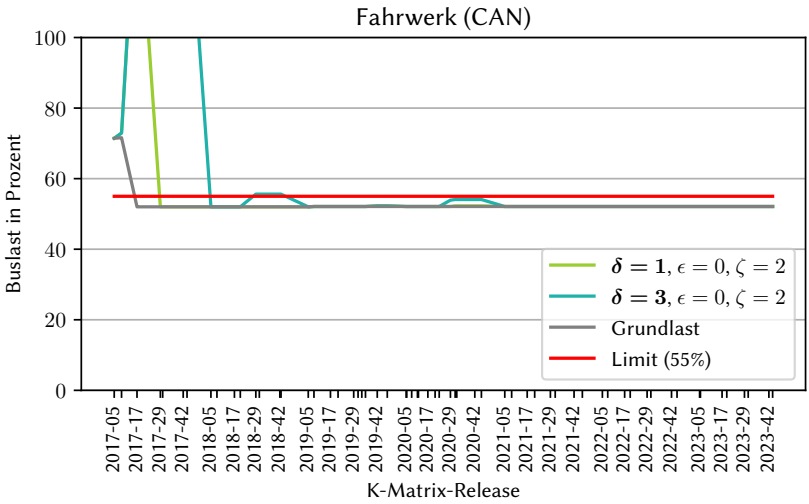

Abbildung 5.14: Unterschiedliche Mindestlebensdauern

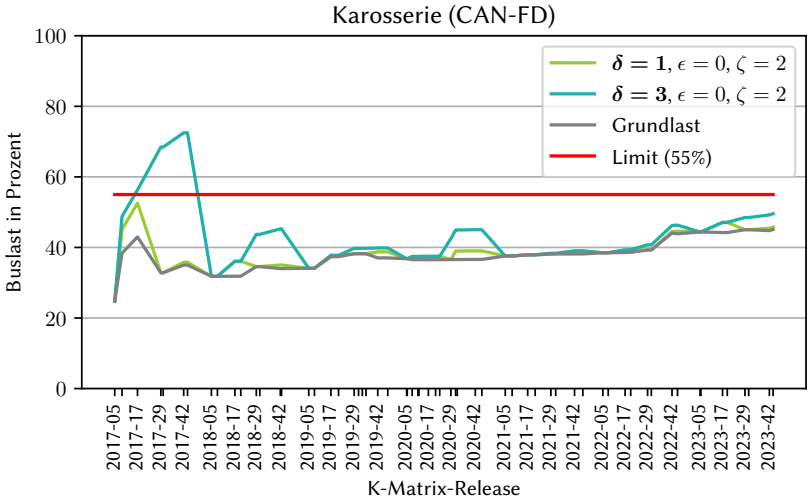

Abbildung 5.15: Unterschiedliche Mindestlebensdauern

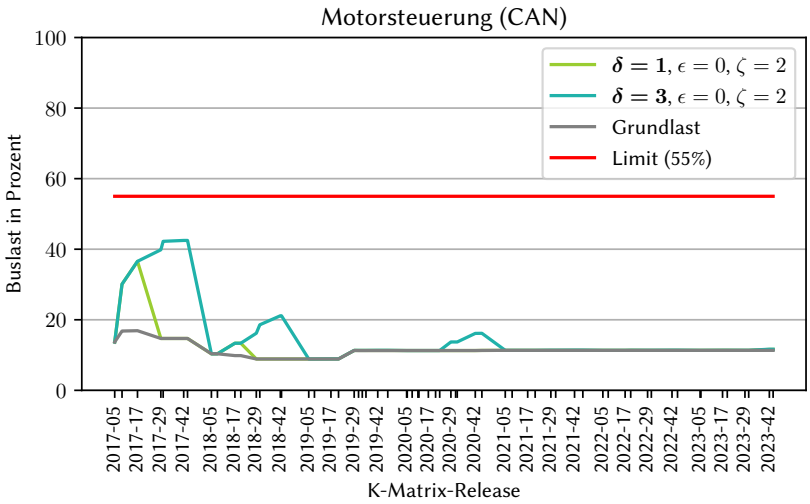

Abbildung 5.16: Unterschiedliche Mindestlebensdauern

5.4.3 Hierarchische Versionierung u. einfache Rückwärtskompatibilität

Um zu beantworten, ob die parallele Umsetzung eines Kompatibilitätsmechanismus auf mehreren Hierarchieebenen der Kommunikationsstruktur sinnvoll ist, wird der Vergleich mit einem einschichtigen Verfahren durchgeführt. Bei einfacher Rückwärtskompatibilität wird bei Änderungen einzelner Signale einer PDU die gesamte PDU in ihrer originalen und in ihrer veränderten Form übertragen. Bei hierarchischer Versionierung wird – falls das möglich ist – die PDU um das veränderte Signal erweitert, während das Originalsignal bestehen bleibt. Falls, z. B. durch die bustechnologiebedingte Größenbegrenzung für PDUs, eine Erweiterung auf dieser Ebene nicht möglich ist, wird wie bei der einfachen Rückwärtskompatibilität die gesamte PDU in verändertem und Originalzustand übertragen.

Mit einem Offset von 0 und einer Mindestlebensdauer von jeweils 1 wurden Simulationen für beide Verfahren durchgeführt. Die Ergebnisse sind in Abb. 5.17 – 5.20 dargestellt.

In allen Fällen übertrifft die zusätzliche Buslast des einschichtigen Verfahrens die des zweischichtigen, teilweise um mehr als Faktor zwei. Es kommt zu mehr Grenzwertüberschreitungen.

Nach den Formeln 5.35 u. 5.36 fällt bei einfacher Rückwärtskompatibilität die PDU-Änderungsquote β genauso stark ins Gewicht wie die Signaländerungsrate α. Bei hierarchischer Versionierung wird β mit γ gewichtet. Der höhere Einfluss der PDU-Änderungsquote fällt bei niedriger Signaländerungsrate auf. Da letztere mit zunehmendem Reifegrad stärker abfällt als erstere (vgl. Abb. 5.5 u. 5.6), nimmt der Abstand mit dem Reifegrad zu.

Für sich betrachtet wird das Buslastlimit mit einfacher Rückwärtskompatibilität, bei den Bussen mit größerem Abstand zwischen Grundlast und Limit, nicht überschritten. In Szenarien, in denen die Kompatibilität nur auf solchen Teilnetzwerken hergestellt werden muss, könnte dieses Verfahren angewandt werden. Rein in Bezug auf die zusätzliche Buslast ist hierarchische Versionierung jedoch vorzuziehen.

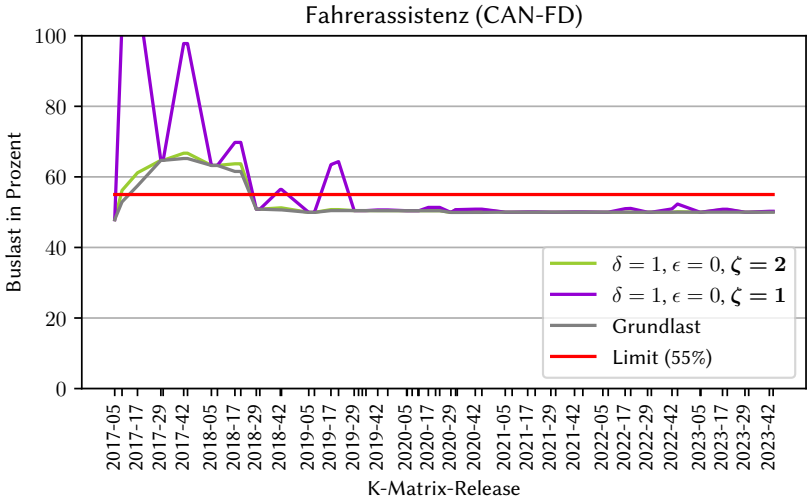

Abbildung 5.17: Hierarchische Versionierung und einfache Rückwärtskompatibilität

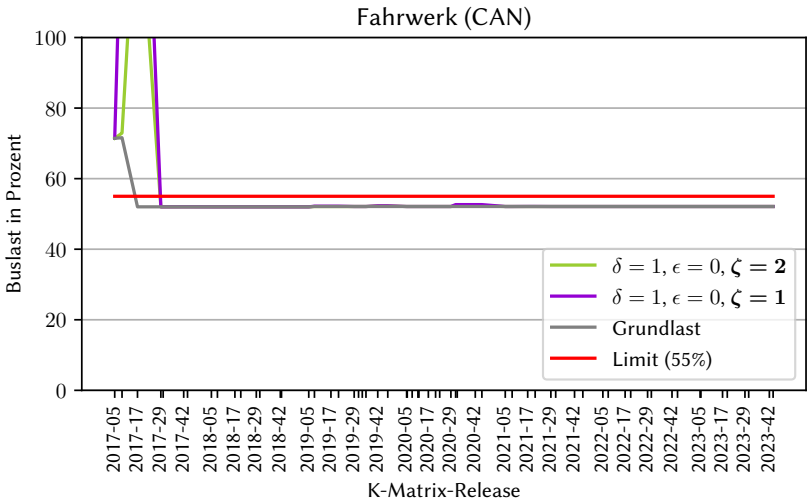

Abbildung 5.18: Hierarchische Versionierung und einfache Rückwärtskompatibilität

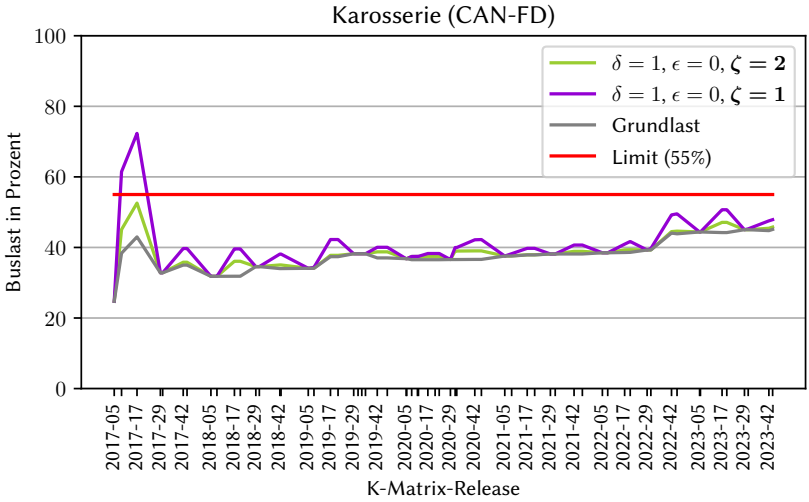

Abbildung 5.19: Hierarchische Versionierung und einfache Rückwärtskompatibilität

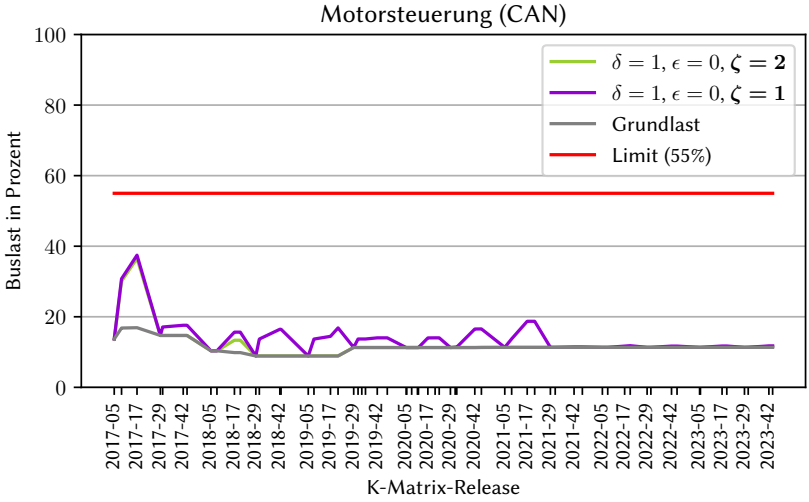

Abbildung 5.20: Hierarchische Versionierung und einfache Rückwärtskompatibilität

6 Praxistauglichkeit

6.1 Buslast

Die Buslast (d. h. die Auslastung der zur Verfügung stehenden Bandbreite, vgl. Abschnitt 3.7) steigt durch Anwendung der in Abschnitt 4.4.5 entwickelten Regeln, wie in Abschnitt 5.4 gezeigt, an. Das Ausmaß dieses Anstiegs ist entscheidend für die Praxistauglichkeit des veränderten Entwicklungsprozesses.

6.1.1 Grenzwert

Um bewerten zu können, ob sich die Zunahme der Buslast in einem akzeptablen Bereich befindet, muss definiert werden, welche maximale Buslast $L_{i,max}$ akzeptabel ist. Eine vollständige Auslastung der Bandbreite, d. h. eine Buslast von 100 %, kann nicht erreicht werden, da bei CAN (allgemeiner: bei allen CSMA-basierten Vernetzungstechnologien, vgl. Abschnitt 2.3.2) mit zunehmender Buslast die Wahrscheinlichkeit steigt, dass die Übertragung einer Nachricht verzögert wird, da der Bus bereits mit der Übertragung einer anderen Nachricht belegt ist. Um die erforderlichen, maximalen Latenzen bei der Übertragung einhalten zu können, muss die Auslastung des Busses begrenzt werden [Tin94].

Ohne Optimierungen sind Auslastungen von 30 % – 40 % möglich, während bei aufeinander abgestimmten Zykluszeiten, Botschaftsprioritäten und Latenzanforderungen bei bis zu 80 % Buslast die Einhaltung der Maximallatenzen garantiert werden kann [DeM05][Dav07]. In der Entwicklungspraxis von Mercedes-Benz hat sich eine Obergrenze von 55 % bewährt. Bei diesem Wert werden die Anforderungen an die Latenzen aller periodischer Signale zuverlässig erfüllt, bei ausreichender Restkapazität für alle sporadischen (und daher in der statischen Buslastberechnung unberücksichtigten) Signale. Bei anderen Fahrzeugherstellern können andere Grenzwerte zum Einsatz kommen, die im Bereich der oben erläuterten 30 % – 80 % liegen werden. Denkbar

wäre auch unterschiedliche Grenzwerte pro Bus zu definieren, je nach Verhältnis aus periodischen und sporadischen Signalen oder Sicherheitsrelevanz (Latenzüberschreitungen sind im Unterhaltungs- u. Komfortbereich tolerabler als in Fahrerassistenz oder Antriebsstrang). Da die vorliegenden Auswertungen auf Basis der realen Daten eines Fahrzeugherstellers erfolgten, wird zur Bewertung der Grenzwert dieses Herstellers, d. h. 55 %, angenommen.

6.1.2 Vergleich Schätzfunktion u. Simulationsergebnisse

Die in Abschnitt 5.3.2 vorgestellten Schätzfunktionen für hierarchische Versionierung (Gl. 5.35) u. einfache Rückwärtskompatibilität (Gl. 5.36) basieren auf den Eigenschaften α_i, β_i u. γ_i des Entwicklungsprozesses, die sich, wie in Abb. 5.5, 5.6 u. 5.7 dargestellt, im zeitlichen Verlauf verändern. Der sich ergebende Wertebereich für die Buslastzunahme $\Delta L_i^+ / L_i$, der in den Tabellen 5.1 u. 5.2 dargestellt, ist reicht für $\delta = 1$ von 10 % – 80 % für einfache Rückwärtskompatibilität oder von 6 % – 56 % für hierarchische Versionierung (für $\delta = 3$ jeweils dreimal so hohe Werte).

Zu dieser nicht unerheblichen Bandbreite kommt eine Unsicherheit hinzu: Die Grundlasten schwanken und befinden sich auf unterschiedlichen Niveaus (vgl. Abb. 5.1). Ein Bus mit Grundlast $L_i = 20\%$ läge bei einer Verdopplung (d. h. $\Delta L_i^+ / L_i = 100\%$) bei $L_i = 40\%$ und deutlich unter dem Grenzwert von $L_{i,max} = 55\%$. Bei einem Bus mit einer Grundlast von $L_i = 50\%$ wäre dagegen ein Zuwachs von $\Delta L_i^+ / L_i = 10\%$ grenzwertig.

Hinzu kommt, dass die Parameter α_i, β_i u. γ_i der Schätzfunktion für die gesamte Vernetzungsplattform gemeinsam erhoben wurden, einzelne Busse jedoch ober- oder unterhalb des Durchschnitts liegen können.

Gemessen an den Simulationsergebnissen (vgl. Abs. 5.4) werden die Verhältnisse (höhere Zuwächse in Bereichen mit größeren Werten für α_i, β_i u. γ_i) und Größenordnungen (Zuwächse um die 150 %) korrekt wiedergegeben. Abweichungen bestehen vorwiegend nach unten (d. h. ein Zuwachs, der geringer ausfällt als geschätzt), in Situationen, in denen auf den jeweiligen Bussen

weniger Änderungen erfolgten als im Rest des Fahrzeugs. Die Funktion eignet sich für Abschätzungen, z. B. um die Auswirkungen von Änderungen am Entwicklungsprozess zu diskutieren.

6.1.3 Phasen der Netzwerkentwicklung

Die in Abschnitt 5.1 – 5.3 gemachten Beobachtungen anhand der realen Daten sowie der Ergebnisse der Simulationen in Abschnitt 5.4 vorgestellten Simulationsergebnisse legen eine Unterteilung des Entwicklungsprozesses in mehrere Phasen nahe. Zwei Kriterien dafür sind:

- Der Bedarf nach zueinander kompatiblen Steuergeräten.
- Die Möglichkeit, unter Einhaltung des vorgegebenen Grenzwertes für die Buslast, Kompatibilität zu erzeugen.

Bedarf

Die Modellierung des Fahrzeugnetzwerks nach der AUTOSAR Entwicklungsmethodik (vgl. Abs. 2.7.6) beginnt innerhalb des V-Modells (vgl. Abs. 3.2.4) in der Spitze des Vs. Mit den regelmäßigen Releases werden Fehler behoben, die im Rahmen der verschiedenen Tests und Erprobungen (vgl. Abs. 3.2.5) entlang des gesamten rechten Schenkels des V-Modells entdeckt wurden.

Bis einschließlich den HIL-Tests einzelner Steuergeräte besteht keine Abhängigkeit zwischen den Steuergeräten und deren Entwicklungszyklen, da jeder HIL-Aufbau mit einer, exakt zum vorliegenden Steuergerät passenden, K-Matrix konfiguriert werden kann. Sobald die ersten Testaufbauten aus mehreren Steuergeräten erstellt werden (im V-Modell die Systemintegration u. Verifikation im rechten, aufsteigenden Schenkel) ist Kompatibilität zwischen diesen Steuergeräten notwendig und somit der Bedarf für Kompatibilitätsmechanismen gegeben.

Möglichkeiten

Die Grundlasten einzelner Netzwerke überschreiten zu Beginn der Entwicklung den Grenzwert von $L_{i,max} = 55\%$ (vgl. Abb. 5.1; Fahrwerk (CAN) in Release 2017-05 und Fahrerassistenz (CAN-FD) bis einschl. Release 2018-17). Die vorgestellten Kompatibilitätsmechanismen können die Buslast weiter erhöhen, nicht aber verringern – eine Einhaltung des Grenzwerts ist ausgeschlossen.

Die Ergebnisse der Simulationen mit unterschiedlichen Mindestlebensdauern (vgl. Abs. 5.4.2) zeigen, dass für $\delta = 1$ ab Release 2018-29 keine Grenzwertüberschreitungen vorliegen, und für $\delta = 3$ ab Release 2019-29. Mit zunehmendem Reifegrad sind höhere Mindestlebensdauern möglich.

Die in Kapitel 5 verwendeten, einheitlichen Mindestlebensdauern sind eine Vereinfachung der individuellen Mindestlebensdauern für einzelne Kommunikationselemente. Werden nur die von beständigeren Systemen und Steuergeräten eingesetzten Signale und Services mit längeren Mindestlebensdauern versehen, fällt der Buslastzuwachs geringer aus, als wenn für alle Elemente eine lange Rückwärtskompatibilität garantiert werden muss. Wird z. B. ein Teil der Elemente mit einer Mindestlebensdauer von einem halben Jahr, und der andere Teil mit einem Jahr versehen, ist eine Buslast zu erwarten, die zwischen den Kurven für $\delta = 1$ und $\delta = 3$ liegt (vgl. Abs. 5.4.2).

Erweiterung des Konzepts

Den Auswertungen folgend wird eine Unterteilung des Netzwerkentwicklungsprozesses in drei Phasen, parallel zum rechten Schenkel des V-Modells, vorgeschlagen:

1 Initialisierungsphase: An der unteren Spitze des V-Modells beginnt die Implementierung der einzelnen Komponenten des Systems (vgl. Abschnitt 3.2.4). In dieser Phase werden keine Kompatibilitätsmechanismen genutzt. Die Buslast überschreitet teilweise ihren Grenzwert. Es erfolgen keine Verbundtests.

2 Entwicklungsphase: Mit der Integration erster Teilsysteme und deren Verifikation beginnen die Verbundtests. Es wird ein Kompatibilitätsmechanismus mit geringer Mindestlebensdauer genutzt.

3 Reifegradphase: Die Änderungsrate ist hinreichend klein, um längere Mindestlebensdauern zu ermöglichen.

6.1.4 Schnellere Zwischenzyklen

Die vorliegenden Auswertungen basieren auf dem vierteljährlichen Release-Zyklus der STAR3 Entwicklung, und haben gezeigt, dass hier Kompatibilitätsdauern von bis zu einem Jahr möglich sind. In einem solchen Szenario gäbe es ein Haupt- und drei Zwischen-Releases pro Jahr (vgl. Abs. 4.4.5). Diese vierteljährlichen Releases basieren auf einem Kompromiss (vgl. Abs. 4.1.1), der einen Teil der Steuergeräteprojekte ausbremst. Für manche Domänen (z. B. Infotainment oder Fahrerassistenz) werden kürzere Zyklen bis zu monatlichen Releases gewünscht.

Um zu beantworten, ob hierarchische Versionierung bei einer größeren Anzahl Zwischen-Releases praktikabel wäre, wird die Schätzfunktion (vgl. Gl. 5.35) herangezogen. Die Anzahl der Zwischen-Releases entspricht dem Wert für δ, welcher linear in die Berechnung der Zusatzlast einfließt. Eine Verdopplung der Zwischen-Releases entspräche einer Verdopplung der Zusatzlast. Dies lässt mögliche Einflüsse einer Beschleunigung der Zwischenzyklen auf die anderen Parameter der Schätzfunktion außer acht.

Der PDU-Füllfaktor γ_i und die Grundlast L_i entsprechen Eigenschaften des Netzwerks, die sich auf die Schätzfunktion auswirken, aber von der Release-Häufigkeit nicht beeinflusst werden. Die Signaländerungsrate α_i und die PDU-Änderungsquote β_i bilden ab, wie viele Änderungen zwischen zwei Releases erfolgt sind.

Ausgehend davon, dass die Menge der Änderungen pro Zeitraum (z. B. 1 Jahr) konstant ist, wären α_i und β_i im gleichen Maße kleiner (da sich die Menge der Änderungen auf mehr Releases aufteilt), wie δ größer wäre – im Endergebnis

bliebe die Zusatzlast unverändert. Wird die größere Release-Häufigkeit genutzt, um insgesamt mehr Änderungen durchzuführen, stiege auch die durch Kompatibilitätsmechanismen erzeugte Zusatzlast an. Bei der Umstellung von klassischer, dem Wasserfallmodell folgenden Softwareentwicklung zu agiler Softwareentwicklung werden häufigere Releases mit geringerem Änderungsumfang angestrebt, um Auswirkungen der jeweiligen Änderungen früher testen (u. ggf. korrigieren) zu können [Sto10]. Sollte dieses Prinzip auf die Fahrzeugnetzwerkentwicklung übertragen werden, entspräche das dem Ersten der beiden Szenarien.

6.1.5 Übertragbarkeit auf andere Fahrzeughersteller

Die durchgeführten Untersuchungen und abgebildeten Statistiken basieren ausschließlich auf Daten aus der Entwicklung eines einzigen Herstellers. Bei anderen Fahrzeugherstellern sind verschiedene Abweichungen möglich.

Wie in Abschnitt 6.1.1 erläutert, sind bei unterschiedlichen Herstellern andere Grenzwerte für die maximale Buslast L_{max} oder individuelle Grenzwerte für unterschiedliche Busse möglich. Bedingt durch die endliche Bandbreite realer Bussysteme sind – unabhängig von deren exakten Höhe – zwingend Grenzwerte vorhanden. Die jeweiligen Grundlasten (und deren Abstand zur Maximallast) variieren in den vorliegenden Daten von Bus zu Bus, bei anderen Herstellern ist mit ähnlichen Schwankungen zu rechnen.

Beim Parameter γ_i müssen bei allen Fahrzeugherstellern vergleichbare Abwägungen getroffen werden: Je voller die PDUs (d. h. je größer γ_i), desto größer ist der Anteil der Nutzdaten (vgl. Abb. 3.9) pro Frame, aber desto schwieriger wird es, innerhalb der PDUs Signale mit gleichen Zykluszeiten z_i zu gruppieren. Fällt bei einem Fahrzeughersteller der Wert für γ_i geringer aus, fällt der Vorteil hierarchischer Versionierung gegenüber einfacher Rückwärtskompatibilität größer aus. Je näher γ_i der 100 % Marke kommt, desto geringer wird der Unterschied.

Bei α_i und β_i bestehen keine technischen Begrenzungen, anhand derer sich die Situation bei anderen Fahrzeugherstellern eingrenzen ließe. Werden in

einem Release sämtliche Signale verändert und zusätzlich neue Signale eingefügt, sind beispielsweise für α_i Werte größer 100 % möglich.

Um sinnvolle Aussagen zur Anwendbarkeit bei einem anderen Hersteller treffen zu können, müssen die Parameter α_i, β_i und γ_i wie in Abschnitt 5.3.1 beschrieben ermittelt werden. Anschließend kann mit der Schätzfunktion (Gl. 5.35) geprüft werden, bei welchen Werten für δ bei den gegebenen Grundlasten die Grenzwerte eingehalten werden.

6.2 Eignung unterschiedlicher Vernetzungstechnologien

Sowohl hierarchische Versionierung, als auch einfache Rückwärtskompatibilität haben eine zentrale Anforderung an die verwendete Netzwerktechnik: Eine nachträgliche Erweiterung einzelner Teilnehmer, ohne Beeinträchtigung der anderen Teilnehmer, muss möglich sein. Für hierarchische Versionierung muss die Kommunikation auf mehreren Ebenen erweiterbar sein. Basissoftware und Middleware nach dem AUTOSAR-Standard (vgl. Abschnitte 2.6 und 2.7) bieten diese Voraussetzung[1]. Hinzu kommen die unterschiedlichen Bustechnologien (vgl. Abschnitt 2.3.2 und Tabelle 2.6), die gemeinsam in Fahrzeugnetzwerken eingesetzt werden (vgl. Abs. 2.4), diese müssen jeweils die Durchführung von Änderungskopien ermöglichen.

6.2.1 CAN und CAN-FD

In zukünftigen Fahrzeugnetzwerken wird CAN bzw. CAN-FD im Rahmen hybrider E/E-Architekturen (vgl. Abs. 2.4) für günstige Sensoren und Aktoren weiter zum Einsatz kommen. Erweiterungen sind sowohl innerhalb eines

[1] Ursprünglich waren die Möglichkeiten zur kompatiblen Erweiterung von Kommunikationselementen nur implizit gegeben. Um zukünftigen, dem entgegenstehenden Entwicklungen vorzubeugen, wurde infolge dieser Arbeit eine Änderung in AUTOSAR eingebracht. Mit Release 2022-11 ist die Anforderung explizit festgehalten in RS_MAIN_00129 [AUT22, S. 31].

Frames (vgl. Abschnitte 2.3.2 und 2.7.5) möglich (d. h. der Frame wird vergrößert), als auch durch Erweiterung des Netzwerks um zusätzliche Frames. Die Forderung nach Erweiterbarkeit auf mehreren Ebenen ist damit erfüllt. Limitiert wird die Erweiterung auf Frame-Ebene durch die maximale Anzahl von 8 (CAN) oder 64 (CAN-FD) Bytes pro Frame (vgl. Abschnitt 3.7 und Gleichung (3.6)).

Die wesentliche Herausforderung bei CAN ist die rein Signal basierte Kommunikation, bei der die Informationsübertragung unabhängig von den Empfängern erfolgt. Ein Steuergerät überträgt die Botschaften aller Schnittstellen, unabhängig davon, ob ein Empfänger für Botschaften dieser Schnittstelle (in Fall von CAN wäre das z. B. ein Frame, mit einer ID, die von keinem im Netz vorhandenen Steuergerät verarbeitet wird) vorhanden ist. Dadurch wirken sich Änderungskopien unmittelbar auf die Buslast aus.

6.2.2 LIN

Der Datenaustausch im Local Interconnect Network wird durch den Master-Knoten gesteuert. Slave-Knoten reagieren jeweils auf die ihnen zugewiesenen IDs. Unbekannte IDs werden ignoriert, sodass nachträglich hinzukommende Botschaften nicht zu Fehlern führen. Kompatible Erweiterung ist, unter der Voraussetzung, dass der Master-Knoten jeweils auf dem neuesten Stand ist, möglich. Ist dies der Fall, kann auf Frame-Ebene erweitert werden, unter Beachtung der maximalen Frame-Größe von 8 Bytes.

LINs sind in der Praxis typischerweise auf ein einzelnes Subsystem (z. B. Steuerung einer Tür oder eines Sitzes) beschränkt. Für eine Entwicklung der Teilnehmer einzelner solcher Netze in unterschiedlichen Entwicklungszyklen (vgl. Abs. 4.1.2), besteht keine Notwendigkeit.

6.2.3 FlexRay

FlexRay verwendet TDMA als Arbitrierungsschema, um Kollisionen zu vermeiden und garantierte Latenzobergrenzen zu bieten[Zim14, S. 99ff]. Zur Einhaltung der Latenzanforderungen ist keine Beschränkung der Buslast (vgl. Abs. 6.1.1) erforderlich. Alle Zeitslots weisen dieselbe Länge auf, sodass kürzere Botschaften ihre Slots nicht vollständig ausfüllen. Bei den entsprechenden PDUs ist eine nachträgliche Ergänzung um zusätzliche Signale möglich. Solange auf einem FlexRay Kanal freie Zeitslots verfügbar sind, können PDUs hinzugefügt werden. Die erforderliche Erweiterbarkeit auf mehreren Ebenen ist damit gegeben.

6.2.4 Ethernet

Mit dem wachsenden Bandbreitenbedarf und um IP-basierte, Service-orientierte Kommunikation nutzen zu können, wird automotive Ethernet in Zukunft vermehrt eingesetzt werden. Service-orientierte Kommunikation auf Basis der SOME/IP Middleware sieht explizit die Möglichkeit zur kompatiblen Erweiterung auf mehreren Ebenen vor: Methoden können um zusätzliche Parameter ergänzt werden, Services können um zusätzliche Methoden ergänzt werden, und die Einführung neuer Services ist ohne die Beeinträchtigung bestehender Services möglich. Das SOME/IP Protokoll sieht die Verwendung unterschiedlicher Service Versionen vor, und das Service Discovery Protokoll [AUT23h], bietet die Möglichkeit explizit bestimmte Service Versionen (unterteilt in Major- u. Minor-Version) anzufordern.

In Hinblick auf hierarchische Versionierung bietet Service-orientierte Kommunikation den besonderen Vorteil, dass die Verbindung zwischen Clients und Servern dynamisch erfolgt. Werden infolge einer Änderungskopie mehrere Versionen eines Service bereitgehalten, werden nur die Versionen übertragen, die im Rahmen der Service Discovery von den vorhandenen Clients angefordert werden. Dadurch kann der Buslastzuwachs geringer ausfallen als bei Signal-basierter Kommunikation auf FlexRay oder CAN.

6.3 Einschränkungen

6.3.1 Patch Versionen

Für die Kommunikationselemente werden gemäß Regel 3 semantische Versionsnummern (vgl. Abs. 3.5.3) verwendet. Fehlerkorrekturen an der Software oder K-Matrix, die sich nicht auf die Kompatibilität der Schnittstellen auswirken, werden durch Inkremente der Patch Version kenntlich gemacht.

In der Praxis wird die Patch Version nur von eingeschränktem Nutzen sein. Änderungen an der Software ohne Auswirkung auf die Kompatibilität einer Schnittstelle sind für System- bzw. Steuergeräteprojekte relevant, nicht aber für die zentrale K-Matrix-Entwicklung. Umgekehrt sind Fehlerkorrekturen an der K-Matrix ohne Auswirkung auf die Schnittstellen für die jeweiligen Projekte nicht relevant.

In Konsequenz wird im praktischen Umgang kein Austausch der Patch-Versionen zwischen Netzwerk- und Steuergeräteentwicklern stattfinden. Zur K-Matrix Entwicklung sind die Patch-Versionen dennoch von Bedeutung, da jedes veränderte Objekt in der Datenbank eine neue Versionsnummer benötigt, um die Änderungshistorie nachvollziehen zu können. Die Unterscheidung zwischen Änderungen mit und ohne Auswirkungen auf die Kompatibilität wird ermöglicht.

6.3.2 Sender- u. Empfänger-seitige Software

Bei Änderungen an der Kommunikation, die kompatibel gehalten werden (vgl. Abs. 4.4.2 u. 4.4.3), ergeben sich Änderungen an der Software. Sender-seitig muss sowohl das neue, als auch das alte Kommunikationsobjekt (d. h. zwei Signale oder zwei PDUs) gesendet werden. Empfänger-seitig muss die Fähigkeit vorhanden sein, sowohl das neue, als auch das alte Kommunikationsobjekt zu empfangen und verarbeiten.

Die Software zum Senden und Empfangen der alten u. neuen Version eines Signals werden im Rahmen der bestehenden Entwicklungsprozesse ohnehin geschrieben (zunächst je für die alte Version des Signals, dann ab dem veränderten Release für die neue Version). Der sich durch Anwendung hierarchischer Versionierung ergebende Unterschied ist, dass die, auf den Extrakten der Zwischen-Releases aufbauenden, Steuergeräte beide Softwarekomponenten parallel integrieren müssen (vgl. Abs. 2.6.4). Der durch AUTOSAR vorgesehene modulare Aufbau der Software mit einzelnen SWCs für einzelne Funktionen (vgl. Abs. 2.7.2) ermöglicht die parallele Integration alter u. neuer SWCs.

Der dadurch entstehende Mehrbedarf an CPU-Zeit und Arbeitsspeicher auf den jeweiligen Steuergeräten liegt vergleichbar mit der Zusatzlast ΔL_i^+ nur in den kompatiblen Zwischen-Releases vor. Mit Ende der Restlebensdauer der veralteten Kommunikationselemente können die entsprechenden Sender- u. Empfängerkomponenten entfernt werden (vgl. Regel 12), um die Ressourcen freizugeben.

Die für den Parallelbetrieb älterer und neuerer SWCs notwendigen, und nach Abschluss der Entwicklungsphase frei werdenden Ressourcen können als Vorhalte für zukünftige Erweiterungen von Software- u. Funktionsumfang eines Steuergeräts eingesetzt werden. Durch die UNECE Richtlinie R155 besteht z. B. die Notwendigkeit während der Fahrzeuglebensdauer die Angriffssicherheit durch nachträgliche Erweiterung von kryptografischen Algorithmen aufrechtzuerhalten [Win22]. Mit der wachsenden Bedeutung von Over-the-Air-Updates (OTAs) und Softwareerweiterungen werden solche Vorhalte zunehmend in ECU-Designs berücksichtigt [Bur18][Ban21].

6.3.3 Prozesstreue

In der automobilen Entwicklungspraxis kommt es, z. B. wegen unvorhergesehenen Problemen in einzelnen Systemen oder ECU-Projekten, oder wegen sich kurzfristig verändernden Anforderungen, zu Abweichungen von den vorgesehenen Prozessen [Ekl05]. Für die praktische Anwendung eines Prozesses

müssen nicht nur dessen Eigenschaften an sich, sondern auch die Auswirkungen von Abweichungen betrachtet werden.

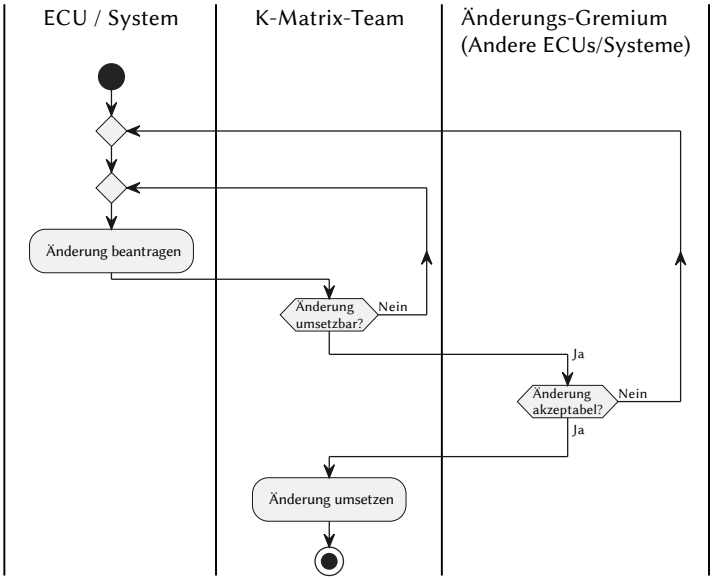

Abbildung 6.1: Der Prozess zur Durchführung von Änderungen an der K-Matrix

Ausgelöst durch neue Anforderungen oder Erkenntnisse aus zuvor durchgeführten Tests (vgl. Abschnitt 3.2.2) wird durch ein ECU- bzw. System-Projekt ein K-Matrix-Änderungsantrag an das K-Matrix-Team gestellt. Dort wird überprüft, ob die Änderung umsetzbar ist. Idealerweise ist der Änderungsantrag hinreichend spezifisch, um direkt umgesetzt werden zu können. Andernfalls werden Rückfragen gestellt, bis ein umsetzbarer Änderungsantrag vorliegt. Dessen Umsetzung muss durch ein Gremium, bestehend aus den durch die Änderung betroffenen Parteien (z. B. sämtlichen Empfängern eines veränderten Signals), genehmigt werden. Wird die Genehmigung erteilt, setzt das K-Matrix-Team die Änderung in der Bordnetzdatenbank um, sodass sie mit dem nächsten K-Matrix-Release an die Steuergeräteprojekte verteilt wird. Andernfalls sind weitere Nachbesserungen durch das die Änderung

beantragende Projekt nötig (s. Abb. 6.1). Idealerweise findet dieser Ablauf innerhalb eines K-Matrix-Release-Zyklus statt, längere Dauern sind möglich (vgl. Abb. 6.2 und 6.3).

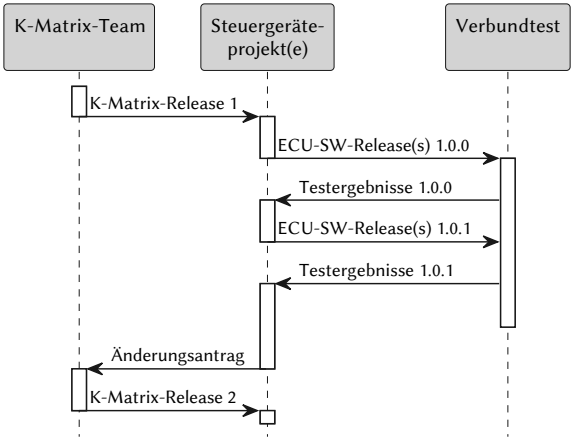

Abbildung 6.2: Die Testergebnisse aus K-Matrix-Release 1 fließen in K-Matrix-Release 2 ein.

Auf Basis der K-Martrix-Releases erstellen die Steuergeräteprojekte Software-Releases (vgl. Abschnitt 3.5.4) welche zunächst einzeln und anschließend im Verbund mit anderen Steuergeräten (vgl. Cluster-HIL und Erprobungsfahrzeug, s. Abschnitt 3.2.5) getestet wird. Erkenntnisse aus diesen Tests können, sofern sie nicht das Interface der ECU betreffen, noch im selben Release-Zyklus für Softwareverbesserungen durch das Steuergeräteprojekt verwendet werden (s. Abb. 6.2). Sind Änderungen an der Schnittstelle erforderlich, muss ein Änderungsantrag an das K-Matrix-Team gestellt und abgestimmt (s. Abb. 6.1) werden. Die beiden, auf K-Matrix-Release 1 basierenden, Software-Releases weisen die Versionsnummern 1.0.0 und 1.0.1 auf, um auszudrücken, dass es sich zwar um unterschiedliche Softwarestände handelt, diese aber keine Unterschiede am Interface aufweisen (vgl. Abschnitte 3.5.3 und 6.3.1).

Eine für die Netzwerkentwicklung typische Situation, in der von dem bestehenden Prozess abgewichen wird, ist das Auftreten von schwerwiegenden (d. h. die Erprobung Fahrzeugaufbauten verhindernden) Fehlern. In solchen

Situationen werden für die betroffenen Steuergeräte, außerhalb des regulären Entwicklungszyklus, korrigierte K-Matrizen erstellt, und durch die Steuergerätelieferanten in die Software integriert (siehe die b- u. c-Releases in den Abschnitten 5.1.1 und 5.1.3). Die außerplanmäßige Neuintegration der Software verursacht, neben dem Aufwand für die zusätzliche K-Matrix-Version, Zusatzkosten, die durch die ECU-Hersteller in Rechnung gestellt werden. Bei der Entscheidung, ob es zu einem außerplanmäßigen Release kommt, werden diese Zusatzkosten, gegen die zu erwartende Beeinträchtigung der Erprobung abgewogen.

Der Bedarf nach (kurzfristigen) Fehlerkorrekturen wird sich durch die Einführung hierarchischer Versionierung nicht verändern. Die Auswirkungen und Umsetzung einer solchen Änderung sowie die Abwägung, ob es zu einem außerplanmäßigen Release kommt, werden beeinflusst.

Es muss berücksichtigt werden, in welchen Release-Zyklen die betroffenen Steuergeräte entwickelt werden. Bei schnelleren Zyklen sinkt die Wahrscheinlichkeit, dass ein außerplanmäßiges Release nötig wird, da die Wartezeit bis zum nächsten, planmäßigen Korrekturzeitpunkt geringer ausfällt (vgl. Abb. 4.3). Bei Steuergeräten, die in langsameren Zyklen entwickelt werden, steigt dagegen die Wahrscheinlichkeit an. Ausgehend von der ursprünglichen Voraussetzung, dass die in längeren Zyklen zu entwickelnden ECUs ausgereifter u. beständiger sind (vgl. Abs. 1.2 u. 4.1), werden Fehlerkorrekturen nur selten notwendig sein.

Tritt der Bedarf auf, die Kommunikation einzelner Steuergeräte außerplanmäßig zu verändern, wäre zu prüfen, ob die Änderung in einem der ohnehin vorgesehenen Zwischen-Releases untergebracht werden kann (d. h. die fehlerhaften ECUs wechseln temporär in einen schnelleren Entwicklungszyklus). So wird die Zahl der Steuergeräte, für die ungeplant zusätzliche Softwareänderungen nötig werden, verringert. Nur wenn dies (z. B. aus zeitlichen Gründen) nicht möglich ist, erfolgt ein zusätzliches Zwischen-Release.

Unabhängig vom Zeitpunkt, zu dem Fehlerkorrekturen eingebracht werden, besteht bei hierarchischer Versionierung eine weitere Möglichkeit vom vorgesehenen Prozess abzuweichen. In Form der Mindestlebensdauern von Kommunikationselementen garantiert der Prozess die Verfügbarkeit von Schnittstellen über definierte Zeiträume. Es könnten sich Situationen ergeben, in denen diese Garantien verletzt werden müssen (vgl. Abschnitt 4.5.4). Beispielsweise könnten auf einzelnen Bussen unerwartet viele Änderungen auftreten, sodass entgegen der in Kap. 5 durchgeführten Untersuchungen, die Grenzwerte für die Buslast nicht eingehalten würden.

Bei der Verletzung von Kompatibilitätsgarantien müssen bei den betroffenen Elementen jeweils die Major-Versionen erhöht werden (vgl. Abschnitt 3.5.3), um die Inkompatibilität kenntlich zu machen. In Konsequenz müssten Steuergeräte, welche die betroffenen Elemente verwenden, zwingend die neue K-Matrix integrieren, obwohl das zu dem Zeitpunkt (ausgehend von der bestehenden Garantie) nicht vorgesehen war. Um zu entscheiden, ob eine Mindestlebensdauer verletzt wird, sind die Kosten, die dadurch entstehen (u. a. Kosten für die ungeplante Neuintegration der Software einer oder mehrerer ECUs), gegen die Kosten, die durch eine Verzögerung der notwendigen Änderungen entstehen (z. B. durch verzögerte Erprobungen) abzuwägen. Die Entscheidung könnte durch ein Gremium, analog zu den bestehenden Änderungsfreigaben (vgl. Abb. 6.1) getroffen werden. In jedem Fall sollte anschließend evaluiert werden, wie die Situation in Zukunft vermieden werden kann, z. B. indem für bestimmte Objekte geringere Lebensdauergarantien gegeben werden.

6.3.4 Begrenzte Geschwindigkeit in der Softwareentwicklung

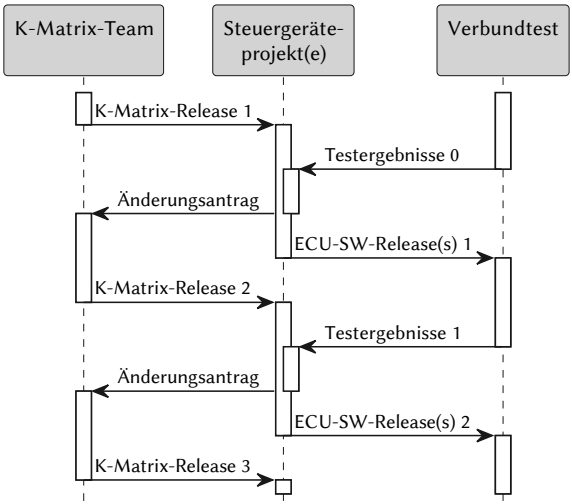

Abbildung 6.3: Die Testergebnisse aus K-Matrix-Release 1 fließen erst in K-Matrix-Release 3 ein.

Die Integration eines neuen K-Matrix-Releases in die Software eines Steuergerätes (vgl. Abschnitt 2.6.4) nimmt 2 – 8 Wochen in Anspruch [Smi22]. Die anschließenden Verbundtests und deren Auswertung nehmen weitere Zeit in Anspruch, bis schließlich Änderungsanträge zu Verbesserung der K-Matrix gestellt werden können (vgl. Abb. 6.2). Dauern Integration und Test länger als ein K-Matrix-Release-Zyklus (vgl. z. B. den A-Zyklus in Abb. 4.3 mit einer Dauer von einem Monat), können die Erkenntnisse aus den Tests nicht im folgenden K-Matrix-Release verwendet werden. Ein möglicher alternativer Ablauf ist in Abb. 6.3 skizziert. Die Testergebnisse der Software zu K-Matrix-Release 1 fließen dabei erst in K-Matrix-Release 3 ein. Durch mehr Automatisierung in der Softwareintegration kann die Dauer einer Entwicklungsschleife (K-Matrix-Release, SW-Release, Testergebnis) reduziert werden, sodass auch bei schnellen Release-Zyklen ein Ablauf wie in Abb. 6.2 möglich ist.

6.3.5 Verringerte Funktionsumfänge in Zwischen-Releases

Eingeschränkte Kompatibilität bedeutet, dass nicht alle Funktionen vorhanden sind, aber zumindest keine Störung vorliegt. Ist der testbare Funktionsumfang zu gering, stellt das den Nutzen der Zwischen-Releases infrage. Die vorgestellten Regeln zur hierarchischen Versionierung stellen sicher, dass in Zwischen-Releases mindestens die Funktionalität des vorangegangenen Haupt-Releases vorhanden ist.

Erfolgt eine rückwärtskompatible Änderung einer Schnittstelle, die ECUs in unterschiedlichen Zyklen betrifft, so ist die neu hinzugekommene Funktionalität nicht sofort als Ganzes testbar. Zumindest bei einzelnen Steuergeräten des Systems sind, zu Zeitpunkten, an denen von anderen Geräten keine neuen Stände vorliegen (und auch nicht erwartet werden), HIL-Tests der neuen Schnittstellen durchführbar. Gegenüber dem bisherigen Prozess mit einheitlichen Releases und Release-konformem Flashen ist dies ein geringer Vorteil, da das Testen individueller Softwarestände auf einzelnen ECUs bereits möglich ist. In Bezug auf den Umgang mit isoliert betrachteten Steuergeräten ist die Möglichkeit, häufiger neue K-Matrix-Releases zu erhalten, und damit häufiger Änderungen an der Konfiguration des COM-Stacks durchführen und testen zu können.

Bei der Gruppierung von ECUs nach Systemen oder Teilnetzen in gemeinsamen Zwischenzyklen, sind die Gruppen-internen, neuen Funktionen jeweils auch zu den Zwischen-Releases testbar – ein erheblicher Gewinn gegenüber der herkömmlichen Strategie.

6.4 Mögliche Variationen

6.4.1 Verblockungen

Die bisherigen Untersuchungen bezogen sich auf die Releases während der Entwicklungsphase einer Vernetzungsplattformn (vgl. Abschnitt 5.1.1). Idealerweise sollte das Konzept auch Vor- und Rückverblockungen (vgl. Abschnitt 2.1.2) berücksichtigen.

Verblockungen bedeuten für die Netzwerkentwicklung, dass eine vorliegende Schnittstelle nicht verändert werden darf, um die betreffenden Steuergeräte in mehreren Baureihen verwenden zu können. Über die gesamte Entwicklungsdauer muss das Fahrzeugnetzwerk zu diesen Schnittstellen kompatibel bleiben. Im Rahmen der hierarchischen Versionierung, lässt sich dieser Umstand durch hohe Mindestlebensdauern abbilden.

Regel 7 sieht vor, die Mindestlebensdauer als eine Anzahl von Haupt-Releases anzugeben – innerhalb des Entwicklungsprozesses einer Baureihe eine sinnvolle Maßeinheit. Mindestlebensdauern, die mehrere Baureihen überspannen sollen, sind nicht sinnvoll als Anzahl von Releases in den Entwicklungsprozessen darstellbar, weil die Menge der K-Matrix-Releases für eine Baureihenentwicklung nicht festgelegt ist.

Stattdessen kann ein zweites, parallel bestehendes Feld zur Angabe der Mindestlebensdauer verwendet werden. Dieses setzt als Einheit die Anzahl Baureihen, oder eine konkrete Zielbaureihe, bis einschließlich derer die Schnittstelle bestehen muss, ein.

Eine Ausphasung findet dann nur nach Ablauf beider Fristen statt. Hat bis zum Serienstart der Zielbaureihe keine Änderungskopie stattgefunden, kann die Schnittstelle unverändert weiterverwendet und für weitere Baureihen verblockt werden[1].

[1] Ein Beispiel für eine sehr lange unverändert eingesetzte Schnittstelle ist die des Anhängersteuergerätes. Solange keine Änderungen an der ISO 11446 [Int12] zustande kommen, wird das Steuergerät immer wieder verblockt.

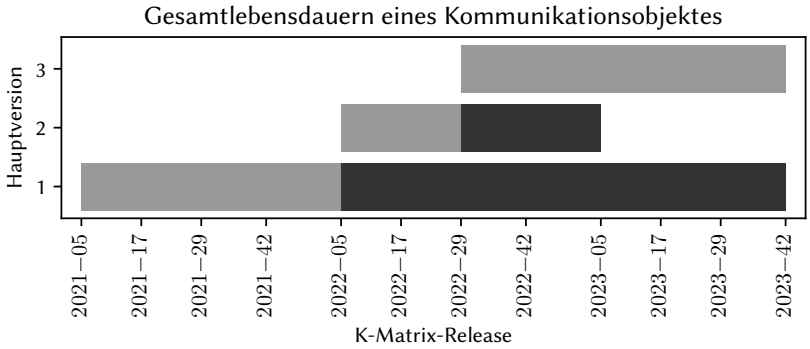

Abbildung 6.4: Der Teil der Gesamtlebensdauer, in der ein Kommunikationselement die jeweils neueste Version darstellt, ist hellgrau markiert.

Werden Änderungskopien durchgeführt, kann die neue Hauptversion eine geringere Mindestlebensdauer aufweisen (vgl. Regel 8), als der verblockte, ursprüngliche Stand. Kommen im Verlauf weitere Änderungskopien zustande, kann sich durch die Ausphasung der zweiten Hauptversion eine Situation ergeben, in der z. B. Version 1.x.y (verblockt) und Version 3.c.d im Fahrzeug vorhanden sind, nicht aber Version 2.a.b (durch Version 3.c.d abgelöst). Eine entsprechende Situation ist in Abb. 6.4 illustriert, mit einer Mindestlebensdauer von 6 Monaten für die Hauptversion 2 und 21 Monaten (oder mehr) für die Hauptversion 1. Ab Release 2023-05 ist Hauptversion 2 nicht mehr vorhanden, während Hauptversion 1 aufgrund der langen Mindestlebensdauer parallel zu Hauptversion 3 existiert. Zwischen den Releases 2022-29 und 2023-05 lagen drei Hauptversionen desselben Kommunikationselements vor. Dies ist auch ein Beispiel dafür, wie eine schlechte Kombination aus Änderungsrate und Mindestlebensdauer (Lange Mindestlebensdauer trotz hoher Änderungsrate), die Buslast stark erhöht (in diesem Fall: verdreifacht) werden kann.

6.4.2 Mindestlebensdauern in Form von Zwischen-Releases

Eine andere, denkbare Abweichung von Haupt-Releases als Einheit für Mindestlebensdauern, ist die Verwendung von Zwischen-Releases. Das wäre primär in Prozesskonfigurationen sinnvoll, in denen zwischen zwei Haupt-Releases viele Zwischen-Releases liegen. Bei schnelllebigen Steuergeräteprojekten mit mehreren inkompatiblen Änderungen zwischen zwei Haupt-Releases würde der Bandbreitenbedarf mit jeder dieser Änderungen ansteigen.

Das Entfernen eines Elements gemäß Regel 12 ist mit an Haupt-Releases gebundenen Mindestlebensdauern nur zu solchen Releases möglich. Wären Mindestlebensdauern an Zwischen-Releases gebunden, ließen sich nicht mehr benötigte Elemente aus den früheren Zwischenständen eines Haupt-Release-Zyklus zu späteren Zwischenständen innerhalb desselben Zyklus wieder entfernen.

Gegen dieses Modell spricht, dass das Entfernen von Elementen eine inkompatible Änderung ist. Diese dürfen ausschließlich zu Zeitpunkten stattfinden, an denen alle Steuergeräte die Möglichkeit haben, sich anzupassen.

6.4.3 Umgang mit Zwischen-Releases

Regel 1 sieht periodische Haupt-Releases vor, sowie kürzere Zwischenzyklen für die schnelllebigeren ECUs. In Abbildung 4.3 wurden zwei unterschiedlich schnelle Sets an Zwischenzyklen angenommen, die jeweils in regelmäßigen Abständen zwischen den Haupt-Releases liegen.

Regel 1 macht keine Vorgabe zu Häufigkeit oder Periodizität der Zwischen-Releases. Es gibt zwei Variationspunkte:

- Zwischen-Releases sind nur sinnvoll, wenn sie für mindestens zwei Steuergeräte (ein Sender und mindestens ein Empfänger) erfolgen. Eine Unterteilung in Gruppen, für die gemeinsame Zwischen-Releases

stattfinden, kann fest (über die ganze Entwicklungsphase) oder für jedes Zwischen-Release individuell erfolgen.

- Die Zeitpunkte für Zwischen-Releases können in festen (evtl. periodischen) Abständen, relativ zu den Haupt-Releases liegen oder dynamisch nach Bedarf erfolgen.

Für eine dynamische Zuordnung von Steuergeräten zu Zwischen-Release-Gruppen spricht, dass auf diese Weise unnötige Zwischenstände vermieden werden können.

Für definierte, periodische Zwischen-Release-Zeitpunkte spricht, die bessere Planbarkeit der K-Matrix-Entwicklung. Nachteil ist die geringere Flexibilität. Ist die feste Frequenz, in der Zwischenstände erfolgen können, größer als die bisherigen vier geplanten Releases pro Jahr, bedeutet das trotzdem eine Zunahme der Flexibilität.

6.4.4 Gruppierung von Änderungskopien

Ein Aspekt der hierarchischen Versionierung, ist der Versuch, kompatible Erweiterungen auf niedrigen Hierarchieebenen durchzuführen, und nur falls dies nicht möglich ist, Änderungskopien auf höherer Ebene durchzuführen. Wenn etwa die Änderungskopie eines Signals, wegen der Maximalgröße einer PDU, nicht möglich ist, muss anstatt des Signals die gesamte PDU dupliziert werden.

Eine Alternative zu diesem Vorgehen wäre, pro Interface alle inkompatibel zu verändernden Signale in einer neu anzulegenden PDU zu sammeln.

Der Vorteil wäre, dass jeweils nur die veränderten Signale, nicht aber die unveränderten der betroffenen PDUs kopiert werden. Durch die geringere Redundanz wäre weniger zusätzliche Bandbreite erforderlich.

Gegen diese Variante spricht, dass die Gruppierung von Signalen in PDUs nicht willkürlich, sondern nach inhaltlicher Zusammengehörigkeit und ähnlichen Zykluszeiten erfolgt. Eine „Sammel-PDU“ müsste mit der höchsten Sendefrequenz des schnellsten, enthaltenen Signals gesendet werden und könnte verschiedene, unabhängige Teilsysteme betreffen. Bei der Ausphasung der „Sammel-PDU“ müsste die längste, bei den Signalen vorkommende, Mindestlebensdauer berücksichtigt werden.

Die Auswertungen der reellen Daten (vgl. Abb. 5.8 oder Abb. 5.17 – 5.20) zeigt, dass die Situation, in der ganze PDUs kopiert werden müssen, selten auftritt. Entsprechend gering wäre der Nutzen von „Sammel-PDUs“.

7 Fazit und Ausblick

7.1 Beantwortung der Forschungsfragen

Forschungsfrage 1

> Wie kann der Widerspruch zwischen der Wiederverwendung von bestehenden Steuergeräten und der Forderung nach schnellen Innovationszyklen und den damit einhergehenden Veränderungen am Gesamtsystem sinnvoll aufgelöst werden?

Zunächst ist anzuerkennen, dass sowohl die Beständigkeit mancher, als auch die Schnelllebigkeit anderer Steuergeräte valide Ziele sind, deren Einhaltung zur Verbesserung (technologisch und wirtschaftlich) des Gesamtproduktes „Fahrzeug“ beitragen. Statt der bisherigen Kompromisslösung, alle Geräte in einem einheitlichen Zyklus zu entwickeln (vgl. Abschnitt 4.1.1), sollte ein Prozess geschaffen werden, der eine Entwicklung in jeweils optimalen Intervallen (vgl. Abb. 4.3 in Abschnitt 4.4.5) ermöglicht.

Die einzelnen Steuergeräte müssen weiterhin in Gesamtfahrzeugaufbauten testbar bleiben (vgl. Abschnitt 3.2.2). Auf Netzwerkfunktionen aufbauende Systeme sollten in größtmöglichem Umfang einsetzbar sein. Voraussetzung dafür ist es, eine – trotz der individuellen Entwicklungszyklen – hohe Kompatibilität (vgl. Abschnitt 4.2.1) zwischen den Steuergeräten sicherzustellen.

Forschungsfrage 2

> Wovon hängt die Kompatibilität zwischen Steuergeräten und Fahrzeugnetzwerken bzw. zwischen Clients und Servern ab, und wie kann diese bewertet werden?

Da in der Praxis nicht einzelne Teilsysteme, sondern Steuergeräte austauschbar sind, und von unterschiedlichen Lieferanten entwickelt werden (vgl. Abschnitt 2.1.3), ist die Schnittstelle zwischen Steuergerät und Fahrzeug für den Entwicklungsprozess die Schlüsselstelle (vgl. Abschnitt 4.2.2). Für diese Schnittstellen werden die zu übertragenden Kommunikationselemente (aus verschiedenen Events und Methoden zusammengesetzte Services, oder aus verschiedenen Signalen bestehende PDUs) betrachtet.

Zur Bewertung der Kompatibilität wird der Funktionsumfang des Gesamtsystems (Fahrzeug) als Maßstab gesetzt (vgl. Abschnitt 4.2.1). Es wird unterteilt in vollständige Kompatibilität (sämtliche Funktionen sind vorhanden), eingeschränkte Kompatibilität (nicht alle Funktionen vorhanden) und vollständige Inkompatibilität (keine Funktion vorhanden). Hinzu kommt der Sonderfall der Scheinkompatibilität (vgl. Definition 3.27), bei der für die beteiligten Kommunikationspartner der Anschein besteht, dass eine Information korrekt übertragen wird, obwohl Sender und Empfänger die betroffenen Bits unterschiedlich interpretieren. Diese Fehlinterpretationen führen zu erratischem Systemverhalten und machen Erprobungen genauso unmöglich wie vollständige Inkompatibilität.

Entwicklungsprozesse basieren auf der inkrementellen Verbesserung des Gesamtsystems, durch Veränderungen an dessen Komponenten. Speziell zur Umsetzung unterschiedlicher Entwicklungszyklen für verschiedene ECUs sind Änderungen an den Schnittstellen und deren Auswirkung auf die Kompatibilität zu bewerten. In Bezug auf Veränderungen an Schnittstellen ergibt sich eine besondere Form der eingeschränkten Kompatibilität: Kommen neue Kommunikationselemente hinzu, während sämtliche alten Elemente bestehen, ist die Änderung „rückwärtskompatibel“ (vgl. Definition 3.29). D. h. die ursprünglichen Funktionen sind vollständig vorhanden, während das Vorhandensein von neuen Funktionen davon abhängt, ob die anderen, am jeweiligen System beteiligten, Steuergeräte auch die neue Interface-Version umsetzen. Der gezielte Zwang zur Rückwärtskompatibilität ermöglicht die Entwicklung in abweichenden Zyklen.

Die Kommunikationselemente können als Hierarchie dargestellt werden, mit dem ECU-Interface als oberster Ebene, dem Services bzw. PDUs untergeordnet werden, die wiederum über Events und Methoden bzw. Signalen bestehen (vgl. Abb. 4.2 in Abschnitt 4.4.1). Auf jeder dieser Hierarchieebenen sind rückwärtskompatible, aber auch inkompatible Änderungen möglich.

Forschungsfrage 3

> Wie kann ein gemeinsamer, praxistauglicher Prozess zur Bewertung und Optimierung der Kompatibilität sowohl in Signal-basierten als auch in Service-orientierten Netzwerken aussehen?

Startpunkt für die Definition eines Prozesses war die Zielsetzung, Änderungen rückwärtskompatibel zu halten. Mit dem Konstrukt der „Änderungskopie“ (vgl. Definition 4.1), d. h. der Duplikation eines inkompatibel zu verändernden Kommunikationselements, und der Durchführung der eigentlichen Änderung am Duplikat, sind sämtliche Änderungen rückwärtskompatibel durchführbar.

Änderungskopien führen zu einer redundanten Informationsübertragung (alte und neue Form werden parallel gesendet, vgl. Abschnitt 4.5.3) und so zu einem gesteigerten Bandbreitenbedarf. Damit die Bandbreite nicht mit jeder Änderung unbegrenzt ansteigt, ist ein Mechanismus notwendig, mittels dessen ältere, redundante Elemente automatisch, und für die beteiligten Systeme vorhersehbar, entfernt werden. Hierzu wurde das Konzept der „Mindestlebensdauern“ eingeführt (vgl. Regel 7), mit denen angegeben wird, wie lange ein Element, ab dessen erster Änderungskopie, verfügbar sein wird. Ein weiterer Mechanismus zur Begrenzung des Bandbreitenbedarfs, ist die Anwendung der Änderungskopien auf der jeweils niedrigsten, möglichen Hierarchieebene (vgl. Regel 6) – je niedriger, desto weniger Elemente werden kopiert. Eine semantische Versionsnummer an jedem Kommunikationselement ermöglicht das unmittelbare Ablesen der Kompatibilität, ohne K-Matrix-Releases miteinander vergleichen zu müssen.

Der Bandbreitenzuwachs könnte in der Stückkosten-sensiblen Pkw-Industrie (vgl. Abschnitt 2.1.2) die Einführung des vorgestellten Verfahrens blockieren. Um Aussagen zur Praxistauglichkeit treffen zu können, wurde die Auswirkung des vorgestellten Verfahrens mathematisch modelliert (vgl. Abschnitt 5.3). Zur Berechnung notwendige Parameter des K-Matrix-Entwicklungsprozesses wurden identifiziert (Änderungsrate, mit der sich Signale zwischen Releases verändern, die Quote veränderter zu unveränderter PDUs, und der mittlere Füllstand der PDUs) und anhand der protokollierten Daten eines realen K-Matrix-Entwicklungsprozesses (vgl. Abschnitt 5.1) bestimmt. Mit den aufgestellten Formeln lässt sich zeigen, wie sich die Prozessparameter (z. B. mehr/weniger Änderungen oder größere/kleinere Mindestlebensdauern) auf die Bandbreite auswirken. Da die ermittelten Eigenschaften des realen Prozesses im zeitlichen Verlauf stark schwanken, ergibt sich für die aufgestellten Formeln ein Ergebniskorridor, der je nach gewählter Mindestlebensdauer von 6 % – 56 % oder 18 % – 168 % breit ist (vgl. Tabelle 5.2).

Für eine präzisere Bewertung wurde eine Simulation erstellt, die dem realen Entwicklungsprozess Release für Release folgt, und für jedes Signal und jede PDU individuell die Regeln des vorgestellten Verfahrens anwendet (vgl. Abschnitt 5.4). Die realen und die durch die Änderungskopien erhöhten Buslasten werden bitgenau berechnet und miteinander verglichen.

Die Ergebnisse der Simulationen zeigten, dass der zusätzliche Bandbreitenbedarf auf den verschiedenen Bussen in einer akzeptablen Größenordnung liegt. Für Mindestlebensdauern von einem Jahr erfolgen zu Beginn der Entwicklungsphase Überschreitungen des Grenzwerts (vgl. Abschnitt 5.4.2). Für optimale Ergebnisse empfiehlt sich daher, die Entwicklung einer Vernetzungsplattform mit kürzeren Mindestlebensdauern der Kommunikationselemente zu beginnen, und mit zunehmendem Reifegrad (und entsprechend geringeren Änderungsraten) längere Kompatibilitätsgarantien zu bieten.

7.2 Einführung in der Entwicklungspraxis

Hierarchische Versionierung verspricht drei wesentliche Vorteile für den Entwicklungsprozess eines Fahrzeugnetzwerkes:

- Durch Zwischen-Releases kann in Steuergeräteprojekten, in denen ein entsprechender Bedarf besteht, die Geschwindigkeit, mit der sich Software und Kommunikation verändern, erhöht werden. Das ermöglicht schnellere Entwicklung, frühere Reife oder höherer Reifegrad und mehr Innovation.
- Erfolgen Haupt-Releases in einem größeren Abstand, als die bisherigen einheitlichen K-Matrix-Releases, sind Softwarestände, die K-Matrixänderungen integrieren, bei beständigeren Steuergeräten seltener notwendig. Dies spart Entwicklungskosten in den jeweiligen ECU-Projekten.
- Durch die semantischen Versionsnummern und die Mindestlebensdauern entsteht Transparenz und Planbarkeit bezüglich der Kompatibilität von Steuergeräteschnittstellen.

Diesen Gewinnen steht ein nicht zu vernachlässigender Anstieg der benötigten Bandbreite entgegen. Den durchgeführten Untersuchungen zufolge wird dieser Nachteil in einem akzeptablen Rahmen bleiben. Als nächster Schritt folgt die praktische Erprobung des Konzepts (vgl. Abschnitt 7.2.2).

7.2.1 Bisher Erreichtes

Infolge dieser Forschung wurden bereits erste Anpassungen am von Mercedes-Benz verwendeten K-Matrix-Entwicklungswerkzeug (vgl. Abschnitt 3.5.1 und [Vet23a]) vorgenommen. Statt einer rein technischen Versionsnummer (d. h. eine einzelne Zahl, die bei jeder Änderung – unabhängig von der Art der Änderung – um eins erhöht wird) werden Elemente mit einer semantischen Versionsnummer, anhand derer die Kompatibilität

vorangegangener Änderungen ablesbar ist, verwendet. Durch die Einführung der semantischen Versionsnummern, ist den aktuell entstehenden K-Matrix-Releases zu entnehmen, wie die Kompatibilität der Elemente zu den Vorgänger-Releases verhält.

Der Editor des K-Matrix-Entwicklungswerkzeugs erhielt eine Erweiterung zur automatischen Durchführung von Änderungen als Änderungskopien. Diese legt basierend auf dem zu verändernden Kommunikationselement ein neues Kommunikationselement mit neuer Major-Version an, hinterlegt die Vorgänger-Nachfolger-Relation in der Datenbank, und bietet bezüglich der untergeordneten Kommunikationselemente die Wahl, diese unverändert zu übernehmen, oder neue Major-Versionen anzulegen.

Die identifizierte Anforderung an den AUTOSAR Standard, dass Empfänger nachträglich erweiterte Kommunikationselemente (z. B. um zusätzliche Signale ergänzte PDUs) tolerieren, um Änderungskopien zu ermöglichen, wurde in Abstimmung mit dem Gremium in den Standard eingebracht. Ab Release 2022-11 wird durch die neue Anforderung RS_MAIN_00129 [AUT22, S. 31] explizit sichergestellt, sodass die bisher implizit vorhandenen Mechanismen dauerhaft erhalten bleiben.

7.2.2 Nächste Schritte

Für weitere Tests sollte ein kleines Set von Steuergeräten (entweder ein Teilsystem, oder die an einen einzelnen, gemeinsamen Bus angebundenen ECUs, vgl. Abs. 4.1.2) ausgewählt werden, für die ein schnellerer Release-Zyklus (z. B. 6-wöchig), als die einheitlichen drei Monate, gewünscht wird. Für dieses Set können probeweise Zwischen-Releases durchgeführt werden, während das restliche Netzwerk in unverändertem Rhythmus weiterentwickelt wird.

Die zentrale Rolle der K-Matrix – die Software jedes einzelnen Steuergeräts hängt davon ab – erfordert ein hohes Maß an Sorgfalt bei Eingriffen in den Prozess. Ein unbrauchbares K-Matrix-Release würde die gesamte E/E-Entwicklung blockieren, bis ein Ersatz geschaffen wird. Durch den unveränderten Prozess für das Gros an ECUs und den experimentellen Teil in einem

Zwischen-Release kann das Risiko minimiert werden: Im Fehlerfall wäre nur das kleine Set von Steuergeräten betroffen. Zur vollständigen Korrektur eines schwerwiegenden Fehlers ließe sich das Zwischen-Release verwerfen und zum nächsten klassischen Release-Termin ein Einheits-Release in bewährter Form liefern.

Bereits ab der ersten Änderungskopie können an den kopierten Elementen Lebensdauermarkierungen gesetzt werden, sodass der Ablauf deren Restlebensdauern beginnt. Die praktische Anwendung des Ausphasungsmechanismus, in Form erster, entfernter Kommunikationselemente, stellt die zweite Stufe der realen Umsetzung des Konzeptes dar. Sobald Elemente entfernt, d. h. gezielt inkompatible Änderungen vorgenommen werden, wirkt sich das auf Steuergeräte aus, welche nicht zu der Probemenge gehören. Alle Steuergeräte, die auszuphasende Elemente verwenden, müssen spätestens zu dem betreffenden Haupt-Release an die veränderten Interfaces angepasste Software erhalten.

Bei hierarchischer Versionierung in der vorgeschlagenen Form wird, mit der Verwendung eines Signals oder eines Services, dessen Mindestlebensdauer akzeptiert (d. h. es wird die Bereitschaft erklärt, innerhalb der deklarierten Zeit einen neuen Softwarestand liefern zu können, falls dies notwendig wird). In der Übergangsphase besteht die Herausforderung, dass für Elemente rückwirkend Mindestlebensdauern festgelegt, und mit allen potenziell betroffenen Steuergeräteprojekten abgestimmt werden müssen.

Der letzte Schritt, zur vollständigen Umsetzung, wäre eine Vergrößerung des Abstands zwischen den Haupt-Releases. Diese fänden dann nur noch sechsmonatig oder jährlich statt, um die Entwicklungskosten der beständigeren Steuergeräte zu reduzieren. Die bisherigen vierteljährlichen Releases würden dann zu kompatibel zu haltenden Zwischen-Releases.

7.3 Zukünftige Forschungsfelder

7.3.1 Optimale Strategien zur Festlegung von Mindestlebensdauern

Anhand der Daten eines vergangenen Entwicklungsprozesses wurde rückwirkend gezeigt, dass Mindestlebensdauern von 6 bis 12 Monaten möglich gewesen wären. In zukünftigen Entwicklungsprozessen müssen Entscheidungen bezüglich der für neue Kommunikationselemente zu vergebenden Mindestlebensdauern während des laufenden Prozesses getroffen werden, wenn die Eigenschaften der nachfolgenden Releases und die bevorstehenden Änderungsmengen nicht bekannt sind.

Vorliegen werden die aktuelle sowie die vergangenen Buslasten, die Mindestlebensdauern der im aktuellen Release enthaltenen Elemente (und daraus ableitbar die zukünftigen Zeitpunkte, zu denen diese entfernt werden) sowie die Änderungsanträge für das jeweils nächste Release.

Es gilt Strategien zu entwickeln, mit denen unter diesen Bedingungen bestmögliche Entscheidungen für die Mindestlebensdauern einzelner Objekte getroffen werden können. Bis es solche Strategien gibt, kann mit einheitlichen Werten (vgl. Abs. 6.1.3) gearbeitet werden.

7.3.2 Ausweitung der Hierarchie auf Hard- u. Software-Ebenen

Die durchgeführte Untersuchung beschränkte sich auf die Kompatibilität der AUTOSAR Kommunikation in den OSI-Schichten 5–7, in denen die K-Matrix-Entwicklung stattfindet. Eine Ausweitung dieses Geltungsbereichs sollte zukünftig untersucht werden.

So könnten beispielsweise als Ebene oberhalb der Steuergeräteschnittstelle das Steuergerät eingeführt werden, mit der Möglichkeit mehrere Schnittstellenversionen pro ECU zu führen. Dadurch könnten Verblockungen zwischenzeitlich weiter entfernten (d. h. mit abweichenden Bustechnologien ausgestatteten) Baureihen ermöglicht werden. So könnten neue Funktionen auch Haltern bestehender Fahrzeuge angeboten (vgl. Abschnitt 2.6.5) oder die Ersatzteillogistik vereinfacht werden. Auch denkbar wäre eine Verwendung durch Zubehörhersteller, die für die Fahrzeuge verschiedener OEMs verwendbare Ersatzsteuergeräte anbieten.

Oberhalb des Steuergerätes könnte eine weitere Hierarchieebene festgelegt werden. Dafür kommen entweder das Teilsystem oder das Teilnetzwerk (z. B. ein einzelner Bus oder eine Domäne) infrage. So könnten Teilnetz bzw. System vollständig an externe Zulieferer beauftragt und ggf. später durch kompatible Teilnetze mit erweitertem Funktionsumfang von anderen Herstellern ersetzt werden.

In [Tra17] wird die E/E-Architektur eines Fahrzeugherstellers als „Hierarchische E/E-Architektur“ bezeichnet. Diese Sichtweise auf die Gesamtarchitektur gilt es, um Kompatibilitätsmechanismen zu ergänzen und in einen iterativen Entwicklungsprozess zu integrieren.

7.3.3 Auswirkung auf Strategien für Over-the-Air-Updates

Die erwartete Zunahme von Software Over-the-Air-Updates [Gui18] erfordert eine Absicherung der verschiedenen Softwarekomponenten und deren Zusammenspiel. Durch die hohe Zahl an im Feld befindlichen Varianten (vgl. Abs. 3.6) stellt dies eine wesentliche Herausforderung dar.

Es ist zu prüfen, ob durch hierarchische Versionierung Teile dieser Herausforderung bewältigt werden können. Indem eine weitere Hierarchieebene für Softwareabsicherungen definiert wird, könnten Softwareabsicherungen als kompatibel / inkompatibel modelliert werden. Im Falle von anstehenden OTAs wären dann nur die inkompatiblen Absicherungen neu durchzuführen.

7.3.4 Übertragung auf Softwareprojekte außerhalb der Automobilindustrie

Eine potenzielle weitere Anwendung für hierarchische Versionierung liegt im Bereich der allgemeinen Softwareentwicklung, außerhalb der Automobilindustrie (z. B. Smartphone Apps, PC-Programme, oder Server-/Cloudsoftware). Größere Softwareprojekte basieren auf zahlreichen Bibliotheken, die teilweise von verschiedenen Herstellern geliefert werden (vergleichbar mit dem Netzwerk eines Fahrzeugs, das aus Steuergeräten verschiedener Lieferanten besteht) und in unterschiedlichen Versionen vorliegen können. Aus diesem Bereich stammt das, durch die hierarchische Versionierung aufgegriffene, Konzept der semantischen Versionierung [Pre13], um die Kompatibilitätseigenschaften unterschiedlicher Versionen kenntlich zu machen.

Einige Softwaremodule oder Bibliotheken bauen selbst auf anderen Bibliotheken und Modulen auf und haben Anforderungen an deren Versionen (dies kann als eine Hierarchie aus Abhängigkeiten dargestellt werden). Bei größeren Projekten wird das Erfüllen sämtlicher Anforderungen zu einer Herausforderung, die teilweise als „Abhängigkeitshölle“ bezeichnet wird [Bir05][Jan08, S. 325].

Möglicherweise lässt sich das Konzept der hierarchischen Versionierung auf diese Problemstellung übertragen, oder an diese anpassen.

Literatur

[Abt24] Abteilung Statistik des Kraftfahrt-Bundesamtes: Bestand. Flensburg, 1. Jan. 2024. URL: https://www.kba.de/DE/Statistik/Fahrzeuge/Bestand/bestand_node.html (besucht am 08. 06. 2024) (siehe S. 67).

[Air23] Airbus SE, Hrsg.: Annual Report 2022. Leiden, Niederlande, 2023 (siehe S. 12).

[Ala20] Alaei, Reza; Moallem, Payman und Bohlooli, Ali: „Statistical based algorithm for reducing bit stuffing in the Controller Area Networks". In: *Microelectronics Journal* 101 (2020). DOI: 10.1016/j.mejo.2020.104794 (siehe S. 102).

[Arn20] Arne Babenhauserheide: Mercurial source control management. Mercurial Community. 2020. URL: https://www.mercurial-scm.org/about (besucht am 09. 06. 2024) (siehe S. 89).

[Aut18] Automobilwoche.de: „Software-Probleme: Audi verschiebt offenbar Auslieferungsstart des e-tron". In: (21. Okt. 2018). URL: https://www.automobilwoche.de/article/20181021/NACHRICHTEN/181029997/software-probleme-audi-verschiebt-offenbar-auslieferungsstart-des-e-tron (besucht am 09. 06. 2024) (siehe S. 79).

[AUT22] AUTOSAR GbR, Hrsg.: Main Requirements. Version FO-R22-11. Hörgertshausen, 24. Nov. 2022. URL: https://www.autosar.org/fileadmin/standards/R22-11/FO/AUTOSAR_RS_Main.pdf (besucht am 08. 06. 2024) (siehe S. 193, 214).

[AUT23a] AUTOSAR GbR, Hrsg.: Application Interfaces User Guide. Version CP-R23-11. Hörgertshausen, 23. Nov. 2023. URL: https://www.autosar.org/fileadmin/standards/R23-11/CP/AUTOSAR_CP_EXP_AIUserGuide.pdf (besucht am 08. 06. 2024) (siehe S. 42, 47, 50).

[AUT23b] AUTOSAR GbR, Hrsg.: Generic Structure Template. Version FO-R23-11. Hörgertshausen, 23. Nov. 2023. URL: https://www.autosar.org/fileadmin/standards/R23-11/FO/AUTOSAR_FO_TPS_GenericStructureTemplate.pdf (besucht am 08. 06. 2024) (siehe S. 43).

[AUT23c] AUTOSAR GbR, Hrsg.: Glossary. Version FO-R23-11. Hörgertshausen, 23. Nov. 2023. URL: https://www.autosar.org/fileadmin/standards/R23-11/FO/AUTOSAR_FO_TR_Glossary.pdf (besucht am 08. 06. 2024) (siehe S. 62).

[AUT23d] AUTOSAR GbR, Hrsg.: Layered Software Architecture. Version CP-R21-11. Hörgertshausen, 23. Nov. 2023. URL: https://www.autosar.org/fileadmin/standards/R23-11/CP/AUTOSAR_CP_EXP_LayeredSoftwareArchitecture.pdf (besucht am 08. 06. 2024) (siehe S. 36, 38, 39, 54).

[AUT23e] AUTOSAR GbR, Hrsg.: Methodology. Version CP-R23-11. Hörgertshausen, 23. Nov. 2023. URL: https://www.autosar.org/fileadmin/standards/R23-11/CP/AUTOSAR_CP_TR_Methodology.pdf (besucht am 08. 06. 2024) (siehe S. 64).

[AUT23f] AUTOSAR GbR, Hrsg.: Requirements on Communication. Version CP-R23-11. Hörgertshausen, 23. Nov. 2023. URL: https://www.autosar.org/fileadmin/standards/R23-11/CP/AUTOSAR_CP_SRS_COM.pdf (besucht am 08. 06. 2024) (siehe S. 41).

[AUT23g] AUTOSAR GbR, Hrsg.: SOME/IP Protocol Specification. Version FO-R23-11. Hörgertshausen, 23. Nov. 2023. URL: https://www.autosar.org/fileadmin/standards/R23-11/FO/AUTOSAR_FO_PRS_SOMEIPProtocol.pdf (besucht am 08. 06. 2024) (siehe S. 6, 38, 60, 62, 272).

[AUT23h] AUTOSAR GbR, Hrsg.: SOME/IP Service Discovery Protocol Specification. Version FO-R23-11. Hörgertshausen, 23. Nov. 2023. URL: https://www.autosar.org/fileadmin/standards/R23-11/FO/AUTOSAR_FO_PRS_SOMEIPServiceDiscoveryProtocol.pdf (besucht am 08. 06. 2024) (siehe S. 195).

[AUT23i] AUTOSAR GbR, Hrsg.: System Template. Version CP-R23-11. Hörgertshausen, 23. Nov. 2023. URL: https://www.autosar.org/fileadmin/standards/R23-11/CP/AUTOSAR_CP_TPS_SystemTemplate.pdf (besucht am 08. 06. 2024) (siehe S. 44–46).

[AUT23j] AUTOSAR GbR, Hrsg.: Virtual Functional Bus. Version CP-R23-11. Hörgertshausen, 23. Nov. 2023. URL: https://www.autosar.org/fileadmin/standards/R23-11/CP/AUTOSAR_CP_EXP_VFB.pdf (besucht am 08. 06. 2024) (siehe S. 41).

[Ban21] Bandur, Victor; Selim, Gehan; Pantelic, Vera und Lawford, Mark: „Making the Case for Centralized Automotive E/E Architectures“. In: *IEEE Transactions on Vehicular Technology* 70.2 (2021), S. 1230–1245. DOI: 10.1109/TVT.2021.3054934 (siehe S. 17, 197).

[Bau67] Baumann, Von Günther: „Eine elektronisch gesteuerte Kraftstoffeinspritzung für Ottomotoren“. In: *Bosch Techn. Ber.* 2 (1967), S. 107–116 (siehe S. 1).

[Bay23] Bayerische Motoren Werke AG, Hrsg.: BMW Group Bericht 2022. München, 2023 (siehe S. 11, 12).

[Ber96] Bernstein, Philip A.: „Middleware: a model for distributed system services“. In: *Communications of the ACM* 39.2 (1996), S. 86–98 (siehe S. 38).

[Bir05] Birsan, Dorian: „On Plug-ins and Extensible Architectures“. EN. In: *Queue* 3.2 (2005), S. 40–46. DOI: 10.1145/1053331.1053345 (siehe S. 218).

[Bis07] Biswas, Pratik K.: „Autonomic Software Release Management for Communications Networks“. In: *2007 10th IFIP/IEEE International Symposium on Integrated Network Management* (München, 21.–25. Mai 2007). International Federation for Information Processing und Institute of Electrical and Electronics Engineers. Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE), 2007, S. 179–188. DOI: 10.1109/INM.2007.374782 (siehe S. 92).

[Boe23] Boeing, Hrsg.: The Boeing Company Annual Report 2022. Chicago, USA, 2023 (siehe S. 12).

[Bor19] Borky, John M. und Bradley, Thomas H.: Effective Model-Based Systems Engineering. Cham: Springer, 2019. 779 S. doi: 10.1007/978-3-319-95669-5 (siehe S. 64, 66, 67, 69).

[Bor21] Borgeest, Kai: Elektronik in der Fahrzeugtechnik. Hardware, Software, Systeme und Projektmanagement. 4., aktualisierte und erweiterte Auflage. Borgeest, Kai (VerfasserIn). Wiesbaden: Springer Vieweg, 2021. 552 S. doi: 10.1007/978-3-658-23664-9 (siehe S. 20).

[Bra18] Braun, Lisa: Modellbasierte Design-Space-Exploration nichtfunktionaler Auslegungskriterien des Fahrzeugenergiebordnetzes. KITopen, 2018. 233 S. doi: 10.5445/IR/1000081298 (siehe S. 11).

[Bru17] Brunner, Stefan; Roder, Jurgen; Kucera, Markus und Waas, Thomas: „Automotive E/E-architecture enhancements by usage of ethernet TSN". In: *2017 13th Workshop on Intelligent Solutions in Embedded Systems (WISES)* (Hamburg, 12.–13. Juni 2017). Institute of Electrical and Electronics Engineers (IEEE), 2017, S. 9–13. doi: 10.1109/WISES.2017.7986925 (siehe S. 25, 27).

[Bun21] Bundesministerium der Justiz: Straßenverkehrs-Zulassungs-Ordnung vom 26. April 2012 (BGBl. I S. 679), die zuletzt durch Artikel 11 des Gesetzes vom 12. Juli 2021 (BGBl. I S. 3091) geändert worden ist. 2021. url: https://www.gesetze-im-internet.de/stvzo_2012/BJNR067910012.html (siehe S. 34).

[Bur18] Burkacky, Ondrej; Deichmann, Johannes; Doll, Georg und Knochenhauer, Christian: Rethinking car software and electronics architecture. McKinsey Center for Future Mobility, 2018. url: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/rethinking-car-software-and-electronics-architecture#/ (besucht am 09. 06. 2024) (siehe S. 197).

[Bur20] Burkacky, Ondrej; Deichmann, Johannes; Klein, Benjamin; Pototzky, Klaus und Scherf, Gundbert: Cybersecurity in automotive. Mastering the challenge. Hrsg. von McKinsey & Company. 2020. URL: https://www.gsaglobal.org/wp-content/uploads/2020/03/Cybersecurity-in-automotive-Mastering-the-challenge.pdf (besucht am 14. 07. 2024) (siehe S. 2, 4, 66).

[Çel14] Çela, Arben; Ben Gaid, Mongi; Li, Xu-Guang und Niculescu, Silviu-Iulian: Optimal Design of Distributed Control and Embedded Systems. SpringerLink Bücher. Cham: Springer, 2014. 288 S. DOI: 10.1007/978-3-319-02729-6 (siehe S. 3).

[Cen17] Center for Automotive Research: Automotive Product Development Cycles and the Need for Balance with the Regulatory Environment. 2017. URL: https://www.cargroup.org/automotive-product-development-cycles-and-the-need-for-balance-with-the-regulatory-environment/ (besucht am 09. 06. 2024) (siehe S. 4, 67).

[Cha14] Chacon, Scott und Straub, Ben: Pro git. Apress, 2014. URL: https://git-scm.com/book/en/v2 (besucht am 09. 06. 2024) (siehe S. 89).

[Con98] Conradi, Reidar und Westfechtel, Bernhard: „Version models for software configuration management“. In: *ACM Computing Surveys* 30.2 (1998), S. 232–282. DOI: 10.1145/280277.280280 (siehe S. 86).

[Coo00] Cook, Stephen C.: „7.5. 2 What the Lessons Learned from Large, Complex, Technical Projects Tell Us about the Art of Systems Engineering“. In: *INCOSE International Symposium*. Bd. 10. Wiley Online Library. 2000, S. 680–687 (siehe S. 67).

[Dai17] Daigmorte, Hugo und Boyer, Marc: „Evaluation of admissible CAN bus load with weak synchronization mechanism“. In: *RTNS '17: Proceedings of the 25th International Conference on Real-Time Networks and Systems* (Grenoble, Frankreich, 4.–6. Okt. 2017). Hrsg. von Schweißguth, Eike Björn; Danielis, Peter; Timmermann, Dirk; Parzyjegla, Helge und Mühl, Gero.

New York, USA: ACM, 2017, S. 277–286. DOI: 10.1145/3139258.3139261 (siehe S. 97).

[Dai23] DAIMLER TRUCK, Hrsg.: Geschäftsbericht 2022. Stuttgart, 2023 (siehe S. 12).

[Dar21] DARKO DURISIC: „AUTOSAR (AUTomotive Open System ARchitecture)“. In: *Automotive Software Architectures. An Introduction.* Hrsg. von STARON, Miroslaw. 2nd ed. 2021. Cham: Springer, 2021, S. 97–136. DOI: 10.1007/978-3-030-65939-4_5 (siehe S. 37, 51).

[Dav07] DAVIS, Robert I.; BURNS, Alan; BRIL, Reinder J. und LUKKIEN, Johan J.: „Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised“. In: *Real-Time Systems* 35.3 (2007), S. 239–272. DOI: 10.1007/s11241-007-9012-7 (siehe S. 187).

[Day83] DAY, J. D. und ZIMMERMANN, H.: „The OSI reference model“. In: *Proceedings of the IEEE* 71.12 (1983), S. 1334–1340. DOI: 10.1109/PROC.1983.12775 (siehe S. 53, 55, 56).

[DeM05] DEMEIS, Rick: „Cars sag under weighty wiring“. In: *EDN* 10 (2005), S. 24. URL: https://www.edn.com/cars-sag-under-weighty-wiring/ (besucht am 09. 06. 2024) (siehe S. 187).

[Deu13] DEUTER, Andreas: „Slicing the V-Model – Reduced Effort, Higher Flexibility“. In: *ICGSE '13: Proceedings of the 2013 IEEE 8th International Conference on Global Software Engineering (ICGSE)* (Bari, Italien, 26.–29. Aug. 2013). Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE), 2013, S. 1–10. DOI: 10.1109/ICGSE.2013.10 (siehe S. 70).

[Dol18] DOLL, Nikolaus: „VW beendet den Unikate-Wahnsinn“. In: *Welt* (7. Dez. 2018). URL: https://www.welt.de/wirtschaft/article185131950/VW-nimmt-seinen-Kunden-die-grosse-Auswahl-weg.html (besucht am 09. 06. 2024) (siehe S. 5, 94).

[Dom20] Dominguez, Xavier; Mantilla-Perez, Paola und Arboleya, Pablo: „Toward Smart Vehicular dc Networks in the Automotive Industry: Process, computational tools, and trends in the design and simulation of vehicle electrical distribution systems“. In: *IEEE Electrification Magazine* 8.1 (2020), S. 61–68. doi: 10.1109/MELE.2019.2962890 (siehe S. 5).

[Don18] Donovan Porter: 100BASE-T1-Ethernet - Die Entwicklung der Automobil-Netzwerke. next-mobility. 2018. url: https://www.next-mobility.news/100base-t1-ethernet-die-entwicklung-der-automobil-netzwerke-a-749666/ (besucht am 09. 06. 2024) (siehe S. 19).

[Drö00] Dröschel, Wolfgang, Hrsg.: Das V-Modell 97. Der Standard für die Entwicklung von IT-Systemen mit Anleitung für den Praxiseinsatz. München und Wien: Oldenbourg, 2000 (siehe S. 69).

[Ebe20] Ebert, Christof und Kirschke-Biller, Frank: „Agile systems engineering for critical systems“. In: *20. Internationales Stuttgarter Symposium. Automobil- und Motorentechnik*. Hrsg. von Bargende, Michael; Reuss, Hans-Christian und Wagner, Andreas. 1. Auflage 2020. Proceedings. Wiesbaden: Springer Vieweg, 2020, S. 573–581. doi: 10.1007/978-3-658-30995-4_49 (siehe S. 70).

[Eis08] Eisner, Howard: Essentials of project and systems engineering management. John Wiley & Sons, 2008 (siehe S. 64, 67).

[Eis19] Eisele, Georg; Kauth, Michael; Steffens, Christoph und Glusk, Patrick: „Automotive megatrends and their impact on NVH“. In: *19. Internationales Stuttgarter Symposium*. Hrsg. von Bargende, Michael; Reuss, Hans-Christian; Wagner, Andreas und Wiedemann, Jochen. Proceedings. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2019, S. 523–539. doi: 10.1007/978-3-658-25939-6_45 (siehe S. 1).

[Ekl05] Eklund, Ulrik; Askerdal, Örjan; Granholm, Johan; Alminger, Anders und Axelsson, Jakob: „Experience of introducing reference architectures in the development of automotive electronic systems“. In: *ACM SIGSOFT Software Engineering Notes* 30.4 (2005), S. 1–6. doi: 10.1145/1082983.1083195 (siehe S. 197).

[Eng14] Engstler, Martin, Hrsg.: Projektmanagement und Vorgehensmodelle 2014. Soziale Aspekte und Standardisierung ; gemeinsame Tagung der Fachgruppen Projektmanagement (WI-PM) und Vorgehensmodelle (WI-VM) im Fachgebiet Wirtschaftsinformatik der Gesellschaft für Informatik e.V. ; 16. und 17. Oktober 2014 in Stuttgart. Bd. 236. GI Edition Proceedings. Bonn: Ges. für Informatik, 2014. 152 S. (siehe S. 70).

[Fre17] Freyer, Ulrich: Nachrichten-Übertragungstechnik. Grundlagen, Komponenten, Verfahren und Anwendungen der Informations-, Kommunikations- und Medientechnik. 7., neu bearbeitete Auflage. Lernbücher der Technik. Freyer, Ulrich (VerfasserIn). München: Hanser und Ciando, 2017. 573 S. url: http://ebooks.ciando.com/book/index.cfm/bok_id/2261494 (siehe S. 18).

[Für09] Fürst, Simon; Mössinger, Jürgen; Bunzel, Stefan; Weber, Thomas; Kirschke-Biller, Frank; Heitkämper, Peter; Kinkelin, Gerulf; Nishikawa, Kenji und Lange, Klaus: „AUTOSAR–A Worldwide Standard is on the Road“. In: *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden.* Bd. 62. 2009 (siehe S. 37).

[Gal09] Gallus, Jürgen: „Elektronik im Kraftfahrzeug“. In: *Handbuch Verkehrsunfallrekonstruktion.* Hrsg. von Burg, Heinz. ATZ / MTZ-Fachbuch. Wiesbaden: Vieweg+Teubner, 2009, S. 823–834. doi: 10.1007/978-3-8348-9974-3_41 (siehe S. 3).

[Gui18] Guissouma, Houssem; Klare, Heiko; Sax, Eric und Burger, Erik: „An Empirical Study on the Current and Future Challenges of Automotive Software Release and Configuration Management“. In: *44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (Prag, 29.–31. Aug.

2018). Hrsg. von Bures, Tomas und Angelis, Lefteris. Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE), 2018, S. 298–305. DOI: 10.1109/SEAA.2018.00056 (siehe S. 2, 37, 92, 217).

[Hak15] Hakuli, Stephan und Krug, Markus: „Virtuelle Integration". In: *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Winner, Hermann; Hakuli, Stephan; Lotz, Felix und Singer, Christina. 3., überarbeitete und ergänzte Auflage. ATZ / MTZ-Fachbuch. Wiesbaden: Springer Vieweg, 2015, S. 125–138. DOI: 10.1007/978-3-658-05734-3_8 (siehe S. 71).

[Han13] Hank, Peter; Müller, Steffen; Vermesan, Ovidiu und van den Keybus, Jeroen: „Automotive Ethernet: In-vehicle Networking and Smart Mobility". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. Design Automation and Test in Europe (Grenoble, France, 18.–22. März 2013). New Jersey: Institute of Electrical and Electronics Engineers (IEEE), 2013, S. 1735–1739. DOI: 10.7873/DATE.2013.349 (siehe S. 26).

[Han22] Hansen, Paul: „On Automotive Electronics. Three-Net Physical Architecture Can Lead to Mass Produced Wire Harnesses". In: *ATZ electronics worldwide* 17.10 (2022), S. 26–29. DOI: 10.1007/s38314-022-0824-y (siehe S. 11).

[Har12] Hartwich, Florian: CAN with flexible data-rate. Nürnberg: CAN in Automation, 2012. URL: https://old.can-cia.org/fileadmin/resources/documents/proceedings/2012_hartwich.pdf (besucht am 10. 06. 2024) (siehe S. 96).

[Has16] Hassan El-Bouloumi: Die zentralisierte E/E-Architektur: Folgen und Herausforderungen für OEMs. POLARIXPARTNER GmbH. 2016. URL: https://www.polarixpartner.com/files/content/documents/whitepaper/2016/20160919_whitepaper_EE_architektur/polarixpartner_whitepaper_EE_architektur_20160919_DE.pdf (besucht am 10. 06. 2024) (siehe S. 4).

[Heb09] Hebig, Regina: Methodology and Templates in AUTOSAR. Hrsg. von Hasso Plattner Institut. 2009. URL: https://hpi.de/fileadmin/user_upload/fachgebiete/giese/Ausarbeitungen_AUTOSAR0809/Methodology_hebig.pdf (besucht am 11. 06. 2024) (siehe S. 74).

[Hei24] Heimes, Frank: Releases. Hrsg. von ubuntu wiki. 2024. URL: https://wiki.ubuntu.com/Releases (besucht am 11. 06. 2024) (siehe S. 91).

[Hel17] Helmling, Markus: „Service-oriented Architectures and Ethernet in Vehicles: Towards Data Centers on Wheels with Model-based Methods“. In: (2017). URL: https://assets.vector.com/cms/content/know-how/_technical-articles/PREEvision/PREEvision_SOA_Ethernet_ElektronikAutomotive_201703_PressArticle_EN.pdf (besucht am 24. 06. 2024) (siehe S. 6, 89).

[Hun02] Hunt, Craig: TCP/IP network administration, 3rd Edition. Bd. 2. O'Reilly Media Inc., 2002. 746 S. (siehe S. 59).

[INC21] INCOSE - International Council on Systems Engineering: Systems Engineering. Transforming Needs to Solutions. 2021. URL: https://www.incose.org/systems-engineering (besucht am 11. 06. 2024) (siehe S. 67).

[Ins16a] Institute of Electrical and Electronics Engineers, Hrsg.: IEEE Standard for Ethernet Amendment 4: Physical Layer Specifications and Management Parameters for 1 Gb/s Operation over a Single Twisted-Pair Copper Cable. New York, 20. Sep. 2016. DOI: 10.1109/IEEESTD.2016.7564011 (siehe S. 60).

[Ins16b] Institute of Electrical and Electronics Engineers, Hrsg.: IEEE standard for Ethernet, Amendment 1: Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable (100BASE-T1). New York, 7. März 2016. DOI: 10.1109/IEEESTD.2016.7433918 (siehe S. 59).

[Int05a] International Organization for Standardization, Hrsg.: ISO 17356-4:2005 – Road vehicles – Open interface for embedded automotive applications – Part 4: OSEK/VDX Communication (COM). Genf, 2005. URL: https://www.iso.org/standard/40118.html (besucht am 12. 06. 2024) (siehe S. 62).

[Int05b] Internet Engineering Task Force, Hrsg.: RFC-4122: A Universally Unique IDentifier (UUID) URN Namespace. Request for Comments. 1. Juli 2005. URL: https://datatracker.ietf.org/doc/html/rfc4122 (besucht am 24. 06. 2024) (siehe S. 49, 273, 276).

[Int12] International Organization for Standardization, Hrsg.: ISO 11446-1:2012 – Road vehicles – Connectors for the electrical connection of towing and towed vehicles – Part 1: 13-pole connectors for vehicles with 12 V nominal supply voltage not intended to cross water fords. Genf, 1. März 2012. URL: https://www.iso.org/standard/51374.html (besucht am 12. 06. 2024) (siehe S. 204).

[Int17] Internet Engineering Task Force, Hrsg.: RFC-8200: Internet Protocol – Version 6 (IPv6) Specification. Request for Comments. 1. Juli 2017. URL: https://datatracker.ietf.org/doc/html/rfc8200 (besucht am 12. 06. 2024) (siehe S. 60).

[Int24a] International Organization for Standardization, Hrsg.: ISO 11898-1:2024 – Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical coding sublayer. Genf, 1. Mai 2024. URL: https://www.iso.org/standard/86384.html (besucht am 12. 06. 2024) (siehe S. 96, 101, 102).

[Int24b] International Organization for Standardization, Hrsg.: ISO/IEC/IEEE 24748-1:2024 – Systems and software engineering – Life cycle management – Part 1: Guidelines for life cycle management. Genf, 1. März 2024. URL: https://www.iso.org/standard/84709.html (besucht am 12. 06. 2024) (siehe S. 64, 67).

[Int80] Internet Engineering Task Force, Hrsg.: RFC-768: User Datagram Protocol. Request for Comments. 1. Aug. 1980. URL:

https://datatracker.ietf.org/doc/html/rfc768 (besucht am 12. 06. 2024) (siehe S. 60).

[Int81a] Internet Engineering Task Force, Hrsg.: RFC-791: Internet Protocol – DARPA Internet Program – Protocol Specification. Request for Comments. 1. Sep. 1981. URL: https://datatracker.ietf.org/doc/html/rfc791 (besucht am 12. 06. 2024) (siehe S. 60).

[Int81b] Internet Engineering Task Force, Hrsg.: RFC-793: Transmission Control Protocol – DARPA Internet Program – Protocol Specification. Request for Comments. 1. Sep. 1981. URL: https://datatracker.ietf.org/doc/html/rfc793 (besucht am 12. 06. 2024) (siehe S. 60).

[Int89a] International Organization for Standardization, Hrsg.: ISO/IEC 7498-2:1989 – Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture. Genf, 1. Feb. 1989. URL: https://www.iso.org/standard/14256.html (besucht am 12. 06. 2024) (siehe S. 53).

[Int89b] International Organization for Standardization, Hrsg.: ISO/IEC 7498-4:1989 – Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management framework. Genf, 1. Nov. 1989. URL: https://www.iso.org/standard/14258.html (besucht am 12. 06. 2024) (siehe S. 53).

[Int94] International Organization for Standardization, Hrsg.: ISO/IEC 7498-1:1994 – Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model. Genf, 1. Nov. 1994. URL: https://www.iso.org/standard/20269.html (besucht am 12. 06. 2024) (siehe S. 53–56, 58).

[Int97] International Organization for Standardization, Hrsg.: ISO/IEC 7498-3:1997 – Information technology – Open Systems Interconnection – Basic Reference Model: Naming and addressing. Genf, 1. Apr. 1997. URL: https://www.iso.org/standard/25022.html (besucht am 12. 06. 2024) (siehe S. 53).

[Jan08] Jang, Michael: Linux Annoyances for Geeks. Getting the Most Flexible System in the World Just the Way You Want It. Sebastopol: O'Reilly Media Inc, 2008. 512 S. URL: http://gbv.eblib.com/patron/FullRecord.aspx?p=443354 (siehe S. 218).

[Joh04] Johnson, R. W.; Evans, J. L.; Jacobsen, P.; Thompson, J.R.R. und Christopher, M.: „The Changing Automotive Environment: High-Temperature Electronics". In: *IEEE Transactions on Electronics Packaging Manufacturing* 27.3 (2004), S. 164–176. DOI: 10.1109/TEPM.2004.843109 (siehe S. 10).

[Kam19] Kampmann, Alexandru; Alrifaee, Bassam; Kohout, Markus; Wustenberg, Andreas; Woopen, Timo; Nolte, Marcus; Eckstein, Lutz und Kowalewski, Stefan: „A Dynamic Service-Oriented Software Architecture for Highly Automated Vehicles". In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. 2019 IEEE Intelligent Transportation Systems Conference - ITSC (Auckland, New Zealand, 27.–30. Okt. 2019). Institute of Electrical and Electronics Engineers (IEEE), 2019, S. 2101–2108. DOI: 10.1109/ITSC.2019.8916841 (siehe S. 6, 79–81, 96).

[Kat14] Katumba, Brian und Knauss, Eric: „Agile Development in Automotive Software Development: Challenges and Opportunities". In: *Product-focused software process improvement. 15th International Conference, PROFES 2014, Helsinki, Finland, December 10-12, 2014 : proceedings*. Hrsg. von Jedlitschka, Andreas; Kuvaja, Pasi; Kuhrmann, Marco; Männistö, Tomi; Münch, Jürgen und Raatikainen, Mikko. Bd. 8892. Lecture Notes in Computer Science 8892. Cham: Springer, 2014, S. 33–47. DOI: 10.1007/978-3-319-13835-0_3 (siehe S. 3).

[Kat15] Katzenbach, Alfred: „Automotive". In: *Concurrent Engineering in the 21st Century*. Hrsg. von Stjepandić, Josip; Wognum, Nel und Verhagen, Wim J.C. Cham: Springer, 2015, S. 607–638. DOI: 10.1007/978-3-319-13776-6_21 (siehe S. 68).

[Kos20] Kossiakoff, Alexander; Biemer, Steven M.; Seymour, Samuel J. und Flanigan, David A.: Systems engineering. Principles and practice. Third edition. Wiley series in systems engineering and management. Hoboken, NJ: John Wiley & Sons, Inc, 2020 (siehe S. 64–68).

[Kra04] Krammer, Josef; Bornat, Pascal; Dengel, Günter; Kuttenkeuler, Stephan; Lukas, Rainer; Oberhofer, Klaus; Ruh, Jens; Stroop, Joachim und Quecke, Hans: „FIBEX – An Exchange Format for Networks Based on Field Busses“. In: *2nd Embedded Real Time Software Congress* (Toulouse, Frankreich). 2004 (siehe S. 45).

[Kug17] Kugele, Stefan; Obergfell, Philipp; Broy, Manfred; Creighton, Oliver; Traub, Matthias und Hopfensitz, Wolfgang: „On Service-Orientation for Automotive Software“. In: *2017 IEEE International Conference on Software Architecture (ICSA)*. 2017 IEEE International Conference on Software Architecture (ICSA) (Gothenburg, Sweden, 3.–7. Apr. 2017). Institute of Electrical and Electronics Engineers (IEEE), 2017, S. 193–202. doi: 10.1109/ICSA.2017.20 (siehe S. 6).

[Liu16] Liu, Bohan; Zhang, He und Zhu, Saichun: „An Incremental V-Model Process for Automotive Development“. In: *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. 2016 23rd Asia-Pacific Software Engineering Conference (APSEC) (Hamilton, New Zealand, 6.–9. Dez. 2016). Institute of Electrical and Electronics Engineers (IEEE), 2016, S. 225–232. doi: 10.1109/APSEC.2016.040 (siehe S. 70).

[Mai20] Maier, Thomas: „Volvo-Rückruf: Probleme beim automatischen Bremssystem“. In: *Auto Service Praxis* (16. März 2020). url: https://www.autoservicepraxis.de/rueckrufe/artikel/volvo-rueckruf-probleme-beim-automatischen-bremssystem-2581604 (besucht am 25. 06. 2024) (siehe S. 79).

[Mar11] Martin Glinz und Harald Gall: Software Engineering. Kapitel 20: Software-Konfigurationsverwaltung. Zürich: Universität

Zürich, 2011. URL: https://www.ifi.uzh.ch/dam/jcr:ffffffff-fc3b-5ce0-0000-00003ed6daef/Kapitel_20_Konfig_Verw.pdf (besucht am 24. 06. 2024) (siehe S. 88).

[Mar16] Martin Brückmann und Philipp Schumacher: Increasing Acceptance of the AUTOSAR Philosophy at Daimler. Unter Mitarb. von AUTOSAR. Gothenburg: Daimler AG, 28. Sep. 2016. URL: https://www.autosar.org/fileadmin/AOC/AOC_2016/Presentations/Daimler_Increasing_acceptance_of_AUTOSAR_BRUECKMANN.pdf (besucht am 27. 12. 2020) (siehe S. 22, 65).

[Mar19] Marner, Kristina; Theobald, Sven und Wagner, Stefan: „Real-Life Challenges in Automotive Release Planning". In: *Proceedings of the 2019 Federated Conference on Computer Science and Information Systems*. 2019 Federated Conference on Computer Science and Information Systems (1.–4. Sep. 2019). Annals of Computer Science and Information Systems. Institute of Electrical and Electronics Engineers (IEEE), 2019, S. 831–839. DOI: 10.15439/2019F326 (siehe S. 92).

[Mau18] Maul, Mario; Becker, Gerhard und Bernhard, Ulrich: „Serviceorientierte EE-Zonenarchitektur. Schlüsselelement für neue Marktsegmente". In: *ATZ elektronik* 13.1 (2018), S. 36–41. DOI: 10.1007/s35658-017-0105-3 (siehe S. 28).

[Mei12] Meinel, Christoph und Sack, Harald, Hrsg.: Internetworking. Technische Grundlagen und Anwendungen. X.media.press. Berlin und Heidelberg: Springer, 2012. 978 S. DOI: 10.1007/978-3-540-92940-6 (siehe S. 59, 61).

[Men20] Menzel, Stefan: „VW will die Software-Probleme rasch lösen. ID.3 kommt im Spätsommer". In: *Handelsblatt* (10. Juni 2020). URL: https://www.handelsblatt.com/unternehmen/industrie/volkswagen-vw-will-die-software-probleme-rasch-loesen-id-3-kommt-im-spaetsommer/25904772.html?ticket=ST-4022637-a4UUfLWBNc0MMZUHMfOb-ap2 (besucht am 25. 06. 2024) (siehe S. 79).

[Mer20a] Mercedes-Benz AG, Hrsg. Stuttgart, 6. Okt. 2020. URL: https://media.mercedes-benz.com/article/791f8f51-2e8e-4c54-871a-cbcea15ee609 (besucht am 08. 09. 2024) (siehe S. 72).

[Mer20b] Mercedes-Benz Gebrauchtteile: Außenspiegel LI. Artikelnummer: A2228100376. 2020. URL: https://www.mbgtc.de/Geschaeftsfeld/PKW-Modell/S-Klasse/Limousine/W222/AMG-S-63/Au-enspiegel-LI-oxid-7.html (besucht am 25. 06. 2024) (siehe S. 14).

[Mer23] Mercedes-Benz Group, Hrsg.: Geschäftsbericht 2022. Stuttgart, 2023 (siehe S. 11).

[Mic15] Michlmayr, Martin; Fitzgerald, Brian und Stol, Klaas-Jan: „Why and How Should Open Source Projects Adopt Time-Based Releases?“ In: *IEEE Software* 32.2 (2015), S. 55–63. DOI: 10.1109/MS.2015.55 (siehe S. 92).

[Mot23] Motortrend: Coding the Car. The Leaders Driving the Software Defined Vehicle. El Segundo, CA, USA: Motortrend, 2023. URL: www.motortrendgroup.com/wp-content/uploads/2023/09/UPDATED-Coding-The-Car-EBook-Download.pdf (besucht am 14. 07. 2024) (siehe S. 2, 4, 66).

[Mül17] Müller, Thomas Manfred und Tappe, Robert: „Elektronik-Architektur für automatisiertes Fahren und digitale Geschäftsmodelle“. In: *ATZ extra* 22.S3 (2017), S. 28–31. DOI: 10.1007/s35778-017-0031-2 (siehe S. 3).

[Nem19] Nemenz, Thomas und Lein Oliver: „Erstellung einer Simulation zur Analyse und Bewertung der Busleistungsfähigkeit anhand der Baureihe 223 (S-Klasse)“. Bachelorarbeit. Göppingen: Hochschule Esslingen, 2019 (siehe S. 96).

[Neu20] Neumann, Tim: Volvo ruft 736.000 Autos in die Werkstatt. Auto Zeitung. 2020. URL: https://www.autozeitung.de/volvo-rueckruf-196885.html# (besucht am 25. 06. 2024) (siehe S. 79).

[Nol01] Nolte, Thomas; Hansson, Hans; Norström, Christer und Punnekkat, Sasikumar: „Using bit-stuffing distributions in CAN analysis“. In: *IEEE Real-Time Embedded Systems Workshop*. Institute of Electrical and Electronics Engineers (IEEE), 2001. URL: http://www.es.mdh.se/pdf_publications/298.pdf (besucht am 25. 06. 2024) (siehe S. 102).

[Obj15] Object Management Group, Hrsg.: Meta Object Facility (MOF) Core Specification. Version 2.5. 1. Juni 2015. URL: https://www.omg.org/spec/MOF/2.5/PDF (besucht am 25. 06. 0204) (siehe S. 43, 45).

[Obj17] Object Management Group, Hrsg.: Unified Modeling Language. Version 2.5.1. 1. Dez. 2017. URL: https://www.omg.org/spec/UML/2.5.1/PDF (besucht am 15. 08. 2019) (siehe S. 42).

[Ott18] Otten, Stefan; Bach, Johannes; Wohlfahrt, Christoph; King, Christian; Lier, Jan; Schmid, Hermann; Schmerler, Stefan und Sax, Eric: „Automated Assessment and Evaluation of Digital Test Drives“. In: *Advanced Microsystems for Automotive Applications 2017. Smart Systems Transforming the Automobile*. Hrsg. von Zachäus, Carolin; Müller, Beate und Meyer, Gereon. Springer eBook Collection Engineering. Cham: Springer, 2018, S. 189–199. DOI: 10.1007/978-3-319-66972-4_16 (siehe S. 71).

[Pag06] Pagel, Mike und Brörkens, Mark: „Definition and Generation of Data Exchange Formats in AUTOSAR“. In: *Model driven architecture - foundations and applications. Second European conference, ECMDA-FA 2006, Bilbao, Spain, July 10 - 13, 2006 ; proceedings*. Hrsg. von Rensink, Arend. Bd. 4066. Lecture Notes in Computer Science 4066. Berlin und Heidelberg: Springer, 2006, S. 52–65. DOI: 10.1007/11787044_5 (siehe S. 47).

[Par13] Parzi: NetzwerkTopologien.png. Hrsg. von wikimedia.org. 2013. URL: https://commons.wikimedia.org/wiki/File:NetzwerkTopologien.png (besucht am 25. 06. 2024) (siehe S. 24).

[Pel05] Pelz, Georg; Oehler, Peter; Fourgeau, Eliane und Grimm, Christoph: „Automotive System Design and Autosar“. In: *Advances in Design and Specification Languages for SoCs. Selected Contributions from FDL'04*. Hrsg. von Boulet, Pierre. The ChDL series. New York, NY: Springer, 2005, S. 293–305. doi: 10.1007/0-387-26151-6_21 (siehe S. 37, 62).

[Per02] Persson, Niclas; Gustafsson, Fredrik und Drevö, Markus: „Indirect tire pressure monitoring using sensor fusion“. In: *Journal of Passenger Vehicle Systems*. Hrsg. von SAE International. 2002, S. 1657–1662. doi: 10.4271/2002-01-1250 (siehe S. 3).

[Pet20] Petzold, Hanjo; Halkenhaeusser, Heike; Rentschler, Ralf; Uttendorfer, Uwe; Lenz, Tobias; Kießling, Gerhard; Feuerbach, Martin und Steck, Andreas: „Produktdatenmanagement über den gesamten Lebenszyklus“. In: *Sonderprojekte ATZ/MTZ* 25.S1 (2020), S. 30–35. doi: 10.1007/s41491-020-0075-2 (siehe S. 95).

[Pil08] Pilato, C. Michael; Collins-Sussman, Ben und Fitzpatrick, Brian W.: Version Control with Subversion: Next Generation Open Source Version Control. O'Reilly Media, Inc, 2008 (siehe S. 89).

[Pis21] Pischinger, Stefan und Seiffert, Ulrich, Hrsg.: Vieweg Handbuch Kraftfahrzeugtechnik. 9. erweiterte und ergänzte Auflage. Wiesbaden: Springer Vieweg, 2021. doi: 10.1007/978-3-658-25557-2 (siehe S. 3, 66, 79).

[Pis93] Piscitello, David M. und Chapin, A. Lyman: Open systems networking: TCP/IP and OSI. Addison-Wesley professional computing series. Addison-Wesley, 1993 (siehe S. 58, 59).

[Pre13] Preston-Werner, Tom: Semantic Versioning 2.0.0. 2013. url: https://semver.org/spec/v2.0.0.html (besucht am 25. 06. 2024) (siehe S. 90, 91, 218).

[Rae14] Raemaekers, Steven; van Deursen, Arie und Visser, Joost: „Semantic Versioning versus Breaking Changes: A Study of the Maven Repository“. In: *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation (SCAM) (Victoria, BC, Canada, 28.–29. Sep. 2014). Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE), 2014, S. 215–224. doi: 10.1109/SCAM.2014.30 (siehe S. 92).

[Rei11] Reif, Konrad: Bosch Autoelektrik und Autoelektronik. Wiesbaden: Vieweg+Teubner, 2011. doi: 10.1007/978-3-8348-9902-6 (siehe S. 66, 79).

[Rei15] Reif, Konrad, Hrsg.: Gasoline Engine Management. Systems and Components. Wiesbaden: Springer Vieweg, 2015. 354 S. doi: 10.1007/978-3-658-03964-6 (siehe S. 9, 10).

[Rho16] RhodeCode: Version Control Systems Popularity in 2016. 2016. url: https://rhodecode.com/insights/version-control-systems-2016 (besucht am 10. 01. 2021) (siehe S. 89).

[Roc75] Rochkind, Marc J.: „The source code control system“. In: *IEEE Transactions on Software Engineering* SE-1.4 (1975), S. 364–370. doi: 10.1109/TSE.1975.6312866 (siehe S. 89).

[Ruf15] Ruf, Florian: „Auslegung und Topologieoptimierung von spannungsstabilen Energiebordnetzen“. Technische Universität München, 2015. url: https://mediatum.ub.tum.de/doc/1220402/ (besucht am 25. 06. 2024) (siehe S. 11).

[Rüp15] Rüping, Thomas; Fürst, Simon und Bechter, Markus: „Zukunft von Autosar Weiter- und Neuentwicklung“. In: *ATZ elektronik* 10.S7 (2015), S. 24–29. doi: 10.1007/s35658-015-0571-4 (siehe S. 74).

[SAE10] SAE International, Hrsg.: ARP4754A: Guidelines for Development of Civil Aircraft and Systems. Warrendale, PA, USA, 21. Dez. 2010. url: https://www.sae.org/standards/content/arp4754a/ (besucht am 25. 06. 2024) (siehe S. 64, 66, 69).

[SAE16] SAE International, Hrsg.: EIAIS632: Systems Engineering. Warrendale, PA, USA, 16. Juni 2016. URL: https://www.sae.org/standards/content/eiais632/ (besucht am 25. 06. 2024) (siehe S. 77).

[Sax08] Sax, Eric, Hrsg.: Automatisiertes Testen eingebetteter Systeme in der Automobilindustrie. Sax, Eric (Hrsg.) München: Hanser, 2008. 219 S. DOI: 10.3139/9783446419018 (siehe S. 64–66, 71).

[Sax17] Sax, Eric; Reussner, Ralf; Guissouma, Houssemeddine und Klare, Heiko: A survey on the state and future of automotive software release and configuration management. Bd. 2017/11. Karlsruhe Reports in Informatics. KITopen, 2017. DOI: 10.5445/IR/1000075673 (siehe S. 37, 92).

[Sch05a] Schade, Ralf und Varchmin, Jörn-Uwe: „Ansatz für eine Datenbusarchitektur auf Basis von FlexRay“. In: *25 Jahre Elektronik-Systeme im Kraftfahrzeug: Rückblick – Ausblick – Visionen*. Hrsg. von Bäker, Bernard. Bd. 50. Haus der Technik - Fachbuchreihe. expert verlag, 2005, S. 53–67 (siehe S. 1, 4).

[Sch05b] Scholz, Peter: Softwareentwicklung eingebetteter Systeme. Grundlagen, Modellierung, Qualitätssicherung. Xpert.press. Berlin und Heidelberg: Springer, 2005. DOI: 10.1007/3-540-27522-3 (siehe S. 73).

[Sch14] Scheidhammer, Georg und Himmel, Jörg: „Gewichteinsparung im Mehrspannungs-Bordnetz“. In: *ATZ - Automobiltechnische Zeitschrift* 116.11 (2014), S. 58–63. DOI: 10.1007/s35148-014-0523-y (siehe S. 3).

[Sch16a] Schäuffele, Jörg: „E/E Architectural Design and Optimization using PREEvision“. In: *SAE 2016 World Congress and Exhibition* (12. Apr. 2016). SAE Technical Paper Series. SAE International, 2016. DOI: 10.4271/2016-01-0016 (siehe S. 4, 74, 89).

[Sch16b] Schäuffele, Jörg und Zurawka, Thomas: Automotive Software Engineering. Grundlagen, Prozesse, Methoden und Werkzeuge

effizient einsetzen. 6. Auflage. ATZ / MTZ-Fachbuch. Wiesbaden: Springer Vieweg, 2016. 359 S. DOI: 10.1007/978-3-658-11815-0 (siehe S. 66, 79, 88).

[Sch17] SCHULLER, Jürgen; BUHLMANN, Markus; JULING, Alexander; SCHWARZ, Ralf und SCHMID, Stefan: „Electronic chassis platform – highly integrated ECU for chassis control functions". In: *7th International Munich Chassis Symposium 2016*. Hrsg. von PFEFFER, Peter E. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2017, S. 349–365. DOI: 10.1007/978-3-658-14219-3_25 (siehe S. 6).

[Sch20a] SCHAAL, Sebastian: Volkswagen: Verzögern Software-Probleme den ID.3-Start? electrive.net. 2020. URL: https://www.electrive.net/2020/02/25/volkswagen-verzoegern-software-probleme-den-id-3-start/ (besucht am 25. 06. 2024) (siehe S. 79).

[Sch20b] SCHEER, Dietmar; GLODD, Oliver; GÜNTHER, Hartmut; DUHR, Yves und SCHMID, Achim: „STAR3 - Eine neue Generation der E/E-Architektur". In: *Sonderprojekte ATZ/MTZ* 25.S1 (2020), S. 72–79. DOI: 10.1007/s41491-020-0056-5 (siehe S. 27).

[Sch94] SCHUMNY, Harald und OHL, Rainer: Handbuch Digitaler Schnittstellen. Springer eBook Collection Computer Science and Engineering. Wiesbaden: Vieweg+Teubner Verlag, 1994. DOI: 10.1007/978-3-322-84916-8 (siehe S. 78).

[Siv19] SIVAKUMAR, P.; DEVI, M. R.S. und KUMAR, B. Vinoth: „Analysis of Software Reusablity Concepts Used In Automotive Software Development Using Model Based Design and Testing Tools". In: *Alliance International Conference on Artificial Intelligence and Machine Learning (AICAAM)* (26.–27. Apr. 2019). Hrsg. von SELVARANI, Rangasamy; KORAH, Reeba und SUDAROLI, Vijayakumar. 2019, S. 78–89. URL: https://www.alliance.edu.in/aicaam-conference-proceedings/Papers/Analysis-of-Software-Reusablity-Concepts.pdf (besucht am 25. 06. 2024) (siehe S. 6).

[Smi22] Smith, Shawn und Khalid, Mohammed A.S.: „Automated Generation and Integration of AUTOSAR RTE Configurations“. In: *2022 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE). 2022 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE) took place 18-20 September 2022 in Halifax, Nova Scotia, Canada.* 2022 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE) (Halifax, NS, Canada, 18.–20. Sep. 2022). Institute of Electrical and Electronics Engineers. Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE), 2022, S. 141–144. DOI: 10.1109/CCECE49351.2022.9918435 (siehe S. 202).

[Sta21] Staron, Miroslaw, Hrsg.: Automotive Software Architectures. An Introduction. eng. 2nd ed. 2021. Staron, Miroslaw (VerfasserIn). Cham: Springer, 2021. 274 S. DOI: 10.1007/978-3-030-65939-4 (siehe S. 22, 64–66, 73, 79).

[Ste08] Steffen, Rainer; Bogenberger, Richard; Hillebrand, Joachim; Hintermaier, Wolfgang; Winckler, Andreas und Rahmani, Mehrnoush: „Design and realization of an ip-based in-car network architecture“. In: *The First Annual International Symposium on Vehicular Computing Systems.* 2008, S. 1–7 (siehe S. 3, 6).

[Sto10] Stober, Thomas und Hansmann, Uwe: Agile Software Development. Best Practices for Large Software Development Projects. Berlin und Heidelberg: Springer, 2010. 179 S. DOI: 10.1007/978-3-540-70832-2 (siehe S. 192).

[Str12] Streichert, Thilo und Traub, Matthias: Elektrik/Elektronik-Architekturen im Kraftfahrzeug. VDI-Buch. Berlin und Heidelberg: Springer, 2012. DOI: 10.1007/978-3-642-25478-9 (siehe S. 10, 25, 73–75, 77, 79).

[Tan07] Tanenbaum, Andrew S. und van Steen, Maarten: Distributed systems. Principles and paradigms. 2. ed., internat. ed. Upper Saddle River, NJ: Pearson/Prentice Hall Pearson Education Internat, 2007. 686 S. (siehe S. 38).

[Tei17] Teixeira, Jose: „Release Early, Release Often and Release on Time. An Empirical Case Study of Release Management“. In: *Open source systems - towards robust practices. 13th IFIP WG 2.13 International Conference, OSS 2017, Buenos Aires, Argentina, May 22-23, 2017, proceedings*. Hrsg. von Balaguer, Federico; Di Cosmo, Roberto; Garrido, Alejandra; Kon, Fabio; Robles, Gregorio und Zacchiroli, Stefano. Bd. 496. IFIP Advances in Information and Communication Technology 496. Cham: Springer, 2017, S. 167–181. doi: 10.1007/978-3-319-57735-7_16 (siehe S. 92).

[Tha18] Tharma, Rajeeth; Winter, Roland und Eigner, Martin: „An Approach for the Implementation of the Digital Twin in the Automotive Wiring Harndess Field“. In: *Proceedings of the DESIGN 2018 15th International Design Conference*. 15th International Design Conference (21.–24. Mai 2018). Design Conference Proceedings. Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Croatia und The Design Society, Glasgow, UK, 2018, S. 3023–3032. doi: 10.21278/idc.2018.0188 (siehe S. 5).

[Tin94] Tindell, Ken und Burns, Alan: „Guaranteeing message latencies on control area network (CAN)“. In: *Proceedings of the 1st International CAN Conference* (Mainz). CAN in Automation. Erlangen, 1994 (siehe S. 187).

[Tra17] Traub, Matthias; Maier, Alexander und Barbehön, Kai L.: „Future Automotive Architecture and the Impact of IT Trends“. In: *IEEE Software* 34.3 (2017), S. 27–32. doi: 10.1109/MS.2017.69 (siehe S. 6, 217).

[Tra18] Traub, Matthias; Vogel, Hans-Jorg; Sax, Eric; Streichert, Thilo und Harri, Jerome: „Digitalization in automotive and industrial systems“. In: *Proceedings of the 2018 Design, Automation & Test in Europe (DATE). 19-23 March 2018, Dresden, Germany*. Design, Automation & Test in Europe Conference & Exhibition (DATE) (Dresden, 19.–23. März 2018). Piscataway, NJ: Institute

of Electrical and Electronics Engineers (IEEE), 2018, S. 1203–1204. DOI: 10.23919/DATE.2018.8342198 (siehe S. 3).

[Tri15] TRIPPNER, Dietmar; RUDE, Stefan und SCHREIBER, Andreas: „Challenges to Digital Product and Process Development Systems at BMW“. In: *Concurrent Engineering in the 21st Century*. Hrsg. von STJEPANDIĆ, Josip; WOGNUM, Nel und VERHAGEN, Wim J.C. Cham: Springer, 2015, S. 555–569. DOI: 10.1007/978-3-319-13776-6_19 (siehe S. 68).

[Ung09] UNGERMANN, Jochen; HAMMER, Bernhard und SIEGWART, Roland: „Entwicklung eines Planungsstandards für die Gesamtfahrzeugerprobung unter Einbeziehung von Erprobungs-und Gewährleistungsdaten“. In: *14. VDI-Fachtagung Erprobung und Simulation in der Fahrzeugentwicklung* (Würzburg, 24.–25. Juni 2009). Verein Deutscher Ingenieure. Düsseldorf: VDI-Wissensforum GmbH, 2009. DOI: 10.3929/ethz-a-010118601 (siehe S. 72).

[Vai17] VAIBHAV: Electronic Control Unit is at the Core of All Automotive Innovations: Know How the Story Unfolded. Embitel. 2017. URL: https://www.embitel.com/blog/embedded-blog/automotive-control-units-development-innovations-mechanical-to-electronics (besucht am 25. 06. 2024) (siehe S. 5).

[Ver21] VEREIN DEUTSCHER INGENIEURE, Hrsg.: VDI/VDE 2206:2021-11: Entwicklung mechatronischer und cyber-physischer Systeme. Berlin: Beuth Verlag GmbH, 1. Nov. 2021. URL: https://www.dinmedia.de/de/technische-regel/vdi-vde-2206/342674320 (besucht am 25. 06. 2024) (siehe S. 68, 69).

[Ver24] VEREIN ZUR WEITERENTWICKLUNG DES V-MODELL XT E.V., Hrsg.: V-Modell XT. Das deutsche Referenzmodell für Systementwicklungsprojekte. Version: 2.4. Unter Mitarb. von DANIEL ANGERMEIER, CHRISTIAN BARTELT, OTTO BAUER, GERD BENEKEN, KLAUS BERGNER, ULRICH BIROWICZ, THOMAS BLIß, CHRISTIAN BREITENSTROM, NILS CORDES, DAVID CRUZ, PATRICK DOHRMANN, JAN FRIEDRICH, MICHAEL GNATZ, ULRIKE

Hammerschall, Istvan Hidvegi-Barstorfer, Helmut Hummel, Dirk Israel, Thomas Klingenberg, Klaus Klugseder, Inga Küffer, Marco Kuhrmann, Michael Kranz, Wolfgang Kranz, Hans-Jürgen Meinhardt, Michael Meisinger, Sabine Mittrach, Hans-Joachim Neußer, Dirk Niebuhr, Klaus Plögert, Doris Rauh, Andreas Rausch, Thomas Rittel, Winfried Rösch, Erik Saas, Joachim Schramm, Marc Sihling, Thomas Ternité, Sascha Vogel, Bernd Weber, Marion Wittmann. München, 2024. URL: https://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/2.4/Dokumentation/V-Modell-XT-Gesamt.pdf (besucht am 25. 06. 2024) (siehe S. 69).

[Vet21a] Vetter, Andreas; Obergfell, Philipp; Guissouma, Houssem; Grimm, Daniel; Rumez, Marcel und Sax, Eric: „Development Processes in Automotive Service-oriented Architectures“. In: *9th Mediterranean Conference on Embedded Computing (MECO)* (Budva, Montenegro, 8.–11. Juni 2020). Institute of Electrical and Electronics Engineers (IEEE), 2021. DOI: 10.1109/MECO49872.2020.9134175 (siehe S. 27, 32).

[Vet21b] Vetter, Andreas und Sax, Eric: „Hierarchical Versioning to Increase Compatibility in Signal-Oriented Vehicle Networks“. In: *Proceedings of the 27th International Conference on Systems Engineering*. 2020-12-14/2020-12-16 (Las Vegas, USA). Hrsg. von Selvaraj, Henry; Chmaj, Grzegorz und Zydek, Dawid. Bd. 182. Lecture Notes in Networks and Systems. Cham: Springer, 2021. DOI: 10.1007/978-3-030-65796-3_42 (siehe S. 107).

[Vet23a] Vetter, Andreas und Schumacher, Philipp: „Custom Toolchain as Bridge between OEM Specific Needs and Standard Software“. In: *AUTOSAR 20th Anniversary 2003-2023*. Hrsg. von Rüping, Thomas und Redlich, Peter. Hörgertshausen, 2023, S. 168–169. URL: https://www.autosar.org/fileadmin/user_upload/AUTOSAR_20th-Book_FINAL_WEB_06-OCT-2023.pdf (besucht am 25. 06. 2024) (siehe S. 4, 89, 213, 273).

[Vet23b] Vetter, Andreas; Schumacher, Philipp; Hedenetz, Bernd und Sax, Eric: „The Impact of Different Compatibility Schemes on the Static Bus Load in Vehicle Networks“. In: *Proceedings of Eighth International Congress on Information and Communication Technology* (London, 20.–23. Feb. 2023). Hrsg. von Yang, Xin-She; Sherratt, R. Simon; Dey, Nilanjan und Joshi, Amit. Lecture Notes in Networks and Systems. Singapore: Springer, 2023. doi: 10.1007/978-981-99-3236-8_44 (siehe S. 149).

[Vol21] Volkswagen AG: Modelle und Konfigurator. 2021. url: https://www.volkswagen.de/de/modelle-und-konfigurator.html (besucht am 22. 01. 2021) (siehe S. 94).

[Vol23] Volkswagen AG, Hrsg.: Geschäftsbericht 2022. Wolfsburg, 2023 (siehe S. 11, 12).

[Web19] Weber, Heiko; Krings, Jörg; Seyfferth, Jonas; Güthner, Hartmut und Neuhausen, Jörn: The 2019 Strategy& Digital Auto Report. Time to get real: opportunities in a transforming market. strategy&, Teil von PricewaterhouseCoopers GmbH, 2019. url: https://web.archive.org/web/20220709112242/https://www.strategyand.pwc.com/de/en/industries/automotive/digital-auto-report-2019/digital-auto-report-2019.pdf (besucht am 25. 06. 0204) (siehe S. 1).

[Wec14] Weckemann, Kay: „Domänenübergreifende Anwendungskommunikation im IP-basierten Fahrzeugbordnetz“. Fakultät für Mathematik, Informatik und Statistik. München: Ludwig-Maximilians-Universität, 2014. 156 S. url: https://edoc.ub.uni-muenchen.de/17275/1/Weckemann_Kay.pdf (besucht am 25. 06. 2024) (siehe S. 6).

[Wed94] Wedekind, Hartmut: „Are the terms “version” and “variant” orthogonal to one another?“ In: *ACM SIGMOD Record* 23.4 (1994), S. 3–7. doi: 10.1145/190627.190628 (siehe S. 86).

[Wei13] Weissinger, Jürgen und Scheer, Michael: „Abstimmung und Erprobung des Gesamtfahrzeugs“. In: *Mercedes-Benz SL. Entwicklung und Technik*. Hrsg. von Ernstberger, Uwe; Weissinger, Jürgen und Frank, Jürgen. ATZ / MTZ-Typenbuch. Wiesbaden: Springer Vieweg, 2013, S. 266–271. doi: 10.1007/978-3-658-00800-0_30 (siehe S. 72).

[Wes01] Westfechtel, B.; Munch, B. P. und Conradi, R.: „A layered architecture for uniform version management“. In: *IEEE Transactions on Software Engineering* 27.12 (2001), S. 1111–1133. doi: 10.1109/32.988710 (siehe S. 86).

[Win22] Windpassinger, Hans: „On the Way to a Software-defined Vehicle“. In: *ATZ electronics worldwide* 17.7-8 (2022), S. 48–51. doi: 10.1007/s38314-022-0779-z (siehe S. 197).

[Wit22] Wittler, Jan Willem; Kühn, Thomas und Reussner, Ralf: „Towards an integrated approach for managing the variability and evolution of both software and hardware components“. In: *Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume B* (Graz, Österreich, 12.–16. Sep. 2022). Hrsg. von Felfernig, Alexander. New York, USA: Association for Computing Machinery, 2022, S. 94–98. doi: 10.1145/3503229.3547059 (siehe S. 86).

[Zim14] Zimmermann, Werner und Schmidgall, Ralf: Bussysteme in der Fahrzeugtechnik. Protokolle, Standards und Softwarearchitektur. 5., aktual. und erw. Aufl. ATZ / MTZ-Fachbuch. Wiesbaden: Springer Vieweg, 2014. doi: 10.1007/978-3-658-02419-2 (siehe S. 20, 24, 29, 66, 70, 73, 74, 79, 96, 195).

Eigene Publikationen

[1] Vetter, Andreas; Obergfell, Philipp; Guissouma, Houssem; Grimm, Daniel; Rumez, Marcel und Sax, Eric: „Development Processes in Automotive Service-oriented Architectures“. In: *9th Mediterranean Conference on Embedded Computing (MECO)* (Budva, Montenegro, 8.–11. Juni 2020). Institute of Electrical and Electronics Engineers (IEEE), 2021. doi: 10.1109/MECO49872.2020.9134175.

[2] Vetter, Andreas und Sax, Eric: „Hierarchical Versioning to Increase Compatibility in Signal-Oriented Vehicle Networks“. In: *Proceedings of the 27th International Conference on Systems Engineering*. 2020-12-14/2020-12-16 (Las Vegas, USA). Hrsg. von Selvaraj, Henry; Chmaj, Grzegorz und Zydek, Dawid. Bd. 182. Lecture Notes in Networks and Systems. Cham: Springer, 2021. doi: 10.1007/978-3-030-65796-3_42.

[3] Puder, Andreas; Vetter, Andreas; Rumez, Marcel; Henle, Jacqueline und Sax, Eric: „A Mixed E/E-Architecture for Interconnected Operating Tables Inspired by the Automotive Industry“. In: *Journal of Medical Robotics Research (JMRR)* (2022). doi: 10.1142/S2424905X22410082.

[4] Puder, Andreas; Vetter, Andreas; Rumez, Marcel; Henle, Jacqueline und Sax, Eric: „A Mixed E/E-Architecture for Interconnected Operating Tables Inspired by the Automotive Industry“. In: *International Symposium on Medical Robotics (ISMR)* (Atlanta, GA, USA, 13.–15. Apr. 2022). Institute of Electrical and Electronics Engineers (IEEE), 2022. doi: 10.1109/ISMR48347.2022.9807578.

[5] Schindewolf, Marc; Stoll, Hannes; Guissouma, Houssem; Puder, Andreas; Sax, Eric; Vetter, Andreas; Rumez, Marcel und Henle, Jacqueline: „A Comparison of Architecture Paradigms for Dynamic Reconfigurable Automotive Networks“. In: *2022 International Conference on Connected Vehicle and Expo (ICCVE)* (Lakeland, FL, USA, 7.–9. März 2022). Institute of Electrical and Electronics Engineers (IEEE), 2022. doi: 10.1109/ICCVE52871.2022.9742775.

[6] Vetter, Andreas und Schumacher, Philipp: „Custom Toolchain as Bridge between OEM Specific Needs and Standard Software“. In: *AUTOSAR 20th Anniversary 2003-2023*. Hrsg. von Rüping, Thomas und Redlich, Peter. Hörgertshausen, 2023, S. 168–169. url: https://www.autosar.org/fileadmin/user_upload/AUTOSAR_20th-Book_FINAL_WEB_06-OCT-2023.pdf (besucht am 25. 06. 2024).

[7] Vetter, Andreas; Schumacher, Philipp; Hedenetz, Bernd und Sax, Eric: „The Impact of Different Compatibility Schemes on the Static Bus Load in Vehicle Networks“. In: *Proceedings of Eighth International Congress on Information and Communication Technology* (London, 20.–23. Feb. 2023). Hrsg. von Yang, Xin-She; Sherratt, R. Simon; Dey, Nilanjan und Joshi, Amit. Lecture Notes in Networks and Systems. Singapore: Springer, 2023. doi: 10.1007/978-981-99-3236-8_44.

[8] Zhai, Yi; Vetter, Andreas und Sax, Eric: „Analysis of Current Challenges of Automotive Software in the View of Manufacturing“. In: *23rd Stuttgart International Symposium* (Stuttgart, 4.–5. Juli 2023). SAE International, 2023. doi: 10.4271/2023-01-1221.

Abbildungsverzeichnis

Tabellenverzeichnis

Notation

Dieses Kapitel gibt eine Übersicht, über die verwendete Notation und Formelzeichen. Bei Elementen mit mehreren Indizes benennt der erste Index das K-Matrix-Release, der zweite identifiziert das jeweilige Kommunikationsobjekt, und der dritte Index benennt ggf. den Bus.

Symbol	Bedeutung	Einführung
$a_{i,j,k}$	Anzahl der Adressbits der PDU Nr. j in Release Nr. i auf Bus Nr. k	Definition 3.55
$\mathbb{B}$	Menge der Busse	Definition 3.41
B_i	Einzelner Bus Nr. i	Definition 3.41
$B_{i,j}$	Release Nr. i von Bus Nr. j	Definition 3.42
$B^{W(i,j)}$	Menge der Busse die die PDU $W_{i,j}$ enthalten	Gleichung (3.1)
$\mathcal{B}$	Bitfunktion	Definition 3.48
b	Stuffingfaktor	Definition 3.56
$\mathbb{C}$	Menge aller Service- bzw. Software-Compositions	Definition 3.19
C_i	Einzelne Service- bzw. Software-Composition Nr. i	Definition 3.19
D_i	Datentypdefinition von τ_i	Definition 3.15
d	Anzahl Datenbits	Definition 3.48
$d_{i,j}$	Anzahl der Datenbits für PDU Nr. j in Release Nr. i	Definition 3.52
E_i	Kanten (engl.: „Edges“) von C_i	Definition 3.19
G_i	Menge der Garantien von S_i	Definition 3.14
g_i^j	Einzelne Garantie Nr. j von S_i	Definition 3.14

I_i	Bezeichner (engl.: „Identifier“) von τ_i	Definition 3.15
$\mathbb{K}$	Menge der K-Matrix-Releases	Definition 3.40
K^H	Menge der Haupt-Releases	Definition 4.2
K^A, K^B	Zwischen-Release-Mengen A und B	Definition 4.3
K_i	Einzelnes K-Matrix-Release Nr. i	Definition 3.40
$\mathcal{K}$	Kompatibilitätsfunktion	Definition 3.23
$\mathcal{K}_b$	Bedingte Kompatibilitätsfunktion	Definition 3.28
$L_{i,j}$	Auslastung des Bus Nr. j in Release Nr. i	Definition 3.62
$L_{i,j,k}$	Der absolute Anteil der PDU Nr. j an der Gesamtauslastung des Bus Nr. k in Release Nr. i	Definition 3.61
$\Delta L_{i,j}$	Änderung der Last des Bus Nr. j zwischen Release Nr. i-1 und i	Definition 5.1
$\Delta L_{i,j,k}$	Änderung des absoluten Lastanteils der PDU Nr. j auf Bus Nr. k zwischen Release Nr. i-1 und i	Definition 5.2
$\Delta L^+_{i,j}$	Zusätzliche Last auf Bus Nr. j im K-Matrix-Release Nr. i	Definition 5.4
$\Delta L^+_{i,j,k}$	Zusätzlicher Lastanteil der PDU Nr. j auf Bus Nr. k im K-Matrix-Release Nr. i	Definition 5.7
$\mathcal{L}$	Lebensdauerfunktion	Definition 5.6
$\mathcal{L}_W$	Lebensdauerfunktion für PDUs	Definition 5.6
$\mathcal{L}_Y$	Lebensdauerfunktion für Signale	Definition 5.6
$n_{i,j,k}$	Anzahl der für PDU Nr. j auf Bus Nr. k in Release Nr. i zu übertragenden Bits	Definition 3.57
$n'_{i,j,k}$	Anzahl der für PDU Nr. j auf Bus Nr. k in Release Nr. i in erhöhter Geschwindigkeit zu übertragenden Bits	Definition 3.58

P_i	Parameter von τ_i	Definition 3.15
$p_{i,j,k}$	Anzahl der Paddingbits für PDU Nr. j in Release Nr. i auf Bus Nr. k	Definition 3.54
Q_i	Qualität von τ_i	Definition 3.15
R_i	Menge der Anforderungen (engl.: „Requirements“) von S_i	Definition 3.14
$R_{i,j}$	Baudrate des Bus Nr. j in Release Nr. i	Definition 3.59
$R'_{i,j}$	Erhöhte CAN-FD Baudrate des Bus Nr. j in Release Nr. i	Definition 3.59
r_i^j	Einzelne Anforderung Nr. j von S_i	Definition 3.14
$\mathbb{S}$	Menge der Services bzw. Softwarekomponenten	Definition 3.14
S_i	Einzelner Service bzw. einzelne Softwarekomponente Nr. i	Definition 3.14
$s_{i,j}$	Signalgröße in Bits für Signal Nr. j in Release Nr. i	Definition 3.49
$\mathbb{T}$	Menge der Informationstypen	Definition 3.15
$\mathcal{T}_G$	Typenfunktion für Abbildung auf Garantien	Definition 3.16
$\mathcal{T}_R$	Typenfunktion für Abbildung auf Anforderungen	Definition 3.16
$\mathcal{T}_Y$	Signaltypfunktion	Definition 3.47
$t_{i,j,k}$	Übertragungsdauer der PDU Nr. j in Release Nr. i auf Bus Nr. k	Definition 3.60
$\mathcal{V}_{\tau_i}^D$	Menge der möglichen Werte für D_i	Definition 3.15
$\mathcal{V}_{\tau_i}^Q$	Menge der möglichen Werte für Q_i	Definition 3.15
$\mathcal{V}_{\tau_i}^P$	Menge der möglichen Werte für P_i	Definition 3.15
V_i	Knoten (engl.: „Vertices“) von C_i	Definition 3.19

$\mathbb{W}$	Menge der PDUs	Definition 3.43
W_i	Einzelne PDU Nr. i	Definition 3.43
$W_{i,j}$	Release Nr. i von PDU Nr. j	Definition 3.44
$W^{Y(i,j)}$	Menge der PDUs die das Signal $Y_{i,j}$ enthalten	Gleichung (3.2)
W_i^+	Menge der PDUs, die im Release Nr. i durch Kompatibilitätsmechanismen entstehen	Definition 5.9
$\mathbb{Y}$	Menge der Signale	Definition 3.45
Y_i	Einzelnes Signal Nr. i	Definition 3.45
$Y_{i,j}$	Release Nr. i von Signal Nr. j	Definition 3.46
Y_i^+	Menge der Signale, die im Release Nr. i durch Änderungskopien entstehen	Definition 5.8
$\mathcal{Z}$	Zykluszeitfunktion für Informationstypen	Definition 3.50
$z_{i,j}$	Zykluszeit für PDU Nr. j in Release Nr. i	Definition 3.51
$\alpha_{i,j}$	Signaländerungsrate auf dem Bus Nr. j im Release Nr. i	Definition 5.11
$\beta_{i,j}$	Verhältnis der veränderten PDUs zur Gesamtanzahl der PDUs auf einem Bus Nr. j eines K-Matrix-Releases Nr. i	Definition 5.12
$\gamma_{i,j,k}$	Füllfaktor einer PDU Nr. k auf Bus Nr. j in Release Nr. i	Definition 5.13
δ	Lebensdauer eines Kommunikationsobjekts	Definition 5.5
δ_j	Mittlere Lebensdauer der PDUs auf dem Bus Nr. j	Definition 5.14
ϵ	Offset als Anzahl Releases, ab dem ersten Release, ab dem Kompatibilitätsmechanismen aktiviert werden	Definition 5.15

ζ	Hierarchietiefe, d. h. die Anzahl der für Kompatibilitätsmechanismen verwendeten Hierarchieebenen	Definition 5.16
η	Anzahl der PDUs	Definition 3.43
θ	Anzahl der Signale	Definition 3.45
$\kappa(i)$	Anzahl der Garantien von S_i	Definition 3.14
λ	Anzahl der Service- bzw. Software-Compositions	Definition 3.19
$\mu(i)$	Anzahl der Anforderungen von S_i	Definition 3.14
π	Anzahl der Informationstypen	Definition 3.15
σ	Anzahl der Services bzw. Softwarekomponenten	Definition 3.14
τ_i	Einzelner Informationstyp Nr. i	Definition 3.15
φ	Anzahl der K-Matrix-Releases	Definition 3.40
ψ	Anzahl der Busse	Definition 3.41

Abkürzungsverzeichnis

ABS	Antiblockierystem
ADT	Application Data Type
ARXML	AUTOSAR XML
AUTOSAR	AUTomotive Open System ARchitecture
BTV	Bauteilverantwortliche
CAN	Controller Area Network
CAN FD	Controller Area Network with Flexible Datarate
COM	AUTOSAR Communication Module
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
DCU	Domain Control Unit
ECU	Electronic Control Unit
ESP	Elektronisches Stabilitätsprogramm
FF	Forschungsfrage
FIBEX	Field Bus EXchange format
FMK	Fertigungs- und Materialkosten
HIL	Hardware in the Loop

ID	Identifier
INCOSE	International Council on Systems Engineering
IP	Internet Protocol
IPDU	Interaction Layer Protocol Data Unit
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
K-Line	Keyword-Line
LIN	Local Interconnect Network
MOF	Meta Object Facility
MOST	Media Oriented Systems Transport
NPDU	Network Layer Protocol Data Unit
NRZ	Non-Return-to-Zero
NVRAM	Nonvolatile Random Access Memory
OBD	On Board Diagnose
OEM	Original Equipment Manufacturer
OMG	Object Management Group
OSI	Open Systems Interconnection
OTA	Over The Air
PCI	Protocol Control Information
PDU	Protocol Data Unit. Siehe Definition 3.3.

QOS	Quality Of Service
RTE	Runtime Environment
SAP	Service Access Point
SCCS	Source Code Control System
SDG	Special Data Group
SDU	Service Data Unit
SIL	Software in the Loop
SOA	Service-orientierte Architektur
SOME/IP	Scalable service-Oriented MiddlewarE over IP
SOTA	Software Over The Air
STAR	(Mercedes) Standard (E/E-)Architektur
SVN	Subversion
SWC	Software Component
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
UDP	User Datagram Protocol
UML	Unified Modeling Language
UUID	Universally Unique Identifier
VCS	Version Control System
VFB	Virtual Functional Bus
VLAN	Virtual Local Area Network

WLAN Wireless Local Area Network

XML Extensible Markup Language

Glossar

150%-K-Matrix K-Matrix, die alle Kommunikationsvarianten beschreibt. Reale Fahrzeuge bestehen jeweils aus einer Teilmenge der 150%-K-Matrix. Siehe Abschnitt 3.6.

Abstandstempomat Ein Fahrerassistenzsystem, bei dem automatisch eine festgelegte Geschwindigkeit oder ein festgelegter Abstand zum vorausfahrenden Fahrzeug gehalten wird.

Arbitrierung Ein Verfahren, um festzulegen wann welches Steuergerät senden darf.

Backbone Engl. für "'Rückgrat"', zentraler Bus eines Fahrzeugs, der unterschiedliche Domänen miteinander verbindet.

Brake-by-Wire Deutsch: Bremsen über Kabel. Anstatt der etablierten, hydraulischen Kraftübertragung vom Bremspedal zum Bremszylinder, wird der Bremszylinder elektrisch angesteuert. Die Information, das gebremst werden soll, wird digital über das Fahrzeugnetzwerk übertragen.

Bug Fix Engl. für "'Fehlerkorrektur"'. Eine Änderung, die ein falsches Verhalten verbessert. Siehe Definition 3.36.

Bus (Daten-) Unter mehreren Kommunikationsteilnehmern geteiltes Kommunikationsmedium,

	über welches die Teilnehmer Daten austauschen können. Siehe Abschnitt 2.3.2.
Domäne	Fahrzeugfunktionen werden in größere Gruppen (z. B Antrieb, Fahrwerk, Komfort, Fahrerassistenz, etc.), die sogenannten Domänen, einsortiert.
Domänencontroller	Auch: Domänengateway oder DCU. Zentrales Steuergerät bzw. Gateway einer Domäne.
E/E-Architektur	Elektrik/Elektronik-Architektur. Die Gesamtheit aus der physischen Netzwerktopologie und den verwendeten Kommunikationsparadigmen. Siehe Abschnitt 2.4.
E/E-Architetkur Plattform	Eine Plattform oder ein "'Baukasten"' aus dem die E/E-Architekturen für mehrere Baureihen abgeleitet werden.
ECU-Extract	Eine ARXML-Datei, welches den Ausschnitt ("'Extract"') aus dem Gesamtsyststem enthält, der eine einzelne ECU modelliert.
Erprobungsträger	Prototypenfahrzeuge, die sämtliche Steuergeräte beinhalten und zur Valdierung des Gesamtsystems dienen.
Ethernet	Kabelgebundene Datenübertragung nach einem Standard der IEEE802.3 Serie.
Flashen	Das Beschreiben des nicht flüchtigen Flash-Speichers einer ECU mit Software oder Daten.

Gateway Ein Steuergerät, welches an mehrere Busse angeschlossen ist und über die Möglichkeit verfügt Botschaften von einem Bus an den anderen weiterzuleiten. Siehe Abschnitt 2.3.3.

Headunit Das zentrale Steuergerät des Infotainmentsystems.

Infotainmentsystem "'Infotainment"' ist ein Kofferwort aus Kofferwort aus "'Information"' (z. B. Navigation u. Verkehrsmeldungen) und "'Entertainment"' (engl. f. "'Unterhaltung"', z. B. Radio u. Musikstreamingdienste).

Konnektivität Der Megatrend "'Konnektivität"' beinhaltet die Vernetzung von Fahrzeugen untereinander, mit Infrastruktur, mit einem Backend des Fahrzeugherstellers und anderen Internetdiensten sowie den Smartphones der Insassen.

Kundenerlebbare Funktionen Anwendungen, die ein Benutzer eines netzwerkbasierten Systems zu Gesicht bekommt. (Im Gegensatz zu zugrunde liegenden Basisfunktionen, die für die Nutzer unsichtbar sind.)

Lead-Baureihe Die erste, auf einer neuen E/E-Architektur Plattform aufgebaute Fahrzeugbaureihe.

Major-Version Hauptversion. Wird nur bei inkompatiblen Änderungen erhöht. Siehe Abschnitt 3.5.3.

Master Netzwerkknoten (Steuergerät), der von sich aus Kommunikation initiieren darf. Siehe Abschnitt 2.3.2.

Middleware Zwischenschicht, um zwischen den heterogenen Geräten eine einheitliche Kommunikation zu ermöglichen. Siehe Abschnitt 2.7.1.

Minor-Version Unterversion. Wird bei kompatiblen Erweiterungen erhöht. Siehe Abschnitt 3.5.3.

Patch-Version Korrekturversion. Wird nach Fehlerkorrekturen (Bug Fixes) erhöht. Siehe Abschnitt 3.5.3.

Release Veröffentlichung, Freigabe, Abgabe. Die Veröffentlichung (oder Weitergabe an ein anderes Glied in der Lieferkette) zu entwickelnder Elemente in einer bestimmten Version. Siehe Abschnitt 3.5.4.

Rohbauplanung Die Phase in der Fahrzeugentwicklung, in der die Geometrie des Fahrzeugs entwickelt wird. Siehe Abschnitt 2.1.1.

Routing Die Entscheidung, welche Pakete von welchen Eingangsbussen zu welchen Ausgangsbussen weitergeleitet werden. Siehe Abschnitt 2.3.3.

Semantische Versionierung Engl.: "'Semantic Versioning"'. Versionierungsschema, in dem die Kompatibilität von Änderungen anhand der Versionsnummern ablesbar ist. Siehe Abschnitt 3.5.3.

Sharing Mobilitätskonzepte die auf der (kurzfristigen) Vermietung von Fahrzeugen oder dem Anbieten von Fahrdienstleistungen basieren.

Signal Über das Fahrzeugnetzwerk übertragene Einzelinformation.

Slave Netzwerkknoten (Steuergerät), die nur senden dürfen, wenn sie durch den Master dazu aufgefordert wurden. Siehe Abschnitt 2.3.2.

Spurhalteassistent Ein Fahrerassistenzsystem, das das Verlassen der Fahrspur durch Warnungen an den Fahrer (passiver Spurhalteassistent) oder Lenkeingriffe (aktiver Spurhalteassistent) verhindern soll.

Steer-by-Wire Deutsch: Lenken über Kabel. Anstatt der etablierten, Kraftübertragung vom Lenkrad über die Lenksäule und das Lenkgetriebe, wird die Lenkung elektrisch angesteuert. Der gewünschte Lenkwinkel, wird digital über das Fahrzeugnetzwerk übertragen.

Tier 1, Tier 2, ... Rang in der Lieferkette. Tier 1: direkter Zulieferer; Tier 2: Zulieferer des Zulieferers; usw. Siehe Abschnitt 2.1.3.

Topologie Die Anordnung und Struktur der Verbindungen der Steuergeräte. Siehe Abschnitt 2.4.

Transceiver Kofferwort aus Transmitter, d. h. Sender und Receiver, d. h. Empfänger. Spezielle, integrierte Schaltkreise, welche die Spannungspegel eines Mikrocontrollers an die

	jeweilige physische Übertragungsschicht anpassen.
Verbauraum	Der Raum innerhalb eines Fahrzeugs, der für den Einbau eines Steuergerätes vorgesehen ist. Siehe Abschnitt 2.1.1.
Verblockung	Ein Steuergerät aus einer Baureihe wird zur Verwendung in einer weiteren Baureihe eingeplant. Siehe Abschnitt 2.1.2.
V-Modell	Vorgehensmodell zur Entwicklung komplexer Systeme. Siehe Abschnitt 3.2.4.
Zugriffsverfahren	Siehe Arbitrierung.

A Kompatible und inkompatible Änderungen an SOME/IP basierter Kommunikation

Die Tabelle in Abbildung A.1 enthält in ihren Spalten verschiedene optionale Funktionen für SOME/P basierten Datenaustausch. In den Zeilen sind mögliche Änderungen an der übertragenen Datenstruktur. Ein Kreuz gibt an, ob die jeweilige Änderung kompatibel durchführbar ist.

Besonders hervorzuheben ist, dass bei vielen Änderungen senderseitig („provide“) das Wegfallen von Elementen kompatibel ist, während empfängerseitig („require“) das Hinzufügen kompatibel ist. Die in Abschnitt 3.4.2 definierte Kompatibilitätsfunktion bildet diese Tatsache korrekt ab, da $K(C_x, C_y) \neq K(C_y, C_x)$ für $x \neq y$.

	Classes of Protocol /Serialization Capabilities							
	Serialization without length fields		Serialization with length fields		Serialization with TLV		Serialization with TLV and optional members	
Change of Interface	Provide	Require	Provide	Require	Provide	Require	Provide	Require
Add a struct member **not** to the end of the struct					X		X	X
Add a struct member to the end of the toplevel struct	X		X		X		X	X
Add a struct member to the end of a sub-struct			X		X		X	X
Remove struct member **not** from the end of the struct						X	X	X
Remove struct member from the end of the toplevel struct		X		X		X	X	X
Remove struct member from the end of a sub-struct				X		X	X	X
Reorder struct members					X	X	X	X
Change the non-highest union member (redefine or remove)								
Add a new union member with previously unused type selector		X		X		X		X
Remove union member with highest type selector	X		X		X		X	
Change of data type: • to a larger one (e.g. uint8 to uint16) • to a smaller one (e.g. uint16 to uint8 • to a semantically different one (e.g. integer to struct, integer to float, string to string with different character size) • byte order • number of dimensions of arrays • size of length field of array, struct or union type selector								
Add new enumeration values		X		X		X		X
Change existing enumeration values								
Remove enumeration values	X		X		X		X	
Change length of fixed size string or array			N/A	N/A	N/A	N/A	N/A	N/A
Decrease maximum length of variable size string	N/A	N/A	X		X		X	
Increase maximum length of variable size string	N/A	N/A		X		X		X
Change maximum length of variable size array	N/A	N/A	X	X	X	X	X	X
Add argument **not** to the end of the argument list of a method request Remove argument **not** from the end of the argument list of a method reponse						X	X	X
Add argument to the end of the argument list of a method request Remove argument from the end of the argument list of a method reponse		X		X		X	X	X
Remove argument **not** from the end of the argument list of a method request Add argument **not** from the end of the argument list of a method response					X		X	X
Remove argument from the end of the argument list of a method request Add argument from the end of the argument list of a method response	X		X		X		X	X
Reorder arguments of methods					X	X	X	X
Change optionality of argument	N/A	N/A	N/A	N/A	N/A	N/A		
Change the return type of a method (e.g void to uint8)								
Add return codes of a method		X		X		X		X
Remove return codes of a method	X		X		X		X	
Change of the name of a service interface, method or event	X	X	X	X	X	X	X	X
Add event ot eventgroup	X		X		X		X	
Remove event from eventgroup		X		X		X		X
Add setter or getter to a field Remove notifier from a field	X		X		X		X	
Remove setter or getter from a field Add notifier to a field		X		X		X		X
Extend service interface by new method, event or field	X		X		X		X	
Remove method, event or field from a service interface		X		X		X		X
Change Method ID or Event ID								
Change data ID of argument Reuse data ID of repviously removed argument	N/A	N/A	N/A	N/A				

Abbildung A.1: Kompatible Änderungen bei SOME/IP (Tabelle: [AUT23g, S. 62ff])

B Umsetzung der HV-Simulation

B.1 Eingangsdaten

Ausgangspunkt für die durchgeführten Simulationen ist die Datenbank des K-Matrix-Entwicklungswerkzeuges „XDIS“ [Vet23a]. Aufgrund der historischen Entwicklung[1] entspricht das interne Datenmodell nicht dem AUTOSAR-Metamodell (vgl. Abschnitt 2.7.5). Aus diesem internen Datenmodell erzeugt „XDIS“ unter Anwendung definierter Transformationsregeln standardkonforme ARXML-Dateien.

Die Daten sind in Form von Objekten strukturiert. Zu jeder Klasse von Objekten existiert eine Tabelle in einer relationalen Datenbank, mit einer Zeile je Objekt. Jedes Objekt weist auf:

- Objekt-ID (identisch über alle Versionen des Objekts)
- UUID (eindeutig einer einzelnen Version eines einzelnen Objektes zuordenbar, vgl. [Int05b])
- XDIS-Versionsnummer (nicht semantisch, vgl. Abschnitt 3.5.3, wird bei jeder Änderung inkrementiert)
- Ein Releasedatum
- Objektspezifische weitere Attribute

Teilweise gibt es Referenzen (in Form der UUID des referenzierten Objektes) auf andere Objekte und Mappingtabellen, mittels derer n:n-Beziehungen (z. B. Signale, die Teil mehrerer PDUs sind, welche wiederum jeweils mehrere Signale enthalten) gespeichert werden. Wichtig ist die Unterscheidung zwischen Objekt-ID (steht für alle Versionen eines Objekts) und UUID (steht für eine Version eines Objekts).

[1] Das von Mercedes-Benz verwendete, Datenbank-basierte K-Matrix-Entwicklungswerkzeug wird seit 2003 (d. h. vor der Gründung des AUTOSAR-Konsortiums und der Veröffentlichung des ersten AUTOSAR-Standards) betrieben und kontinuierlich weiterentwickelt.

Eine Klasse „Vernetzungsplattform“ (vgl. Abschnitt 5.1.1) weist ein Objekt je K-Matrix-Release K_i auf, welches jeweils mit den Releases der Busse B_i verknüpft ist. Die Busse sind mit den Interfaces, die Interfaces mit den PDUs und die PDUs mit den Signalen verknüpft.

Der Zugriff auf die Datenbank ist nur innerhalb des Firmennetzwerks und nach einer Authentifizierung möglich. Im regulären Entwicklungsbetrieb erfolgt dies ausschließlich über das Modellierungswerkzeug „XDIS“, welches über diverse Eingabemasken eine konsistente (vgl. Definition 2.1) Modellierung sicherstellt. Im Rahmen der Simulation greift die eigens geschriebene Software direkt auf die Datenbank zu.

B.2 Implementierung

B.2.1 Get-Methoden

Zunächst wurden einige Methoden definiert, um die Datenbankobjekte der AUTOSAR-Struktur entsprechend (vgl. Abb. 4.2) abrufen zu können. Neben der Authentifizierung und dem Zugriff auf die Datenbank abstrahieren diese Methoden auch das abweichende, XDIS-interne Datenmodell. Der Datenbankzugriff erfolgt mit einem Benutzeraccount, welcher ausschließlich über Leserechte verfügt, um unbeabsichtigte Beschädigungen an der Produktivdatenbank während der Entwicklung auszuschließen.

B.2.2 Versionsnummern und Release-Zeitpunkte

Die XDIS-Versionsnummer eines Objektes wird bei jeder Änderung um eins erhöht, aber nicht jedes Objekt wird in jedem Release verändert. Dadurch wird eine Unterscheidung zwischen den XDIS-Versionsnummern einzelner Objekte, und der XDIS-Versionsnummer der K-Matrix (genauer: XDIS-Versionsnummer des Plattformobjektes, im Folgenden: „Release-Nummer“) notwendig.

Zusätzlich zu den vierteljährlichen a-Releases gibt es weitere K-Matrix-Releases, die b- und c-Releases (vgl. Abschnitte 5.1.2 und 5.1.3). Da die Simulation aber a-Releases als Grundlage für Offset ϵ und Mindestlebensdauer δ verwenden muss (vgl. Abschnitt 5.4), ist eine Ermittlung der a-Releases erforderlich. Deren Release-Nummern werden in einer (chronologisch) geordneten Liste abgelegt.

Dazu wird jedes Plattformobjekt, in Abhängigkeit dessen Veröffentlichungsdatums, einem a-Release zugeordnet. Anschließend werden sämtliche Plattformobjekte in chronologischer Reihenfolge aufgerufen, und überprüft, ob zu dem jeweiligen a-Release ein jüngeres Plattformobjekt existiert. Falls nicht, wird die Liste der a-Releases um die jeweilige Release-Nummer ergänzt.

Für die b- und c-Releases (d. h. alle Releasenummern, die nicht in der Liste der a-Releases vorkommen) gilt stets, dass sie kompatibel zu den unmittelbar vorangegangenen Releases zu halten sind. Bei den a-Releases, muss jeweils anhand von ϵ und δ überprüft werden, ob diese kompatibel zu halten sind.

Bei $\delta = 1$ kann jedes zweite K-Matrix-Release (konkret im XDIS-Kontext: Jedes zweite a-Release) inkompatibel geändert werden. Bei $\delta = 3$ ist es jedes vierte (vgl. Abschnitt 5.4.2) und allgemein jedes $(\delta + 1)$-te Release. Durch den Offset ϵ wird das Auftreten kompatibler Releases verzögert. Für die Simulation ist nicht der Unterschied zwischen a-, b- und c-Releases relevant, sondern der Unterschied zwischen Releases, in denen inkompatibel geändert werden darf (eine Teilmenge der a-Releases) und solchen, in denen das nicht möglich ist (alle anderen). Diese werden unter Berücksichtigung von δ und ϵ aus der Liste der a-Release-Nummern ermittelt, und in einer chronologisch geordneten Liste abgelegt.

B.2.3 Erstellen der alternativen K-Matrizen

Um die alternativen K-Matrizen zu erstellen, sind zusätzlich zu den Get-Methoden (vgl. Abschnitt B.2.1) weitere Methoden erforderlich, um neue, den alternativen K-Matrizen zugehörige, Datenbankobjekte anzulegen. Die neuen Datenbankobjekte werden nicht in der produktiven XDIS-Datenbank,

sondern einer lokalen SQLite-Datenbank abgelegt. Das Format, der neu anzulegenden Objekte, entspricht dem der ursprünglichen Objekte, mit Ausnahme dreier zusätzlicher Felder:

- Der „HV-Typ“ fasst die Simulationsparameter (vgl. Abschnitt 5.4) als String im Schema „δ-ϵ-ζ“ zusammen.
- Eine Major-Version (vgl. Abschnitt 3.5.3).
- Eine Minor-Version (vgl. Abschnitt 3.5.3).
- Ein Feld „OriginalObjekt“, welches die UUID des entsprechenden Objektes in der ursprünglichen K-Matrix enthält.

Filtern nach einem spezifischen „HV-Typ“ liefert sämtliche Objekte zu einer alternativen K-Matrix, während die Ergebnisse der weiteren Simulationen ausgeblendet werden. Für das Anlegen und den späteren Abruf dieser neuen Objekte werden eigene Methoden erstellt. Im Folgenden werden die unveränderten Kommunikationsobjekte aus der XDIS-Datenbank kurz als „XDIS-Objekte“ und die Objekte in der simulierten, nach den Regeln der hierarchischen Versionierung (HV) erstellten Objekte als „HV-Objekte“ bezeichnet.

Neue UUIDs werden auf Basis von UUID3, d. h. zeitinvariant, auf Basis eines Strings erzeugt (vgl. [Int05b]). So wird sichergestellt, dass bei wiederholten Simulationsläufen dieselben Objekte dieselben UUIDs erhalten (um z. B. die Fehlersuche zu vereinfachen). Der String besteht aus der UUID des ursprünglichen XDIS-Objekts, der Release-Nummer und dem HV-Typ. Letzterer wird einbezogen, um sicherzustellen, dass in den Simulationen mit unterschiedlichen Parametern keine identischen UUIDs vorkommen. Die Release-Nummer in die UUID einzubeziehen ist notwendig, in Situationen, in denen aus einem XDIS-Objekt mehrere HV-Objekte entstehen (welche sich in der UUID unterscheiden müssen). Dies ist dann der Fall, wenn ein XDIS-Objekt zu einem kompatiblen Release verändert wurde, sodass das HV-Objekt Redundanzen enthält, welche die Rückwärtskompatibilität sicherstellen. Diese Redundanzen werden im nächsten, inkompatiblen Release entfernt. Wird das XDIS-Objekt zum nächsten, inkompatiblen Release nicht verändert, verweist das neue HV-Objekt auf dasselbe XDIS-Objekt wie das ersetzte HV-Objekt.

B.2.4 Hauptschleife

Für die Simulation eines K-Matrix-Entwicklungsprozesses, nach den in Abschnitt 4.4.5 aufgestellten Regeln, werden die realen K-Matrix-Releases und die daran vorgenommen Änderungen aus der XDIS-Datenbank entnommen. Auf dieser Basis werden alternative K-Matrix-Releases erstellt, die dem Zustand entsprechen, der bei Anwendung hierarchischer Versionierung vorgelegen hätte. Beispielsweise werden veränderte Objekte kopiert, und wegfallende Objekte werden bis zum nächsten inkompatiblen a-Release erhalten.

Die Hauptschleife der Simulation iteriert über alle Plattformobjekte (d. h. K-Matrix-Releases der Vernetzungsplattform). Pro Plattformobjekt wird für jeden Bus eine Funktion zu dessen Bearbeitung aufgerufen. Dieser wird, neben dem zu bearbeitenden Bus, die Release-Nummer übergeben, und die Information, ob es sich um ein kompatibles oder inkompatibles Release handelt.

Kompatibler Zweig

Im kompatiblen Zweig werden die verschiedenen XDIS-Objekte (ausgehend von den Bussen, über die Interfaces und PDUs bis zu den Signalen) auf Änderungen überprüft. Dazu werden die XDIS-Objekte aus dem aktuellen und dem vergangenen Release abgerufen. Anhand der UUID wird überprüft, ob das Objekt verändert ist.

Falls eine Änderung vorliegt, wird diese in der alternativen K-Matrix mittels einer Änderungskopie (vgl. Definition 4.1) umgesetzt. Andernfalls wird das HV-Objekt aus dem vorangegangen Release dem aktuellen Release zugeordnet. Falls kein Vorgängerobjekt existiert, handelt es sich um ein neu angelegtes Objekt. In dem Fall wird ein neues HV-Objekt mit der semantischen Version 1.0 angelegt. Außerdem muss überprüft werden, ob im vorangegangenen Release Objekte existierten, welche im vorliegenden Release fehlen. Diese müssen in kompatiblen Releases erhalten werden, um die Rückwärtskompatibilität sicherzustellen.

Inkompatibler Zweig

Im inkompatiblen Zweig werden die eingeführten Redundanzen entfernt (und neue Änderungen direkt inkompatibel eingebracht). Für die Simulation bedeutet das, dass die XDIS-Objekte des jeweiligen Releases unverändert als HV-Objekte übernommen werden können. Lediglich die zusätzlichen Attribute (HV-Typ, Major- u. Minor-Version) müssen erzeugt werden. Die semantischen Versionsnummern der HV-Objekte im inkompatiblen Release ergeben sich jeweils aus der semantischen Versionsnummer desselben Objektes im vorangegangenen Release und der Änderung (vgl. Regel 3 und Abschnitt 4.4.2).

Erfolgten in den vergangenen, kompatiblen Releases Änderungen, die durch Änderungskopien kompatibel gehalten wurden (d. h. beim HV-Objekt im vorangegangen Release liegt eine Minor-Version größer Null vor), *oder* liegen zwischen den XDIS-Objekten des vorangegangenen und des aktuellen Releases Änderungen vor, wird die Major-Version um eins erhöht und die Minor-Version auf null zurückgesetzt. Ist die Minor-Version bereits null (d. h. es liegen keine, durch Änderungskopien eingeführte Redundanzen vor) *und* zwischen den XDIS-Objekten des vorangegangenen und des aktuellen Releases liegen keine Änderungen vor, kann das bestehende HV-Objekt unverändert übernommen werden.

B.2.5 Bus-Ebene und Interface Ebene

Die Hauptschleife iteriert in jedem Release über alle Busse und ruft eine Methode auf, um die Busobjeke in der alternativen K-Matrix anzulegen. Das Anlegen eines Busses beinhaltet das Anlegen von Objekten für sämtliche an den Bus angeschlossenen Interfaces. Sowohl für die Busse als auch die Interfaces gibt es dabei drei mögliche Pfade:

Neue Major-Version

In drei Szenarien wird eine neue Hauptversion angelegt. Im ersten Release wird für jedes Objekt die Version 1.0 angelegt. Ferner werden neue Hauptversionen angelegt, wenn zum Zeitpunkt eines inkompatiblen Releases ein HV-Objekt mit bestehenden Redundanzen vorliegen (d. h. in vorigen Releases wurden Änderungskopien durchgeführt, daran erkennbar, dass die Minor-Version ungleich null ist). Oder wenn unmittelbar zum inkompatiblen Release Änderungen vorliegen, die in einem inkompatiblen Release nicht kompatibel gehalten werden.

Neue Minor-Version

Eine neue Minor-Version wird angelegt, wenn in einem kompatibel zu haltenden Release Änderungen vorliegen, die mittels Änderungskopien kompatibel umgesetzt wurden.

Unveränderte Übernahme

In zwei weiteren Szenarien kann das im vorigen Release erstellte HV-Objekt unverändert in das aktuelle Release übernommen werden: In einem kompatibel zu haltenden Release, in dem keine Änderungen vorliegen oder in einem Release, in dem inkompatibel geändert werden darf, in dem aber keine zu entfernenden Redundanzen vorliegen (erkennbar daran, dass die Minor-Version null ist).

B.2.6 PDU-Ebene und Signalebene

Auf der PDU-Ebene finden die eigentlichen Änderungen der Kommunikation statt. Hier kommen neue PDUs hinzu und alte PDUs fallen weg oder

werden verändert. In den beiden letzteren Fällen wird in der alternativen K-Matrix Rückwärtskompatibilität geschaffen (im ersten Fall liegt diese automatisch, auch in der originalen K-Matrix, vor). Dabei entstehen Kommunikationsobjekte, die sich in mehr als nur den Metadaten (vgl. Abschnitt B.2.3) von den Originalobjekten unterscheiden: Durch Änderungskopien auf PDU-Ebene existieren PDUs zeitgleich, die in der originalen K-Matrix nur in unterschiedlichen Releases existieren. Und durch Änderungskopien auf Signalebene entstehen PDUs, die in der ursprünglichen K-Matrix nicht vorkommen.

Auf der PDU-Ebene wirkt sich der Unterschied zwischen einfacher Rückwärtskompatibilität und hierarchischer Versionierung aus. In Abhängigkeit des Simulationsparameters ζ wird die Durchführung von Änderungskopien beeinflusst:

Einfache Rückwärtskompatibilität

Der einfachere Fall ist die Umsetzung einfacher Rückwärtskompatibilität. Dabei wird stets auf PDU-Ebene Kompatibilität hergestellt, indem in kompatiblen Releases Änderungskopien für alle veränderten PDUs durchgeführt werden.

Hierarchische Versionierung

Im Fall hierarchischer Versionierung erfolgen die Änderungskopien gemäß Regel 6 auf der niedrigsten möglichen Hierarchieebene. Es wird überprüft, ob die Möglichkeit besteht, die PDU durch Änderungskopien der darin enthaltenen, veränderten Signale, kompatibel zu halten. Dazu wird einerseits ermittelt, welche Größe die PDU durch die zusätzlichen Signale erreichen würde, und andererseits welche maximale Größe die PDU erreichen darf (unter Berücksichtigung aller Busse, über die die PDU übertragen wird; vgl. Definition 3.53). Liegt die neue PDU-Größe im zulässigen Bereich, werden die Änderungskopien auf Signalebene durchgeführt, andernfalls wird – genau wie bei einfacher Rückwärtskompatibilität – eine Änderungskopie der gesamten

PDU umgesetzt. Da in der Praxis die meisten PDUs deutlich unter ihrer maximal möglichen Größe liegen (vgl. Abb. 5.8) ist zu erwarten, dass in vielen Fällen die Änderungskopien auf Signalebene möglich sind.

B.2.7 Ergebnisse

Ergebnis der verschiedenen Simulationsdurchläufe ist eine SQLite-Datenbank mit mehreren, anhand des „HV-Typs“ (vgl. Abschnitt B.2.3) zu unterscheidenden, alternativen K-Matrizen. Diese bilden den Zustand ab, der vorgelegen hätte, wenn in der Entwicklung, die in Abschnitt 4.4.5 vorgeschlagenen Regeln angewandt worden wären.

Zur Bewertung der verschiedenen Szenarien, und um die in Abschnitt 5.4 gezeigten Abbildungen zu erstellen, wurde für jedes Szenario (alle „HV-Typen“ und die originale K-Matrix), jedes Release und jeden Bus die Buslast (vgl. Abschnitt 3.7) berechnet. Aus den einzelnen Werten wurden, mittels Pythons Matplotlib, für jeden Bus Graphen erstellt, mit den Releases auf der Zeitachse, der Auslastung auf der y-Achse, und unterschiedlichen Farben für die unterschiedlichen Szenarios.