Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Florian Bartols

Echtzeit-Ethernet Restbussimulation: Frühzeitiges Modellbasiertes Testen in Fahrzeugnetzwerken der nächsten Generation

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Florian Bartols

Echtzeit-Ethernet Restbussimulation: Frühzeitiges Modellbasiertes Testen in Fahrzeugnetzwerken der nächsten Generation

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Bettina Buth

Abgegeben am 26. März 2014

Florian Bartols

Titel der Masterarbeit
Echtzeit-Ethernet Restbussimulation: Frühzeitiges Modellbasiertes Testen in Fahrzeugnetzwerken der nächsten Generation

Stichworte
Echtzeit-Ethernet, TTEthernet, Restbussimulation, Bussysteme, Automotive Anwendungen, Modellbasiertes Testen, UML, SysML, UML-MARTE, Abstrakte Testfälle, FIBEX

Kurzzusammenfassung
Die zunehmende Komplexität automobiler Netzwerke und deren steigende Bandbreitenanforderungen bei gleichzeitiger Einhaltung von Zeitaussagen erfordern neue Konzepte in der Vernetzung elektronischer Komponenten im Fahrzeug. Echtzeit-Ethernet ist ein geeigneter Kandidat diese Problemstellung zu lösen. Um frühzeitig im Entwicklungsprozess Komponenten und Systeme zu testen hat sich die Restbussimulation als adäquates Mittel etabliert.
In dieser Thesis wird eine Restbussimulationsmethodik für Echtzeit-Ethernet-Netzwerke entwickelt, mit der es möglich ist, Systeme in frühen Entwicklungsphasen auf funktionale und nicht-funktionale Anforderungen mittels abstrakter Testfälle zu testen. Dazu wird auf die Modellierung von zeitlichen Anforderungen von Echtzeit-Ethernet-Systemen eingegangen und ein abstraktes Testfallmodell entwickelt, das Testfälle nicht-funktionaler Leistungsanforderungen beschreibt und gleichzeitig als Simulationsmodell für den Restbussimulator dient. Die entwickelte Methodik wird anschließend exemplarisch an einer Automotive-Echtzeit-Ethernet-Anwendung angewendet.

Title of the master thesis
Real-time Ethernet Clustersimulation: Model based testing in next-generation in-vehicle networks at early development stages

Keywords
Real-time Ethernet, TTEthernet, Cluster Simulation, Bussystems, Automotive Applications, Modelbased Testing, UML, SysML, UML-MARTE, Abstract testcases, FIBEX

Abstract
The increasing complexity of automotive networks, their time constraints, and their increasing bandwidth demands requires new concepts for connecting electronic automotive components. Real-time Ethernet is a suitable candidate to solve this problem. In order to allow testing components and systems at early development stages, cluster simulation has been established as an adequate manner.
In this thesis, a method for cluster simulation of real-time Ethernet systems is developed, in order to allow testing systems in early design stages, by using abstract test cases. Additionally, a requirement modeling approach for non-functional requirements of real-time Ethernet systems is presented. An abstract test case model is developed, which describes test cases for non-functional performance requirements. It is simultaneously utilized as the simulation model for the cluster simulation environment. The developed methodology is exemplarily applied to an automotive real-time Ethernet system.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Moderne Kraftfahrzeuge sind mit ihrem hohen Anteil an elektronischen Komponenten mittlerweile zu sehr komplexen, verteilten Systemen angewachsen. Sie verfügen teilweise über mehr als 70 Steuergeräte, die miteinander über das Bordnetzwerk vernetzt sind (siehe Abbildung 1.1) und tauschen ca. 2500 verschiedene Nachrichten aus (vgl. Schäuffele / Zurawka 2013; Marscholik / Subke 2007; Navet / Song / Simonot-Lion u. a. 2005).

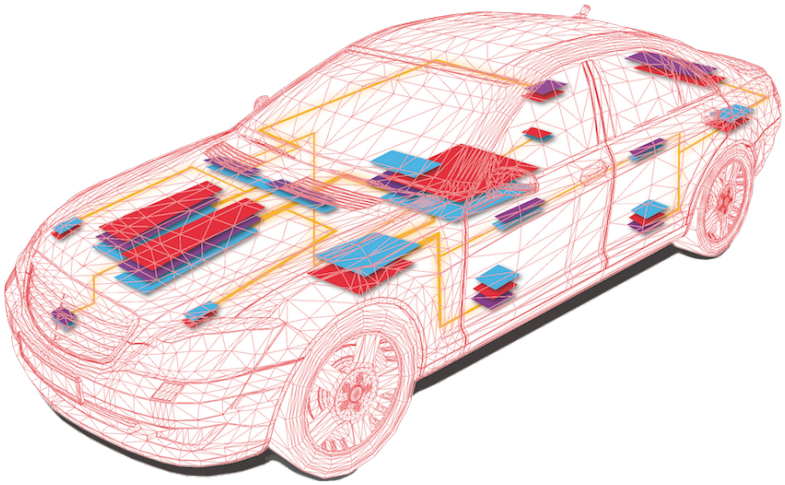


Abbildung 1.1: *Elektronische Komponenten und deren Vernetzung in einem modernen Oberklassefahrzeug (Quelle: CoRE-Arbeitsgruppe [a])*

Auf der einen Seite hat die hohe Anzahl an elektronischen Komponenten im PKW und deren Entwicklung einen großen Einfluss auf die Gesamtkosten im Automobil. Lag der Anteil 1985 noch bei ca. 3%, liegt dieser mittlerweile bei über 20% (vgl. Elektroniknet.de 2008). Dieser Trend wird sich auch in der Zukunft fortsetzen, da immer mehr Funktionen in Software realisert werden. Um dem entgegen zu wirken, müssen die Entwicklungskosten gering gehalten werden, indem frühzeitig im Entwicklungsprozess getestet wird (vgl. Riegraf / Behh / Kraus 2007). Je später Fehler entdeckt werden, desto kostenintensiver ist deren Beseitigung. Da der Entwicklungsprozess in der Automobilindustrie durch viele verschiedene Zulieferer geprägt ist,

werden Systeme an verschiedenen Standorten durch verschiedene Hersteller entwickelt, sodass frühzeitiges Testen erschwert wird, wenn nicht alle Komponenten des Systems während der Entwicklung verhanden sind.
Eine gängige Methode verteilte Anwendungen frühzeitig zu testen, ist die Restbussimulation. Dabei werden einzelne oder mehrere Komponenten durch eine Simulation ersetzt und die reale zu testende Komponente wird mit dem Restbussimulator über die Kommunikationsschnittstelle verbunden. Das Verhalten der simulierten Komponenten wird modellbasiert von der Spezifikation des Gesamtsystems abgeleitet, damit sich die Restbussimulation für den Testling transparent verhält. Auf diese Weise kann die Funktion verifiziert werden, ohne dass das gesamte System vorhanden sein muss. Es unterstützt die Integrationsphase, in der die einzelnen Komponenten zu einem System zusammengefasst werden, da diese schon im Vorfeld auf Implementierungsfehler überprüft werden können. Die Überprüfung von nicht-funktionalen Anforderungen, die z. B. die Reaktionsgeschwindigkeit auf eine empfangene Nachricht festlegt, ist dabei nur schwer durchführbar, da nicht nur Daten analysiert werden müssen, sondern auch Zeitspannen.

Auf der anderen Seite hat die stetige Zunahme an elektronischen Komponenten Auswirkungen auf die Vernetzung dieser Systeme. Das heutige im Fahrzeug befindliche Netzwerk ist zu einer heterogenen Struktur verschiedenster Technologien angewachsen, dessen Gesamtkomplexität nur noch schwer zu beherrschen ist, insbesondere wenn Systeme mit hohen Bandbreitenanforderungen domänenübergreifend Daten austauschen müssen. Für zeitkritische Echtzeitanwendungen, muss das Fahrzeugnetzwerk dabei nicht-funktionale Anforderungen erfüllen. Garantierte zeitliche Aussagen sind bei der Nachrichtenübertragung einzuhalten, damit Daten unter allen Umständen rechtzeitig übermittelt werden und keine Verletzung einer Deadline hervorrufen. Aus diesem Grund müssen neue Ideen und Konzepte in der Vernetzung von elektronischen Komponenten im Fahrzeug Einzug halten, damit neue Anwendungen und Funktionen umgesetzt werden können.
Die Verwendung von Echtzeit-Ethernet kann die Probleme heutiger Fahrzeugnetzwerke lösen. Es stellt hohe Bandbreiten bei gleichzeitiger Echtzeitfähigkeit zur Verfügung, sodass es ein geeigneter Kandidat ist, als Technologie im Fahrzeugnetzwerk der nächsten Generation verwendet zu werden. Für den produktiven Einsatz im Fahrzeug bedarf es allerdings auch entsprechender Werkzeuge, die den Entwicklungsprozess unterstützen.

Damit die Entwicklungskosten auch im Fahrzeugnetzwerk der nächsten Generation gering gehalten werden, muss eine Möglichkeit zum frühzeitigen Testen vorhanden sein. Für Echtzeit-Ethernet sind zum Erstellungszeitpunkt der Arbeit noch keine geeigneten Werkzeuge vorhanden, die eine Restbussimulation ermöglichen. Hierbei müssen die Charakteristiken, wie z. B. verschiedene Nachrichtenklassen und Synchronisationsmechanismen, vollständig vom Restbussimulator unterstützt werden, damit eine Kommunikation mit dem Testling möglich ist. Um sowohl funktionale als auch nicht-funktionale Anforderungen überprüfen zu können, muss die Restbussimulationsplattform in der Lage sein, Zeitabschnitte bestimmen und messen zu können.

1.1 Problemstellung und Zielsetzung

Das Hauptziel dieser Arbeit ist es, eine Restbussimulationsmethodik für Echtzeit-Ethernet Netzwerke zu entwickeln, um eine Testmöglichkeit für modellbasiertes, frühzeitiges Testen im Fahrzeugnetzwerk der nächsten Generation zu ermöglichen. Dabei wird der Fokus auf eine modellbasierte Entwicklung von Echtzeit-Ethernet-basierten Anwendungen und Systemen gelegt, sodass Testfälle anhand von Modellen abgeleitet werden können. Insbesondere sollen nicht-funktionale Anforderungen innerhalb dieser Modelle modelliert werden können, die das zeitliche Verhalten von Systemen beschreiben. Auf diese Weise lassen sich nicht nur Testfälle für funktionale, sondern auch für nicht-funktionale Anforderungen ableiten, sodass entwickelte Systeme systematischer getestet werden können und eine höhere Testabdeckung erreicht wird.
Damit eine Restbussimulation für Echtzeit-Ethernet basierte Anwendungen im Fahrzeugnetzwerk der nächsten Generation durchgeführt werden kann, um sowohl funktionale, als auch nicht-funktionale Leistungsanforderungen zu überprüfen, müssen deshalb folgende Bereiche erarbeitet und diskutiert werden:

- Prinzipien von Echtzeit-Ethernet als Fahrzeugnetzwerk der nächsten Generation
- Modellierungsmöglichkeiten von nicht-funktionalen Anforderungen innerhalb von Automotivesystemen
- Ein geeignetes Testfallmodell, das zur Überprüfung dieser nicht-funktionalen Anforderungen verwendet werden kann, um modellbasiertes Testen zu ermöglichen
- Implementierung einer geeigneten Restbussimulationsplattform, die das Testfallmodell ausführt und eine Analyse der Testergebnisse zulässt

1.2 Inhaltlicher Aufbau dieser Arbeit

Diese Arbeit ist wie folgt aufgebaut: In Kapitel 2 auf der nächsten Seite werden Hintergrundinformationen dieser Arbeit und verwandte Arbeiten vorgestellt. Dabei wird insbesondere auf die Systementwicklung, die Restbussimulation als Werkzeug und Echtzeit-Ethernet als Fahrzeugnetzwerk der nächsten Generation eingegangen. Kapitel 3 auf Seite 36 beschreibt Modellierungsmöglichkeiten, insbesondere aus der Sicht von zeitlichen Anforderungen für Echtzeit-Ethernet Systeme und stellt ein abstraktes Testfallmodell vor, mit dem es möglich ist, diese zeitlichen Anforderungen zu überprüfen. Unterschiede zu bisherigen Restbussimulationsansätzen werden in Kapitel 4 auf Seite 62 diskutiert, gefolgt von einer Beschreibung der Anforderungen und benötigten Komponenten einer Restbussimulationsplattform. Das Konzept und die Implementierung der Plattform werden im nachfolgenden Kapitel 5 auf Seite 74 beschrieben, sodass im anschließenden Kapitel 6 auf Seite 95 beispielhaft ausgewählte Anforderungen einer Anwendung per Restbussimulation getestet werden. Das letzte Kapitel 7 auf Seite 104 schließt diese Arbeit mit einer Zusammenfassung, einem Fazit und einem Ausblick ab.

2 Hintergrund und verwandte Arbeiten

Dieses Kapitel gibt Informationen zum Hintergrund des modellbasierten Entwicklungsprozesses und erklärt Grundlagen, die zum Verständnis dieser Arbeit nötig sind. Zuerst wird dabei der Systementwicklungsprozess im Automobil erklärt, zu dem die Modellierung von Systemen und Anwendungen gehört. Anschließend wird die Restbussimulation als Methodik zum Testen von verteilten Systemen beschrieben. Echtzeit-Ethernet als zukünftiges Fahrzeugnetzwerk der nächsten Generation wird im darauf folgenden Abschnitt dargestellt. Zum Schluss dieses Kapitels werden produktiv eingesetzte Restbussimulatoren und vergleichbare wissenschaftliche Arbeiten vorgestellt und eine Abgrenzung zu der umgesetzten Arbeit gegeben.

2.1 Systems Engineering in der Automobilindustrie

Moderne Fahrzeuge haben weit mehr als 2500 Funktionen , die mittlerweile zum größten Teil in Software realisiert werden. Sie übernehmen sowohl Komfort-, als auch Assistenzfunktionen, die den Fahrer während der Fahrt unterstützen und unterhalten (vgl. Schäuffele / Zurawka 2013; Marscholik / Subke 2007; Navet / Song / Simonot-Lion u. a. 2005). Dadurch ist das Automobil zu einem technisch komplexen Konsumgut geworden, dessen Anforderungen sich jedoch wesentlich von denen der sonstigen Konsumgüterelektronik unterscheiden. Besonders hervorzuheben sind dabei die hohen Anforderungen an Sicherheit, Zuverlässigkeit und Verfügbarkeit und die vergleichsweise lange Produktlebensdauer von mehr als 20 Jahren.
Des Weiteren erfordert der Entwicklungsprozess gerade im Automobilbereich eine interdisziplinäre und dezentrale Zusammenarbeit, da der Entwicklungsprozess durch Automobilfabrikanten und Zulieferer geprägt ist. Software-Entwickler müssen mit Ingenieuren aus der Antriebs-, Fahrwerks- und Elektronik-Entwicklung kommunizieren, die gleichzeitig oft aus verschiedenen Unternehmen stammen. So müssen Spezifikationen durch den Fabrikanten erarbeitet werden, die durch den Zulieferer in Produkte umgesetzt werden. Dazu bedarf es einer gemeinsamen „Sprache“ in der Entwicklungsmethodik, die von allen Beteiligten verstanden wird und somit eine enge Zusammenarbeit der verschiedenen Disziplinen und Unternehmen möglich macht. Damit die Entwicklung eines Fahrzeuges zu einem Erfolg wird, müssen weiterhin geeignete Methoden vorhanden sein, die die kontinuierlich steigende Komplexität automobiler Systeme beherrschen, ein konsequentes Qualitäts-, Risiko- und Kostenmanagement

ermöglichen und eine interdisziplinäre Zusammenarbeit erlauben. Dieser Abschnitt beschreibt zuerst die Eigenschaften automobiler Softwaresysteme, stellt ein verbreitetes Vorgehensmodell der Projektdurchführung vor und gibt einen Überblick über eine Entwicklungsmethodik, die es erlaubt, sowohl die Komplexität zu beherrschen, als auch die Grenzen der Interdisziplinarität zu überwinden.

2.1.1 Eigenschaften von Automotivesoftwaresystemen

Moderne Fahrzeuge sind komplexe, verteilte und heterogene Systeme, die nahezu alle Bereiche aus der Informationsverarbeitung abdecken und die Funktionen mittlerweile hauptsächlich in Software realisieren. So gibt es unkritische Teilsysteme, die z. B. für das Anzeigen von Informationen für den Fahrer und der Unterhaltung aller Passagiere, so genannte Infotainmentsysteme, zuständig sind. Ebenso sind sicherheitskritische Systeme, die Echtzeitregelungsaufgaben übernehmen, im Fahrzeug vorhanden. Funktionen im Fahrzeug werden deshalb in der Regel in fünf Domänen (Multimedia, Komfort, Passagiersicherheit, Antriebs-/Fahrwerkstrang & Wartung) aufgeteilt (vgl. Broy / Krüger / Pretschner u. a. 2007), die jeweils separate Eigenschaften haben und anhand ihren Anforderungen eingeteilt werden. In der Vergangenheit wurde für jede Domäne ein eigenes, in sich geschlossenes System realisiert, dessen Daten ausschließlich innerhalb des jeweiligen Systems ausgetauscht wurden. Jedoch bedarf die weiter fortschreitende Entwicklung neuer Sicherheits-, Assistenz- und Informationsanwendungen Möglichkeiten zum Datenaustausch zwischen diesen Systemen, sodass eine insgesamt heterogene Struktur entsteht, bei der z. B. Daten der Antriebsstrangdomäne an das Infotainmentsystem weitergereicht werden. Automotivesoftwaresysteme unterscheiden sich außerdem darin, in welcher Form anliegende Daten zu verarbeiten sind, bzw. wann anstehende Aktionen ausgeführt werden. Hierbei wird zwischen *diskreten*, *kontinuierlichen* und *hybriden Mischsystemen*, die beide vorherigen Eigenschaften besitzen, unterschieden, die nachstehend beschrieben werden.

Diskrete Systeme Diskrete Systeme werden in werte-/ereignisdiskrete und zeitdiskrete Systeme eingeteilt, wobei die Ausgangsdaten oder auszuführenden Aktionen von den diskreten Werten/Zeitpunkten bestimmt werden. Werte/ereignisdiskrete Systeme unterteilen den Dateneingang in verschiedene Wertebereiche, denen jeweils eine Aktion zugeteilt ist. Die Aktion wird ausgeführt, sobald der Eingangswert innerhalb des entsprechenden Wertebereichs liegt. Es wird dementsprechend eine Diskretisierung des Wertes durchgeführt und die Verarbeitung hat keine Relation zur aktuellen Zeit. Bei zeitdiskreten Systemen spricht man von der Diskretisierung der Zeit, bei der die aktuell zu verarbeitenden Daten zu bestimmten Zeitpunkten ausgelesen, bzw. Aktionen generiert werden. Auf der einen Seite eignen sich ereignisdiskrete Systeme in Bereichen, in denen Aktionen durch bestimmte Ereignisse beeinflusst werden, z. B. bei einer Betätigung ei-

nes Knopfes. Auf der anderen Seite eignen sich zeitdiskrete Systeme für periodisch anfallende Aktionen, z. B. für das Auslesen eines Sensorwertes.

Kontinuierliche Systeme Im Gegensatz zu diskreten Systemen wird in kontinuierlichen Systemen keine Diskretisierung durchgeführt, um eine Aktion auszuführen. Sie sind vergleichbar mit stetigen mathematischen Funktionen, bei der Aktionen stufenlos zu keinen bestimmten Punkten ausgeführt werden. Kontinuierliche Systeme finden bei der Berechnung physikalischer Vorgänge Verwendung und werden deshalb für regelungstechnische Anwendungen bevorzugt.

Hybride Mischsysteme Hybride Systeme vereinen die Eigenschaften diskreter und kontinuierlicher Systeme, sodass Aktionen kontinuierlich ausgeführt oder zu bestimmten Ereignissen besondere Reaktionen ausgelöst werden. Sie sind deshalb gut geeignet physikalische Prozesse mit verschiedenen Betriebsarten umzusetzen, z. B. automatische Klimaanlagen, die die Temperatur im Fahrgastraum regeln.

Kontinuierliche Systeme werden mittels Differentialgleichungen beschrieben. Diskrete Systeme hingegen werden in der Regel mit Modellen beschrieben, die das Auftreten von Ereignissen darstellen können (z. B. Zustandsautomaten oder Ablaufdiagramme). Um hybride Systeme zu modellieren, haben sich hybride Zustandsautomaten durchgesetzt, die es erlauben kontinuierliches mit diskretem Verhalten zu verbinden (vgl. Henzinger 2000). Alle drei Arten von Systemen finden sich in den fünf Anwendungsdomänen im Automobil wieder, die nachfolgend mit ihren Eigenschaften aufgelistet sind.

1. **Multimediaanwendungen** (z. B. Infotainment und Navigation) sind in der Regel Systeme mit weichen Echtzeitanforderungen. Sie entsprechen zum größten Teil diskreten Systemen.

2. Auch **Komfortanwendungen** unterliegen typischerweise weichen Echtzeitanforderungen. In dieser Domäne werden sowohl diskrete, als auch kontinuierliche Systeme verwendet, wenngleich die Anzahl diskreter Systeme überwiegt. Sie können z. B. zur Steuerung der Klimaautomatik oder der elektrischen Fensterheber verwendet werden.

3. Anwendungen, die für die **Sicherheit** der Passagiere in besonderen Umständen sorgen, wie z. B. das Zünden eines Airbags oder das Straffen des Sicherheitsgurtes bei einem Aufprallunfall, unterliegen strikten Echtzeitanforderungen und werden durch diskrete Systeme beschrieben.

4. Anwendungen aus dem **Antriebs- und Fahrwerkstrang** (z. B. adaptive Fahrwerksteuerung) unterliegen ebenfalls strikten Echtzeitanforderungen. Im Gegensatz zu Sicherheitsanwendungen werden in dieser Domäne hauptsächlich kontinuierliche Systeme verwendet, da hier regelungstechnische Aufgaben zu bearbeiten sind, die ein ununterbrochenes Ausführen während des Betriebes voraussetzen.

5. **Wartungs- und Infrastrukturanwendungen**, die Softwareupdates zulassen, sind typischerweise durch keine oder weiche Echtzeitanforderungen und einer diskreten Ausführung gekennzeichnet.

Die Entwicklung dieser Systeme wird, wie bereits erwähnt, durch die Zunahme an Funktionalität und Interdomänenkommunikation immer komplexer und der Entwicklungsprozess immer unübersichtlicher. Daher bedarf es einer geeigneten Methode, diese Probleme zu überwinden und den Entwicklungsprozess zu vereinfachen.

2.1.2 Das V-Modell als Vorgehensmodell

Das allgemeine V-Modell (vgl. IABG 2012) ist ein weit verbreitetes Vorgehensmodell in der Automobilindustrie (vgl. Meier 2008; Schäuffele / Zurawka 2013), das seinen Namen aus der Darstellungsform als „V“ erhalten hat (siehe Abbildung 2.1 auf der nächsten Seite) und kann in zwei Entwicklungsprozessbereiche aufgeteilt werden. Die linke Seite beschreibt den Konstruktionsprozess, in dem das System entworfen wird, während die rechte Seite den Verifikations- bzw. Validierungsprozess darstellt. Durch die Verifikation wird überprüft, ob das System richtig entwickelt wurde, während die Validierung sicherstellt, ob das richtige System entwickelt wurde. Die Konstruktions- und Verifikation-/Validierungsprozesse werden in einzelne Phasen aufgeteilt, wodurch eine durchgängige phasenorientierte, top-down Herangehensweise ermöglicht wird, bei der jeder Konstruktionsphase eine entsprechende Verifikationsphase gegenüber steht und eine Verfeinerung der Konstruktion im Laufe des Prozesses vorgibt. Dadurch ermöglicht dieses Modell den geforderten durchgängigen Entwicklungsprozess von der anfänglichen Analyse der Anforderungen bis zum abschließenden Akzeptanztest. Die Gegenüberstellung wird dadurch erreicht, dass Testfälle aus den erarbeiteten Spezifikationen der Konstruktionsphase erstellt werden, die in der entsprechenden Validierungsphase auf dem System ausgeführt werden. Die Ist-Ergebnisse werden anschließend mit den Soll-Ergebnissen der Spezifikation verglichen, um das Verhalten zu überprüfen. Nachfolgend werden die einzelnen Phasen und dessen Ziele vorgestellt, sodass der Entwicklungsprozess innerhalb des V-Modells deutlich wird:

Abstrakte Benutzeranforderungsspezifikation Der erste Schritt im V-Modell ist die Analyse der Benutzeranforderungen des zu entwickelnden Systems und die daraus resultierende Erstellung einer abstrakten Anforderungsspezifikation. Diese Spezifikation beinhaltet das gewünschte Verhalten des Systems aus der Sicht des Benutzers.

Funktionaler Systementwurf Aufbauend auf der abstrakten Benutzeranforderungsspezifikation wird der funktionale Systementwurf erarbeitet. In dieser Phase werden Funktionen definiert, die das geforderte Verhalten umsetzen sollen. Zusätzlich werden die Schnittstellen der verschiedenen Funktionen spezifiziert, sodass abschließend Anforderungen an die Funktionen existieren.

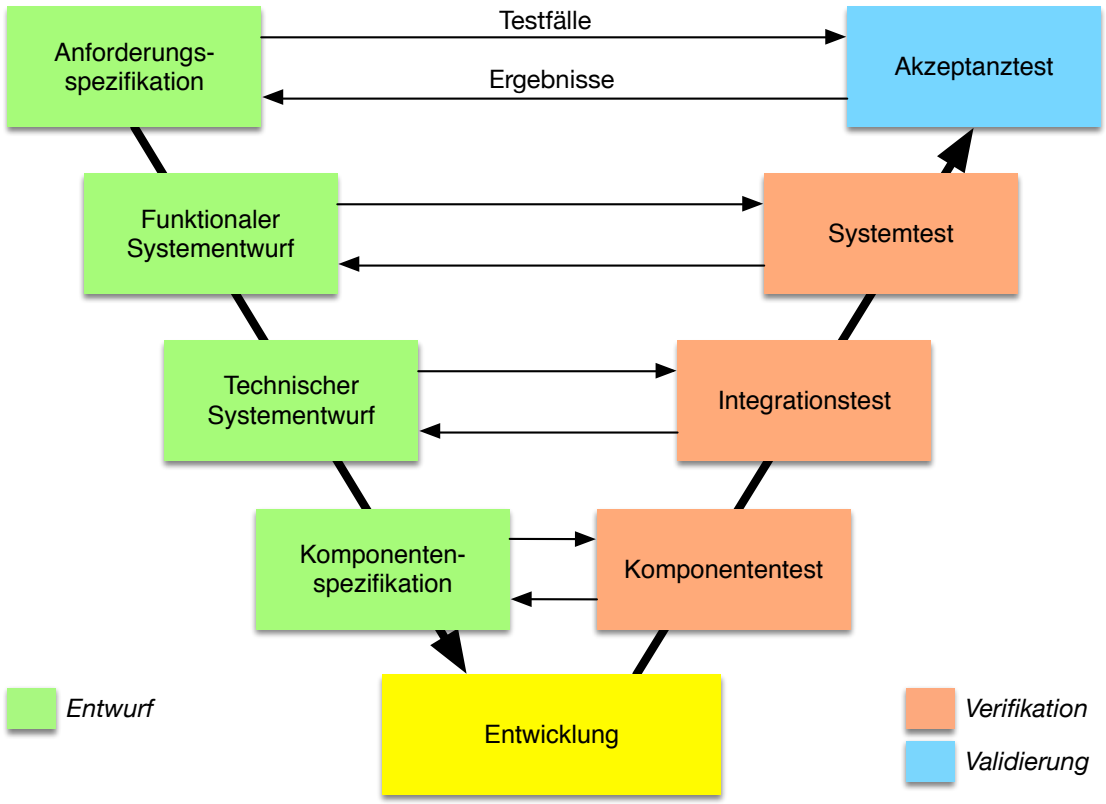


Abbildung 2.1: *Das V-Modell als Vorgehensmodell (nach Schäuffele / Zurawka 2013)*

Technischer Systementwurf Nachdem der funktionale Systementwurf durchgeführt ist, kann das System von der technischen Seite aus spezifiziert werden. In dieser Phase wird definiert, aus welchen Teilkomponenten das System besteht, welche Funktionen auf welchen Komponenten ausgeführt werden und wie die Kommunikation stattfindet.

Komponentenspezifikation Auf Basis des technischen und funktionalen Systementwurfs werden die einzelnen Komponenten des Systems mit allen implementierungsrelevanten Aspekten spezifiziert, sodass sich ein konkretes Modell ergibt.

Implementierung der Komponenten Anhand der Spezifikation der vorherigen Phase werden die einzelnen Komponenten implementiert.

Komponententest Als erster Schritt nach der Implementierung erfolgt der Test der einzelnen Komponenten. Hierbei dient die Komponentenspezifikation dazu, dass modellierte

Verhalten zu überprüfen, sodass Implementierungsfehler rechtzeitig aufgedeckt und behoben werden können.

Integrationstest Ist der Komponententest erfolgreich abgeschlossen, werden die einzelnen Komponenten in ein größeres oder Gesamtsystem integriert. Anschließend wird die Kommunikation und das Zusammenspiel überprüft. Innerhalb dieser Phase lassen sich frühzeitig Mängel innerhalb der Kommunikation feststellen. Dafür werden die Spezifikationen aus dem technischen Systementwurf verwendet.

Systemtest Beim anschließenden Systemtest wird das gesamte System anhand der Spezifikationen des funktionalen Systementwurfs überprüft. Im Gegensatz zum Integrationstest werden die Tests unter nahezu realen Bedingungen auf dem System ausgeführt. Hier wird das korrekte Verhalten der Funktionen sowie die Einhaltung der definierten Schnittstellen sichergestellt.

Akzeptanztest Zum Abschluss wird das entwickelte System dem Akzeptanztest unterzogen. Dieser Test prüft gegen die abstrakte Benutzeranforderungsspezifikation und ermittelt, ob das richtige System entwickelt wurde und dient zur Validierung.

Wichtig zu erwähnen ist, dass das V-Modell sowohl auf das gesamte System, als auch auf Teilsysteme angewendet wird. Bei komplexen Systemstrukturen, wie in einem Automobil, ist es üblich, die Herangehensweise „Divide et Impera“ – Teile und Beherrsche – umzusetzen. Hierbei wird das gesamte System in verschiedene kleinere Systeme aufgeteilt, um einen besseren Überblick zu erhalten und den Entwicklungsprozess zu vereinfachen. Dieser Schritt kann beliebig auf die entstandenen Teilkomponenten angewendet werden, sodass diese einfacher zu entwickeln sind.

2.1.3 Modellbasierte Entwicklung von Automotivesystemen

Wie bereits erwähnt sind gerade Anwendungen im Automobil zu sehr komplexen Softwaresystemen angewachsen, die sehr hohe Anforderungen an Sicherheit und Zuverlässigkeit des Systems erfüllen müssen. Durch diese Eigenschaften muss es schon während konstruktiver Phasen möglich sein, Fehler im Entwurf zu erkennen und zu beseitigen. Der Entwicklungsprozess wird so nicht gefährdet und eine bessere Vorhersagbarkeit der nächsten Schritte wird erreicht. Dieses erlaubt eine bessere Integration verschiedener Module und Systeme (vgl. Bauer / Broy / Romberg u. a. 2005). Zusätzlich ist es nötig eine gemeinsame Kommunikationsbasis im interdisziplinären und dezentralen Entwicklungsprozess zu verwenden, damit Spezifikationen unternehmensübergreifend verwendet, im Team Probleme betrachtet und Lösungen erarbeitet werden können. Aus diesen Gründen hat sich die modellbasierte Entwicklung als Methodik im Softwareentwicklungsprozess auch in der Automobilindustrie durchgesetzt, sodass bekannte Sprachen aus der gängigen generischen Softwareentwicklung bzw. Systementwicklung (z. B. *Unified Modelling Language (UML)* (vgl. OMG [d]) oder

System Modelling Language (SysML) (vgl. OMG [c])) zur Modellierung verwendet werden können (vgl. Apvrille / Becoulet 2012; Andrianarison / Piques 2010). Zusätzlich sind eigene Sprachen entworfen worden (u. a. *Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL)* (vgl. Cuenot / Chen / Gerard u. a. 2007) oder *Automotive Model-based Development (AutoMoDe)* (vgl. Bauer / Broy / Romberg u. a. 2005)), um die speziellen Anforderungen im Automobil adressieren zu können, die nicht oder nur umständlich mit generischen Sprachen modelliert werden können.

Generell teilen sich alle Modellierungssprachen die Eigenschaft, verschiedene Modelle und Diagramme zur Modellierung zur Verfügung zu stellen. Sie können in die Konstruktionsphasen aus dem V-Modell klassifiziert werden, sodass Diagrammnotationen für die Modellierung von Benutzeranforderungen, zur Beschreibung des funktionalen und technischen Entwurfs und der Komponenten vorhanden sind. Die Notationen der verschiedenen Sprachen unterscheiden sich in ihren Details, basieren jedoch auf einem gemeinsamen Fundament an Modellen. So sind in nahezu allen Sprachen Zustandsautomaten, Blockdiagramme und Sequenzdiagramme vorhanden. Auf dieser Basis ist es möglich, sowohl Verhalten und Funktionen eines Systems, als auch Struktur und Datenfluss zu beschreiben.

Durch die verschiedenen vorhandenen Modelle und Diagramme der Modellierungssprachen kann der Top-Down Entwicklungsprozess gut umgesetzt werden, da das System in verschiedenen Schichten entwickelt werden kann. Da alle Modellierungssprachen eine eigene grafische Notation bereitstellen, ist es möglich, die erarbeiteten Diagramme als gemeinsame Sprache zu verwenden, mit der alle beteiligten Personen im Entwicklungsprozess arbeiten können. Komplexe Zusammenhänge können detailliert und anschaulich dargestellt werden.

Die wichtigste Eigenschaft einer modellbasierten Entwicklung ist jedoch die Möglichkeit analytische Methoden auf den Modellen anzuwenden, um fehlerhaftes Verhalten oder Designschwächen frühzeitig zu erkennen. Die meisten Modellierungssprachen weisen ein formales Gerüst auf, sodass geeignete Werkzeuge dazu verwendet werden, Modelle statisch zu untersuchen. Falls eine analytische Betrachtung aufgrund der Systemkomplexität zu schwierig ist, können auch Simulationen aus dem Modell erstellt werden, die das Verhalten des entwickelten Systems widerspiegeln. Zusätzlich kann aus den überprüften Modellen Code generiert werden, sodass die Implementierungsphase der Komponenten deutlich erleichtert und vereinfacht wird, wodurch Kosten in der Entwicklung eingespart werden können.

2.1.4 Modellbasiertes Testen im Entwicklungsprozess

Neben der modellbasierten Entwicklung von Software und Systemen findet auch das modellbasierte Testen Einzug im Entwicklungsprozess von Systemen im Automobil (vgl. Bringmann / Krämer 2008; Lamberg / Beine / Eschmann u. a. 2004). Das Ziel bei der Ausführung

Testfällen ist es Fehler in der Implementierung von Funktionen, Komponenten oder Systemen aufzudecken und wird im V-Modell durch die Verifikations- & Validierungsprozesse (siehe Abbildung 2.1 auf Seite 8) dargestellt. Modellbasiertes Testen ist eine Variante, die ausdrücklich auf Modellen aufbaut, die das Verhalten des zu prüfenden Systems (im weiteren Verlauf der Arbeit auch *System-under-Test – SUT* genannt) widerspiegeln (vgl. Utter / Pretschner / Legeard 2006; Zander / Schiefdecker / Mosterman 2011). Die Idee dieser Variante ist es, Testfälle systematisch aus den Modellen zu erzeugen, die in den verschiedenen Verifikationsphasen wiederverwendet werden können. Die Erstellung hängt somit nicht von der Intuition des Testingenieurs ab. Modellbasiertes Testen wird hauptsächlich für die Blackbox-Testmethodik eingesetzt (vgl. Zander / Schiefdecker / Mosterman 2011), die im Gegensatz zur Whitebox-Testmethodik ohne Kenntnisse der Komponentenimplementierung erstellt werden. In diesem Fall dienen die erstellten Modelle als Spezifikation des SUT.

Die Testfälle werden dabei direkt aus den Modellen des SUT abgeleitet und können manuell durch den Testingenieur oder automatisch, z. B. durch Graphenanalyse bei Zustandsdiagrammen (vgl. Weißleder / Schlingloff 2011) erzeugt werden. Bei der Erzeugung wird weiterhin unterschieden, ob die Testfälle zur Laufzeit (online) oder vor der Ausführung (offline) erstellt werden. Online erzeugte Testfälle können verwendet werden, um reaktiv auf das Verhalten des SUT zu reagieren, während ihr offline erzeugtes Pendant mehrmals mit den gleichen Parametern ausgeführt werden kann.
Die Modelle aus denen die Testfälle abgeleitet werden, können entwickelte Systemmodelle sein oder speziell entwickelte Testmodelle, die vom gesamten System abstrahieren und nur benötigte Teile darstellen. Dabei können die Modelle sowohl funktionale, als auch nichtfunktionale Anforderungen darstellen, sodass beide Arten von Anforderungen überprüft werden können. Dabei ist es wichtig, dass die Modelle in der Entwicklungsphase vorher auf Fehlerfreiheit (z. B. durch analytische Methoden) überprüft wurden.

Die Taxonomie des modellbasierten Testens beschreibt allerdings nicht nur die Erzeugung von Testfällen anhand von Modellen, sondern weiterhin die Ausführung der Fälle auf speziellen Testplattformen. Offline erzeugte Testfälle müssen ein gewisses Level an Abstraktion vorweisen, damit sie auf den unterschiedlichen Plattformen ausgeführt und wiederverwendet werden können. Die Testfälle werden anschließend auf den jeweiligen Testplattformen in die speziellen plattformabhängigen, ausführbaren Fälle umgewandelt, die dann die speziellen Stimuli der Testumgebung bereitstellen.
Die erzielten Ergebnisse können anschließend online zur Laufzeit oder offline nachdem der Test ausgeführt wurde, ausgewertet werden. Die Auswertung kann automatisch oder manuell erfolgen.

Durch modellbasiertes Testen können entwickelte Systeme systematischer auf Fehler überprüft werden und sorgen durch die Wiederverwendbarkeit der Testfälle auf verschiedenen Testplattformen für eine Verkürzung der Verfikationsphasen, da nicht für jede Plattform

Testfälle erstellt werden müssen. Daraus folgt, dass die Entwicklungszeit und -kosten reduziert werden können.

2.2 Restbussimulation als Methodik im Entwicklungsprozess

Die Restbussimulation ist in der Automobilindustrie eine weit verbreitete Methode in der Entwicklung von elektronischen Steuergeräten (vgl. Riegraf / Behh / Kraus 2007) und wird im V-Modell innerhalb des Verifikationsprozesses angewendet (siehe rechte Seite in Abb. 2.1 auf Seite 8). Im Gegensatz zu einer reinen Softwaresimulation wird in einer Restbussimulation nicht das ganze verteilte System simuliert. Stattdessen wird ein hybrider Ansatz umgesetzt, bei der simulierte und reelle Teile gleichzeitig während der Simulation in Betrieb sind. Der Name „Restbus“ lässt sich an der Eigenschaft herleiten, dass die Vernetzung von Komponenten im Automobil zum größten Teil durch Feldbusse realisiert wird. Die simulierten Teile spiegeln in dieser Methodik den Rest eines Busses, also den Rest des verteilten Systems wider, die mit den realen Teilnehmern interagieren. Dieser Abschnitt beschreibt zuerst prinzipielle Grundlagen einer Simulation. Anschließend wird die Restbussimulation als Methodik vorgestellt und anhand eines Beispiels erläutert.

2.2.1 Grundlagen zur Simulation

Der Begriff „Simulation“ stammt aus dem lateinischen und bedeutet *etwas nachzubilden* oder *nachzuahmen* bzw. *ähnlich machen*. Simulationen werden häufig während der Analyse von komplexen Systemen und Prozessen eingesetzt, bei denen eine theoretische und analytische Betrachtung oft nur schwer durchführbar ist. Sie unterstützen den Analyseprozess dahingehend, dass das zu untersuchende System virtuell nachgebildet wird und anschließend Experimente durchgeführt werden. Sie sind vielerorts die einzige Möglichkeit Systeme in besonderen Umständen zu untersuchen, da die gleichen Experimente mit realen Systemen entweder zu gefährlich oder zu teuer sind.
Simulationen können aber auch eingesetzt werden, um einem bestehenden, realen System das Vorhandensein eines oder mehrerer virtueller Systeme vorzutäuschen. Anschließend kann die Wechselwirkung beider Systeme untersucht werden (vgl. Galla 1999; Schlager 2008), welches unter realen Bedingungen nur schwer durchzuführen ist.

Beide Simulationsvarianten haben die Gemeinsamkeit, dass ein echtes System in einem virtuellen Pendant nachgebildet werden muss. Man spricht deshalb von der Bildung eines *Simulationsmodells*, dass das echte System repräsentiert und einem *Simulator*, der dieses Modell ausführt. Bei der Nachbildung des Systems, sei es zum einen durch Werkzeuge aus der modellbasierten Entwicklung oder zum anderen bei einer manuellen Umsetzung, wird oft von der im Original verwendeten Struktur, Funktion oder des Verhaltens abstrahiert, da

nicht alle Merkmale des realen Systems relevant sind und die Modellbildung durch diese nur unnötig erschwert. Inwieweit vom realen System abstrahiert werden kann, hängt von den erhofften Ergebnissen und den Anforderungen an die Simulation ab und muss im Vorweg der Simulation bestimmt werden. Soll z. B. die Auslastung eines Netzwerks unter bestimmten Voraussetzungen analysiert werden, so kann das Anwendungsverhalten der einzelnen Knoten vernachlässigt werden. Im Gegensatz dazu, wenn das Verhalten der Knoten in bestimmte Situationen analysiert werden soll, darf nicht abstrahiert werden.

Um das erstellte Simulationsmodell auszuführen, gibt es drei verschiedene Techniken, die sich darin unterscheiden, wie die Zustandsübergänge innerhalb des Simulationsmodells berechnet werden (vgl. Helal 2008). In Anlehnung an die Art des Originalsystems (siehe 2.1.1 auf Seite 5), wird auch bei Simulationen des nachgebildeten Modells zwischen *diskreten*, *kontinuierlichen* und *hybriden* Methoden unterschieden. Der Zustandsübergang der drei Techniken ist in Abbildung 2.2 dargestellt. Nachfolgend werden die drei Simulationstechniken beschrieben.

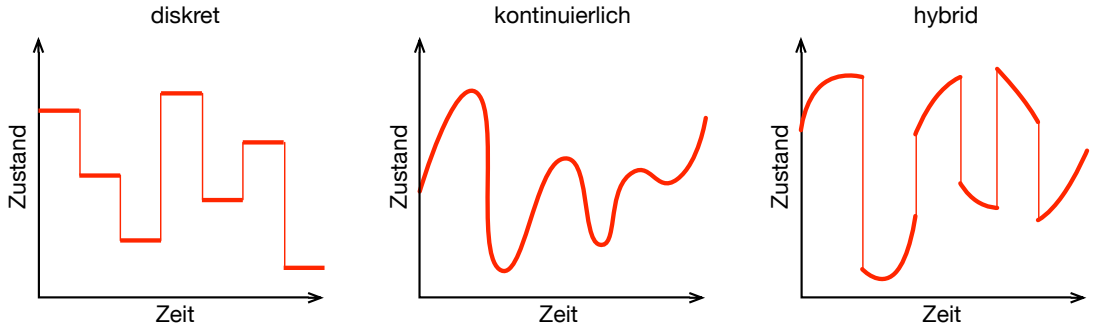


Abbildung 2.2: *Ausführungsvarianten von Simulationsmodellen im Überblick (nach Steinbach 2011)*

Diskrete Simulation Diskrete Simulationen erneuern Zustände sprunghaft zu bestimmten Ereignissen oder Zeitpunkten und lassen sich dementsprechend zusätzlich in *diskrete ereignisbasierte* und *diskrete zeitbasierte* Simulationen unterscheiden. Beide Varianten verwenden eine Ereignisliste mit Ereignispaaren, die den Aktionszeitpunkt sowie die eigentliche Aktion beinhalten. Der Unterschied beider Varianten liegt im zeitlichen Verhalten der Ausführung der Ereignisse. In Abbildung 2.3 auf Seite 15 ist der Ablauf beider Varianten während der Ausführung dargestellt. Während in diskreten zeitbasierten Simulationen ein Taktgeber existiert (siehe rechte Seite in Abbildung 2.3), der die Simulationszeit kontinuierlich mit dem gleichen Wert inkrementiert, wird die Simulationszeit bei einer diskreten ereignisbasierten Simulation direkt aus dem nächsten Ereignis entnommen (linke Seite in Abb. 2.3). Die Ereignisse werden bei der ersten

Variante nur ausgeführt, wenn der Aktionszeitpunk des Ereignisses mit der Simulationszeit übereinstimmt. Im Gegensatz ist die Taktlänge bei der zweiten Variante variabel und entspricht der Differenz zweier aufeinanderfolgender Ereignisse. Generell sind diskrete ereignisbasierte Simulationen performanter, da die Ereignisse direkt nacheinander ausgeführt werden können und keine Wartezeiten entstehen. Für Simulationen, die in Echtzeit ausgeführt werden sollen, sind diskrete zeitbasierte Simulationen zu bevorzugen, da hier die Simulationszeit der realen Zeit entsprechen muss.
Diskrete Simulationen werden in der Regel für Systeme verwendet, bei denen das Aufstellen eines mathematischen Modells nur schwer oder gar unmöglich ist, bzw. außerhalb des Rahmens der Untersuchung liegt. Die Analyse von paketorientierten Netzwerken ist gut durch Ereignisse zu modellieren und ist z. B. im Simulationswerkzeug OMNeT++ (vgl. OMNeT++ Community) umgesetzt worden.

Kontinuierliche Simulation In einer kontinuierlichen Simulation ändern sich die Zustände des Modells stetig über die Zeit. Das heißt, dass der aktuelle Zustand abhängig von der Zeit ist und kann, wie ihr reales Pendant, in Differentialgleichungen beschrieben werden. Auf dem Simulator werden sie anschließend in der Regel durch numerische Verfahren (z. B. durch Euler- oder Runge-Kutta-Verfahren (vgl. Bossel 2004, S.131)) berechnet. Kontinuierliche Simulationen eignen sich deshalb besonders für physikalische oder biologische Prozesse und werden sowohl im ingenieurs- als auch im naturwissenschaftlichen Umfeld verwendet. Matlab/Simulink (vgl. MathWorks 2014b; MathWorks 2014a) ist ein gängiges Werkzeug für die kontinuierliche Simulation.

Hybride Simulation Wenn das Simulationsmodell sowohl kontinuierliche als auch diskrete Eigenschaften besitzt, spricht man von einer hybriden oder diskret-kontinuierlichen Simulation. In hybriden Simulationen können sich Zustände kontinuierlich über die Zeit oder sprunghaft zu bestimmten Ereignissen ändern und werden verwendet, wenn genauere Simulationsergebnisse benötigt werden, da eine Annäherung eines kontinuierlichen an ein diskretes System Über- bzw. Unterbewertungen der Ergebnisse liefern kann (vgl. Helal 2008). Hybride Simulationen werden genutzt, um reaktive, technische Systeme zu untersuchen (vgl. Schwarz 2002). Als Beispielwerkzeug dieser Methodik sei Matlab/Simulink mit der Stateflow Erweiterung (vgl. MathWorks 2014c) genannt, die es erlaubt Zustandsautomaten oder Ablaufpläne zu modellieren, in deren Zuständen Differentialgleichungen gelöst werden.

Eine Restbussimulation basiert in der Regel auf Simulationstechniken, die eine Diskretisierung des Zustands zulassen, da auch in dieser Methodik Nachrichten erzeugt und versendet werden und dem Auftreten von Ereignissen entspricht. Da eine Restbussimulation in der Regel Daten in Echtzeit produzieren und verarbeiten soll, sind als Ausführmethodik diskrete zeitbasierte Verfahren zu bevorzugen, obgleich die Leistungsfähigkeit beeinträchtigt werden kann. Das zeitlich korrekte Ausführen hat in dieser Simulationsvariante höhere Priorität im

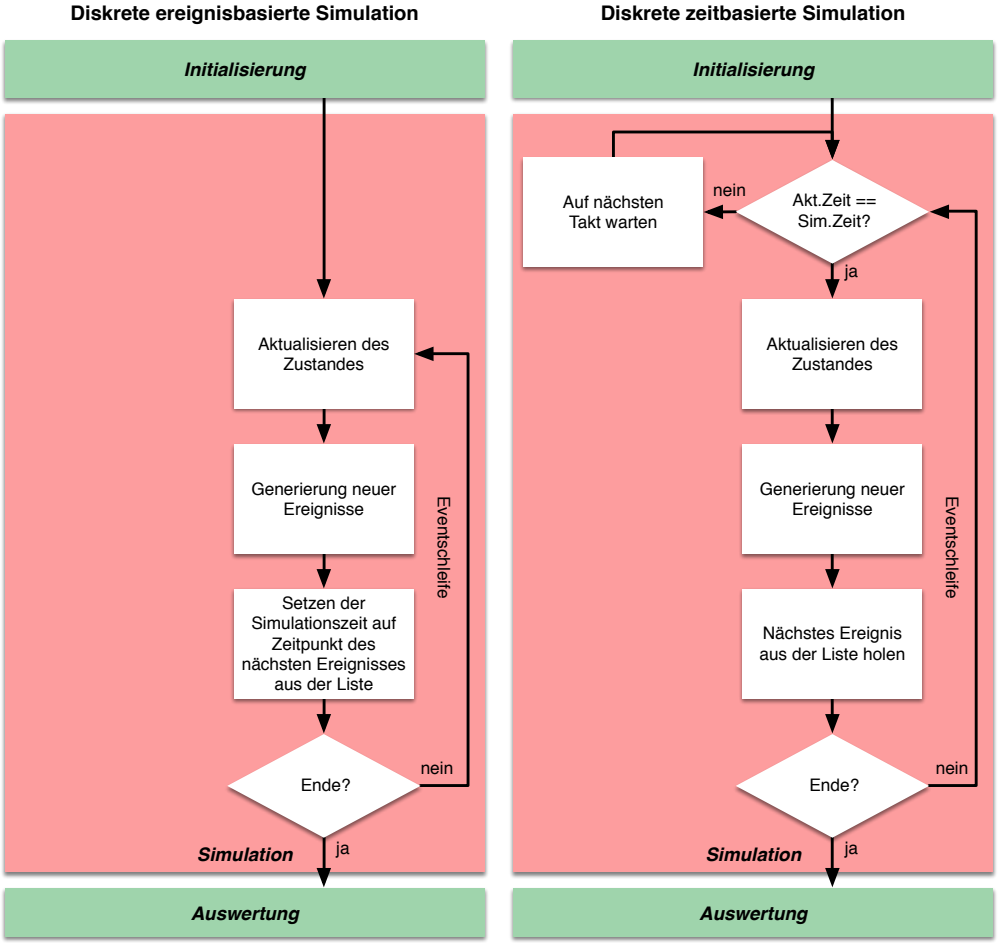


Abbildung 2.3: *Gegenüberstellung der Abläufe diskreter Simulationen (nach Steinbach 2011)*

Gegensatz zum Simulationsdurchsatz. In Abhängigkeit des zu untersuchenden Systems können zusätzlich auch kontinuierliche Aspekte einfließen, wenn z. B. zusätzlich zum Versenden von Nachrichten auch das Verhalten der Knoten simuliert werden soll. Verschiedene kommerzielle Restbussimulatoren unterstützen deshalb die Berechnung von Differentialgleichungen (siehe Abschnitt 2.4.1 auf Seite 30).

2.2.2 Grundlagen und Abgrenzung zur Hardware-in-the-Loop Simulation

Der dezentrale Entwicklungsprozess von verteilten Automotivesystemen erschwert das frühzeitige Testen der entwickelten Teilsysteme und die Unterstützung der Integration von Teilsystemen. So kann es vorkommen, dass entwickelte Systeme auf andere Systeme angewiesen sind, auf die noch nicht zugegriffen werden kann. In der Regel kann dieses Problem auftreten, wenn die benötigten Steuergeräte von einem anderen Zulieferer entwickelt werden oder wenn sie sich noch in einer Konstruktionsphase befinden und damit nicht einsatzbereit sind. Damit die entwickelten Systeme dennoch frühzeitig getestet werden können, haben sich die *Restbussimulation* und *Hardware-in-the-Loop-Simulation* als de facto Standard etabliert (vgl. Zimmermann / Schmidgall 2011; Schäuffele / Zurawka 2013), die eine Simulation der benötigten Systeme ermöglicht. In beiden Verfahren wird das reale zu testende System über die Ein- und Ausgänge mit dem Simulator verbunden und durch diesen mit Daten getriggert. Beide Simulationstechniken sind eng miteinander verbunden, sodass eine klare Grenze schwer zu ziehen ist. In der vorangeschrittenen Entwicklungsphase werden beide Verfahren außerdem in einer Mischform verwendet, sodass die Grenzen ganz verschwimmen . Nachfolgend werden beide Verfahren vorgestellt, um die Unterschiede beider Varianten zu verdeutlichen.

Restbussimulation Bei einer Restbussimulation wird das SUT, das aus einem einfachen oder mehreren vernetzen Knoten bestehen kann, mittels der realen Kommunikationsschnittstelle (in der Regel das Netzwerkinterface) mit dem Simulator verbunden. Auf dem Simulator wird anschließend das Simulationsmodell des restlichen, noch nicht vorhandenen Teils des verteilten Systems ausgeführt. Der Restbussimulator sendet während der Simulation reguläre Datenpakete, die vom SUT über den Protokollstack verarbeitet werden. Um dieses zu ermöglichen, muss er das verwendete Kommunikationsprotokoll vollständig unterstützen, indem z. B. Zeitsynchronisierungsmechanismen oder Arbitrierungsverfahren durchgeführt werden. Fehlen protokollspezifische Steuerdaten im System, verlässt das SUT den Funktionszustand und geht in der Regel in einen Fehlerzustand über. Der Simulator ist deshalb dafür verantwortlich, dass Nachrichtenpakete entsprechend der technischen Systemspezifikation der simulierten Knoten generiert werden, sodass die Simulation für das SUT transparent erfolgt. In dieser Simulationsmethodik wird das SUT lediglich mit regulären Daten stimuliert, sodass dessen Verhalten nur auf dem abstrakten Datenlevel überprüft werden kann. Dabei

werden die Werte der empfangen Datenpakete am Restbussimulator mit den Sollwerten aus der Spezifikation verglichen.

Hardware-in-the-Loop-Simulation Im Gegensatz zur reinen Restbussimulation erlaubt die Hardware-in-the-Loop-Simulation zusätzlich die Überprüfung der Reaktion vom SUT mit der Umwelt. Dazu wird die Simulationsplattform genutzt, um die Umgebungsbedingungen des SUT zu simulieren, indem die Sensorein- und Aktorausgänge des SUT mit dem Simulator verbunden werden. Damit ist es möglich sowohl bestimmte Eingangsereignisse zu erzeugen, indem elektrische Pegel an die Eingänge angelegt werden, als auch die Reaktion der Ausgänge zu überwachen. Diese Form der Simulation benötigt jedoch eine deutlich höhere Rechenleistung, da nicht nur Daten generiert werden müssen, sondern auch Umgebungsbedingungen erzeugt werden. Generell setzen sich Hardware-in-the-Loop-Simulationsumgebungen deshalb aus mehreren Teilsystemen zusammen, die synchronisiert zusammenarbeiten und Daten austauschen müssen, sodass diese Methodik komplexer in der Umsetzung im Vergleich zur Restbussimulation ist.

Wichtig zu erwähnen ist, dass aktuelle Automobile über eine Vielzahl von Systemen für unterschiedliche Anwendungen verfügen, die über die verschiedenen Bordnetze miteinander kommunizieren. Daraus folgt, dass eine hohe Komplexität durch die gesamte Menge aller Nachrichten bei der Übertragung entsteht. Während der frühen Validierungsphasen im Entwicklungsprozess, bei dem das SUT nur aus einer Komponente besteht und dessen Verhalten überprüft werden soll, ist es deshalb wichtig, die hohe Komplexität der Nachrichten zu verringern. Ansonsten kann der Test und die Konfiguration des Simulators zu viel Zeit beanspruchen und die Performance des Simulators eventuell negativ durch unnötige Berechnung von Daten beeinflusst werden. Dieses wird erreicht, indem nur die für das SUT relevanten Teile des Systems simuliert werden. Nachrichten, die weder mit dem SUT kommunizieren, noch für das Übertragungsprotokoll nötig sind, können demnach vernachlässigt werden. Dieses Prinzip ist im nachfolgenden Anwendungsbeispiel dargestellt und verdeutlicht den Zusammenhang.

2.2.3 Anwendungsbeispiel, Workflow und Einsetzbarkeit

In Abbildung 2.4 ist als Beispiel ein Teil eines Bordnetzwerks mit einer fiktiven geschwindigkeitsabhängigen Fahrwerksteuerung dargestellt, die nach Geschwindigkeit und Beschleunigung das Fahrverhalten über die Stoßdämpferhärte des Automobils beeinflusst. Die zugehörigen Sensoren und Aktoren sind in diesem Fall nicht direkt über elektrische Verbindungen mit dem Steuergerät verbunden, sondern werden mit Daten über das Bordnetzwerk angesprochen. Zu der Anwendung gehören die Stoßdämpfer vorn und hinten, sowie der Tachometer. Am Bordnetz sind weiterhin zusätzliche Steuergeräte für den Motor und Bremsen angeschlossen, die eine unabhängige Anwendung ausführen.

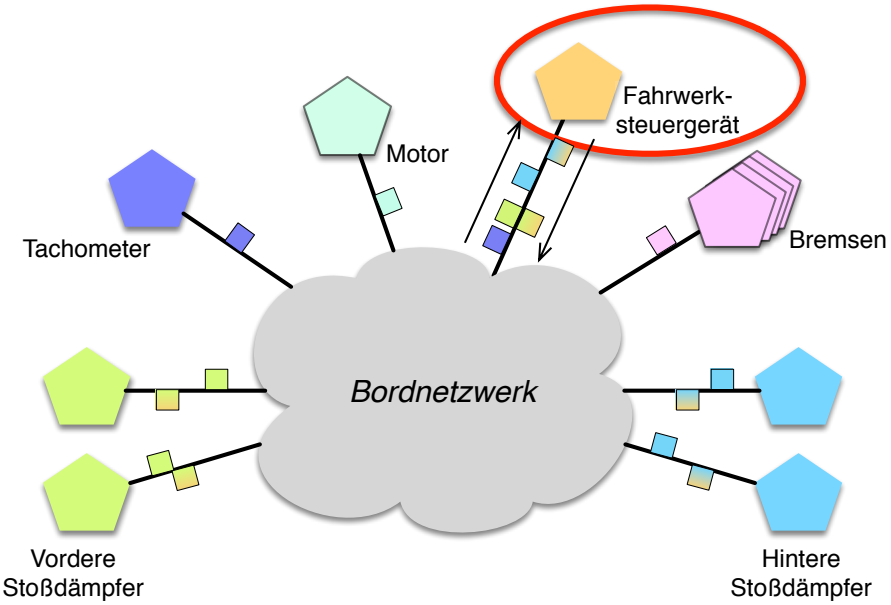


Abbildung 2.4: *Beispielanwendung einer geschwindigkeitsabhängigen Fahrwerksteuerung*

Die Stoßdämpfer senden ihren aktuellen Zustand, z. B. Hublänge des Federbeins, an das Fahrwerksteuergerät und dieses berechnet mit den Daten, die der Tachometer liefert, die neuen Dämpfereinstellungen, die über das Bordnetzwerk an die Aktoren übermittelt werden. Motor- und Bremsendaten werden in dieser fiktiven Fahrwerksanwendung nicht berücksichtigt, sodass diese Geräte keine Auswirkungen auf das Fahrwerksteuergerät haben.

Das Testen des entwickelten Fahrwerksteuergerätes wird mit Hilfe einer Restbussimulation in Abbildung 2.5 dargestellt. Dabei werden die Geräte, die unabhängig von der Anwendung sind (Motor und Bremse), nicht simuliert. Der Tachometer und Dämpfer werden im Simulator so konfiguriert, dass sich dieser wie das echte System verhält, indem originalgetreue Nachrichten vom Simulator an das Steuergerät gesendet und vom Gerät empfangen werden. Während der Testausführung können bestimmte Verhaltensmuster am Simulator konfiguriert werden, die sich in den übertragenen Daten widerspiegeln. Dabei werden die Testfälle anhand der Spezifikation des Fahrwerksteuergerätes abgeleitet, sodass eine Restbussimulation in der Regel der Blackbox-Testmethodik entspricht. Die anschließende Analyse der Ist- und Sollwerte der vom Restbussimulator empfangenen Daten bestimmt das Ergebnis des Tests.

Mit dieser Konfiguration ist es möglich, das Steuergerät auf dem abstrakten Datenlevel zu testen und zu untersuchen, ohne auf die hohe Nachrichtenkomplexität des gesamten Bordnetzwerks einzugehen, die das Auffinden von etwaigen Fehlern erschweren würde. Im

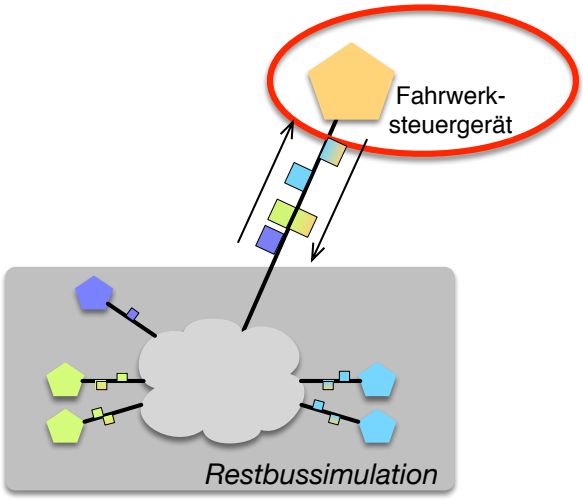


Abbildung 2.5: *Restbussimulation aller relevanten Knoten zum Testen des Steuergerätes*

Bezug zum V-Modell hängt der Einsatz der Restbussimulation vom Reifegrad des SUT ab, sodass sich dieser auf Höhe der Komponenten- oder Integrationstestphase befindet.

2.2.4 Datengenerierung innerhalb einer Restbussimulation

Bei der Ausführung des Simulationsmodells wird wie im modellbasierten Testen (vgl. Abschnitt 2.1.4 auf Seite 10) zwischen *online* und *offline* berechneten Daten unterschieden. Bei online generierten Daten, wird neben der statischen Aspekte, die die Kommunikation des Restbussimulators mit dem SUT beschreiben, zusätzlich für jeden simulierten Knoten ein Verhaltensmodell ausgeführt, das dem Verhalten des realen Systems entspricht. Die Daten, die während der Simulation über das Kommunikationsmedium versendet werden, werden durch das Verhaltensmodell berechnet. Bei offline generierten Daten werden diese vor der eigentlichen Simulationsausführung erzeugt. Die erste Variante ermöglicht eine einfachere Umsetzung reaktiv auf das Verhalten des SUT einzugehen, während die zweite Variante ein weniger komplexes Simulationsmodell voraussetzt und dadurch weniger Rechenleistung zur Simulation benötigt wird. Darüber hinaus ermöglicht die zweite Variante ebenfalls zuvor aufgezeichneten Datenverkehr (z. B. von einer Testfahrt mit Datenlogger) während der Simulation zu verwenden und so das SUT mit realen Daten zu triggern.

2.3 Echtzeit-Ethernet als Fahrzeugnetzwerk der nächsten Generation

In diesem Abschnitt wird Echtzeit-Ethernet als Fahrzeugnetzwerk der nächsten Generation vorgestellt. Zunächst wird dazu die aktuelle Situation im Fahrzeug, sowie Eigenschaften einer Echtzeit-Ethernet basierten Kommunikationsstruktur herausgestellt. Anschließend werden verwandte Umgebungen präsentiert, in denen bereits Ethernet mit einer Echtzeiterweiterung eingesetzt wird. Die Vorstellung von Time-Triggered Ethernet (TTEthernet) als eine Variante von Echtzeit-Ethernet und dessen Eigenschaften schließen diesen Abschnitt ab.

2.3.1 Etablierte Kommunikationsstrukturen im Fahrzeug

Bisherige Kommunikationsarchitekturen im Fahrzeug erreichen in naher Zukunft ihre Grenzen, da durch die Zunahme von Elektronik und Software ein stetiges Steigen der Anzahl zu übertragener Daten zu erwarten ist. Bisherige Netzwerktechnologien im Automobil können mit ihren geringen Bandbreiten diesen Nachrichtenzuwachs nicht bewerkstelligen oder können die geforderten Echtzeiteigenschaften nicht bereitstellen. Gerade in Bezug auf zukünftige Fahrerassistenzsysteme sind diese Eigenschaften sehr wichtig, da z. B. durch den gleichzeitigen Einsatz von Kameras und Laserscannern ein hohes Datenaufkommen zu erwarten ist. Die Datenübertragung muss gleichzeitig das Einhalten von Zeitaussagen erfüllen, damit eine Sensorfusion mit diesen Daten möglich ist (vgl. Kaempchen / Fuerstenberg / Dietmayer 2004).

Eine etablierte Kommunikationsstruktur eines Oberklassefahrzeugs ist in Abbildung 2.6 auf der nächsten Seite zu sehen. Hierbei handelt es sich um ein heterogenes Netzwerk verschiedener Systeme mit verschiedenen Anforderungen, die in den fünf vorgestellten Anwendungsdomänen (siehe Abschnitt 2.1.1 auf Seite 5) Verwendung finden. Anwendungen, die für die Sicherheit der Passagiere sorgen oder sich im Antriebs- bzw. Fahrwerkstrangumfeld befinden sind rot bzw. orange gekennzeichnet und zeichnen sich, wie erwähnt, durch strikte Echtzeitanforderungen aus. Gelb bzw. hellgrün kennzeichnet Anwendungen aus dem Komfortbereich mit weichen Echtzeitanforderungen. Multimediaanwendungen sind in diesem Bild dunkelgrün dargestellt; grau/schwarz definieren den Diagnosebereich. Ein zentrales Gateway sorgt dafür, dass domänenübergreifend Nachrichten zwischen verschiedenen Netzwerktechnologien ausgetauscht werden können. Generell ist die aktuelle Struktur durch vier Hauptnetzwerktechnologien realisiert (vgl. Zimmermann / Schmidgall 2011), die im folgenden kurz vorgestellt werden.

LIN Das Bussystem *Local Interconnect Network (LIN)* (vgl. Grzemba / Wense 2005) wird eingesetzt, um kostengünstig einfache Sensor-/Aktor-Anwendungen, wie z. B. elektrische Fensterheber, zu ermöglichen, bei denen eine geringe Bandbreite (LIN ist mit

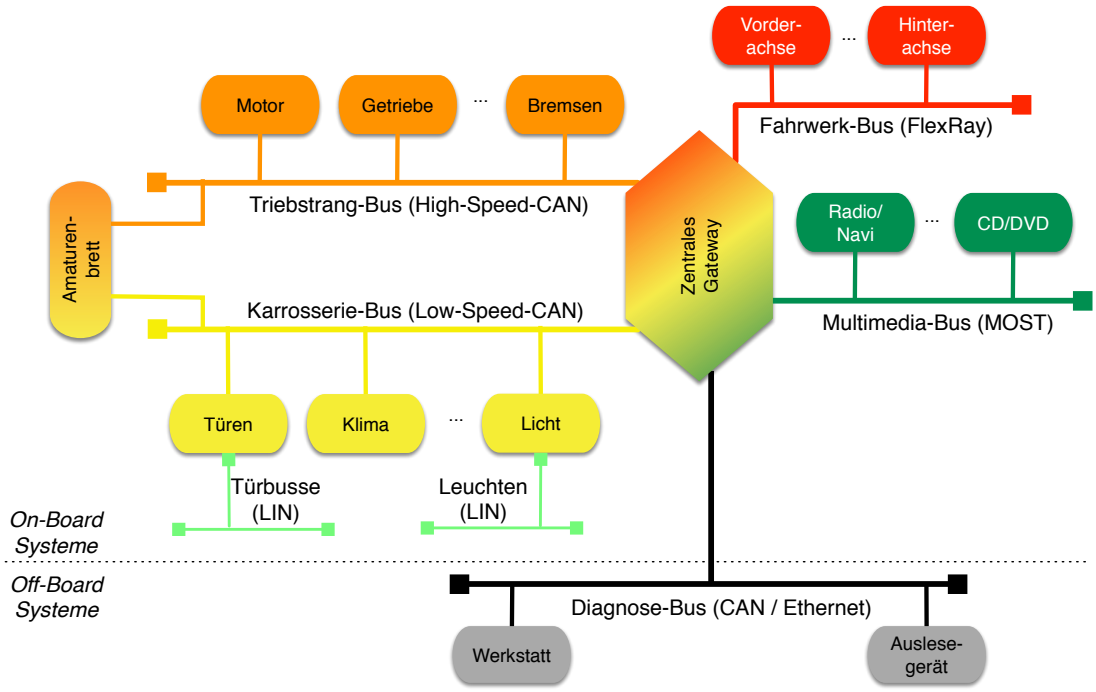


Abbildung 2.6: *Etablierte Netzwerkarchitektur in aktuellen Fahrzeugen (nach Zimmermann / Schmidgall 2011)*

2,4 kbit/s, 9,6 kbit/s und 19,2 kbit/s spezifiziert) ausreicht. Dabei arbeitet LIN nach dem Master-/Slave-Prinzip, bei dem der Master die Kommunikation treibt, indem dieser eine Anfrage sendet und genau ein Slave mit dem Datenpaket antwortet. LIN wird oft im Zusammenspiel mit einem CAN-Bus verwendet, sodass der LIN-Master als Gateway zwischen beiden Bussen fungiert und Nachrichten weiterleitet.

MOST *Media Oriented Systems Transport (MOST)* (vgl. Grzemba 2007) ist ein Bussystem, das für Multimediaanwendungen konzipiert worden ist, um speziell den steigenden Bedarf an Bandbreite dieser Domäne gerecht zu werden. So ist MOST mit 25 Mbit/s, 50 Mbit/s und 150 Mbit/s spezifiziert, jedoch ohne garantierte Zusagen der Paketlaufzeiten. Ein MOST-Netz wird in der Regel als Ring aufgebaut, indem Daten unidirektional und zyklisch von einem zum nächsten Knoten geleitet werden. Ein Timingmaster sendet regelmäßig spezielle Nachrichten, auf die sich alle Teilnehmer synchronisieren.

CAN *Controller Area Network (CAN)* (vgl. Wolfhard / Obermöller 2011; International Organization for Standardization 2003) ist das seit 1991 am häufigsten eingesetzte Bussystem in der Automobilindustrie mit üblichen Bandbreiten von 125 kbit/s (Low-Speed

CAN) bis zu 500 $^{kbit}/_{s}$ (High-Speed CAN). Es ermöglicht eine priorisierte Datenübertragung, bei der die maximale Latenz der höchstprioren Nachricht begrenzt ist, sodass es für Echtzeitanwendungen geeignet ist. Die Priorisierung wird anhand des Nachrichtenidentifiers durchgeführt, bei der die niedrigste ID die höchste Priorität hat. Die Übertragung erfolgt ereignisorientiert per CSMA/CR (Carrier Sense Multiple Access / Collision Resolution), bei der jeder Teilnehmer auf den Bus zugreifen kann und während der Arbitrierungsphase entschieden wird, ob er senden darf. Die Erweiterung TTCAN erlaubt eine zeitgesteuerte Übertragung von Nachrichten, bei der auch die Latenz niederpriorer Nachrichten begrenzt ist. Allerdings hat sich TTCAN durch die zu geringe Bandbreite nicht durchsetzen können und wurde durch FlexRay ersetzt (vgl. Zimmermann / Schmidgall 2011).

FlexRay *FlexRay* (vgl. Rausch 2008) ist ein zeitgesteuertes Bussystem, welches speziell für Automotiveanwendungen (*X-by-Wire* - z. B. bremsen oder lenken per Kabel) entwickelt worden ist und ermöglicht sowohl ereignis-, als auch zeitorientierte Übertragung, sodass es sich für die Verwendung von Echtzeitanwendungen eignet, bei der die maximale Nachrichtenlatenz begrenzt sein muss. Im Gegensatz zu den bereits erwähnten Systemen erlaubt FlexRay einen Betrieb mittels zweier getrennter Kanäle, bei der jedem Kanal 10 $^{Mbit}/_{s}$ zu Verfügung stehen. Die Übertragung von zeitgesteuerten Daten erfolgt in statisch definierten Bereichen im Übertragungszyklus, während ereignisbasierte Daten in optionalen dynamischen Bereichen im Zyklus übertragen werden. Als Topologie eignen sich beim FlexRay-System sowohl Bus- als auch Sternvarianten.

Wie in Abbildung 2.6 auf der vorherigen Seite zu sehen ist, bedarf es eines Gateways, um domänenübergreifend Daten verschicken zu können. Daten müssen von einer Technologie in eine andere übersetzt werden, was das Bordnetzwerk nicht nur komplex macht, sondern auch zusätzliche Verzögerungen erzeugt, wodurch die Echtzeitfähigkeit einiger Anwendungen beeinträchtigt werden kann. Richard Bogenberger von der BMW Group Forschung und Technik sagte bereits 2008:

> *„Eigentlich ist das Bordnetz im Gesamtfahrzeug bereits heute nicht mehr vernünftig zu beherrschen“* (Badstübner 2008)

Diese Aussage resultiert aus der Eigenschaft, dass das Bordnetzwerk, die Anwendung und insbesondere deren Echtzeitfähigkeit auf korrekte Funktion mittels formaler Analyse verifiziert werden müssen (vgl. Schäuffele / Zurawka 2013). Außerdem muss das Versenden von Nachrichten und der zugehörige Nachrichtenschedule bei zeitgesteuerten Netzwerktechnolgien konfiguriert werden, um keine Nachrichtenkollisionen zu erzeugen. Gateways erschweren diese Analyse und die Konfiguration dieser Netzwerke, da die zusätzlichen Verzögerungen des Gateways mit in der Analyse betrachtet werden müssen. Daraus ergibt sich eine längere Entwicklungszeit, die höhere Kosten verursacht.
Ein weiteres Problem der aufgelisteten Netzwerktechnolgien ist, dass keines ausreichend

Bandbreite bei gleichzeitiger Bereitstellung von Echtzeitfähigkeit für zukünftige Anwendungen zur Verfügung stellen kann. So stellt MOST zwar hohe Bandbreiten zur Verfügung, kann aber, bedingt durch die Topologie, keine garantierten Aussagen zum zeitlichen Verhalten machen. FlexRay auf der anderen Seite ist mit 10 $^{\text{Mbit}}/_{\text{s}}$ pro Kanal in der Bandbreite sehr stark eingeschränkt, sodass es sich hauptsächlich für regelungstechnische Anwendungen eignet.

Damit auch zukünftige Anwendungen im Automobil Einzug finden können, bedarf es einer neuen Technologie, die die hohe Komplexität beherrscht, gleichzeitig hohe Bandbreiten zur Verfügung stellt und Echtzeitkommunikation erlaubt. Ethernet mit einer Echtzeiterweiterung als Basis für das Bordnetz der Zukunft, ist ein geeigneter Kandidat, die genannten Problemstellungen zu lösen (vgl. Badstübner 2008; Steinbach / Lim / Korf u. a. 2012).

2.3.2 Verwendung von Echtzeit-Ethernet in verwandten zeitkritischen Umgebungen

Ethernet als Netzwerkprotokoll ist in seiner Standardausführung kein echtzeitfähiges Übertragungsprotokoll, da es alle Datenströme fair behandelt und keine Priorisierung bei der Weiterleitung der Nachrichten im Switch durchführt. Durch die Erweiterung 802.1Qp (vgl. IEEE 802.1 TSN Task Group [a]) wird diese zwar eingeführt, sodass weiche Echtzeiteigenschaften für hochpriore Nachrichten erreicht werden, jedoch werden Daten weiterhin asynchron ohne Rücksicht auf andere Teilnehmer versendet. Nachrichten anderer Teilnehmer mit gleicher Priorität können im schlechtesten Fall bei der Weiterleitung im Switch blockiert werden.

Die Herausforderung Ethernet mit harter Echtzeitfähigkeit zu erweitern besteht darin, zeitkritische und unkritische Nachrichten unterscheiden zu können. Weiterhin müssen die Zeitkritischen priorisiert werden, um ein gleichzeitiges Übertragen dieser am gleichen Port im Switch zu verhindern. Deterministisches Verhalten kritischer Daten kann ohne diese drei Eigenschaften nicht erzielt werden. Verschiedene echtzeitfähige Ethernet-Protokolle verfolgen voneinander abweichende Strategien, diese beiden Eigenschaften zu erfüllen und lassen sich in drei Klassen aufteilen:

Token basierte Netzwerke sind über einen virtuellen Token synchronisiert und erlauben es nur demjenigen Knoten eine Übertragung zu starten, der momentan diesen Token besitzt. Der Token wird an jeden Teilnehmer im Netz weitergereicht, sodass eine faire Übertragung für jeden Teilnehmer erreicht wird. Allerdings ist der Verlust des Tokens, die Verlusterkennung und dessen Neuerzeugung ein komplexer Vorgang. Dadurch, dass das Versenden der Nachrichten nur auf Endsystemen durchgeführt wird, können in dieser Variante standard Ethernet Switches verwendet werden. Weiterhin erhöht sich die Zykluszeit, indem der Token weitergereicht wird mit der Anzahl sendender Teilnehmer,

sodass die Übertragungsrate, Latenz und Jitter eine direkte Korrelation zu dieser Anzahl hat. Erfolgreich eingesetzt wird diese Strategie in der Prozessautomatisierung im Protokoll *EtherCAT* (**etg-ef-rtefa-09**).

Bandbreiten limitierende Protokolle sind unsynchronisierte Echtzeit-Ethernet Varianten und ermöglichen eine echtzeitfähige Übertragung, indem die Bandbreite für jeden Knoten festgelegt wird und diese nicht überstiegen werden darf. Somit wird sichergestellt, dass jeder Teilnehmer im Netz genügend Bandbreite zur Verfügung hat, um Nachrichten zu versenden. Da in dieser Variante die Switches dafür Sorge tragen, dass Teilnehmer nur in ihren Bandbreitenkontigenten Nachrichten übertragen, müssen spezielle Switches verwendet werden, die das Sendeintervall überprüfen. Durch die asynchrone Übertragung ergeben sich rechnerisch höhere Latenzen und Jitter als bei zeitgesteuerten Protkollen, da hier immer vom schlechtesten Fall ausgegangen wird, indem alle Sender gleichzeitig senden. Vorteilhaft ist, dass keine komplexen Verfahren zur Synchronisierung durchgeführt werden müssen. AFDX *Avionics Full Duplex (X) Switched Ethernet (AFDX)* (vgl. Aeronautical Radio Incorporated 2002; Aeronautical Radio Incorporated 2009) nutzt dieses Prinzip und wird derzeit im Airbus u. a. A380 und in der Boeing 787 verwendet.

Zeitgesteuerte Protokolle verwenden einen zeitlich koordinierten Zugriff auf das Übertragungsmedium (Time Devision Multiple Access), das durch eine gemeinsame synchronisierte Zeitbasis aller Teilnehmer erreicht wird. Jedem Teilnehmer (Endsysteme und Switches) ist es nur zu bestimmten Zeitpunkten erlaubt, eine Übertragung zu starten und Nachrichten weiterzuleiten. Deshalb ist es nötig, dass alle Teilnehmer eine möglichst genaue und synchronisierte Uhr besitzen, damit sie immer in ihren zugewiesenen Timeslot senden. Im Vergleich zu unsynchronisierten Varianten erlauben zeitgesteuerte Protokolle geringere Latenzen und sehr geringe Jitter, da genau festgelegt wird, zu welchem Zeitpunkt eine Nachricht versendet wird. Allerdings müssen hier sowohl Endsysteme, als auch Switches das Protokoll vollständig unterstützen. Es muss ein Sychronisationsmechanismus vorhanden sein, mit dem es möglich ist, eine globale synchronisierte Zeitbasis für alle Teilnehmer zu erreichen, wodurch ein geringer Nachrichtenoverhead entsteht. Als Beispiele seien *Profinet* (vgl. PROFIBUS & PROFINET International; Schnell / Wiedemann 2008), das u. a. in der Prozessautomatisierung eingesetzt wird und TTEthernet (vgl. Steiner 2008) genannt.

Eine weitere Variante von Echtzeit-Ethernet, dessen Einsatz im Automobil untersucht und evaluiert (vgl. Steinbach / Lim / Korf u. a. 2012; Lim / Herrscher / Waltl u. a. 2012; Kuther 2011) wird, ist das IEEE802.1 *Audio-Video-Bridging (AVB)* (vgl. IEEE 802.1 AVB Task Group), das seinen Ursprung in der Vernetzung von Rundfunktechnik hat. AVB ist ebenfalls ein Bandbreiten limitierendes Protokoll und garantiert für alle Netzwerktopologien eine maximale Latenz für Nachrichten der höchsten Priorität. Zum Bearbeitungszeitpunkt dieser Arbeit wird durch die Time-Sensitive-Networking Task-Group eine erweiterte Spezifikation

für AVB-Ethernet erarbeitet, die eine zeitgesteuerte Übertragung von statisch konfigurierten Nachrichten ermöglicht(vgl. IEEE 802.1 TSN Task Group [b]).

2.3.3 Grundlagen zu Time-Triggered Ethernet

In dieser Arbeit wird Time-Triggered-Ethernet als Protokollvariante von Echtzeit-Ethernet betrachtet, da es in der CoRE-Forschungsgruppe (vgl. CoRE-Arbeitsgruppe [a]) in verschiedenen Projekten untersucht und verwendet wird. Es zählt sowohl zu den zeitgesteuerten, als auch zu den bandbreitenlimitierenden Protokollen. Time-Triggered Ethernet (TTEthernet) (vgl. Steiner 2008) wurde von TTTech (vgl. TTTech Computertechnik AG) in Zusammenarbeit mit Honeywell (vgl. Honeywell International), basierend auf Forschungsergebnissen der TU Wien (vgl. Kopetz / Ademaj / Grillinger u. a. 2005), entwickelt und ist durch die *Society of Automotive Engineers* als AS6802 standardisiert (vgl. SAE AS-2D Committee 2011).

TTEthernet ermöglicht es Nachrichten mit harten Echtzeitanforderungen und normale Nachrichten über das gleiche physikalische Medium zu übertragen. Damit dieses möglich ist, werden drei Nachrichtenklassen definiert, die unterschiedliche Prioritäten, Übertragungsprinzipien und Zeitanforderungen aufweisen.

Time-Triggered (TT)-Nachrichten besitzen die höchste Nachrichtenpriorität und erfüllen die strengsten Zeitanforderungen und werden zeitgesteuert versendet und empfangen. Dazu wird ein offline generierter und berechneter Schedule auf jedem Teilnehmer (Endsysteme und Switches) ausgeführt, indem jeder konfigurierter Zeitpunkt zur Übertragung und zum Empfang einer Nachricht genau ausgeführt wird. Mit dieser offline generierten Konfiguration kann ein präzises deterministisches Verhalten erzielt werden, das eine begrenzte Latenz und minimalen Jitter zur Folge hat. TT-Nachrichten eignen sich daher sehr gut für die Übertragung von kritischen Steuerdaten.

Rate-Constrained (RC)-Nachrichten werden über eine Limitierung der Bandbreite gesteuert und entsprechen der AFDX-Protokollspezifikation Arinc-664 (vgl. Aeronautical Radio Incorporated 2009). Nachrichten dieser Klasse werden garantiert übertragen, allerdings mit geringerer Priorität als *TT-Nachrichten*. Die Bandbreitenlimitierung wird mittels *Bandwidth-Allocation-Gaps (BAG)* erreicht, die die minimale Zeitspanne zweier aufeinanderfolgender Nachrichten der gleichen Art definiert. Nachrichten, die diese Einschränkung nicht erfüllen, können im Switch verworfen werden. Im Gegensatz zu TT-Nachrichten erfolgt die Übertragung asynchron zum konfigurierten Schedule, wodurch die Latenz variieren kann. Allerdings können dessen Maximalwerte analytisch berechnet werden (vgl. Charara / Scharbarg / Ermont u. a. 2006), sodass RC-Nachrichten gut für asynchrone Echtzeitkommunikation geeignet sind.

Best-Effort (BE)-Nachrichten entsprechen standard Ethernet und sind am niedrigsten priorisiert. BE-Nachrichten werden versendet, wenn keine Übertragung von TT-Nachrichten ansteht und keine RC-Nachricht im Buffer des Senders vorhanden ist. Sie verbrauchen damit die restliche vorhandene Bandbreite und eine garantierte Nachrichtenübertragung ist nicht möglich. Auch BE-Nachrichten werden asynchron übertragen aber für unkritischen Verkehr verwendet.

Die nachfolgende Abbildung 2.7 verdeutlicht das Übertragungsprinzip von TTEthernet. In diesem Beispiel sendet das Fahrwerk zyklisch TT-Nachrichten, die Lichtsteuerung RC-Nachrichten und das Multimediasystem BE-Nachrichten zum gleichen Zeitpunkt an einen TTEthernet-Switch, der alle Nachrichtenklassen integriert und weiterleitet.

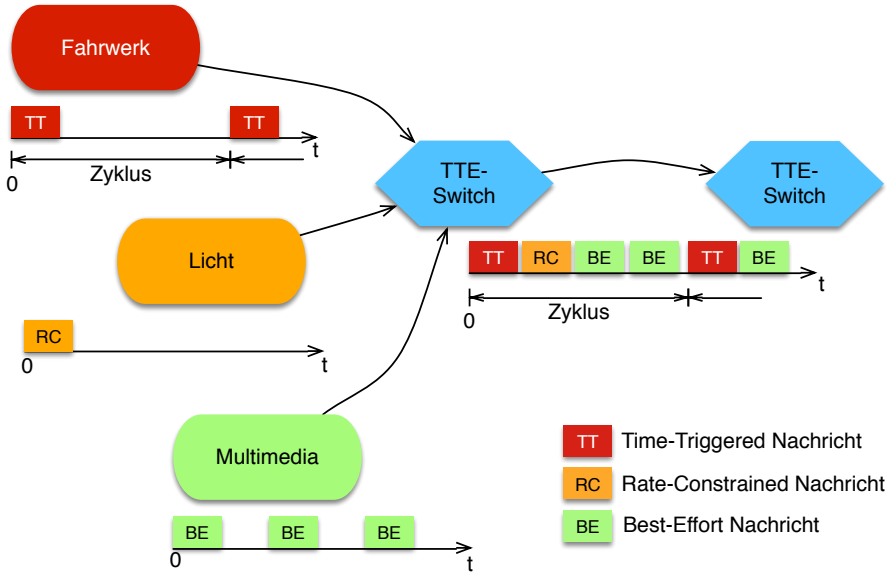


Abbildung 2.7: *Integration der Nachrichtenklassen im TTEthernet (nach Steinbach 2011)*

TT-Nachrichten werden auf dem kompletten Datenpfad von allen Teilnehmern zeitgesteuert übertragen. In der Abbildung 2.7 leitet der Switch sie gemäß des konfigurierten Schedules priorisiert weiter. RC-Nachrichten werden mit der zweithöchsten Priorität nach der zeitgesteuerten Übertragung der TT-Nachrichten weitergeleitet und zuletzt BE-Nachrichten, die die restliche Bandbreite nutzen können.

TT- und RC-Nachrichten stellen in TTEthernet die Echtzeitnachrichten dar und werden daher auch Critical-Frames[1] (CT-Frames) genannt. Damit CT-Frames von BE-Nachrichten

[1] Critical - hier Zeitkritisch

unterschieden werden können, bedarf es einer Möglichkeit Nachrichten mit einer Kennung zu versehen. In TTEthernet wird dieses erreicht, indem die Zieladresse des Ethernetframes nicht als Ziel, sondern als Identifier einer Nachricht verwendet wird, die zur Designzeit konfiguriert wird. Das sechs Byte große Zieladressfeld (siehe Abbildung 2.8) wird in ein CT-Marker und in eine CT-ID unterteilt. Dabei gibt der CT-Marker vor, ob es sich um CT-Frames handelt. Die CT-ID unterscheidet zwischen unterschiedlichen Nachrichten. TT- und RC-Nachrichten unterscheiden sich deshalb nur anhand ihr CT-ID, sodass ein Switch oder ein Endsystem entscheiden muss, mit welcher Strategie die Nachricht behandelt wird.

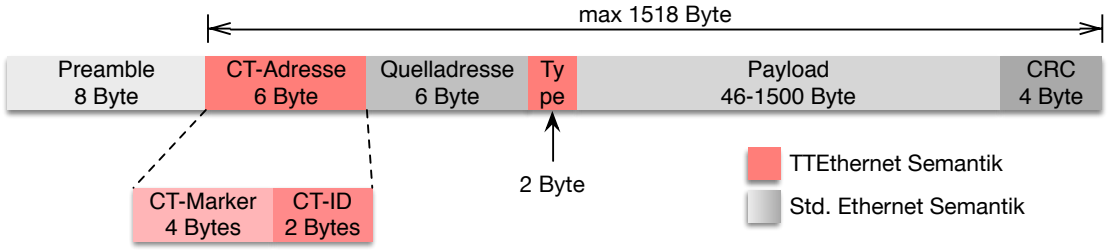


Abbildung 2.8: *Identifizierung der Nachrichtenklasse durch Interpretation der Zieladresse*

Eine weitere Besonderheit von TTEthernet-Netzwerken ist die Verwendung von statisch definierten Routen von CT-Nachrichten. Hierbei greift TTEthernet das Prinzip der *virtuellen Links (Vlink)* auf, die im Arinc664-Standard definiert wurden. Vlinks werden zusammen mit den CT-IDs zur Designzeit konfiguriert und beschreiben einen virtuellen Pfad durch das Netzwerk, die durch genau einen Sender und einer beliebigen Anzahl von Empfängern gekennzeichnet ist. Jede CT-ID wird genau einem virtuellen Link zugeordnet, sodass sich durch dieses Prinzip leicht auch eine Gruppenkommunikation erzielen lässt. Durch den Einsatz der statisch definierten Routen lassen sich für kritische Nachrichten nach oben begrenzte Latenzen und damit ein sehr deterministitsches Verhalten erreichen.

Damit eine zuverlässige Übertragung von TT-Nachrichten mit geringem Jitter bei allen Teilnehmern möglich ist, muss eine gemeinsame Zeitbasis vorhanden sein. TTEthernet definiert daher einen Zeitsynchronisierungsprozess, der transparent zum Übertragungsprotokoll arbeitet und keine zusätzliche Hardware erfordert. Es werden spezielle Synchronisationsnachrichten *Protocol Control Frames* (PCF) versendet, die auf Endsystemen und Switches verarbeitet werden. Synchronisationsnachrichten basieren auf RC-Nachrichten, sodass auch hier virtuelle Links den Nachrichtenpfad beschreiben und dementsprechend statisch definiert werden.
Endsysteme und Switches werden in verschiedene Rollen eingeteilt, die unterschiedliche Aufgaben während der Synchronisation ausführen. In Abbildung 2.9 auf der nächsten Seite ist der Synchronisationsprozess dargestellt. Im ersten Schritt senden die *Synchronization Master*

(SM), ihre aktuelle Zeit zum *Compression Master (CM)*. Dieser berechnet eine Durchschnittszeit aus allen Werten und sendet diese im zweiten Schritt an die *Synchronization Clients (SC)* und *Synchronization Master*, die sich anschließend auf die neue Zeit synchronisieren.

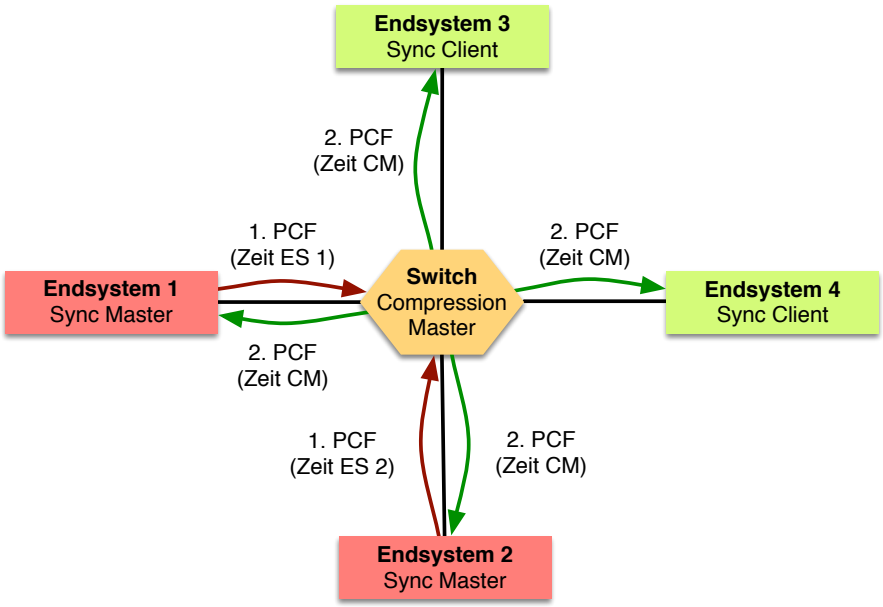


Abbildung 2.9: *Zweiwegezeitsynchronisierung im TTEthernet*

Als Sonderform kann eine Zeitsynchronisation auch mit nur einem Synchronization Master ermöglicht werden. In diesem Fall kann auf den Compression Master verzichtet werden, da es nur eine Zeitbasis gibt. Alle Teilnehmer im Netz führen dann die Rolle als Synchronization Client aus und synchronisieren sich auf die Zeit des Masters. Dieses Prinzip eignet sich besonders für eine Restbussimulation, da Steuergeräte in der Regel die Rolle eines Synchronization Client übernehmen, sodass der Restbussimulator als Synchronization Master fungiert.

2.3.4 Eigenschaften einer Echtzeit-Ethernet-basierten Kommunikationsstruktur

Echtzeit-Ethernet basierte Kommunikationsstrukturen im Automobil können durch die hohen verfügbaren Bandbreiten von Ethernet dafür sorgen, dass aktuelle Bandbreitenprobleme gelöst werden können. Durch die Echtzeiteigenschaften können zudem Anwendungen umgesetzt werden, die hohe Bandbreiten bei gleichzeitigem Echtzeitverhalten voraussetzen.
Durch die Verwendung einer einzigen Netzwerktechnologie ist es weiterhin möglich, eine

flache Netzwerktopologie ohne zusätzliche Gateways zu erzeugen. Die zusätzlich entstehenden Verzögerungen bei der Weiterleitung der Daten von einer in eine andere Technologie lassen sich einsparen, sodass das Netzwerk insgesamt eine höhere Datendurchsatzrate zulässt. Außerdem wird die Komplexität während der Designphase verringert. Das Entfallen der Gateways und das einheitliche Übertragungssystem erlauben eine schnellere und einfachere Verifikation des Zeitverhaltens, da kein Gatewayverhalten modelliert werden muss. Entwicklungszeiten lassen sich damit verkürzen.

Eine mögliche *flache* Echtzeit-Ethernet Netzwerkstruktur im Fahrzeug ist in Abbildung 2.10 dargestellt. Im Vergleich zur aktuellen Struktur (vgl. Abbildung 2.6 auf Seite 21) fällt auf, dass keine Gateways mehr vorhanden sind, die zusätzliche Verzögerungen hervorrufen können. Alle Teilnehmer befinden sich im gleichen physikalischen Netzwerk, die über Echtzeit-Ethernet Switches miteinander verbunden sind. Durch die Möglichkeit der Priorisierung der Nachrichten ist es außerdem möglich, dass sämtliche Kommunikation über das gleiche Backbone-Netzwerk durchgeführt wird. Anwendungen, die zyklischen Datentransfer mit hohen Echtzeitanforderungen aufweisen (rot), können über zeitgesteuerte Nachrichten übertragen werden (TT-Nachrichten). Ereignisbasierte Kommunikation (gelb) kann mittels RC-Nachrichten weiterhin unter Berücksichtigung der Echtzeitanforderungen versendet werden. Multimediaanwendungen können kosteneffizient über das gleiche physikalische Netzwerk mittels BE-Nachrichten kommunizieren. Ein Nachrichtenverlust in der Multimediadomäne kann durch höhere Anwendungsprotokolle (z. B. TCP) kompensiert werden.

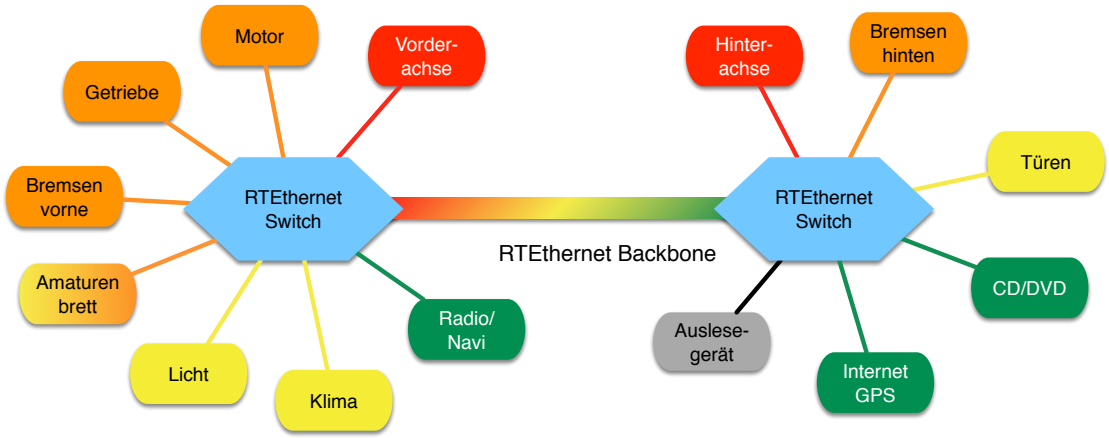


Abbildung 2.10: *Ein flaches Echtzeit-Ethernet-basiertes Fahrzeugnetzwerk der nächsten Generation*

Die Eigenschaften einer Ethernet basierten Kommunikationsstruktur im Fahrzeug ist durch Till Steinbach in seiner Masterthesis (vgl. Steinbach 2011; Steinbach / Korf / Schmidt

2011) mit einer Softwaresimulation evaluiert worden. Dazu wurde für die ereignisbasierte Simulationsumgebung OMNeT++ ein Simulationsmodell entwickelt, mit dem es möglich ist, Echtzeit-Ethernet-Netzwerke zu simulieren. Das Simulationsmodell implementiert das TTEthernet-Protokoll. Die erzielten Ergebnisse zeigen, dass Echtzeit-Ethernet-Netzwerke im Automobil die Zeit- und Bandbreitenanforderungen einer zukünftigen kamerabasierten Fahrerassistenzanwendung erfüllen. Da in dieser Arbeit eine 100 $^{\text{Mbit}}/_{\text{s}}$ Ethernetverbindung evaluiert wurde, kann davon ausgegangen werden, dass durch die Verwendung von 1 $^{\text{Gbit}}/_{\text{s}}$ oder 10 $^{\text{Gbit}}/_{\text{s}}$ Verbindungen, Echtzeit-Ethernet auch für zukünftige Anforderungen sehr gut gerüstet ist.

2.4 Vorstellung verwandter Arbeiten

Der folgende Abschnitt stellt verwandte und vergleichbare Arbeiten vor. Zuerst werden produktiv eingesetzte Restbussimulatoren verschiedener Hersteller vorgestellt, gefolgt von wissenschaftlichen Arbeiten und Arbeiten zum modellbasierten Testen. Die Abgrenzung zu dieser Arbeit schließt diesen Abschnitt ab.

2.4.1 Überblick produktiv eingesetzter Restbussimulatoren

Gerade im produktiven Umfeld ist die Entwicklung von Restbussimulatoren ein attraktiver Bereich für Werkzeughersteller aus der Automotivesoftwareentwicklung, sodass sich viele Produkte am Markt etabliert haben, die jeweils unterschiedliche Eigenschaften aufweisen und verschiedene Protokolle unterstützen. Der grundlegende Aufbau einer Restbussimulation ist bei allen Produkten ähnlich und ist in der Abbildung 2.11 auf der nächsten Seite abgebildet. In der Regel besteht eine Restbussimulation aus drei Hauptkomponenten: Ein Host-Computer zur Konfiguration, dem Restbussimulator und dem System-Under-Test. Der Host-Computer ist eine normale, basierte Workstation mit entsprechender Konfigurationssoftware. Die Simulationsplattform ist mit einem echtzeitfähigem Betriebssystem versehen oder besteht aus einem Mikrocontroller, auf dem die Restbussimulation mit Echtzeiteigenschaften ausgeführt wird. Das SUT wird über die verwendete Netzwerktechnologie mit dem Simulator verbunden. Die Produkte unterscheiden sich in der Regel durch die Konfiguration, Software und verwendete Hardware. Außerdem ist die Unterstützung der Fahrzeugnetzwerksysteme produktspezifisch. Nachfolgend sind zwei Produkte beschrieben:

CANoe ist ein Tool des Herstellers Vector Informatik (vgl. Vector Informatik 2013). Es ist flexibel konfigurierbar und in der Entwicklung von Steuergeräten weit verbreitetet in der Automobilindustrie. Bei der Konzeption des Netzwerks kann im vornherein mittels

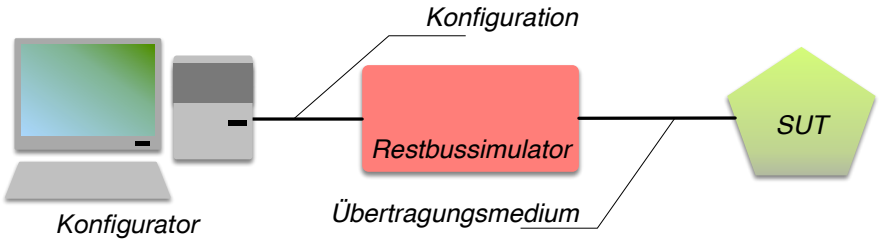


Abbildung 2.11: *Genereller Aufbau einer Restbussimulation*

einer Kommunikationsmatrix[2] und einer reinen Softwaresimulation festgestellt werden, ob das entworfene Netzwerk den Anforderungen entspricht.
Als Modellierungswerkzeug/-Schnittstelle der simulierten Knoten kann auf MATLAB und dessen Erweiterungen Simulink und Stateflow (vgl. Neuffer / Böke 2010; Swager 2008) oder auf die selbst entwickelte Programmiersprache CAPL zurückgegriffen werden, die die Daten online während des Betriebes erzeugen. Das Werkzeug eignet sich deshalb sowohl diskrete als auch kontinuierliche Systeme zu simulieren. Zu Diagnosezwecken kann es während des Betriebes des realen Netzwerks eingesetzt werden, um Daten zu sammeln und Signale während des Betriebes zu manipulieren. CANoe erlaubt es verschiedene Netzwerksysteme mittels angepassten Hardwarekomponenten zu unterstützen. Derzeit werden als Bussysteme CAN, LIN, Most, FlexRay und standard Ethernet unterstützt.

FlexConfig & FlexXCon midget wird vom Hersteller Eberspächer entwickelt (vgl. Eberspächer 2014) und ist eine Hardware-/Softwareplattform, die die Bussysteme CAN und FlexRay für eine Restbussimulation unterstützt. Zusätzlich kann sie verwendet werden, verschiedene Gatewayarchitekturen, z. B. zwischen zwei gleichen oder unterschiedlichen Bussystemen, zu implementieren. Die getrennten Netzwerke können auf diese Weise miteinander verbunden werden (vgl. Gugenhan / Schmidt 2011). Weiterhin kann es auch während der Diagnose von realen Netzwerken verwendet werden und ermöglicht ebenfalls eine Manipulation der Daten während der Laufzeit. Zur Konfiguration des Simulationsmodells steht dem Entwickler Ansi-C zur Verfügung, durch das das Verhalten der simulierten Knoten entwickelt werden kann.

Eingereichte und zugesprochene Patente, die eine Restbussimulation beschreiben, verdeutlichen das Interesse der Industrie, Systeme frühzeitig im Entwicklungsprozess zu testen. Nachfolgend werden zwei Patente zum Thema Restbussimulation vorgestellt.

[2] Eine Kommunikationsmatrix beschreibt die Kommunikation zwischen den verschiedenen Geräten und die verwendeten Nachrichten

Patent BMW Die Bayrische Motoren Werke AG hat eine Restbussimulation für FlexRay-Systeme entwickelt (vgl. Häfen / Fries 2008). Dieser Ansatz beschreibt einerseits den bereits angesprochenen generischen Aufbau der Restbussimulation mittels einer standard Konfigurations- und einer echtzeitfähigen Simulationsplattform. Andererseits wird beschrieben, mit welchen Hilfsmitteln die Restbussimulation konfiguriert wird. Besonders zu erwähnen ist, dass die Konfiguration durch ein standardisiertes Datenformat (Fieldbus Data Exchange Format - FIBEX (vgl. ASAM 2013)) durchgeführt wird, das die Auswahl der zu simulierenden Knoten ermöglicht. Der Ansatz verdeutlicht, dass nicht alle im Netzwerk konfigurierten Nachrichten zu simulieren sind, sondern nur die für das SUT relevanten.

Patent Honeywell In dem Ansatz von Honeywell wurde das Testen und Analysieren von verteilten Time-Triggered Ethernet Komponenten ermöglicht (vgl. Smithgall / Hall / Varadarajan 2013), was einer Restbussimulation entspricht. Hier wird die Implementierung einer Testplattform beschrieben, mit der einzelne Komponenten bzw. Subsysteme zu testen sind, ohne die physikalische Topologie inklusive aller Komponenten eines bestehenden Netzwerks zu verändern. Dieses wird erreicht, indem der Nachrichten Schedule der Testplattform mit einem minimalen Versatz vor dem Schedule des bestehenden Netzes (150 µs bei 100 $^{\text{Mbit}}/_{\text{s}}$) ausgeführt wird. Mit dieser Eigenschaft empfängt das SUT TT-Nachrichten vom Restbussimulator früher und verwirft die später eintreffenden Nachrichten des bestehenden Netzwerks. Eine weitere Eigenschaft dieses Ansatzes ist, dass für jede simulierte Nachricht ein Netzwerkinterface benutzt werden muss, mit dem erzeugte Nachrichten übertragen werden.
Diese Variante kann sowohl im Entwicklungsprozess, als auch zu Diagnosezwecken während Wartungs- und Reparatureinsätzen eingesetzt werden. Das bestehende Netzwerk wird in diesem Fall nicht verändert, da der Restbussimulator einfach in das bestehende Netzwerk integriert werden kann ohne das System zu beeinflussen.

2.4.2 Vorstellung wissenschaftlicher Arbeiten

Die Restbussimulation im wissenschaftlichen Umfeld ist durch die Anzahl bereits vorhandener Tools für aktuelle Bussysteme sehr geprägt, sodass diese zwar erfolgreich eingesetzt werden, allerdings nicht selber entwickelt werden müssen. Die nachfolgend vorgestellten Arbeiten befassen sich mit einer Restbussimulation eines Time-Triggered Protokolls, ähnlich des FlexRay-Busses, und der Anbindung einer ereignisgesteuerten Simulationsumgebung an ein Echtzeit-Ethernet Netzwerk.

Clustersimulation in Time-Triggered Real-Time Systems Diese Arbeit beschreibt die komplette Entwicklung eines Restbussimulators für den TTP/C-Bus (vgl. Galla 1999;

Galla / Pallierer 1999). TTP/C (vgl. Elmenreich / Ipp 2003) ist ein Bussystem, welches zur *Time-Triggered Architecture* gehört. Damit verfügt es über eine zeitgesteuerte Übertragung und ist vom Anwendungsgebiet mit dem FlexRay-Bus vergleichbar. Knoten dieses Protokolls haben eine fest vorgegebene Architektur und werden in ein Host- und Kommunikationssubsystem unterteilt, die mittels „Communication Network Interface“ (in der Regel Dualport-Memory) miteinander verbunden sind. Das Hostsubsystem ist für die Ausführung der eigentlichen Applikation verantwortlich, während das Kommunikationsubsystem das Übertragen der Daten übernimmt. Die Restbussimulation erfolgt in dieser Arbeit konform zum generellen Aufbau eines Knotens und kann daher auch in zwei Teile getrennt werden. Die Simulation der Knoten im Hostteil führt dazu, dass die Funktionalität auf einem Knoten abgebildet werden muss. Durch die geringe Rechenkapazität der damaligen Mikrocontroller ist es erforderlich, dass die Komplexität vereinfacht und von der eigentlichen Funktionalität abstrahiert wird. Das Scheduling der simulierten Knoten läuft zudem synchron zu dem des Busses, sodass jeder simulierte Knoten in einem separaten Zeitslot ausgeführt wird. Dabei muss überprüft werden, ob die Zeit ausreicht, den Knoten zu simulieren. In dieser Arbeit muss deshalb die Worst-Case-Execution-Time (WCET) der simulierten Knoten bestimmt werden $\tau_{sim}^{WCET} < t_i^{Send} - t_i^{SimFunk}$. Die Ausführungszeit τ_{sim}^{WCET} darf nicht länger dauern als die Differenz zwischen dem Sendezeitpunkt t_i^{Send} einer Nachricht und dem Zeitpunkt, an dem die Funktion des simulierten Knotens $t_i^{SimFunk}$ aufgerufen wird. Erfolgreich eingesetzt wurde dieser Restbussimulator bei der Entwicklung eines Steer-by-Wire-Prototypen, bei dem die Entwicklung der einzelnen Komponenten entsprechend des gängigen Entwicklungsprozesses im Automobil, dezentral durchgeführt wurde.

A Hardware/Software Platform for Real-time Ethernet Cluster Simulation in OMNeT++ Die Verbindung der ereignisbasierten Simulationsplattform OMNeT++ mit Echtzeit-Ethernet-Netzwerken ist im Konferenzbeitrag und der Abschlussarbeit von Oleg Karfich beschrieben (vgl. Karfich / Bartols / Steinbach u. a. 2013; Karfich 2013). Hierbei handelt es sich um eine Hardware-/Softwareplattform, die eine Möglichkeit bereitstellt, das Verhalten von Echtzeit-Ethernet-Protokollvarianten zu simulieren und Simulationsimplementierungen mit realen Netzwerkteilnehmern zu koppeln. Die Plattform basiert auf einer Softwareimplementierung des TTEthernet-Protokolls auf einem Microcontroller (vgl. Müller / Steinbach / Korf u. a. 2011), der für das protokollspezifische und zeitliche korrekte Übertragen der Nachrichten zuständig ist und einer x86-basierenden Workstation, die die Simulationsumgebung ausführt. Als Simulationsmodell wird *CoRE4inet* (vgl. CoRE-Arbeitsgruppe [b]) verwendet, welches eine Implementierung des TTEthernet Standards darstellt. Verbunden sind beide Subsysteme über eine Dualport-Memory-Schnittstelle. Das ausgeführte Simulationsmodell musste mit dem realen Netzwerk synchronisiert werden, sodass Nachrichten rechtzeitig aus der Simulationsumgebung gesendet und empfangen werden. Dieses wurde

erreicht, indem der Simulationsscheduler auf die hochpräzise Zeitbasis des Mikrocontrollers zugreifen kann und somit eine Synchronisierung zulässt. Gleichzeitig wurde die Linux-Echtzeit-Kernel-Erweiterung (vgl. Ts'o / Hart / Kacur 2010) auf dem Host verwendet, um eine Priorisierung der Simulationsumgebung durchführen zu können. In dieser Arbeit wurde erreicht, dass empfangene Nachrichten aus dem echten Netzwerk in die Simulationsumgebung und umgekehrt rechtzeitig übertragen werden können. Die Ergebnisse zeigen, dass die Performance dieser Herangehensweise für das Senden und Empfangen von Ethernetframes minimaler Größe genügend ist. Anwendungen, die dieser Einschränkung entsprechen, lassen sich so simulieren.

2.4.3 Modellbasiertes Testen im Automotiveumfeld

Modellbasiertes Testen im Automotiveumfeld ist ein stark bearbeitetes Themengebiet. Da viele Arbeiten außerhalb des Kontextes der Restbussimulation liegen, wird nachfolgend eine Arbeit präsentiert, in der ein mathematisches formales Gerüst entwickelt wurde, mit dem es möglich ist Testfälle, abstrakt zu spezifizieren und anschließend auf einer Testplattform auszuführen.

Pawel Skruch et al. haben in der Arbeit **Model-Based Testing in Embedded Automotive Systems** (vgl. Skruch / Panek / Kowalczyk 2011) ein formales Modell zur Beschreibung von abstrakten Testfällen entwickelt. Auf Basis der Zustandsraummodellierung wird das SUT mit seinen Ein- und Ausgängen beschrieben, sodass im nächsten Schritt auf der Basis der theoretischen SUT-Beschreibung abstrakte Testfälle für funktionale Anforderungen erzeugt werden können. Die Entwicklung wurde an einem Beispiel verdeutlicht, bei dem ein CAN-Steuergerät getestet wurde. Hierbei wurden die abstrakten Testfälle durch eine 1:1 Relation der abstrakten Testdaten zu den realen Nachrichten in ausführbare Testfälle umgewandelt und auf der Testplattform ausgeführt. Die Möglichkeiten, Testfälle zuerst in einer abstrakten Notation zu formulieren, entspricht dem modellbasierten Testen, sodass die gleichen Testfälle auf unterschiedlichen Plattformen ohne Probleme portiert werden können.

2.4.4 Abgrenzung der vorgestellten Arbeiten zu dieser Arbeit

Einige Konzepte der vorgestellten Arbeiten können verwendet werden und finden sich dementsprechend auch in dieser Arbeit wieder. Andere Konzepte können aufgrund unterschiedlicher Technologien nicht umgesetzt werden. Damit eine Verbindung der vorgestellten vergleichbaren Arbeiten zu dieser Arbeit möglich ist, grenzt der folgende Abschnitt die eigene Arbeit ab. Zuerst werden die Gemeinsamkeiten und anschließend die Unterschiede erläutert.

Das Konzept des generellen Aufbaus einer Restbussimulation, das in den vorgestellten kommerziellen Produkten und Patenten umgesetzt wurde, findet sich auch in dieser Arbeit wieder. So wird die Restbussimulation über einen Konfigurations-PC und einer echtzeitfähigen Hardware-Softwareplattform, die für das Übertragen der Nachrichten verantwortlich ist, durchgeführt. Ebenfalls wird das Konzept übernommen FIBEX zur Konfiguration verwenden zu können, welches in produktiv eingesetzten Restbussimulatoren verwendet wird. So ist es möglich in FIBEX beschriebene Echtzeit-Ethernet-Netzwerke zu importieren und die entsprechende Konfiguration zu erstellen.
Des Weiteren wird das Abstraktionskonzept und die Beschränkung auf die relevanten Nachrichten in dieser Arbeit übernommen. So werden nur die Nachrichten erzeugt, die vom SUT empfangen werden. Die Abstraktion des Verhaltens der zu simulierenden Knoten wird in dieser Arbeit noch weiter fortgeführt, indem offline generierte Daten versendet werden und das Verhalten der simulierten Knoten vorher bestimmt wird. Die Arbeit von Oleg Karfich stellt eine gute Basis zu der in dieser Arbeit implementierten Restbussimulation dar. So finden sich Teile der Hardware-/Softwareplattform auch in diesem Ansatz wieder: Die Plattform besteht aus einem Mikrocontroller und einer x86-basierten Workstation.
Das Konzept der Verwendung abstrakter Testfälle von Skruch et al. wird auch in dieser Arbeit umgesetzt und dahingehend erweitert, dass nicht nur funktionale, sondern auch nichtfunktionale Anforderungen, insbesondere zu zeitlichen Aussagen, eines zu testenden Systems überprüft werden können, sodass sich zusätzliche Testmöglichkeiten für Echtzeitsysteme ergeben.

Die Möglichkeit eine Rapid-Prototyping Umgebung zu erstellen, die mit kommerziellen Produkten möglich ist, liegt nicht im Fokus dieser Arbeit und wird deshalb nicht umgesetzt, da bei der Ausführung von Testfällen ein festgelegtes Verhalten erstrebenswert ist. Das Verhalten der zu simulierenden Knoten wird deshalb in dieser Arbeit offline erzeugt, sodass online berechnetes reaktives Verhalten nicht umgesetzt werden muss. Auf der anderen Seite wird die Komplexität bei der Ausführung der Restbussimulation deutlich reduziert, sodass keine aufwendigen Simulationsmodelle implementiert werden müssen, die das Verhalten beeinträchtigen.
Dadurch, dass die Restbussimulation im Entwicklungsprozess eingesetzt werden soll um Implementierungsfehler erkenntlich zu machen, müssen keine komplexen Topologien, wie im Patent von Honeywell beschrieben, verwendet werden, die den parallelen Betrieb des realen Netzwerks zulassen.

3 Modellbasierte Entwicklung von Echtzeit-Ethernet-Anwendungen

In diesem Kapitel wird auf die Besonderheiten einer modellbasierten Entwicklung von Echtzeit-Ethernet Anwendungen im Automobil eingegangen, um Testfälle modellbasiert ableiten zu können. Das Hauptaugenmerk liegt auf der Modellierung von zeitlichen Anforderungen, die anhand einer Automotive-Echtzeit-Ethernet-Anwendung erklärt wird. Sowohl die Modellierung von Spezifikationen, als auch die Modellierung der dazugehörigen Testfälle werden präsentiert. Zunächst wird in diesem Kapitel deshalb die Anwendung in natürlicher Sprache beschrieben, gefolgt von einer systematischen Auflistung der Anforderungen. Anschließend werden Anforderungen an eine Modellierungssprache gestellt, die eine Möglichkeit zur Darstellung zeitlicher Informationen zulässt. Spezifikationen von Automotive-Echtzeit-Ethernet-Anwendungen werden in ausgewählten Modellierungssprachen anschließend erstellt.

3.1 Beschreibung der Beispielanwendung

Als Beispielanwendung dient eine diskrete ereignisbasierte Scheinwerfersteuerung, um die Modellierung einer Echtzeit-Ethernet-basierten Automotiveanwendung zu verdeutlichen.

3.1.1 Beschreibung der Scheinwerfersteuerung in natürlicher Sprache

Die Scheinwerfersteuerung ist Teil eines in der Forschungsgruppe CoRE (vgl. CoRE-Arbeitsgruppe [a]) entwickelten Echtzeit-Ethernet Demonstrationssystems, das die Eigenschaften einer Echtzeit-Ethernet-basierten Netzwerkarchitektur im Fahrzeug verdeutlicht und als Prototyp verwendet wird. Es beinhaltet mehrere Anwendungen, die über das gleiche physikalische Netzwerk Daten austauschen und entsprechen der in Abschnitt 2.1.1 auf Seite 5 vorgestellten Klassen *Multimedia*-, *Komfort*- und *Antriebsstranganwendungen*, wobei die Scheinwerfersteuerung zur Antriebsstrangkategorie zu zählen ist und damit Echtzeitanforderungen aufweist.
Das Demonstrationssystem ist in Abbildung 3.1 auf der nächsten Seite dargestellt. Neben der Scheinwerfersteuerung ist eine Drive-by-Wire-Anwendung (ebenfalls Antriebsstrang), ein Kamerasystem (Komfort) und eine Videostreamanwendung (Multimedia) in einem flachen

Netzwerk umgesetzt, das der in Abschnitt 2.3.4 auf Seite 28 präsentierten Topologie entspricht. Als informative Bedienschnittstelle für den Benutzer steht ein Infotainmentsystem zur Verfügung (siehe rechte Seite in Abb. 3.1). Es dient als digitales Armaturenbrett für die Scheinwerfersteuerung und als Benutzerschnittstelle der Drive-by-Wire- und der Multimediaanwendung.

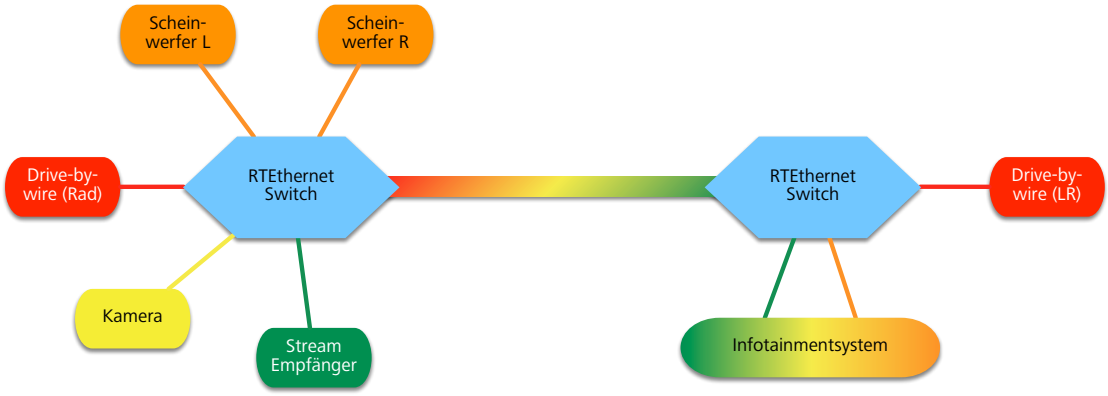


Abbildung 3.1: *Echtzeit-Ethernet Demonstrationssystem für das Fahrzeugnetzwerk der nächsten Generation*

Die verwendeten Scheinwerfer im Demonstrationssystem bestehen aus realer Automotivehardware und entstammen einem VW-Tiguan[3]. Die Modellierung der Scheinwerfersteuerung muss es erlauben, alle im Scheinwerfer befindlichen Leuchten separat ansteuern zu können. Die Scheinwerfer beinhalten Blinkerleuchten, einen Xenon-Hauptscheinwerfer zur Verwendung von Fern- und Abblendlicht sowie ein regelbares LED-Leuchtband, das für Tagfahrlicht benutzt wird. Damit Fern- und Abblendlicht mit derselben Leuchte umgesetzt werden können, gibt es eine steuerbare Verschlussblende, die geöffnet und geschlossen werden kann. Im geschlossenen Zustand wird Abblendlicht umgesetzt, im geöffneten Fernlicht.
Um die verschiedenen Betriebsmodi der Scheinwerfer zu aktivieren, implementiert das Infotainmentsystem ein digitales Armaturenbrett. Dieses erlaubt den gewünschten Betriebsmodus auszuwählen und sorgt dafür, dass der aktuelle Scheinwerferzustand angezeigt wird.

3.1.2 Definition des Anforderungsmodells

Wie in den Grundlagen beschrieben, ist der erste Schritt im V-Modell die Erarbeitung der abstrakten Benutzeranforderungsspezifikation. Sie spezifiziert das gewünschte Verhalten des Systems. Anschließend werden die Anforderungen gemäß des V-Modells in den verschiedenen

[3] http://emosite.volkswagen.de/esn/tiguan/?culture=de_DE#tiguan

Konstruktionsphasen verfeinert. Da die Klassifikation der Anforderungen in die klassischen funktionalen und nicht-funktionalen zu radikal ist (vgl. Glinz 2007) und die Nachvollziehbarkeit bei der Erstellung der Anforderungen erschwert wird, wird die vorgeschlagene Anforderungstaxonomie nach Martin Glinz verwendet (siehe Abbildung 3.2). Diese ermöglicht im Gegensatz zur klassischen Einteilung eine deutlich differenzierte Herangehensweise nichtfunktionale Anforderungen zu beschreiben und einzuordnen. Dadurch wird die Nachvollziehbarkeit bei der Erarbeitung von Anforderungen erhöht.
Gleichzeitig ist die Taxonomie allgemein genug, um sich nicht in Details während der Anforderungsspezifikation zu verlieren (vgl. Glinz 2007), wodurch der Entwicklungsprozess gestört werden kann. Dieses Modell erlaubt einen besseren Verifikationsprozess, da Testfälle systematischer zu erzeugen sind.

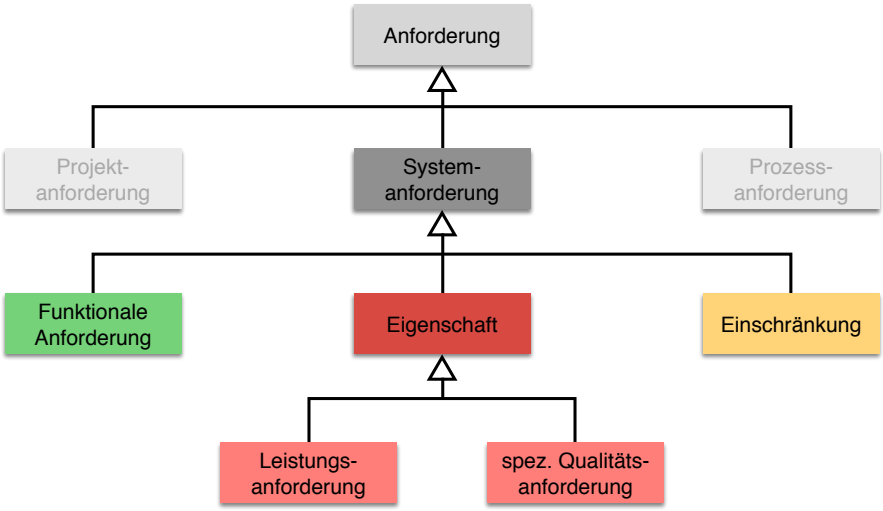


Abbildung 3.2: *Taxonomie der Anforderungen (nach Glinz 2007)*

Die einzelnen Anforderungsklassen und dessen Eigenschaften dieser Taxonomie werden dabei in *funktionale Anforderungen*, *Leistungsanforderungen*, *spezifische Qualitätsanforderungen* und *Einschränkungen* aufgeteilt, die nachfolgend erläutert werden.

Funktionale Anforderungen beschreiben Systemverhalten, Daten und Reaktionen auf Eingänge, unabhängig ihrer späteren Umsetzung. In dieser Kategorie kann wie bisher mit der Regel „Was das System umsetzen soll“ gearbeitet werden, um Anforderungen in diese Kategorie einzuordnen. Die Regel „Wie soll das System das Verhalten umsetzen“ reicht nicht mehr aus, die Art der nicht-funktionalen Anforderungen zu bestimmen. Viel mehr müssen mehrere Aspekte betrachtet werden, die die Eigenschaften des Systems beschreiben können.

Leistungsanforderungen definieren Systemaspekte, die z. B. das Zeitverhalten, Reaktionsgeschwindigkeiten auf Eingaben und den Datendurchsatz definieren.

Spezifische Qualitätsanforderungen erläutern Sicherheits-, Zuverlässigkeits- oder Wartbarkeitszusagen an das System. Sie beschreiben u. a. besondere Attribute der Güte während der Ausführung.

Einschränkungen stellen Implementierungsansätze von Systemverhalten dar und schränken die Umsetzung des Systems auf bestimmte Merkmale ein. Erste einfache Lösungen für eine Umsetzung lassen sich in dieser Hinsicht darstellen.

Die Klassifizierung in diese Kategorien ist auch besonders für Echtzeit-Systeme geeignet, da gerade Zeitverhalten, Datendurchsatz und Reaktionsgeschwindigkeiten klar als Leistungsanforderungen definiert werden und damit zu den Systemeigenschaften zählen. Testfälle, die diese Anforderungen überprüfen sollen, lassen sich systematischer erstellen, da die vorgestellte Taxonomie eine bessere Nachvollziehbarkeit gegenüber der klassischen Klassifikation ermöglicht. Klassen von Testfällen lassen sich somit erzeugen, die genau diese Anforderungen überprüfen.

3.1.3 Anforderungen der Scheinwerfersteuerung

Nachfolgend sind die Anforderungen der Scheinwerfersteuerung nach der vorgestellten Taxonomie definiert. Dabei ist der Schwerpunkt auf Anforderungen aus dem informationstechnischen Hintergrund gelegt und Aspekte der Elektrotechnik und Mechanik werden nicht dargestellt. In der Reihenfolge der Ebenen des V-Modells werden die Anforderungen präsentiert.

Anforderungen der abstrakten Benutzeranforderungsebene

Als erstes werden die abstrakten Benutzeranforderungen vorgestellt. Tabelle 3.1 auf der nächsten Seite listet die funktionalen Anforderungen auf, die beschreiben was für Aufgaben das System umsetzen soll. Die dazugehörigen Leistungsanforderungen werden durch Tabelle 3.2 auf Seite 41 präsentiert.

ID	Name	Beschreibung
$AB_{FA}1$	Unabhängigkeit	Der rechte und linke Scheinwerfer sollen unabhängig von einander angesteuert werden können
$AB_{FA}2$	Steuerbarkeit	Alle Leuchten der Scheinwerfer sollen separat angesteuert werden können
$AB_{FA}3$	Datenreaktion	Die Scheinwerfer dürfen nur auf bekannte Nachrichten reagieren; unbekannte Nachrichten werden verworfen
$AB_{FA}4$	Periodische Zustands-übermittlung	Die Scheinwerfer sollen ihren aktuellen Zustand periodisch an das Bordnetz übermitteln
$AB_{FA}5$	Zustandsempfang	Das Infotainmentsystem soll zur Steuerung der Scheinwerfer verwendet werden
$AB_{FA}6$	Zustands-darstellung	Das Infotainmentsystem soll zur Darstellung des aktuellen Zustands der Scheinwerfer verwendet werden
$AB_{FA}7$	Tagfahrlicht-funktion	Die Scheinwerfer sollen eine Tagfahrlichtfunktion bereitstellen
$AB_{FA}8$	Abblendlicht-funktion	Die Scheinwerfer sollen eine Abblendlichtfunktion bereitstellen
$AB_{FA}9$	Fernlichtfunkion	Die Scheinwerfer sollen eine Fernlichtfunktion bereitstellen
$AB_{FA}10$	Fahrtrichtungs-anzeigerfunktion	Die Scheinwerfer sollen eine Fahrtrichtungsanzeigerfunktion bereitstellen
$AB_{FA}11$	Überholmanöver-anzeigerfunktion	Die Scheinwerfer sollen eine Überholmanöveranzeigerfunktion bereitstellen
$AB_{FA}12$	Warnblinklicht-funktion	Die Scheinwerfer sollen eine Warnblinklichtfunktion bereitstellen
$AB_{FA}13$	Zukunftssicherheit	Die Datenkommunikation soll genügend Bandbreite für zukünftige Erweiterungen bereitstellen

Tabelle 3.1: *Auflistung der funktionalen Anforderungen der abstrakten Benutzeranforderungsebene*

ID	Name	Beschreibung	Wert
$AB_{LA}1$	Reaktions-geschwindigkeit	Die Reaktion auf Benutzereingaben soll für den Menschen unmittelbar erscheinen (vgl. Kamke / Walcher 1994)	$\leq$ 100ms
$AB_{LA}2$	Blinklichtintervall	Die Frequenz des Blinkens ist während der Ausführung der Fahrtrichtungsanzeiger-, Überholmanöveranzeiger- und Warnblinklichtfunktion durch §54 StVZO (vgl. StVZO 2013) begrenzt	1,5Hz($\pm$ 0,5Hz)

Tabelle 3.2: *Auflistung der Leistungsanforderungen der abstrakten Benutzeranforderungsebene*

Anforderungen des funktionalen Systementwurfs

Der nächste Schritt ist die Verfeinerung der abstrakten Benutzeranforderungen auf dem Level des funktionalen Systementwurfs. Dabei stellt Tabelle 3.3 auf der nächsten Seite die funktionalen Anforderungen dar, die im Wesentlichen die umzusetzenden Funktionen beschreiben. Die Leistungsanforderungen des funktionalen Systementwurfs werden in der Tabelle 3.4 auf Seite 43 aufgelistet.

ID	Name	Beschreibung
$FS_{FA}1$	Unterscheidbarkeit	Der rechte und linke Scheinwerfer ist unterscheidbar
$FS_{FA}2$	Hauptscheinwerfer	Der Hauptscheinwerfer wird ein- und ausgeschaltet werden können
$FS_{FA}3$	Verschlussblende	Die Verschlussblende wird geöffnet und geschlossen werden können
$FS_{FA}4$	Blinkerleuchte	Die Blinkerleuchte wird ein- und ausgeschaltet werden können
$FS_{FA}5$	LED-Leuchten	Die LED-Leuchten werden in der Helligkeit verstellt werden können
$FS_{FA}6$	Tagfahrlicht-funktion	Die Tagfahrlichtfunktion wird durch die LED-Leuchten realisiert werden
$FS_{FA}7$	Abblendlicht-funktion	Die Abblendlichtfunktion wird durch den Hauptscheinwerfer und der LED-Leuchten realisiert werden
$FS_{FA}8$	Fernlichtfunktion	Die Fernlichtfunktion wird durch den Hauptscheinwerfer und der Verschlussblende realisiert werden
$FS_{FA}9$	Fahrtrichtungs-anzeigerfunktion	Der Fahrrichtungsanzeiger wird durch die Blinkerleuchte erreicht werden
$FS_{FA}10$	Überholmanöver-anzeigerfunktion	Die Funktion des Überholmanöveranzeigers wird durch die Blinkerleuchte realisiert werden
$FS_{FA}11$	Warnblinklicht--funktion	Die Funktion des Warnblinklichts wird durch die Blinkerleuchte realisiert werden
$FS_{FA}12$	Fahrtrichtungs-anzeiger-bedienung	Die Funktion des Fahrtrichtungsanzeigers wird nach Betätigung dauerhaft bis zum expliziten Beenden durch den Benutzer ausgeführt werden
$FS_{FA}13$	Warnblinklicht-bedienung	Die Funktion des Warnblinklichts wird nach Betätigung dauerhaft ausgeführt werden
$FS_{FA}14$	Überholmanöver-anzeiger-bedienung	Der Überholmanöveranzeiger wird nach Betätigung eine dreimalige alternierende Reihenfolge ausführen und anschließend automatisch stoppen
$FS_{FA}15$	Zustandsquittierung	Die Scheinwerfer werden einen neu übernommenen Zustand quittieren
$FS_{FA}16$	Datenreaktion	Der Scheinwerfer wird auf unbekannte Zustände nicht reagieren
$FS_{FA}17$	Datenkommunikation	Die Kommunikation der Scheinwerfer mit dem Bordnetz wird durch unkritische Multimediakommunikation nicht gestört werden

Tabelle 3.3: *Auflistung der funktionalen Anforderungen des funktionalen Systementwurfs*

ID	Name	Beschreibung	Wert
$FS_{LA}1$	Blinklichtfrequenz	Die Frequenz des Blinkens zwischen AN → AUS ist begrenzt	1Hz
$FS_{LA}2$	Funktionsbereich LEDs	Der Funktionsbereich der Helligkeit LEDs ist beschränkt	0 – 100%
$FS_{LA}3$	Zustandsrückmeldu	Die Zeit für eine Rückmeldung des Zustands ist beschränkt	1000 µs

Tabelle 3.4: *Auflistung der Leistungsanforderungen des funktionalen Systementwurfs*

Anforderungen des technischen Systementwurfs

Aufbauend auf dem funktionalen Systementwurf werden die Anforderungen des technischen Systementwurfs aufgestellt. Die funktionalen Anforderungen listet Tabelle 3.5 auf. In den Tabellen 3.6 und 3.7 auf der nächsten Seite werden die Leistungsanforderungen sowie Einschränkungen dieser Ebene präsentiert.

ID	Name	Beschreibung
$TS_{FA}1$	Datenkommunikation	Die Kommunikation der Scheinwerfer mit dem Bordnetz wird durch Echtzeit-Ethernet realisiert werden
$TS_{FA}2$	Nachrichtenklasse	Die Übertragung der Daten wird durch eine echtzeitfähige (Critical) Nachrichtenklasse erfolgen
$TS_{FA}3$	Zeitbasis	Die Scheinwerfer werden eine synchrone Zeitbasis zum Rest des Systems besitzen

Tabelle 3.5: *Auflistung der funktionalen Anforderungen des technischen Systementwurfs*

ID	Name	Beschreibung	Wert
$TS_{LA}1$	Netzwerkanbindung	Die Netzwerkanbindung wird die Bandbreite des gesamten Netzwerks unterstützen	100 $^{MBit}/_{s}$
$TS_{LA}2$	Bandbreitenbegrenzung	Die verwendete Bandbreite der Scheinwerfersteuerung ist begrenzt und darf 70% der Gesamtbandbreite nicht übersteigen	$\leq$ 70% 70 $^{Mbit}/_{s}$
$TS_{LA}3$	Nachrichtenlatenz	Die Ende-zu-Ende Latenz der Datenübertragung zwischen Sender und Empfänger ist begrenzt	$\leq$ 500 µs

Tabelle 3.6: *Auflistung der Leistungsanforderungen des technischen Systementwurfs*

ID	Name	Beschreibung	Wert
$TS_{ES}1$	Datenfelder	Die Kommunikation der Scheinwerfsteuerung wird durch eine Datenstruktur realisiert, deren Größe begrenzt ist	2 Byte
$TS_{ES}2$	Zustand Leuchten	Die Position des Datenfeldes des Zustands der Leuchten ist festgelegt	1. Byte
$TS_{ES}3$	Zustand LED	Die Position des Datenfeldes des Zustands der LEDs ist festgelegt	2. Byte
$TS_{ES}4$	Bitposition HS	Der Zustand des Hautpscheinwerfers (Wert 0 = aus, 1 = ein) ist festgelegt	Bit 0
$TS_{ES}5$	Bitposition FL	Der Zustand des Verschlusses Wert (0 = geschlossen, 1 = geöffnet) ist festgelegt	Bit 1
$TS_{ES}6$	Bitposition BL	Der Zustand des Blinkers (Wert 0 = aus, 1 = ein) ist festgelegt	Bit 2
$TS_{ES}7$	Bitposition UM	Der Zustand des Überholmanöver (Wert 0 = aus, 1 = ein) ist festgelegt	Bit 3
$TS_{ES}8$	Wertebereich LED	Der Wertebereich der Helligkeit der LED ist vorzeichenbefreit festgelegt	0 - 100
$TS_{ES}9$	CriticalMarker	Der Critical-Marker der Nachrichten ist festgelegt	0x0304 0506
$TS_{ES}10$	Critical-ID Steuernachricht SW/R	Die Critical-ID der Nachricht, die vom Infotainment an den Scheinwerfer rechts gesendet wird, ist festgelegt	0x0400
$TS_{ES}11$	Critical-ID Steuernachricht SW/L	Die Critical-ID der Nachricht, die vom Infotainment an den Scheinwerfer links gesendet wird, ist festgelegt	0x0401
$TS_{ES}12$	Critical-ID Statusnachricht SW/R	Die Critical-ID der Nachricht, die vom Scheinwerfer rechts an das Bordnetz gesendet wird, ist festgelegt	0x0410
$TS_{ES}13$	Critical-ID Statusnachricht SW/L	Die Critical-ID der Nachricht, die vom Scheinwerfer links an das Bordnetz gesendet wird, ist festgelegt	0x0411

Tabelle 3.7: *Auflistung der Einschränkungen des technischen Systementwurfs*

Anforderungen aus der Komponentenspezifikation

Zuletzt präsentieren die Tabellen 3.8 und 3.9 auf der nächsten Seite die Anforderungen der Komponentenspezifikation.

ID	Name	Beschreibung	Wert
$KS_{LA}1$	Reaktionszeit Zustandsquittierung	Die Reaktionsgeschwindigkeit auf empfangen Steuerbefehle ist begrenzt und wird innerhalb einer definierten Zeitspanne mit dem Versenden der Quittierung erfolgen	500 µs
$KS_{LA}2$	Senderate periodische Zustandübermittlung	Das Zeitintervall der Zustandsübermittlung des Scheinwerfers ist festgelegt	5000µs
$KS_{LA}3$	Jitter der Reaktionszeit	Der Jitter der Reaktionsgeschwindigkeit ist begrenzt	$\leq$ 50µs
$KS_{LA}4$	Jitter der Senderate	Der Jitter der periodischen Senderate der Scheinwerfer ist begrenzt	$\leq$ 10µs

Tabelle 3.8: *Auflistung der Leistungsanforderungen der Komponentenspezifikation*

ID	Name	Beschreibung	Wert
$KS_{ES}1$	Synchronisation	Die Scheinwerfer werden eine festgelegte Rolle im Zeitsynchronisierungsprozess ausführen	Sychronization Client
$KS_{ES}2$	Nachrichtenklasse	Die Übertragung der Nachrichten ist festgelegt	RC-Nachricht
$KS_{ES}3$	Zustands-quittierung	Der Scheinwerfer quittiert den neuen Zustand mit den Daten des Steuerbefehls	Daten (Steuerbefehl)

Tabelle 3.9: *Auflistung der Einschränkungen auf der Komponentenspezifikation*

Im Vergleich zu den abstrakteren, höheren Ebenen im V-Modell sieht man, dass, je technischer und detaillierter das System beschrieben wird, desto größere Rollen spielen Einschränkungen und Leistungsanforderungen innerhalb der Scheinwerfersteuerung. Die aufgelisteten Anforderungen beschreiben das Scheinwerfersystem aus der Sicht der zu erfüllenden Funktionen, sodass diese im nächsten Schritt modelliert werden können. Damit eine Spezifikation erstellt werden kann, bedarf es verschiedener Modellierungsmöglichkeiten, die im nächsten Abschnitt beschrieben werden.

3.2 System- und Softwaremodellierung von Automotive-Echtzeit-Ethernet-Anwendungen

Nachdem die Beispielanwendung im vorherigen Abschnitt vorgestellt wurde, beschreibt dieser Abschnitt die System- und Softwaremodellierung von Automotive-Echtzeit-Ethernet-Anwendungen, insbesondere aus der Sicht der Modellierung von zeitlichen Aussagen. Dabei werden zuerst Anforderungen an eine Modellierungssprache gestellt und anschließend verschiedene Modelle vorgestellt. Sie erlauben den Spezifikationsentwurf von Echtzeit-Ethernet-basierter Anwendungen, in denen die zuvor präsentierten Anforderungen zu finden sind.

3.2.1 Anforderungen an Modellierungssprachen

Softwaresysteme im Automobil zeichnen sich durch eine hohe Komplexität aus, da sie sowohl diskrete, als auch kontinuierliche Charakteristiken aufweisen und verschiedene Leistungsanforderungen im zeitlichen Kontext erfüllen müssen. Damit Softwaresysteme im Automobil auch für Echtzeit-Ethernet-Anwendungen modelliert werden können, bedarf es geeigneter Modelle, die diese speziellen Anforderungen erfüllen.

Da Modellierungssprachen für verschiedene Zwecke erstellt werden können, teilen sich Anforderungen an eine Modellierungssprache generell in die Teilgebiete *anwendungsbezogene*, *formale* und *anwenderbezogene Anforderungen* auf (vgl. Frank / van Laak 2003).

Anwendungsbezogene Anforderungen definieren den Bezug zum Modellierungszweck und dessen Domäne. Um Echtzeit-Ethernet-Anwendungen mittels Modellen spezifizieren zu können, müssen sowohl statische, als auch dynamische Systemmerkmale modelliert werden können. Statische Systemmerkmale werden zur Modellierung von Eigenschaften verwendet, die sich während der Laufzeit des Systems nicht verändern. Zu diesen Eigenschaften zählen u. a. die Netzwerktopologie mit allen beteiligten Komponenten sowie der Datenpfad auf dem Nachrichten versendet werden.
Dynamische Systemmerkmale beschreiben Attribute des Systems, die sich während der Laufzeit verändern können. Dazu gehören Funktionen die ausgeführt werden und Verhaltensspezifikationen des Systems. Dadurch, dass Automotivesoftwaresysteme gleichzeitig kontinuierlich und diskret sind, müssen sich alle Eigenschaften auch in der Modellierungssprache wiederfinden.
Besonders wichtig bei der Spezifizierung von Automotiveanwendungen ist das Modellieren von zeitlichem Verhalten, damit Echtzeitanforderungen spezifiziert werden können. Insbesondere bei Echtzeit-Ethernet-basierten Kommunikationsstrukturen ist dieses ein wesentliches Attribut. Anhand dieser Anforderungen wird das Netzwerk konzipiert und

die Konfiguration von TT-, RC- und BE-Nachrichten erstellt. Die Modellierungssprachen müssen daher in der Lage sein sowohl Einschränkungen, als auch die dazugehörigen Zeitpunkte in der Prozessverarbeitung darzustellen.

Formale Anforderungen sind von besonderer Bedeutung, wenn Modelle maschinell weiterverwendet werden sollen. Erfüllt ein Modell formale Anforderungen können z. B. die Integrität eines Modells überprüft oder Modelltransformationen getätigt werden. Dabei muss ein Modell syntaktisch und semantisch korrekt definiert sein und darf keine Widersprüche sowie Redundanzen aufweisen. Modelle, die für Echtzeit-Ethernet-basierte Anwendungen entworfen werden, müssen einen formalen Hintergrund aufweisen, wenn sie zur weiteren Verarbeitung verwendet werden. So können z. B. durch eine Modelltransformation, Simulationsmodelle erstellt werden.

Anwenderbezogene Anforderungen definieren die Eigenschaften der Modelle, die direkt mit dem Benutzer in Verbindung gebracht werden, z. B. Visualisierung und Handhabbarkeit. Modellierungssprachen für Echtzeit-Ethernet-basierte Anwendungen müssen in der Lage sein, die Attribute der Anwendung anschaulich darzustellen. Außerdem muss die Sprache intuitiv zu verwenden sein, damit sie den Entwicklungs- und Spezifikationsprozess unterstützt. Gerade dieser Punkt ist bei Echtzeit-Ethernet-Anwendungen im Automotivekontext sehr wichtig, da es durch den interdisziplinären und dezentralen Entwicklungsprozess einen hohen Bedarf an intuitiv zu verstehenden Modellen gibt. Problemstellungen können verdeutlicht und von allen Beteiligten verstanden werden.

Im nächsten Abschnitt werden zwei Modellierungsmöglichkeiten für Echtzeit-Ethernet-basierte Anwendungen vorgestellt.

3.2.2 Grafische Modellierungsmöglichkeiten

Eine gute Möglichkeit zur Modellierung der Spezifikation bieten standardisierte grafische Modelle, die aus der generischen Software- bzw. Systemmodellierung bekannt sind. Die *Unified Modeling Language (UML)* (vgl. OMG [d]) und *Systems Modeling Language (SysML)* (vgl. OMG [c]) sind von der Object Management Group (vgl. OMG [a]) entworfen und spezifiziert. Sie ermöglichen eine intuitive Verwendung bei gleichzeitiger Formalität. Sie haben durch ihre weite Verbreitung bewiesen, dass sie effektiv und unterstützend im Entwicklungsprozess eingesetzt werden.
Allerdings können UML und SysML nicht ohne weiteres zur Modellierung von eingebetteten Echtzeitsystemen eingesetzt werden, da wesentliche Elemente zur Modellierung von nichtfunktionalen Anforderungen fehlen (z. B. Verbrauch von Ressourcen) (vgl. Demathieu / Thomas / Andre u. a. 2008; Quadri / Sadovykh / Indrusiak 2012). Des Weiteren ist das UML-Zeitmodell zu einfach und unpräzise, da es keine Unterscheidungen zwischen logischen und physikalischen Uhren bereitstellt, sodass UML-Timining Diagramme für die Modellierung von

Echtzeitsystemen nicht ausreichen (vgl. Yu / Talpin / Besnard u. a. 2010). Durch die Verwendung des UML-Profils *Modeling and Analysis of Real-time Embededded Systems (MARTE)* (vgl. OMG [b]; OMG 2011), welches eine Spracherweiterung für die UML darstellt, lassen sich diese fehlenden Elemente erweitern.

Nichtsdestotrotz bieten beide Sprachen ein breites Fundament an Diagrammen, die für verschiedene Anwendungszwecke verwendet werden können. Durch Zustandsautomaten und Sequenzdiagramme kann mit diesen Sprachen diskretes Verhalten beschrieben werden. Da in dieser Arbeit eine ereignisgesteuerte Anwendung mit Fokus auf diskretem Verhalten entworfen wird, sind Zustandsautomaten und Sequenzdiagramme zur Modellierung des Scheinwerfersystems ausreichend. Alternativ stellt die SysML mit dem eingeführten Zusicherungsdiagrammen zusätzlich eine Möglichkeit zur Verfügung, kontinuierliches Verhalten durch die Modellierung von Differentialgleichungen zu spezifizieren (vgl. Johnson / Paredis / Burkhart u. a. 2007; OMG 2010), sodass Verhaltensmodelle für Automotivesoftwaresysteme modelliert werden können. Erste Versuche SysML zur Modellierung von Automotivesystemen zu verwenden sind in (Andrianarison / Piques 2010; Piques / Andrianarison 2012; Apvrille / Becoulet 2012) zu finden.

Auf dem breiten Fundament der UML setzt das UML-Profil MARTE auf. MARTE ist zur Modellierung und Analyse von Echtzeitsystemen entwickelt worden und ermöglicht es, durch die Einführung der *Value Specific Language (VSL)*, klassische UML-Modelle mit Annotationen zu versehen. Die VSL wird u. a. zur Modellierung von Leistungsanforderungen verwendet und erlaubt die Darstellung verschiedenster Maßeinheiten und mathematischer Ausdrücke. Damit ist sie ein geeigneter Kandidat die zeitlichen Leistungsanforderungen zu beschreiben (vgl. Demathieu / Thomas / Andre u. a. 2008) und den Modellierungsprozess Echtzeit-Ethernet-basierter Anwendungen zu unterstützen. MARTE unterstützt die Modellierung von zeitbezogenen Eigenschaften durch ein detailliertes Uhrenmodell, sodass auf ideale, synchronisierte Uhren zugegriffen werden kann. Realistische, physikalische Uhren mit definierbaren Abweichungen und Auflösungen können durch den Modellierer entwickelt werden. Bei der Modellierung von Zeitverhalten muss auf ein Uhrenmodell referenziert werden, sodass ersichtlich ist, um was für eine Art Uhrenmodell es sich handelt. Es muss darauf geachtet werden, welches Uhrenmodell zur Modellierung verwendet wird, da sich nicht jedes Modell für sämtliche Spezifikationen eignet. Müssen z. B. zeitliche Leistungsanforderungen modelliert werden, bei denen bestimmte Zeitspannen dargestellt werden, muss darauf geachtet werden, dass das ideale Uhrenmodell ohne Abweichungen verwendet wird. Bei der Spezifikation von Anwendungen, die auf mehrere Komponenten verteilt werden, um z. B. eine Zeitsynchronisierungsmethode zu modellieren, sollte ein realistisches Uhrenmodell entwickelt werden. Differenzen der verschiedenen lokalen Uhren lassen sich auf diese Weise beschreiben.

Nachfolgend werden Diagramme vorgestellt, die das diskrete Verhalten eines Scheinwerfers und dessen zeitliche Leistungsanforderungen mittels der MARTE-Erweiterung darstellen. Im Fokus dieser Arbeit liegt die Modellierung zeitlicher Leistungsanforderungen, sodass nicht

alle Merkmale der Scheinwerfersteuerung gezeigt werden. Die vollständige Modellierung der Scheinwerfersteuerung kann dem Anhang entnommen werden.

Durch das Uhrenmodell und der VSL ist es möglich, zeitgesteuertes Verhalten innerhalb von Zustandsautomaten zu beschreiben. In Diagramm 3.3 auf der nächsten Seite ist der Zustandsautomat des Scheinwerferprozessmodells dargestellt. Es ist das zeitgesteuerte Verhalten zu erkennen, das die periodische Zustandsübermittlung (siehe Anforderung $AB_{FA}4$) des aktuellen Scheinwerferzustands sowie das sofortige Quittieren (Anforderung $FS_{FA}15$) von übernommenen Zuständen darstellt. Die periodische Zustandsübermittlung ist als „`TimedEvent`" an der Transition zwischen den Zuständen *Idle* und *SendeStatusNachricht* gekennzeichnet, das Quittieren als exit-Anweisung beim Verlassen des Zustands `SetzeLichtStatus`. Als Uhrenmodell wurde in diesem Fall eine ideale Uhr verwendet, da es sich hierbei um eine Spezifikation handelt. Modelliert wird diese Eigenschaft mittels der Annotation «`TimedProcessing`» `on MARTE_Library::TimedLibrary::idealClock`.

In Diagramm 3.4 auf Seite 51 ist die Leistungsanforderung der Reaktionsgeschwindigkeit vom Quittieren einer empfangenen Steuerungsnachricht dargestellt (vgl. Anforderungen $KS_{LA}1$ & $KS_{LA}3$). Die periodische Zustandsübermittlung ist im nächsten Diagramm 3.5 auf Seite 51 abgebildet. Als Diagrammtyp wurde ein Sequenzdiagramm gewählt, da dieser Diagrammtyp den Nachrichtenfluss, das Auftreten bestimmter Ereignisse, wie z. B. das Empfangen von Nachrichten, und die Aktivität eines Objektes übersichtlich darstellt und intuitiv zu modellieren ist. Mittels VSL-Annotationen sind die Zeitpunkte des Empfangs der Steuerungsnachricht (`SetzeScheinwerferStatus`) und des Sendens der Quittierungsnachricht (`SendeAktuellenStatus`) modelliert. Die Zeitpunkte sind durch ein vorangestelltes `@` gekennzeichnet und sind mit dem Nachrichtenfluss verbunden.

Auch in diesem Fall wurde das ideale Uhrenmodell der MARTE-Bibliothek referenziert, da eine Spezifikation einer Zeitspanne modelliert werden soll, die eine genaue Uhrzeit voraussetzt. Die Leistungsanforderung ist als «`timedConstrained`» im Diagramm modelliert. Die Zeitspanne ist durch die Differenz des Zeitpunktes an dem die Nachricht `SendeAktuellenStatus` versendet wurde und des Empfangszeitpunktes der `SetzeScheinwerferStatus`-Nachricht modelliert (`(nachrichtEmpfangen-antwortVersendet) = (500,µs)`). Zusätzlich wurde in diesem Diagramm die Leistungsanforderung des Jitters der Verarbeitungsgeschwindigkeit dargestellt und ist mittels der Annotation `Jitter (nachrichtEmpfangen-antwortVersendet)` $\leq$ `(100,µs)` modelliert. Der Jitter ist in diesem Fall als ein Zeitfenster definiert, mit der Abweichungen der idealen Reaktionsgeschwindigkeit modelliert werden können. Das heißt, dass eine Reaktionsgeschwindigkeit von 400 – 600 µs im erlaubten Rahmen liegt.

Diagramm 3.5 auf Seite 51 stellt die Leistungsanforderung der periodischen Zustandsübermittlung, sowie dessen Jitter dar (siehe Anforderungen $KS_{LA}2$ & $KS_{LA}4$). Da auch in diesem

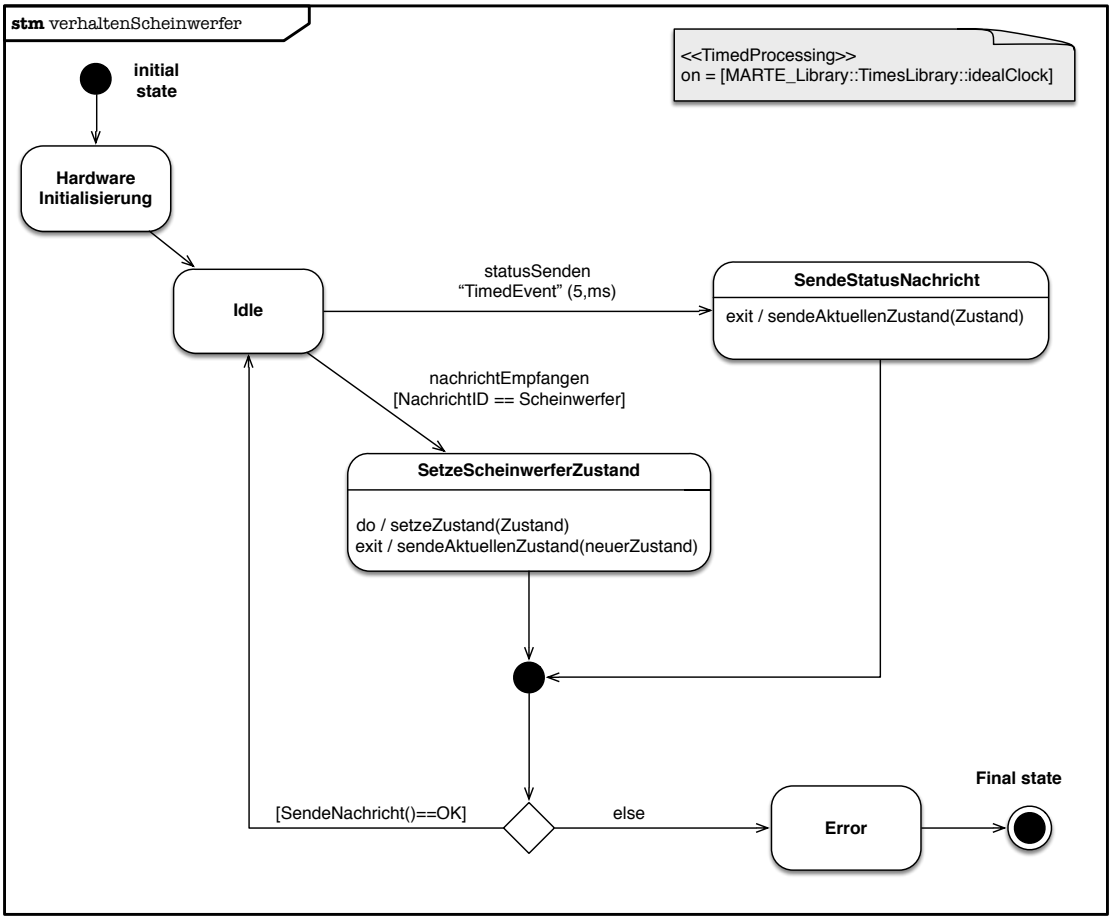


Abbildung 3.3: *UML-MARTE Zustandsautomat des Scheinwerferprozesses*

Fall eine Zeitspanne spezifiziert wird, die mit der Übertragung von Nachrichten in Verbindung steht, bietet sich das Sequenzdiagramm und das ideale Uhrenmodell an.

In diesem Diagramm werden die Leistungsanforderungen durch «`timedConstrained`» modelliert. Da in diesem Fall eine periodische Wiederholung der Übertragung stattfindet, wurde zusätzlich ein "`timedEvent`" in der `Wiederhole`-Schleife modelliert, der diese steuert. Diese Wiederholung steht im direkten Zusammenhang mit der im Diagramm 3.3 vorgestellten zeitgesteuerten Ausführung des Zustands `SendeStatusNachricht`. Auch in diesem Fall ist der Jitter als ein Zeitfenster definiert, der die Abweichung der idealen Senderate darstellt, sodass eine Senderate von 4990 – 5010 µs zulässig ist.

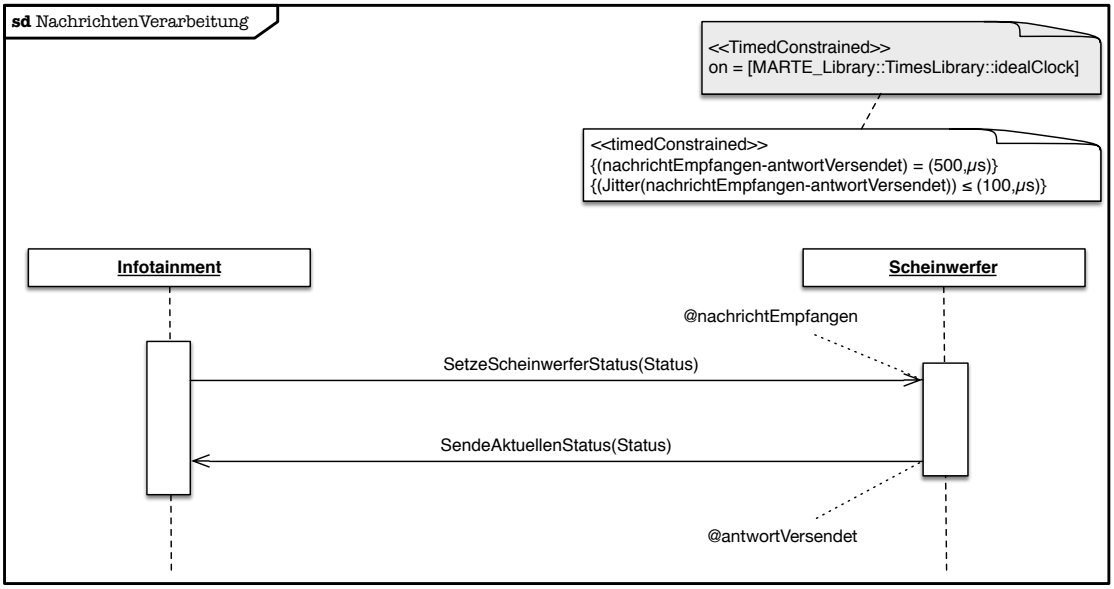


Abbildung 3.4: *UML-MARTE Sequenzdiagramm zur Modellierung der Reaktionszeit*

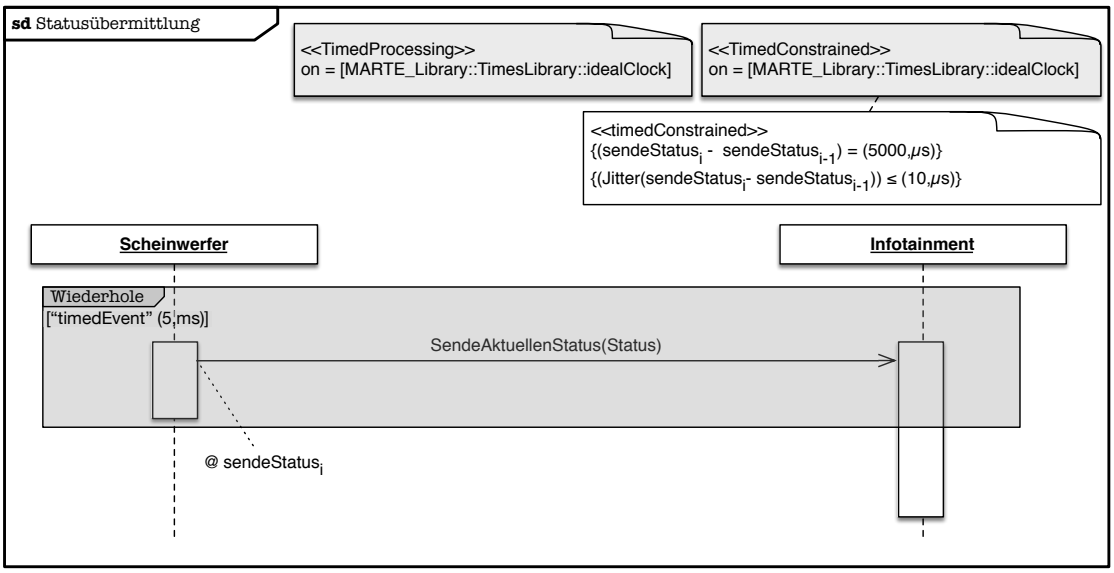


Abbildung 3.5: *UML-MARTE Sequenzdiagramm zur Modellierung der Senderate*

Die Modellierung von nicht-funktionalen Anforderungen mittels der VSL im UML-Profil MARTE stellt eine gute Möglichkeit dar, zeitliche Leistungsanforderungen auch in Echtzeit-EthernetzAnwendungen zu beschreiben. Allerdings ist die MARTE-Erweiterung momentan (Stand Januar 2014) nur für UML verfügbar, sodass nicht alle Attribute, insbesondere kontinuierliches Verhalten, von Softwaresystemen im Automobil in der gleichen Modellierungssprache abgebildet werden können. Quadri et al. hat die parallele Verwendung von MARTE und SysML in (Quadri / Sadovykh / Indrusiak 2012) beschrieben. Wenn beide Sprachen verwendet werden, kann es zu Konflikten führen wenn sowohl Konzepte von SysML, als auch von MARTE im gleichen Diagramm verwendet werden. Die vorgestellte Lösung von Quadri et al. ist die Einführung einer Abstraktionsschicht, die beide Sprachen zusammenführt. Durch die Zusammenführung beider Sprachen lassen sich komplette Automotivesoftwaresysteme modellieren, die sowohl kontinuierliche, als auch diskrete Eigenschaften sowie Echtzeitanforderungen, die durch zeitliche Leistungsanforderungen dargestellt werden, besitzen.

Unter der Verwendung der modellbasierten Testmethodik für eine Restbussimulation können diese Modelle als Spezifikation verwendet werden, um Testfälle sowohl für funktionale, als auch für Leistungsanforderungen abzuleiten. Damit die Testfälle bei einer Restbussimulation verwendet werden können, fehlen allerdings Strukturinformationen, die die Datenein- und Datenausgänge zu bestimmten Zeitpunkten des SUTs beschreiben.

3.2.3 Formale, mathematische Modellierung mittels Zustandsräumen

Eine weitere Möglichkeit, Systeme zu beschreiben, ist durch die Verwendung mathematischer Modelle gegeben. Die Zustandsraummodellierung (vgl. Lunze 2013; Skruch / Panek / Kowalczyk 2011) wird als Basis verwendet, um sowohl kontinuierliche, als auch diskrete Systeme abstrakt beschreiben zu können. Dieses Konzept weist eine praktische, kompakte und formale Darstellung auf Systeme, Einheiten, Module oder Funktionen mit mehreren Ein- und Ausgängen zu modellieren und ist in Abbildung 3.6 auf der nächsten Seite dargestellt. Das Zustandsraummodell wird u. a. in der Regelungstechnik verwendet, um dynamische Systeme analysieren zu können (vgl. Lunze 2013). Ein System besteht aus einer Menge von Eingängen $u(t)$, Zuständen $x(t)$ und Ausgängen $y(t)$, die jeweils durch Vektoren über die Zeit repräsentiert werden. Die Ein- und Ausgänge können Signale, Daten oder auch Nachrichten beschreiben, die vom System verarbeitet bzw. produziert werden. Diese Abstraktion ermöglicht es, ganze Systeme oder nur Subsysteme zu beschreiben. Es kann daher im Entwicklungsprozess im Automobil eingesetzt werden, in dem große Systeme in kleinere Subsysteme aufgeteilt werden, um die Komplexität zu beherrschen.

Die folgenden Formeln (siehe Gl.3.1 – 3.4 auf der nächsten Seite) beschreiben das Zustandsraummodell und definieren den Zusammenhang der Ein- und Ausgänge eines dynamischen Systems sowie dessen Zustände. Im weiteren Verlauf dieser Arbeit wird das formale Modell als Basis verwendet, ein abstraktes Testfallmodell (vgl. Skruch / Panek / Kowalczyk 2011)

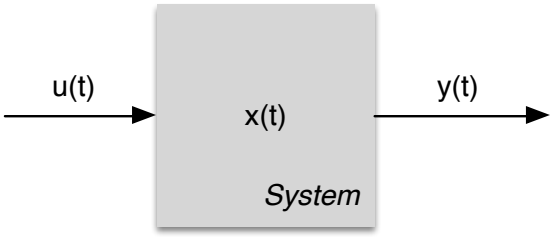


Abbildung 3.6: *Überblick des Systemmodells (nach Skruch / Panek / Kowalczyk 2011)*

für zeitliche Leistungsanforderungen zu erweitern, damit diese modelliert und überprüft werden können. Aus diesem Grund ist es wichtig, den Zusammenhang der Ein- und Ausgänge als Mehrgrößensystem darzustellen.

$$\frac{\mathrm{d}x_c(t)}{\mathrm{d}t} = f_c(t, u(t), x(t)), \quad x_c(0) = x_c^0 \tag{3.1}$$

$$x_d[k+1] = f_d(k, u(t), x(t)), \quad x_d[0] = x_d^0 \tag{3.2}$$

$$y(t) = g(t, u(t), x(t)) \tag{3.3}$$

$$x(t) = \begin{bmatrix} x_c(t) & x_d(t) \end{bmatrix}^T \tag{3.4}$$

wobei: $t > 0$, $k \in \mathbb{N}$ und:

$$f_c : \mathbb{R} \times \mathbb{R}^r \times \mathbb{R}^n \to \mathbb{R}^n$$
$$f_d : \mathbb{N} \times \mathbb{R}^r \times \mathbb{R}^n \to \mathbb{R}^n$$
$$g : \mathbb{R} \times \mathbb{R}^r \times \mathbb{R}^n \to \mathbb{R}^m$$

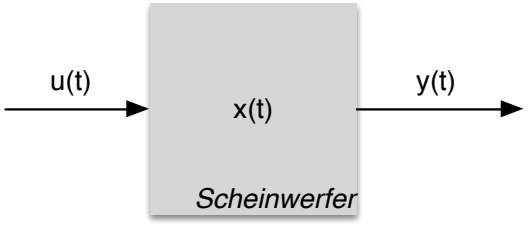
In diesem Modell ist $x(t) \in \mathbb{R}^n$ der Zustandsvektor, $u(t) \in \mathbb{R}^r$ der Eingangs- und $y(t) \in \mathbb{R}^m$ der Ausgangsvektor des Mehrgrößensystems. $f_c(\cdot)$ beschreibt die Zustandsübergangsfunktion für den kontinuierlichen Anteil und $f_d(\cdot)$ die Zustandsübergangsfunktion für den diskreten Anteil. Die Startzustände x_c^0 und x_d^0 sind festgelegt. Die Funktion $g(\cdot)$ beschreibt den Ausgang des Systems in Abhängigkeit der Zeit, des Eingangs und des aktuellen Zustands.

Der interne Zustandsvektor $x(t) = \begin{bmatrix} x_c(t) & x_d(t) \end{bmatrix}^T$ entspricht der kleinst möglichen Menge der Systemvariablen, die den aktuellen Zustand des Systems zu jeder Zeit wiedergeben. Dabei repräsentiert $x_c(t)$ die kontinuierlichen und $x_d(t)$ die diskreten Zustandsvariablen. Sowohl kontinuierliche, als auch diskrete Zustandvariablen haben Auswirkungen auf die Berechnung des nächsten Zustands (siehe Funktionen 3.1 & 3.2) und die Berechnung des Ausgangs (Funktion 3.3).

Der Eingangsvektor $u(t) = \begin{bmatrix} u_1(t) & u_2(t) & \dots & u_r(t) \end{bmatrix}^T$ besteht aus r-begrenzten Elementen, wobei diskrete Signale (z. B. das Betätigen eines Knopfes) durch die Menge

$u_i(t) = \{u_{i1}, u_{i2}, \ldots, u_{ij}\}$ und analoge Signale (z. B. eingehende Spannungen) durch das Intervall $u_i(t) \in [u_i^{min}, u_i^{max}]$ repräsentiert werden, sodass der Wertebereich aller Eingänge definiert ist. Deshalb gilt: $i = 1, 2, \ldots r$.
Der Ausgangsvektor $y(t) = \begin{bmatrix} y_1(t) & y_2(t) & \ldots & u_m(t) \end{bmatrix}^T$ besteht aus m-begrenzten Elementen. Auch in diesem Fall werden diskrete Signale durch die Menge $y_i(t) = \{y_{k1}, y_{k2}, \ldots, y_{ik}\}$ bzw. analoge Signale durch das Intervall $y_i(t) \in [y_i^{min}, y_i^{max}]$ repräsentiert. Der Wertebereich ist für alle Ausgänge definiert und es gilt: $i = 1, 2, \ldots m$.
Analog zu den Ein- und Ausgangsvektoren ist der Zustandsvektor $x(t) = \begin{bmatrix} x_1(t) & x_2(t) & \ldots & x_n(t) \end{bmatrix}^T$ n-begrenzt. Die Menge der diskreten Zustände ist durch $x_i(t) = \{x_{i1}, x_{i2}, \ldots, x_{il}\}$ dargestellt. Analoge Zustände werden durch das Intervall $x_i(t) \in [x_i^{min}, x_i^{max}]$ repräsentiert. Im Konsens zum Ein- und Ausgangsvektor ist der Wertebereich aller Zustände definiert und es gilt: $i = 1, 2, \ldots n$.

Nachfolgend wird das Zustandsraummodell am Beispiel der Scheinwerfersteuerung vorgestellt und ist in Abbildung 3.7 präsentiert. Auf die Modellierung der Zustandsübergangsfunktionen wird in diesem Fall verzichtet, da die formale Definition dieser Funktionen nicht im Fokus dieser Arbeit liegt und als Zustandsautomat (siehe Abbildung 3.3) präsentiert wurde. Wie bereits beschrieben, soll dieses Modell als Basis für die Erstellung von Testfällen dienen, sodass der Scheinwerfer als Blackbox ohne Sicht auf die internen Zustände angesehen werden kann.



u(t) = SetzeScheinwerferZustand(t)
y(t) = SendeAktuellenZustand(t)

Abbildung 3.7: *Abbildung des Scheinwerfers als Zustandsraummodell*

Da die Scheinwerfersteuerung ein diskretes System beschreibt, können Funktionen und Vektoren kontinuierlicher Attribute entfallen, sodass das Scheinwerfersystem wie folgt formal definiert werden kann:

$$x_d[k+1] = f_d(k, u(k), x(k)), \quad x_d[0] = x_d^0 \tag{3.5}$$

$$y(t) = g(t, u(t), x(t)) \tag{3.6}$$

$$x(t) = x_d(t) \tag{3.7}$$

wobei $u(t) = SetzeScheinwerferZustand(t)$ und $y(t) = SendeAktuellenZustand(t)$. Der Scheinwerfer besteht aus genau einem Ein- und einem Ausgang. Die Werte des Ein- und Ausgangs setzten sich aus der gleichen Menge zusammen. $SetzeScheinwerferZustand(t) = SendeAktuellenZustand(t) = \{BLINKER\_AN, BLINKER\_AUS \ldots\}$ Der nächste Zustand des Scheinwerfers (x) und dessen Reaktion (y) sind abhängig von der Zeit (t), dem aktuellen Eingang (u) der Nachrichten und des aktuellen Zustands(x).

Dieses Modell eignet sich gut für die formale Analyse eines modellierten dynamischen Systems, um z. B. Regelkreise überprüfen zu können (vgl. Lunze 2013). Für die Verwendung als Modellierungsprache im Entwicklungsprozess, fehlen diesem Modell allerdings grafische Notationen, die es dem Entwickler und Leser ermöglichen, das Verhalten des Systems intuitiv zu modellieren bzw. zu verstehen. Lediglich statische Attribute (Eingänge, Ausgänge, etc.) können in diesem Modell grafisch modelliert werden. Durch die eindeutig definierten Ein- und Ausgänge sowie Zustände als Vektor, ist dieses Modell als Fundament für die Erstellung von Testfällen geeignet.

3.2.4 Diskussion der Modellierungsvarianten

Die beiden vorgestellten Modellierungsvarianten stellen unterschiedliche Eigenschaften zur Modellierung von Echtzeit-Ethernet-Anwendungen bereit. Die UML-Erweiterung MARTE ist aufgrund der grafischen Notation gut geeignet während des Konstruktionsprozesses eingesetzt zu werden. Sie erlaubt es zeitliche Leistungsanforderungen innerhalb von Verhaltensmodellen, z. B. Sequenzdiagramme oder Zustandsautomaten, mit der VSL anschaulich zu beschreiben. Die Zustandsraummodellierung ist als formale Basis zur Beschreibung des Systems geeignet, da sowohl kontinuierliches, als auch diskretes Verhalten enthalten ist. Des Weiteren stellt das Modell eine Möglichkeit zur Verfügung, formal Ein- und Ausgänge und dessen Wertebereiche zu definieren.

Dementsprechend ist die Modellierung in zwei Stufen zu betrachten, bei der in den Konstruktionsphasen Modellierungsvarianten mit grafischen Darstellungsmöglichkeiten zu bevorzugen sind. Sie unterstützen den Entwicklungsprozess, da das Verhalten des Systems intuitiv zu modellieren ist. Welche Diagrammtypen sich besonders eignen, hängt von der zu entwickelnden Anwendung und dessen Anforderungen ab.
In der zweiten Stufe unterstützen formale, mathematische Modellierungsvarianten Entwickler bei der Ableitung und Erstellung von Testfällen. Insbesondere durch die formale Definition der Systemein- und ausgänge können Testfälle systematisch erstellt werden.

Testfälle werden in dieser Arbeit unter der Verwendung beider Modelle abgeleitet. Dabei werden mit dem Zustandsraummodell die Wertebereiche der Ein- und Ausgänge formal modelliert. Anhand der MARTE-Sequenz- und der MARTE-Zustandsdiagramme, in denen das Verhalten des SUTs modelliert worden ist, können die Testfälle abgeleitet werden.

3.3 Modellierung von Testfällen

Während der Systemmodellierung werden im Konstruktionsprozess Testfälle aus den Spezifikationen abgeleitet, die in den Verifikationsphasen ausgeführt werden. So werden zuerst Testfälle bestimmt, die die Anforderungen der abstrakten Benutzerspezifikation eines Systems validieren. Anschließend werden, je konkreter die Systemmodellierung voranschreitet, verfeinerte Testfälle abgeleitet, um die Implementierung zu verifizieren. In der Automotiveindustrie wird modellbasiertes Testen angewendet und die erzeugten Testfälle werden auf verschiedenen Testplattformen ausgeführt. Um dieses zu erreichen ist es wichtig eine weitere Abstraktionsstufe einzuführen, die es erlaubt Testfälle formal ohne Hintergrund zur späteren Testumgebung zu modellieren. Die Modellierung von logischen, abstrakten Testfällen, die unabhängig von der späteren Testplattform sind, wird in dieser Arbeit durch die Erweiterung eines *abstrakten Testfallmodells* ermöglicht. Der Aufbau dieses Modells ist in diesem Abschnitt beschrieben.

3.3.1 Darstellung abstrakter Testfälle

Die Modellierung von Testfällen hängt stets auch von der Modellierung des zu testenden Systems ab. Da sich die Systembeschreibung als Zustandsraummodell aufgrund der Abstraktion und Formalität anbietet, ist durch Skruch et al. ein abstraktes Testfallmodell für funktionale Anforderungen entwickelt worden (vgl. Skruch / Panek / Kowalczyk 2011). Mit diesem Modell ist es möglich, sowohl diskretes, als auch kontinuierliches Verhalten zu überprüfen. Das Testfallmodell ermöglicht es, Eingänge zu einem bestimmten Zeitpunkt mit Daten zu belegen. Außerdem kann beschrieben werden, welche Daten zu diesem Zeitpunkt am Ausgang anliegen. Es können keine Aussagen über die Reaktionszeit zwischen dem Eingang und dem Ausgang des Systems modelliert werden. Ein abstrakter Testfall ist demnach als 4-Tupel (siehe Tupel 3.8) oder falls der Zustand des SUT keine Rolle spielt, oder nicht durch die Testumgebung festgestellt werden kann (z. B. bei der Blackbox-Testmethodik), auch als Tripel (Tupel 3.9) dargestellt:

$$\begin{aligned} AS_{FA} &= (T, U, X_{soll}, Y_{soll}) \\ &\in (\mathbb{R}^{1\times k} \times \mathbb{R}^{r\times k} \times \mathbb{R}^{n\times k} \times \mathbb{R}^{m\times k}) \end{aligned} \tag{3.8}$$

$$\begin{aligned} AS_{FA} &= (T, U, Y_{soll}) \\ &\in (\mathbb{R}^{1\times k} \times \mathbb{R}^{r\times k} \times \mathbb{R}^{m\times k}) \end{aligned} \tag{3.9}$$

Analog zum Zustandsraummodell definiert U die Systemeingänge, X die Systemzustände und Y die Ausgänge des Systems. Diese Elemente des Tupels sind als Sequenz von Vektoren definiert. Zusätzlich beschreibt T die Zeitpunkte, an denen die Eingänge, die erwarteten Ausgänge sowie Zustände einen konkreten Wert vorweisen. Die Sequenzen dürfen auch leere

Elemente enthalten, wenn z. B. für einen bestimmten Zeitpunkt kein Eingang generiert wird und nur auf die Reaktion des Systems geachtet werden soll. In Relation zum Zustandsraummodell ist die Anzahl der Eingänge, Zustände und Ausgänge begrenzt: $u(t) \in \mathbb{R}^r$, $x(t) \in \mathbb{R}^n$ und $y(t) \in \mathbb{R}^m$. Die Repräsentation eines abstrakten Testfalls kann auf diese Weise folgende Form aufweisen:

$$T = \begin{bmatrix} t_0 & t_1 & \dots & t_k \end{bmatrix}$$

$$U = \begin{bmatrix} u(t_o) & u(t_1) & \dots & u(t_k) \end{bmatrix} = \begin{bmatrix} u_1(t_o) & u_1(t_1) & \dots & u_1(t_k) \\ u_2(t_o) & u_2(t_1) & \dots & u_2(t_k) \\ \vdots & \vdots & \ddots & \vdots \\ u_r(t_o) & u_r(t_1) & \dots & u_r(t_k) \end{bmatrix}$$

$$X_{soll} = \begin{bmatrix} x(t_o) & x(t_1) & \dots & x(t_k) \end{bmatrix} = \begin{bmatrix} x_1(t_o) & x_1(t_1) & \dots & x_1(t_k) \\ x_2(t_o) & x_2(t_1) & \dots & x_2(t_k) \\ \vdots & \vdots & \ddots & \vdots \\ x_n(t_o) & x_n(t_1) & \dots & x_n(t_k) \end{bmatrix}$$

$$Y_{soll} = \begin{bmatrix} y(t_o) & y(t_1) & \dots & y(t_k) \end{bmatrix} = \begin{bmatrix} y_1(t_o) & y_1(t_1) & \dots & y_1(t_k) \\ y_2(t_o) & y_2(t_1) & \dots & y_2(t_k) \\ \vdots & \vdots & \ddots & \vdots \\ y_m(t_o) & y_m(t_1) & \dots & y_m(t_k) \end{bmatrix}$$

Die Beschreibung von Testfällen in dieser Art hat zum einen den Vorteil, dass die Testfälle für funktionale Anforderungen sehr abstrakt und zugleich formal aufgestellt werden können. So ist es möglich, dass die Sequenzen von Ein- bzw. Ausgängen Sequenzen von Nachrichten, Daten oder elektrischen Signalen entsprechen.

3.3.2 Abstraktes Testfallmodell zur Verifizierung von Leistungsanforderungen

In Echtzeit-Ethernet-Netzwerken müssen bei der Verifizierung eines Systems zusätzlich Leistungsanforderungen, insbesondere an zeitliche Aussagen, wie z. B. die Reaktionsgeschwindigkeit oder die Senderate einer bestimmten Nachricht, überprüft werden. Aus diesem Grund ist die Modellierung abstrakter Testfälle mit den Parametern des letzten Abschnitts 3.3.1 auf der vorherigen Seite unzureichend, da gerade diese zeitlichen Informationen fehlen. Der Vektor T gibt dabei lediglich an, zu welchem Zeitpunkt die Eingänge mit Werten belegt werden und welche Werte an den Ausgänge bzw. welche Systemzustände zu diesem Zeitpunkt erwartet werden. Die Definitionen 3.9 und 3.8 müssen deshalb um weitere Parameter erweitert werden, die es ermöglichen zeitliche Aussagen zu überprüfen.

Wie bereits in der Modellierung von Anforderungen beschrieben, muss insbesondere in Echtzeitsystemen das Empfangen, Senden und Verarbeiten von Nachrichten die spezifizierten Zeitaussagen einhalten. Deshalb wird das Modell der abstrakten Testfälle mit den Parametern *Latenz*, *Senderate* und den jeweiligen dazugehörigen Jitter (*Jitter der Latenz* & *Jitter der Rate*) erweitert. Dabei definiert die Latenz die Reaktionsgeschwindigkeit des Systems und gibt eine maximale Zeitspanne an, die eine Erwiderung auf einen Eingang dauern darf. Die Senderate definiert in diesem Fall die maximale Zeitspanne zweier aufeinanderfolgender Signale, Daten oder Nachrichten am Ausgang der gleichen Art. Die jeweiligen Jitter definieren um diese Zeitspannen ein Zeitfenster, sodass Abweichungen der modellierten Latenz und Senderate dargestellt werden können.
Der interne Zustand spielt für die Verifizierung der zeitlichen Leistungsanforderungen in erster Linie keine Rolle und kann dementsprechend entfallen. In dieser Arbeit wird die modellbasierte Testmethodik angewendet, bei der die Testfälle auf Basis von Spezifikationen erzeugt werden. Die Implementierung einer Funktion spielt somit keine Rolle. Das Modell der abstrakten Testfälle für Leistungsanforderungen an zeitliche Aussagen ist nachfolgend als 7-Tupel dargestellt (siehe 3.10):

$$\begin{aligned} AS_{LA} &= (T, U, Y_{soll}, L_{soll}, R_{soll}, J_{Lsoll}, J_{Rsoll}) \\ &\in (\mathbb{R}^{1\times k} \times \mathbb{R}^{r\times k} \times \mathbb{R}^{m\times k} \times \mathbb{R}^{a} \times \mathbb{R}^{b} \times \mathbb{R}^{y} \times \mathbb{R}^{z}) \end{aligned} \tag{3.10}$$

Hierbei entsprechen die ersten drei Parameter denjenigen des vorherigen Modells (siehe Definition 3.9 auf Seite 56). Zusätzlich werden die Vektoren $L_{soll}, R_{soll}, J_{Lsoll}, J_{Rsoll}$ definiert. Sie beschreiben die Latenz (L_{soll}), die Senderate (R_{soll}) und den dazugehörigen Jitter der Latenz (J_{Lsoll}) sowie den Jitters der Senderate (J_{Rsoll}). Für ein Element aus der Leistungsanforderung der Latenz gilt: $l_a(u_g, y_h) \in L_{soll}$, $g = 1,2,\dots r$, $h = 1,2,\dots m$ und $a \geq 1$. Das bedeutet, dass die Latenz aus einem Ein-/Ausgangspaar definiert ist und mindestens ein Element enthalten muss. Für ein Element aus den Leistungsanforderungen der Senderate R_{soll} gilt analog: $r_b(y_h) \in R_{soll}$, $h = 1,2,\dots m$ und $b \geq 1$. Für ein Element aus den Leistungsanforderungen des Jitters der Latenz J_{Lsoll} gilt weiterhin: $j_{Ly}(l_p) \in J_{Lsoll}$, $p = 1,2,\dots a$ und $y \geq 1$. Auf diese Weise wird die Relation des Jitters zur Latenz modelliert. Außerdem gilt für ein Element der Anforderungen des Jitters der Rate J_{Rsoll} analog: $j_{Rz}(r_q) \in J_{Rsoll}$, $q = 1,2,\dots b$ und $z \geq 1$. Im Gegensatz zu den Ein- (U) und Ausgängen (Y) sind die Latenz, die Senderate und die dazugehörigen Jitter nicht als Sequenz in Abhängigkeit von T definiert, sodass sie für den gesamten Testfall gültig sind.

Mit diesem Modell ist es möglich, neben den zu erwartenden Ausgängen eines Systems, die erwartete Latenz eines Ein-/Ausgangspaars zu definieren. Die erwartete Senderate eines Ausgangs wird ebenfalls dargestellt. Gleichzeitig erlaubt das Modell die dazugehörigen Jitter der Latenz und der Rate zu beschreiben, die die Abweichungen der idealen Zeitspannen darstellen. Ein abstrakter Testfall für die Validierung zeitlicher Leistungsanforderungen kann

formal folgendermaßen als Matrix modelliert werden und ist anschaulich in der Tabelle 3.10 auf der nächsten Seite dargestellt:

$$T = \begin{bmatrix} t_0 & t_1 & \dots & t_k \end{bmatrix}$$

$$U = \begin{bmatrix} u_1(t_o) & u_1(t_1) & \dots & u_1(t_k) \\ u_2(t_o) & u_2(t_1) & \dots & u_2(t_k) \\ \vdots & \vdots & \ddots & \vdots \\ u_r(t_o) & u_r(t_1) & \dots & u_r(t_k) \end{bmatrix}$$

$$Y_{soll} = \begin{bmatrix} y_1(t_o) & y_1(t_1) & \dots & y_1(t_k) \\ y_2(t_o) & y_2(t_1) & \dots & y_2(t_k) \\ \vdots & \vdots & \ddots & \vdots \\ y_m(t_o) & y_m(t_1) & \dots & y_m(t_k) \end{bmatrix}$$

$$L_{soll} = \begin{bmatrix} l_1(u_1, y_1) \\ l_2(u_1, y_2) \\ \vdots \\ l_a(u_r, y_m) \end{bmatrix}$$

$$R_{soll} = \begin{bmatrix} r_1(y_1) \\ r_2(y_2) \\ \vdots \\ r_b(y_m) \end{bmatrix}$$

$$J_{Lsoll} = \begin{bmatrix} j_{L_1}(l_1) \\ j_{L_2}(l_2) \\ \vdots \\ j_{L_y}(l_a) \end{bmatrix}$$

$$J_{Rsoll} = \begin{bmatrix} j_{R_1}(r_1) \\ j_{R_2}(r_2) \\ \vdots \\ j_{R_z}(r_b) \end{bmatrix}$$

T	t_0	t_1	$\dots$	t_k
U	$u_1(t_0)$	$u_1(t_1)$	$\dots$	$u_1(t_k)$
	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	$u_r(t_0)$	$u_r(t_1)$	$\dots$	$u_r(t_k)$
Y_{soll}	$y_1(t_0)$	$y_1(t_1)$	$\dots$	$y_1(t_k)$
	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	$y_m(t_0)$	$y_m(t_1)$	$\dots$	$y_m(t_k)$
L_{soll}		$l_1(u_1, y_1)$		
		$\vdots$		
		$l_a(u_r, y_m)$		
R_{soll}		$r_1(y_1)$		
		$\vdots$		
		$r_b(y_m)$		
J_{Lsoll}		$j_{L_1}(l_1)$		
		$\vdots$		
		$j_{L_y}(l_a)$		
J_{Rsoll}		$j_{R_1}(r_1)$		
		$\vdots$		
		$j_{R_z}(r_b)$		

Tabelle 3.10: *Abstrakter Testfall in Tabellennotation*

Wichtig bei diesem Modell zu erwähnen ist, dass der Beobachtungspunkt des zeitlichen Verhaltens technisch bedingt beim Testgerät (in diesem Fall der Restbussimulator) liegt (siehe Abbildung 3.8 auf der nächsten Seite). Die Zeitpunkte, an denen Nachrichten an das SUT versendet werden, müssen am Restbussimulator gemessen werden. Das SUT wird als Blackbox angesehen, sodass man keinen Zugang zum physikalischen Eingang besitzt. Das heißt, dass zu dem spezifizierten Zeitverhalten zusätzlich noch die jeweiligen möglichen physikalischen Übertragungslaufzeiten (bei 100 $^{\text{MBit}}/_{\text{s}}$ z. B. 5,12 µs für einen Ethernetframe mit 64Byte) mit einbezogen werden müssen. Das trifft insbesondere zu, wenn die Verarbeitungszeit, also die Latenz eines Systems, überprüft werden soll, da in der Regel die zeitlichen Einschränkungen der Reaktionsgeschwindigkeit am jeweiligen Objekt hinzugefügt (siehe z. B. Sequenzdiagramm 3.4 auf Seite 51) werden.

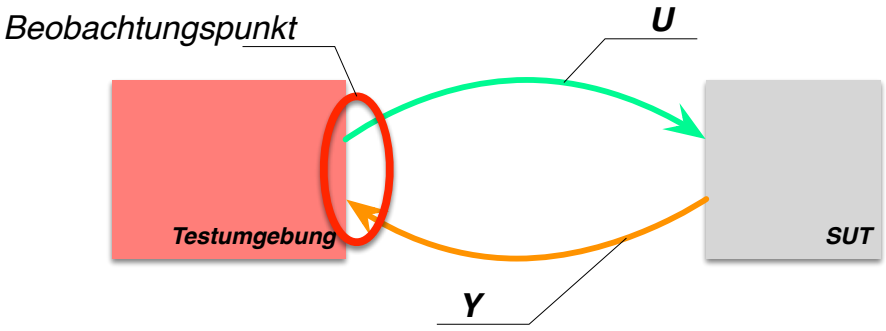


Abbildung 3.8: *Beobachtungspunkt bei der Überprüfung der zeitlichen Leistungsanforderungen*

Im Kontext der Scheinwerfersteuerung wird nachfolgend ein abstrakter Testfall für dieses System als Beispiel vorgestellt, der zum einen funktionale Anforderungen und zum anderen zeitliche Leistungsanforderungen modelliert. Der Testfall ist in Tabelle 6.1 auf Seite 98 dargestellt und stellt einen Testfall zur Überprüfung der Funktionalität des Abblendlichts sowie zur Überprüfung zeitlicher Leistungsanforderungen der Reaktionsgeschwindigkeit und der Senderate der periodischen Zustandsübermittlung dar.

T	1s	5s	9s
U	$u_1 =$ ALLES_AUS	$u_1 =$ BLINKER_EIN	$u_1 =$ BLINKER_AUS
Y_{soll}	$y_1 =$ ALLES_AUS	$y_1 =$ BLINKER_EIN	$y_1 =$ ALLES_AUS
L_{soll}	$l_1(u_1, y_1) = 500\mu s$		
R_{soll}	$r_1(y_1) = 5000\mu s$		
J_{Lsoll}	$j_{L1}(l_1) \leq 100\mu s$		
J_{Rsoll}	$j_{R1}(r_1) \leq 10\mu s$		

Tabelle 3.11: *Beispiel eines abstrakten Testfalls in Tabellennotation*

Die erstellten Testfälle können anschließend, wie beim modellbasierten Testen üblich, auf einer speziellen Testplattform, z. B. bei einer Restbussimulation, ausgeführt werden. Dazu werden die abstrakten Testfälle innerhalb der Testplattform konkret in ausführbare Fälle umgewandelt, damit sie den Aspekten der jeweiligen Plattform entsprechen.

4 Anforderungen an eine Echtzeit-Ethernet Restbussimulation

Nachdem im vorherigen Kapitel Modellierungsvarianten für Echtzeit-Ethernet Anwendung im Automobil vorgestellt wurden, befasst sich dieses Kapitel mit dem Thema *Echtzeit-Ethernet Restbussimulation*. Zuerst wird deshalb ein Vergleich zu bisherigen Restbussimulationsansätzen präsentiert. Aufbauend auf den Unterschieden und den in den Grundlagen präsentierten Eigenschaften des Übertragungsprotokolls TTEthernet werden die Anforderungen und benötigten Komponenten an eine Echtzeit-Ethernet basierte Restbussimulation gestellt.

4.1 Vergleich zu bisherigen Restbussimulationsansätzen

Aufgrund der Tatsache, dass in etablierten Fahrzeugnetzwerkstrukturen hauptsächlich Feldbusse eingesetzt werden, unterscheiden sich etablierte und zukünftige Restbussimulatiosansätze für Echtzeit-Ethernet-Netzwerke. Die Unterschiede und Gemeinsamkeiten werden in diesem Abschnitt erläutert.

4.1.1 Medienzugriffsverfahren und Topologien in busbasierten Restbussimulationen

Es wurden Restbussimulatoren (z. B. CANoe) für die im Automobil etablierten Netzwerktechnologien LIN, CAN und FlexRay entwickelt. Diese Technologien basieren auf einer feldbusbasierten Kommunikationsarchitektur, die einen gemeinsamen Zugriff auf das Übertragungsmedium vorgibt. Jeder Teilnehmer befindet sich dementsprechend in der gleichen Kollisionsdomäne, sodass nur eine Nachricht zur Zeit kollisionsfrei übertragen werden kann, unabhängig von der Anzahl der Teilnehmer im Netzwerk. Gängige Strategien für diese Umsetzung sind CSMA/CR (z. B. CAN), bei der die höchstpriorisierte Nachricht verzögerungsfrei übertragen werden kann und zeitlich koordiniertes TDMA (z. B. bei FlexRay), bei dem jeder Teilnehmer ein Timeslot zur Übertragung zur Verfügung hat.
Im Gegensatz zum gemeinsamen Medienzugriffsverfahren bei busbasierten Technologien erlaubt Echtzeit-Ethernet die parallele Übertragung von Nachrichten im gleichen Netzwerk. Echtzeit-Ethernet baut auf Switched Ethernet auf und erweitert es um Echtzeitfähigkeit.

Jeder Teilnehmer befindet sich bei Switched Ethernet in einer eigenen Kollisionsdomäne, sodass dieser zu jeder Zeit eine Datenübertragung starten kann. Die Datenströme werden vom Switch an verschiedenen Ports empfangen, mittels einer passenden Queing-Strategie integriert (z. B. priorisiertes Queing (vgl. IEEE 802.1 TSN Task Group [a]) und an den jeweiligen Ausgangsport geleitet. Parallele Datenströme von verschiedenen Teilnehmern werden demnach gleichzeitig im Switch weitergeleitet.
Auf die Weiterleitungsstrategie muss insbesondere geachtet werden, wenn mehrere zu testende Systeme zu einem gemeinsamen SUT zusammengefasst und getestet werden. Während sich der Testaufbau bei einer busbasierten Restbussimulation nicht ändert und auch bei mehreren SUTs identisch ist, ergeben sich bei Echtzeit-Ethernet basierten Restbussimulationen mehrere Möglichkeiten, die SUTs mit dem Restbussimulator zu koppeln.

Topologieaufbau bei einer busbasierten Restbussimulation

Der allgemeine Aufbau einer Restbussimulation für busbasierte Netzwerke ist in Abbildung 4.1 dargestellt und besteht aus dem Restbussimulator, den SUTs und dem Bus als Übertragungsmedium, an dem alle Teilnehmer angeschlossen sind. Thomas M. Galla hat in seiner Dissertation (vgl. Galla 1999) beschrieben, dass der Restbussimulator eine höhere Performance erreicht, wenn dieser mit mehreren Netzwerkinterfaces ausgestattet wird.

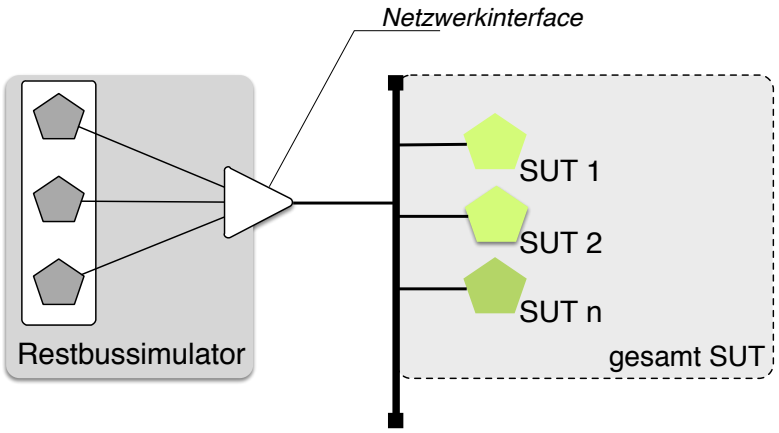


Abbildung 4.1: *Testaufbau einer Busbasierten Topologie (nach Galla 1999)*

Restbussimulationstopologien bei Echtzeit-Ethernet

Im Gegensatz zur einheitlichen Topologie einer busbasierten Restbussimulation ist die Topologie bei einer Echtzeit-Ethernet Restbussimulation abhängig von der Anzahl zu testender

Geräte. So lassen sich die Topologien in zwei Ansätze klassifizieren. Der erste Ansatz basiert auf einer direkten Verbindung zwischen dem Simulator und dem SUT. Beim zweiten wird der Restbussimulator indirekt über einen zusätzlichen Echtzeit-Ethernet Switch mit dem SUT verbunden. In einer getätigten Vorarbeit (vgl. Bartols 2013) wurden beide Klassen von der technischen und ökonomischen Seite verglichen und diskutiert, wobei sich drei relevante Aufbauten herausgestellt haben. Andere Topologien sind zwar denkbar aber entweder zu teuer, zu komplex oder besitzen Einschränkungen bei der geforderten Echtzeitfähigkeit, speziell wenn SUTs untereinander kommunizieren. Nachfolgend sind deshalb nur die relevanten Topologien beschrieben.

Zuerst wird der triviale Fall einer Echtzeit-Ethernet-basierten Restbussimulation beschrieben, der in Abbildung 4.2 dargestellt ist. Hierbei besteht der Restbussimulator aus einem Netzwerkinterface, das direkt mit dem SUT verbunden ist und ist physikalisch auf ein SUT begrenzt. Dieser Ansatz wird während des Komponententests eingesetzt.

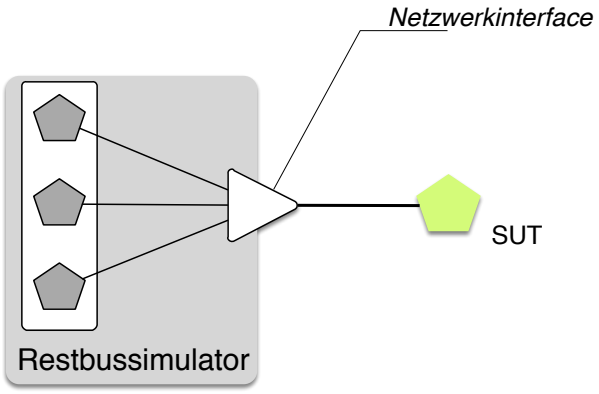


Abbildung 4.2: *Testaufbau einer direkt verbundenen Echtzeit-Ethernet Restbussimulation*

Der Vorteil dieses Ansatzes liegt an der einfachen Umsetzbarkeit. Bei der Konfiguration ist die Anzahl der zu generierenden Nachrichten auf ein SUT begrenzt und eine parallele Übertragung von unabhängigen Nachrichten ausgeschlossen. Außerdem kann der statische Schedule der TT-Nachrichten direkt vom SUT abgeleitet werden. Der Nachteil dieses Ansatzes ist, dass er nicht für Integrationstests verwendet werden kann, wenn mehrere Systeme zusammengefasst und als zusammenhängendes SUT betrachtet werden.

Damit eine Restbussimulation auch während der Intergrationsphase verwendet werden kann, bedarf es einer Möglichkeit, mehr als nur ein SUT mit dem Simulator zu koppeln. Die einfachste Möglichkeit dieses zu erreichen, ist die Verwendung eines zusätzlichen Echtzeit-Ethernet Switches, der es ermöglicht mehrere SUTs zu koppeln und ist in Abbildung 4.3

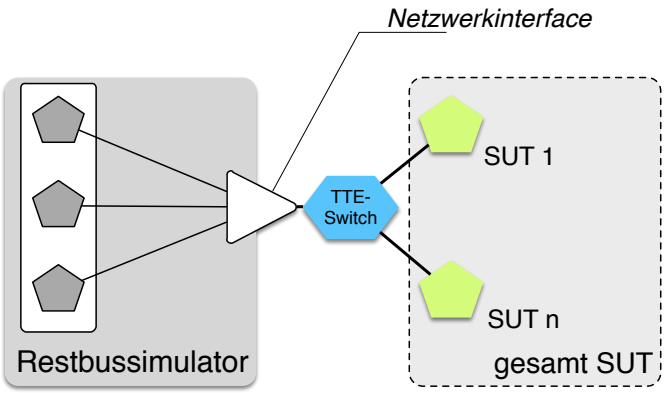


Abbildung 4.3: *Testaufbau einer Echtzeit-Ethernet Restbussimulation mit einem zusätzlichem Switch für mehrere mehrere SUT*

präsentiert. Der Simulator verwendet weiterhin nur ein Netzwerkinterface, um die Verbindung zum SUT herzustellen.

Wenn ein SUT aus mehreren Subsystemen zusammengesetzt ist, besteht bei einer Echtzeit-Ethernet Restbussimulation die Möglichkeit, dass die Subsysteme unabhängig voneinander verschiedene Nachrichten zum gleichen Zeitpunkt versenden bzw. empfangen. Dadurch kann es zum einen zu *Konflikten beim Scheduling* von TT-Nachrichten am Simulator kommen und zum anderen zu *Bandbreitenkonflikten*, da die Ethernetverbindungen der Subsysteme am Switch zusammengefasst werden.
Schedulingkonflikte können auftreten, wenn unterschiedliche TT-Nachrichten an unterschiedlichen Subsystemen zum gleichen Zeitpunkt empfangen werden müssen. Eine parallele Übertragung der Nachrichten ist physikalisch ausgeschlossen, da der Restbussimulator mit nur einem Netzwerkinterface an das SUT gekoppelt ist. Die Nachrichten müssen sequentiell vom Simulator versendet werden, sodass anschließend der Echtzeit-Ethernet Switch die parallele Übertragung dieser Nachrichten durchführt. Schedulingkonflikte können auch auf dem Weg vom Subsystem zum Restbussimulator auftreten, wenn parallel unterschiedliche Nachrichten zum Restbussimulator gesendet werden. Die unterschiedlichen Nachrichten werden im Switch integriert und anschließend sequentiell zum Restbussimulator übertragen.
Bandbreitenkonflikte können aufgrund der Integration mehrerer Ethernet-Verbindungen auftreten, insbesondere wenn die Subsysteme viel Bandbreite belegen. Deshalb muss darauf geachtet werden, dass die verwendete Bandbreite aller Subsysteme geringer ist als die vorhandene Bandbreite zwischen dem Restbussimulator und dem Echtzeit-Ethernet Switch.

Bei dieser Topologie ist zu beachten, dass das Zeitverhalten der Nachrichten beeinflusst wird, wenn diese im Switch weitergeleitet werden. Daher ist diese Topologie bei der Überprü-

fung zeitlicher Leistungsanforderungen nicht verwendbar. Für die Überprüfung funktionaler Anforderungen kann sie jedoch im Integrationsprozess verwendet werden.

Falls die Scheduling- und Bandbreitenkonflikte zu gravierend sind, muss der Simulator mit der gleichen Anzahl an Netzwerkinterfaces ausgestattet werden, wie es Subsysteme gibt. Dieser Aufbau ist in Darstellung 4.4 abgebildet. Mit diesem Aufbau können die Scheduling und Bandbreitenkonflikte gelöst werden, da keine Integration der Nachrichten mehr erforderlich ist.

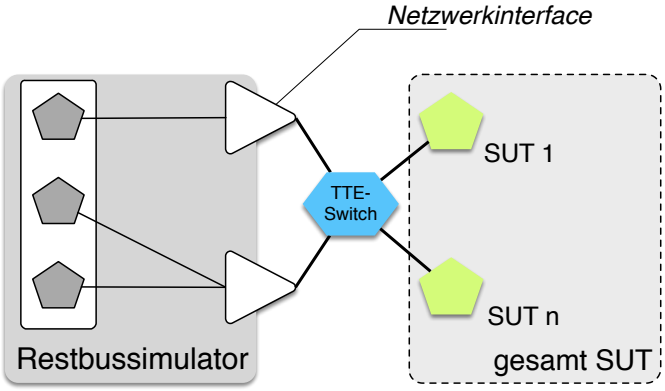


Abbildung 4.4: *Testaufbau einer Echtzeit-Ethernet Restbussimulation mit mehreren Netzwerkinterfaces für mehrere SUT*

Andererseits muss darauf geachtet werden, dass die durch den Simulator generierten Nachrichten an die richtigen Netzwerkinterfaces gereicht werden, sodass die Konfiguration und Implementierung einer solchen Topologie komplexer, im Vergleich zu der Topologie mit nur einem Interface, ausfällt.

4.1.2 Verschiedenen Nachrichtenklassen und Synchronisationsprozess

Ein weiterer Unterschied zwischen Bus- und Echtzeit-Ethernet basierten Restbussimulationen besteht darin, dass das verwendete Derivat TTEthernet drei verschiedenen Nachrichtenklassen mit unterschiedlichen Eigenschaften und Prioritäten unterstützt (siehe Abschnitt 2.3.3 auf Seite 25). Bis auf FlexRay unterstützen alle anderen im Automobil verwendeten Technologien nur eine Nachrichtenklasse, dessen Priorität anhand der Nachrichten- bzw. der Slot-ID vorgegeben wird. FlexRay unterstützt neben den konfigurierten, zeitgesteuerten Nachrichten im statischen Segment eine ereignisbasierte Kommunikation im dynamischen Segment.
Echtzeit-Ethernet unterscheidet zwischen drei verschiedenen Nachrichtenklassen, die bei einer Echtzeit-Ethernet Restbussimulation berücksichtigt werden müssen. Daraus folgt, dass

die Komplexität im Vergleich zu einer busbasierten Restbussimulation höher ist. So müssen neben der zeitgesteuerten Übertragung von TT-Nachrichten, bandbreitenlimitierende RC-Nachrichten und unkritische BE-Nachrichten unterstützt werden. Zudem ist es bei der Nachrichtengenerierung simulierter Knoten wichtig, dass hochpriore TT-Nachrichten immer Vorrang haben und nicht durch die Übertragung anderer Nachrichten gestört werden.

Auch bei der Zeitsynchronisierung gibt es Unterschiede, die die Ausführung einer Restbussimulation beeinflussen. Das TTEthernet Protokoll definiert drei verschiedene Rollen im Synchronisierungsprozess, die je nach Rolle verschiedene Funktionen ausüben. Im Gegensatz zu FlexRay, bei der Nachrichten mit regulärem Dateninhalt im statischen Segment für die Zeitsynchronisierung genutzt werden, werden bei TTEthernet dedizierte „Protocol Control Frames“ zur Übertragung verwendet. Sie können nicht für Nutzdaten genutzt werden. Der Restbussimulator muss dementsprechend das jeweilige passende Gegenstück im Synchronisierungsprozess ausführen und im Fall des „Synchronization Master“ (Abschnitt 2.3.3 auf Seite 25) PCF versenden, damit das SUT aus dem unsynchronisierten in den synchronisierten Zustand überführt wird. Teilnehmer im TTEthernet führen erst im synchronisierten Zustand ihre eigentliche Funktion aus.

4.2 Anforderungen und nötige Komponenten einer Echtzeit-Ethernet-Restbussimulation

Damit eine Echtzeit-Ethernet Restbussimulation durchgeführt werden kann, müssen essentielle Komponenten vorhanden sein. Sie beschränken sich nicht nur auf eine passende Hardware-/Softwareplattform, die die Restbussimulation ausführen, sondern auch auf unterstützende Datenformate. Diese enthalten sowohl statische, als auch dynamische Informationen und werden dazu verwendet, die Konfiguration zu erstellen.

4.2.1 Anforderungen an eine geeignete Hardware-/Softwareplattform

Die wichtigste Komponente bei einer Restbussimulation ist eine geeignete Hardware-/Softwareplattform. Einerseits muss sie protokollspezifisch mit dem SUT zu kommunizieren und andererseits eine Möglichkeit bieten, das Simulationsmodell des „restlichen Netzwerks“ auszuführen. Die Hardware muss genügend Rechenleistung bereitstellen, um sowohl das echtzeitfähige Protokoll entsprechend auszuführen, als auch bei der Ausführung des Simulationsmodells performant zu arbeiten. Auf Seite der Software sollte der Simulator einfach konfigurierbar sein, um möglichst flexibel verschiedene Verhaltensmuster darstellen zu können, wodurch Entwicklungszeit und dementsprechend Entwicklungskosten eingespart werden.

Echtzeit-Ethernet Restbussimulationen müssen im Kommunikationsprotokoll zwischen TT-, RC- und BE-Nachrichten unterscheiden. Um vollständige Kompatibilität zu bieten, müssen alle Nachrichtenklassen von der Plattform unterstützt werden. TT-Nachrichten dürfen unter keinen Umständen bei der Übertragung verzögert werden, sodass deren konfigurierter Timeslot nicht verpasst werden darf. Bei RC-Nachrichten ist darauf zu achten, dass sie gemäß ihrer konfigurierten Bandbreiterestriktion (Bandwidth-Allocation-Gap) übertragen werden. Zudem ist darauf zu achten, dass die Hierarchie der Nachrichtenklassen beachtet wird und hochpriore Nachrichten Vorrang haben. BE-Nachrichten werden bei der Simulation durch TT- und RC-Nachrichten verdrängt. Des Weiteren muss der Zeitsynchronisierungsdienst von der Plattform unterstützt werden und alle im Sychronisierungsprozess beteiligten Rollen ausgeführt werden können. Damit die zeitlichen Leistungsanforderungen der Latenz und Senderate der Nachrichten überprüft werden können, muss die Hardwareplattform zu dem in der Lage sein, versendete und empfangene Nachrichten mit einem präzisen Zeitstempel zu versehen. Wie angesprochen soll die Softwareplattform mittels Werkzeugen einfach zu konfigurieren sein. Auf der einen Seite muss die Softwareplattform eine Möglichkeit zur statischen Konfiguration der Netzwerkattribute bereitstellen, sodass die Restbussimulationsplattform für andere SUTs im gleichen oder in anderen Netzwerken verwendet werden kann. Auf der anderen Seite ist es wichtig, dass das Verhalten des Restbussimulators möglichst einfach angepasst werden kann, um verschiedene Szenarien schnell und effizient ausführen zu können, ohne die statische Konfiguration zu ändern. Der Restbussimulator benötigt deswegen definierte Schnittstellen, die es erlauben, beide Attribute klar definieren zu können.

Generell eignen sich bei der Umsetzung der Kommunikationseigenschaften alle Hardwareplattformen, für die ein vollständig implementierter Echtzeit-Ethernet-Stack vorhanden ist. Allerdings muss die Softwareplattform zusätzlich in der Lage sein, die geforderte Flexibilität bereitzustellen, damit der Restbussimulator für verschiedene Anwendungen und Systeme verwendet werden kann. Wichtig dabei ist, dass die Hardwareplattform den Echtzeit-Ethernet-Stack dabei unter Echtzeitbedingungen ausführt, damit das Übertragen der TT-Nachrichten protokoll- und konfigurationsspezifisch ermöglicht wird. Die Ausführung des Simulationsmodells darf außerdem unter keinen Umständen zu viel Rechenzeit benötigen, damit das Verhalten des „restlichen Netzwerks“ nicht negativ beeinflusst wird.

4.2.2 Anforderungen an die Konfigurationsmöglichkeit der Restbussimulation

Eine benutzerfreundliche Konfigurationsmöglichkeit ist ein wichtiger Bestandteil der Restbussimulation. Zum einen werden Informationen benötigt, die die *statische Konfiguration* ermöglichen. Diese beschreiben die Eigenschaften, die sich während der Restbussimulation nicht ändern und können als Gerüst der Simulation angesehen werden. Zu diesen Informationen gehören u. a. der Aufbau der zu sendenden Daten und dessen Zeitinformationen.

Zum anderen werden Informationen benötigt, die für die *dynamische Konfiguration* verwendet werden, um das Verhalten einer Restbussimulation zu beschreiben. Das entspricht der Erstellung des Simulationsmodells.

Datenmodell zur statischen Konfiguration

Eine Möglichkeit die statische Konfiguration der Restbussimulation durchzuführen, besteht in der Verwendung eines passenden Datenmodells. Es soll das Fahrzeugnetzwerk, alle Teilnehmer sowie die transferierten Nachrichten und deren Inhalte beschreiben. Durch Auswahl des SUT kann anschließend die statische Konfiguration werkzeugunterstützt generiert werden.

Für CAN Netzwerke war das von Vector Informatik entwickelte proprietäre CANdb Format (vgl. Vector Informatik) der Standard, um CAN-basierte Netzwerkstrukturen vollständig zu beschreiben. Vollständig in diesem Kontext bedeutet, dass das Netzwerk mit allen Teilnehmern, Nachrichten, deren Übertragungsstrategien sowie Inhalt beschrieben wird. Mittlerweile wird es durch den offenen Standard „Fieldbus Data Exchange Format“ (FIBEX) abgelöst, das auch unter den Namen ASAM MCD-2 NET bekannt ist (vgl. ASAM 2013). Es ist ein XML-basiertes Datenformat, das zum Datenaustausch zwischen verschiedenen Konfigurations- und Analysewerkzeugen im Automotivekontext eingesetzt wird und alle etablierten Bussysteme, sowie standard Ethernet unterstützt.
Die FIBEX-Spezifikation ist objektorientiert umgesetzt und stellt für alle Attribute eines Fahrzeugnetzwerks generische Datentypen bereit. Die jeweiligen Technologiespezifikationen erben von den generischen Datentypen und erweitern sie mit ihren charakteristischen Eigenschaften. Da Echtzeit-Ethernet in der Spezifikation von FIBEX noch nicht unterstützt wird, wurde in einer Vorleistung (vgl. Bartols 2012) eine Erweiterung implementiert, durch die es möglich ist, Echtzeit-Ethernet Netzwerke die durch TTEthernet repräsentiert werden, vollständig mit allen charakteristischen Eigenschaften zu beschreiben.

Die Architektur der *fibex4ttethernet* Implementierung ist in Abbildung 4.5 auf der nächsten Seite dargestellt. Der FIBEX-Core *fibex* stellt alle generischen Datenstrukturen zur Beschreibung von Fahrzeugnetzwerken bereit. *Fibex4ttethernet* erbt zusätzlich die technologiespezifischen Ethernet-Eigenschaften aus *fibex4ethernet* und erweitert diese um die charakteristischen Merkmale des Echtzeit-Ethernet-Protokolls.

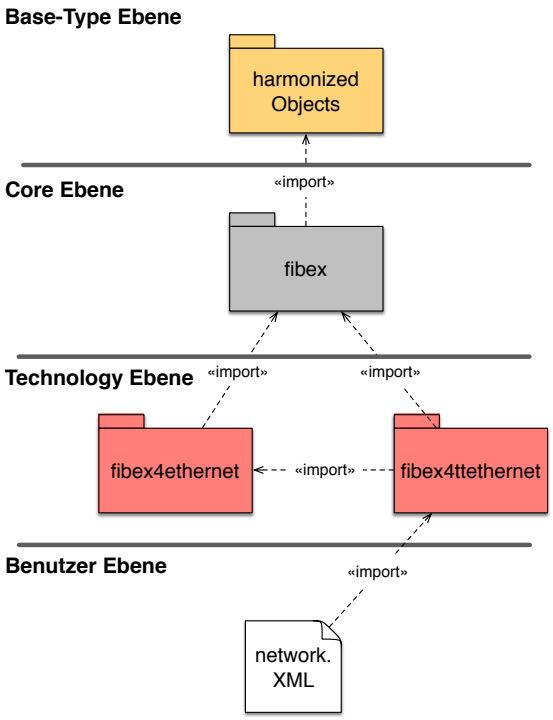


Abbildung 4.5: *Architektur des FIBEX4TTEthernet Datenmodells für Echtzeit-Ethernet-Netzwerke*

Dabei musste das zeitgesteuerte Verhalten des Echtzeit-Ethernet Protokolls dargestellt werden. Timeslots in busbasierten Netzwerken sind global gültig und werden deshalb für das gesamte Netzwerk konfiguriert. Zum Zeitpunkt der Umsetzung unterstützte die FIBEX Version 4.1 nur zeitgesteuerte Feldbusnetzwerke, sodass die bereitgestellten generischen Attribute des FIBEX-Cores nicht im gleichen Kontext verwendet werden konnten. In TTEthernet müssen aufgrund der eigenen Kollisionsdomäne die Timeslots zwischen zwei Teilnehmern konfiguriert werden. Die Konfiguration ist dadurch nicht mehr global gültig und wird an den Netzwerkschnittstellen aller Teilnehmer durchgeführt.

Dieses ist erreicht worden, indem die vom FIBEX-Core bereitgestellten Datentypen für zeitgesteuertes Verhalten in den Endsystemen und Switches implementiert worden sind, sodass das Scheduling der Nachrichten zwischen Teilnehmern konfigurierbar ist. Weiterhin sind die charakteristischen Eigenschaften der Zeitsynchronisierung und die Definition virtueller Links eingeführt, sodass Echtzeit-Ethernet-Netzwerke vollständig in FIBEX beschrieben und anschließend zur Erstellung der statischen Konfiguration verwendet werden können.

Modell zur dynamischen Verhaltenskonfiguration

Neben der statischen Konfiguration der Restbussimulation muss auch das Verhalten zu konfigurieren sein. Es wird dazu verwendet, die statisch konfigurierten Nachrichtengerüste mit sinnvollen Inhalten zu versehen. In der Regel wird das Verhalten durch ein entwickeltes Simulationsmodell vorgegeben, das auf dem Simulator ausgeführt wird. Je nach Datengenerierungstechnik (siehe Abschnitt 2.2.4 auf Seite 19) kann das Verhalten entweder zur Laufzeit berechnet oder offline generiert werden.

Um das Verhalten der simulierten Knoten bei der Ausführung von Testfällen zu beschreiben, bieten sich offline-berechnete Daten an, da das Verhalten vorher genau geplant werden kann und unerwartetes Simulationsverhalten während der Ausführung ausgeschlossen ist. Auf diese Weise wird erreicht, dass das SUT gegen die gestellten Anforderungen überprüft werden kann. Soll eine Restbussimulation als Rapidprototyping-Plattform verwendet werden, müssen die Daten, die das Verhalten der simulierten Knoten darstellen, zur Laufzeit berechnet werden. Die Erstellung von offline-berechneter Daten ist allerdings sehr aufwendig wenn alle Bedingungen und Fälle abzubilden sind. In dieser Arbeit liegt der Fokus auf der Ausführung von Testfällen. Daher sollte das Modell eine Beschreibung für offline berechnetes Verhalten bereitstellen.
Damit dieses bei Testfällen spezifiziert werden kann, muss bestimmt werden, zu welchen Zeitpunkten Nachrichten generiert werden und welche Daten diese Nachrichten beinhalten. Die in dieser Arbeit entwickelte Erweiterung des abstrakten Testfallmodells (siehe Abschnitt 3.3 auf Seite 56) ist für die Verhaltensbeschreibung geeignet. Durch die explizite Angabe der Ein- und Ausgänge des SUT, sowie der Zeitpunkte, an denen Nachrichten am Eingang des SUTs gesendet werden, kann das Verhalten der simulierten Knoten dementsprechend genau geplant werden. Für jeden Testfall werden die Informationen der abstrakten Testfälle neu bestimmt, um ein gewünschtes Verhalten des Restbussimulators zu erzeugen.
Auf diese Weise lassen sich mit den entwickelten abstrakten Testfällen zwei verschiedene Aspekte einer Restbussimulation beschreiben. Zum einen dienen sie als Simulationsmodell und zum anderen werden sie zur Überprüfung des SUT verwendet, sodass ein aufwendiges Modellieren beider Aspekte entfällt und die Durchführung einer Restbussimulation in dieser Hinsicht vereinfacht wird.

4.2.3 Anforderungen an eine Analyse- und Auswertungsmöglichkeit

Damit Schlussfolgerungen aus den ausgeführten Testfällen gezogen werden können, muss eine Analysemöglichkeit bereitgestellt werden. Eine Restbussimulation besitzt keine Verbindung zu den Sensoren und Aktoren des SUT. Daher muss die Auswertung auf der abstrakten Datenbasis erfolgen. Dabei sind zwei Varianten möglich. Auf der einen Seite kann die Analyse online zur Laufzeit der Simulation erfolgen um eine direkte Rückmeldung der Ergebnisse

eines Testfalls zu erhalten. Auf der anderen Seite kann eine offline durchgeführte Analyse eine genauere Untersuchung ermöglichen, wenn zusätzliche Attribute hinzugezogen werden sollen, die während der Laufzeit noch nicht betrachtet wurden.

Bei einer online durchgeführten Analyse muss die Simulationsplattform in der Lage sein, auf den Dateninhalt von empfangenen Paketen zuzugreifen und den Inhalt richtig interpretieren. Auf diese Weise ist es möglich, reaktiv auf das Verhalten des SUT zu reagieren. Dabei ist es wichtig, dass die Plattform die Relation zwischen den Nachrichten am Ein- und Ausgang des SUT herstellt und den kausalen Zusammenhang erkennen kann. Daraus folgt, dass zusätzlich zum Simulationsmodell ein weiteres Modell zur Analyse ausgeführt werden muss, wodurch die Komplexität steigt und das Laufzeitverhalten beeinflusst werden kann.
Offline durchgeführte Analysen setzen die Aufzeichnung von gesendeten und empfangenen Nachrichten der Restbussimulation voraus, damit die Relation von Ein- und Ausgang des SUTs möglich ist. In diesem Fall werden die Daten nicht während der Laufzeit ausgewertet und nur aufgezeichnet, sodass das Verhalten durch die Analyse nicht beeinflusst wird. Der aufgezeichnete Verkehr kann nach dem Testlauf auf einem anderen Rechner untersucht werden, sodass funktionale und zeitliche Leistungsanforderungen untersucht werden können.

Da die Restbussimulation in dieser Arbeit zum Testen von funktionalen und zeitlichen Leistungsanforderungen verwendet werden soll und die Berechnung des Verhaltens ebenfalls offline durchgeführt wird, ist eine offline basierte Analyse zu bevorzugen. Ein weitere Vorteil dieser Variante ist, dass das Laufzeitverhalten nicht beeinflusst wird, da keine Auswertung durchgeführt wird. Sie muss dementsprechend in der Lage sein, den gesendeten und empfangenen Datenverkehr aufzuzeichnen und für die anschließende Analyse bereitzustellen. Damit die zeitlichen Leistungsanforderungen überprüft werden können, müssen alle Nachrichten mit Zeitstempel versehen werden.

4.2.4 Modell zur Berechnung Möglicher Topologiekonflikte

Wie bereits zu Beginn dieses Kapitels beschrieben, unterscheidet sich die Topologie einer Echtzeit-Ethernet Restbussimulation von bisherigen busbasierten Topologien. Drei Varianten haben sich als relevant herausgestellt, die unterschiedliche Eigenschaften aufweisen. Damit überprüft werden kann, ob eine Restbussimulation mit nur einem Netzwerkinterface durchführbar ist, muss berechnet werden, ob etwaige Konflikte auftreten können.

Auf Basis der Vorarbeit, in der die verschiedenen Testtopologien diskutiert wurden, (vgl. Bartols 2013) wurde ein mathematisches Modell entwickelt, das eine Berechnung möglicher Bandbreiten-, bzw. Schedulingkonflikte zulässt, um die Verwendbarkeit eines Restbussimulators mit nur einem Netzwerkinterface zu überprüfen. Das Modell basiert auf Mengendefinitionen, die das Echtzeit-Ethernet Protokoll und den Nachrichtenfluss darstellt, sowie Funktionen, die die Berechnung möglicher auftretender Konflikte ermöglichen.

Die Funktionen erlauben es, alle gesendeten und empfangenen Nachrichten eines ausgewählten SUT zu bestimmen, um die genaue Bandbreite von Time-Triggered- und die worst-case Bandbreite von Rate-Constrained-Nachrichten zu errechnen. Die Gesamtbandbreite aller Nachrichten wird anschließend akkumuliert, um etwaige Bandbreitenkonflikte erkennen zu können. Da Best-Effort-Nachrichten im Echtzeit-Ethernet keine Information zu dessen benötigter Bandbreite aufweisen, werden diese in diesem Modell vernachlässigt. Darüber hinaus erlauben die Funktionen die Berechnung der Zeitpunkte, an denen TT-Nachrichten in Abhängigkeit des ausgewählten SUT vom Restbussimulator zu empfangen, bzw. zu senden sind. Mögliche Schedulingkonflikte können auf diese Weise erkannt werden, wenn mehrere Nachrichten zum gleichen Zeitpunkt zu empfangen bzw. zu senden sind.
Dieses Modell ist als formale Definition entwickelt, das eine Implementierung in beliebigen Programmiersprachen zulässt, um die Berechnung durchzuführen. Es ist daher unabhängig vom Eingabeformat. Während der Umsetzung wurde dennoch darauf Wert gelegt, FIBEX als Format zu unterstützen, da alle benötigten Informationen durch FIBEX bereitstellt werden und als Standardformat für Werkzeuge im Entwicklungsprozess gilt.

5 Architektur und Implementierung eines Echtzeit-Ethernet Restbussimulators

Dieses Kapitel stellt das Konzept und die Realisierung des zu entwickelnden Echtzeit-Ethernet Restbussimulators vor. Dabei werden zuerst verschiedene Implementierungskonzepte der einzelnen Komponenten eines Restbussimulators vorgestellt und deren Alternativen diskutiert. Die Implementierung der Komponenten des Simulators wird im darauf folgenden Abschnitt beschrieben.

5.1 Konzept der Systemarchitektur

Zuerst wird in diesem Kapitel ein Konzept zur Systemarchitektur beschrieben, das die Anforderungen der benötigten Hardware-/Software-Plattform aus dem letzten Kapitel umsetzen soll. Zunächst wird deshalb ein Überblick über die Systemarchitektur gegeben, die die prinzipielle Funktionsweise einer Echtzeit-Ethernet Restbussimulation beschreibt.

5.1.1 Überblick der Systemarchitektur

Damit die Architektur einer Echtzeit-Ethernet Restbussimulation deutlich wird, ist ein abstrakter Überblick der Systemarchitektur in Abbildung 5.1 auf der nächsten Seite dargestellt. Sie stellt die einzelnen Komponenten und deren Aufgaben vor, sodass im Anschluss die Architektur und Konzepte der einzelnen Komponenten vorgestellt werden können.

Das Konzept einer Echtzeit-Ethernet Restbussimulation ist im Wesentlichen identisch mit der im Grundlagenkapitel (siehe Abschnitt 2.4.1 auf Seite 30) beschriebenen Architektur und besteht aus einer Workstation, mit der die Restbussimulation konfiguriert wird, der eigentlichen Hardware-/Softwareplattform, die die Restbussimulation ausführt und dem zu testenden System (SUT). Die Konfiguration der in dieser Arbeit vorgestellten Restbussimulation wird auf einer Workstation erstellt und beinhaltet sowohl die statischen, als auch dynamischen Merkmale.

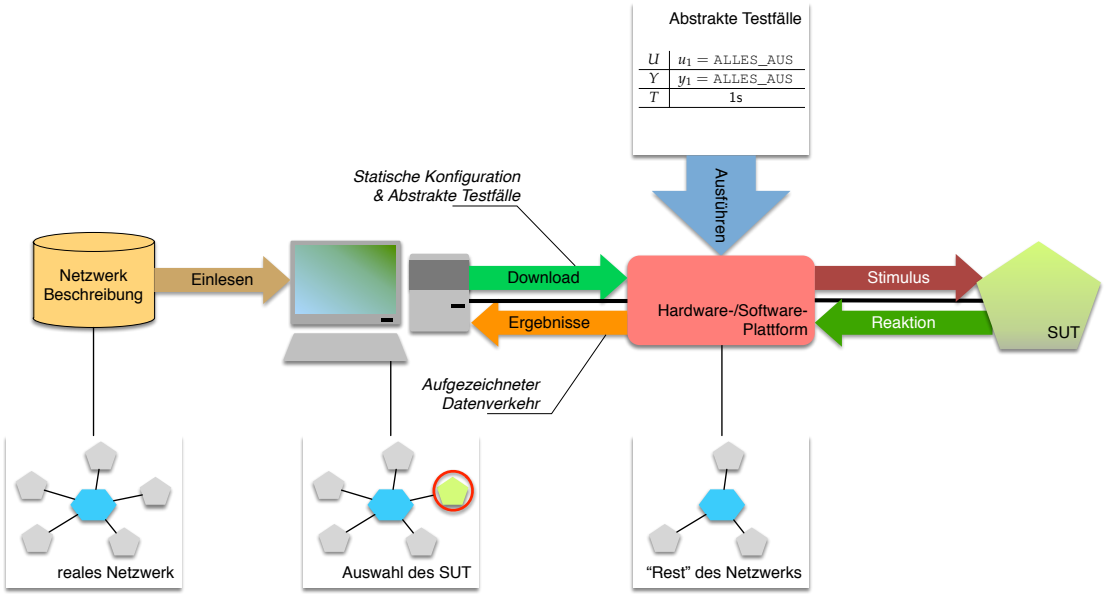


Abbildung 5.1: *Überblick aller Systemkomponenten in einer Echtzeit-Ethernet Restbussimulation*

Zuerst wird bei der Konfiguration die vollständig beschriebene Netzwerkarchitektur in FIBEX durch den Konfigurator eingelesen. Anschließend muss das aus einem einzelnen oder mehreren Knoten bestehende SUT ausgewählt werden um die statische Konfiguration des Restbussimulator zu erstellen. Wenn das SUT aus mehreren Teilsystemen besteht, muss vor der Erstellung überprüft werden, ob bei der Restbussimulation mit nur einem Netzwerkinterface Bandbreiten- oder Schedulingkonflikte auftreten können. Falls keine Konflikte auftreten, kann anschließend die Erstellung der statischen Konfiguration erfolgen. Hierbei werden die Informationen des Nachrichtenschedulings und des Synchronisierungsprozesses direkt aus dem FIBEX-Dokument entnommen und für die Restbussimulation abgeleitet. Des Weiteren wird das Gerüst der zu übertragenen Nachrichten daraus erstellt.

Danach muss das Simulationsmodell entwickelt werden, das das Verhalten der simulierten Knoten beschreibt. Da in dieser Arbeit das Verhaltensmodell direkt aus Testfällen abgeleitet werden kann, muss für das formale Modell ein maschinenlesbares Format entwickelt werden, das vom Simulator verwendet werden kann. Die statische Konfiguration und das Verhaltensmodell können so auf dem Konfigurator erstellt und anschließend auf die Simulationsplattform geladen und ausgeführt werden.

Die Hardware-/Software-Plattform zeichnet während der Simulation den gesamten Datenverkehr auf. Nachdem ein Testfall ausgeführt worden ist, können die Ergebnisse offline auf

dem Konfigurationsrechner betrachtet und bewertet werden, um die Anforderungen des SUT zu verifizieren.

5.1.2 Konzepte einer geeigneten Hardware-/Softwareplattform

Wie bereits im vorigen Kapitel beschrieben, ist die wesentliche Komponente einer Restbussimulation die Hardware-/Softwareplattform. Wichtig zu erwähnen ist, dass in diesem Konzept der Restbussimulation die Ausführung des Simulationsmodells unabhängig von der Ausführung des Echtzeit-Ethernet-Netzwerkstacks ist, da im Vergleich zur Arbeit von Oleg Karfich keine Echtzeit-Ethernet Protokollmodelle in der Simulationsumgebung ausgeführt werden. Das Simulationsmodell muss zeitlich präzise betrieben werden, damit die Simulationsereignisse zum richtigen Zeitpunkt bereitgestellt werden, insbesondere wenn dieser mit hoher Auflösung angegeben ist.

Die Hardware-/Softwareplattform muss zum einen flexibel zu konfigurieren sein und zum anderen Echtzeitanforderungen bei der Übertragung von Nachrichten aufweisen. Der erste Ansatz basiert auf einer *reinen x86-Architektur*. Als Betriebssystem wird Linux mit einer Echtzeiterweiterung eingesetzt, sodass der Netzwerkstack und die Simulationsumgebung auf der gleichen Plattform ausführt wird. Im zweiten Ansatz werden das Simulationsmodell und der Echtzeit-Ethernet-Netzwerkstack ebenfalls auf derselben Plattform ausgeführt. Diese Variante basiert auf einer *Mikrocontroller-Architektur*, mit einem speziell entwickelten Echtzeit-Ethernet-Netzwerkstack. Die letzte Variante entspricht einer Mischform, bei der ein *x86-Host und ein Mikrocontroller* gekoppelt werden und der Echtzeit-Ethernet-Netzwerkstack auf dem Mikrocontroller und das Simulationsmodell auf dem Host ausgeführt wird. Zum Zeitpunkt dieser Arbeit unterstützen alle verfügbaren Echtzeit-Ethernet-Netzwerkstacks nur ein Netzwerkinterface. Aufgrund dieser Einschränkung basieren die Konzepte auf einer Restbussimulationsplattform mit nur einem Interface. Nachfolgend werden die drei Varianten und dessen Eigenschaften vorgestellt.

Restbussimulationsplattform auf Basis einer x86-Architektur

Die Restbussimulationsvariante auf der Basis einer x86-Architektur ist in Abbildung 5.2 auf der nächsten Seite präsentiert und besteht aus einem Rechner zur Konfiguration, der x86-Plattform als Restbussimulator und dem SUT. Die Verbindung zwischen dem Restbussimulator und dem SUT wird in diesem Fall durch eine Netzwerkkarte, für die ein Echtzeit-Ethernet-Netzwerkstack zur Verfügung steht, hergestellt. Damit die Echtzeit-Anforderungen des Echtzeit-Ethernet-Protokolls erfüllt werden, muss die Plattform ein echtzeitfähiges Betriebssystem ausführen. Die Wahl fällt auf die Realtime-Kernel-Erweiterung von Ingo Molnar

(vgl. Ts'o / Hart / Kacur 2010; Yaghmour / Masters / Ben-Yoseff u. a. 2008). Diese Erweiterung verfügt über ein verbessertes Interrupt-Handling und erlaubt Kernel- und Userspace-Anwendungen mit Echtzeitprioritäten zu versehen, sodass beide Anwendungstypen auf der gleichen Hierachiestufe ausgeführt und anhand ihrer RT-Priorität gescheduled werden.

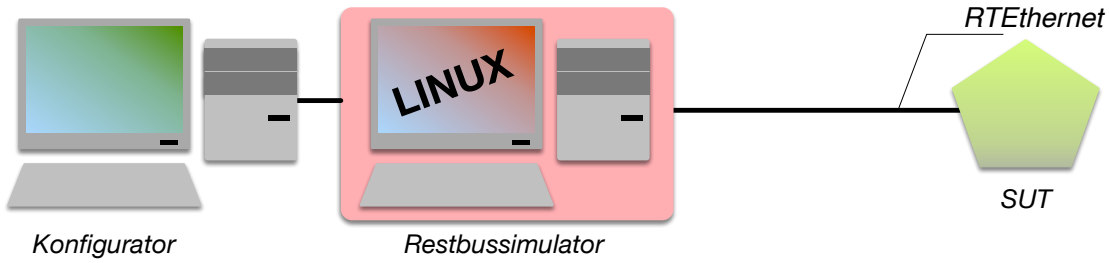


Abbildung 5.2: *Architektur einer Restbussimulation mit einer x86-Architektur und Linux als Betriebssystem*

Der Vorteil dieses Ansatzes besteht in der Implementierung der Simulationsumgebung, welche das Simulationsmodell ausführen soll. Die Implementierung des Simulators wird durch die Vielzahl an bereitstehenden Softwarebibliotheken unterstützt, sodass auf ein breites Angebot an Standardfunktionen (z. B. das Einlesen von XML-Dokumenten) zurückgegriffen werden kann. Ein weiterer Punkt besteht darin, dass keine weitere Hardware erforderlich ist und der Restbussimulator nur aus einer Komponente besteht. Zusätzlicher Aufwand, der bei einer Kopplung mit anderen Hardwarekomponenten entstehen kann, entfällt auf diese Weise.

Auf der anderen Seite hat dieser Ansatz trotz Real-time Kernelerweiterung im Vergleich zu Systemen ohne Betriebssystem eine ungenauere zeitliche Präzision. Das Betriebssystem führt viele Aufgaben parallel aus, die das zeitliche Verhalten beeinflussen können. Abhilfe kann hier die Verwendung eines Netzwerkinterfaces schaffen, das ein- und ausgehende Nachrichten unabhängig vom Betriebssystem auf Hardwarebene mit einem Zeitstempel versieht. Mit einem Interface, das kein unabhängiges Stempeln zulässt, ist die in dieser Arbeit vorgestellte Möglichkeit, zeitliche Leistungsanforderungen mit hoher Auflösung zu überprüfen, nicht durchführbar.
Darüber hinaus ist die Verfügbarkeit des passenden Echtzeit-Ethernet-Netzwerkstack ein Problem. Ein proprietärer Stack der Firma TTTech ist nur für zwei Typen von Netzwerkinterface-Karten vorhanden. Er ist für eine alte Linux-Kernel-Version implementiert, sodass die Umsetzung auf diese Kernel-Version und die Netzwerkinterfaces limitiert ist. Die alte Kernel-Version kann weitere Probleme bereiten, wenn die Restbussimulationsplattform aus neuerer Hardware zusammengesetzt ist, die nicht durch die alte Kernel-Version unterstützt wird. Eine Alternative zum proprietären Echtzeit-Ethernet Stack ist die Verwendung der Arbeit von Johannes Reidl (vgl. Reidl 2013), die einen Echtzeit-Ethernet-Netzwerkstack für Interfaces mit einer

hardwareseitigen Zeitstempelfunktionalität für aktuelle Kernel bereitstellt. Allerdings stand diese zum Implementierungszeitpunk der Restbussimulationsplattform noch nicht zur Verfügung.

Restbussimulationsplattform auf Basis eines Mikrocontrollers

Die zweite Architekturvariante besteht aus einem Mikrocontroller, auf dem die gesamte Funktionalität der Restbussimulation ausgeführt wird und ist in Abbildung 5.3 dargestellt. Für diesen Mikrocontroller wurde ein Echtzeit-Ethernet-Netzwerkstack bereits implementiert (vgl. Müller / Steinbach / Korf u. a. 2011; Müller 2011), sodass die geforderte protokollkonforme Nachrichtenübertragung ermöglicht wird. Die Hardware-Plattform des Mikrocontrollers (Hilscher netX-500 (vgl. Lipfert 2008)) stellt ein eigenes Systemzeit-Modul mit Nanosekunden-Genauigkeit zur Verfügung, das empfangene Ethernetframes mit dem Empfangszeitpunkt versieht.

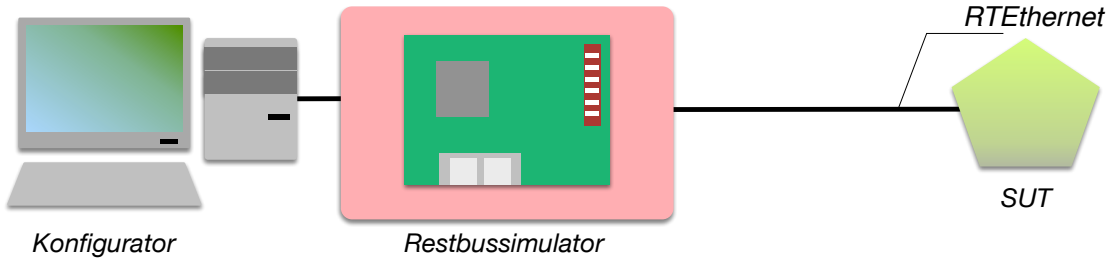


Abbildung 5.3: *Architektur einer Restbussimulation mit einem Mikrocontroller*

Im Vergleich zum zuvor vorgestellten ersten Ansatz ohne Netzwerkkarte mit Zeitstempelfunktionalität weist diese Plattform eine höhere Genauigkeit aufgrund des genannten separaten Systemzeit-Moduls auf. Durch die Verwendung des in der Forschungsgruppe entwickelten Netzwerkstacks ist es möglich, sämtliche Vorgänge auf dem Mikrocontroller nachvollziehen und ungewolltes Verhalten beseitigen zu können. Dieser Stack ist speziell gegen die Anforderungen von Echtzeit-Ethernet implementiert worden. Zeitgesteuertes Übertragen kann auf diese Weise mit einer sehr hohen Präzision ausgeführt werden. Die Implementierung des Echtzeit-Ethernet-Netzwerkstacks ist durch den Einsatz in verschiedenen Abschlussarbeiten und Projekten erprobt und hat seine Einsatzfähigkeit gezeigt. Der größte Vorteil dieser Architektur besteht in der hardwareseitigen Ausführung des Zeitstempelns der empfangenen Ethernetframes. Dieses ermöglicht es, zusammen mit der hochpräzisen Systemzeit, die geforderten zeitlichen Leistungsanforderungen mit hoher Genauigkeit zu überprüfen.

Andererseits ist das Einlesen von Testfällen nicht möglich, da der Mikrocontroller keine Datei-Operationen unterstützt und jeder Testfall als Quellcode kompiliert werden müsste. Die geforderte Flexibilität bei der Ausführung verschiedener Testfälle ist somit nicht erreichbar. Durch die fehlenden Datei-Operationen ist außerdem die geforderte offline Auswertung im Nachhinein nicht möglich, da die gesendete und empfangene Nachrichten nicht abgespeichert werden können.

Kombinierte Restbussimulationsplattform

Die letzte Variante ist eine Kombination aus den beiden vorherigen Architekturen und ist in Darstellung 5.4 abgebildet. Sie besteht aus einem x86-Host, der an den vorgestellten Mikrocontroller gekoppelt wird. Sie vereinigt die positiven Eigenschaften beider Architekturen. In diesem Fall wird der Mikrocontroller nur verwendet, um den Echtzeit-Ethernet-Netzwerkstack auszuführen und der x86-Host dient dazu, die Simulationsumgebung auszuführen. Damit die Verbindung zwischen Mikrocontroller und Host während der Ausführung priorisiert werden kann, wird der Linux Real-time Kernel Patch verwendet.

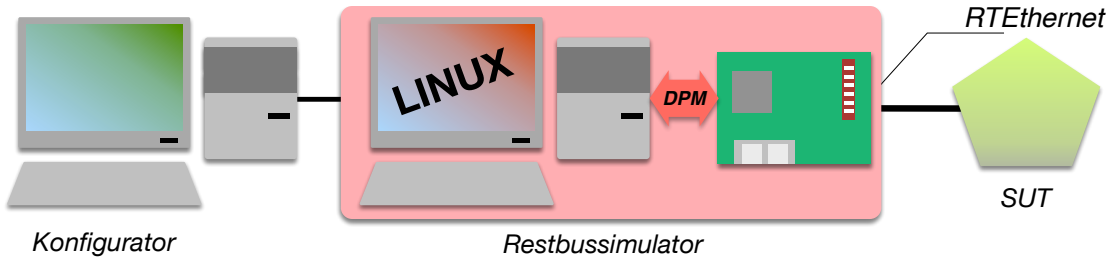


Abbildung 5.4: *Architektur einer Restbussimulation mit einem x86-Host, gekoppelt an einen Mikrocontroller*

Die Kopplung beider Hardware-Plattformen kann über verschiedene Weisen realisiert werden, wobei sich die Nutzung der *Dual-Port-Memory(DPM)*-Schnittstelle als geeignet herausgestellt hat. Im Vergleich zu einer möglichen Kopplung mit dem seriellen RS232-Interface stellt sie eine höhere Bandbreite zur Verfügung. Gegenüber einer Kopplung mit der zweiten Ethernet-Schnittstelle des Mikrocontrollers ermöglicht das DPM-Interface direkt auf den Speicher des Mikrocontrollers zuzugreifen. Dabei muss explizit kein Code auf dem Controller ausgeführt werden, sodass das Lesen und Schreiben von Daten via DPM unabhängig von der aktuell ausgeführten Anwendung ist. Außerdem erlaubt die Schnittstelle das Erzeugen von Interrupts in beide Richtungen, sodass die Kommunikation in beide Richtungen gesteuert werden kann. Für die Implementierung des Restbussimulators kann auf die Arbeit von Oleg

Karfich (vgl. Karfich / Bartols / Steinbach u. a. 2013; Karfich 2013) zurückgegriffen werden, die die Verwendung und Konfiguration der Schnittstelle bereits beschreibt und ein Konzept zur Verfügung stellt.
Die Kombination erlaubt es die zeitliche Präzision des Mikrocontrollers zu nutzen. Auch das Einlesen von Testfällen ist auf diese Weise möglich, da das Simulationsmodell auf dem Host ausgeführt wird. Das Speichern der Ergebnisse ist in diesem Fall ebenfalls möglich, sodass die geforderte Auswertung der Testfälle offline durchgeführt werden kann, ohne das Laufzeitverhalten der Restbussimulation zu beeinflussen.

Allerdings bedeutet dieser Ansatz, dass zusätzliche Hardware verwendet werden muss, sodass die Architektur komplexer im Vergleich zu den vorherigen Plattformen ist. Damit die DPM-Schnittstelle des Mikrocontrollers betrieben werden kann, muss eine dedizierte PCI-Karte für den Host vorhanden sein, die den Datenaustausch beider Systeme ermöglicht. Darüber hinaus bedeutet die Kopplung, dass ein Kommunikations-Protokoll zwischen den beiden Endgeräten entwickelt werden muss, was den Implementierungsaufwand gegenüber der ersten beiden Varianten erhöht. Weiterhin müssen zwei unabhängige Systeme konfiguriert werden, sodass der Konfigurationsaufwand steigt. In diesem Fall muss der Echtzeit-Ethernet-Netzwerkstack des Mikrocontrollers sowie die Simulationsplattform konfiguriert werden.

Die Architektur eines x86-basierenden Restbussimulators ist generell sinnvoll, da diese, gegenüber einer Architektur mit Kopplung zweier Hardwareplattformen, eine geringere Komplexität aufweist. Dazu muss allerdings ein Netzwerkinterface mit hardwareseitiger Zeitstempelfunktionalität vorhanden sein. Die Kombination eines Mikrocontrollers und einem x86-Host ist zum Implementierungszeitpunkt dieser Arbeit dennoch die geeignetste Alternative. Trotz des Mehraufwandes während der Implementierung der Restbussimulationsplattform wird sowohl hohe zeitliche Genauigkeit, als auch eine hohe Rechenleistung erreicht. Die Implementierung und Ausführung des Simulationsmodells wird dadurch vereinfacht und es kann auf eine bereits durchgeführte Arbeit zurückgegriffen werden. Ein weiterer Vorteil ist, dass keine Fremdsoftware verwendet wird und sämtliche benötigten Komponenten auch als Quellcode zur Verfügung stehen, sodass die Implementierung und Fehlersuche erleichtert wird.

5.1.3 Architektur zur Erstellung der statischen Konfiguration

Damit die statische Konfiguration des Restbussimulators durch Werkzeuge unterstützt werden kann, bedarf es einer Möglichkeit, diese aus dem vorliegenden FIBEX-Dokument zu erstellen. Insbesondere, da in dieser Arbeit die Umsetzung der Hardware-/Softwareplattform durch zwei Komponenten implementiert werden soll, kann durch eine automatisch generierte Konfiguration der Prozess vereinfacht werden. Auf der einen Seite muss darauf geachtet werden, dass für die zwei Komponenten unterschiedliche Informationen bereitstehen und ausgewählt werden müssen, damit die Restbussimulationsplattform richtig konfiguriert wird. Auf der anderen Seite müssen die Informationen entsprechend der Zielplattform aufbereitet

werden.
Da sich die statische Konfiguration der Restbussimulation für ein gewähltes SUT nicht zur Laufzeit ändern kann, ist es sinnvoll, diese direkt im Quellcode zu verankern. Sie wird so zur Compilezeit des Restbussimulators durchgeführt. Eine andere Variante könnte durch spezielle Konfigurationsfiles ermöglicht werden, die zur Laufzeit ausgewertet werden. Letztere Möglichkeit scheidet jedoch aus, da Dateioperationen auf dem Mikrocontroller nicht möglich sind. Weiterhin stellt der Echtzeit-Ethernet-Netzwerk-Stack bereits eine spezielle Konfigurationsschnittstelle zur Verfügung, die im Quellcode verankert ist. Auf Seite der x86-Plattform, die die Simulationsumgebung ausführen soll, muss eine Schnittstelle zur Verfügung stehen, die die Konfiguration statischer Attribute ermöglicht.
Dementsprechend müssen die Informationen, die im FIBEX-Dokument enthalten sind, zum einen in passenden Quellcode für den Mikrocontroller und zum anderen in Quellcode für die Simulationsumgebung umgewandelt werden. Die Quellcodegenerierung muss dabei die Konfiguration des Echtzeit-Ethernet-Netzwerkstacks und die Erstellung eines Gerüstes der zu übertragenden Nachrichten in der Simulationsumgebung umfassen. Auf diese Weise ist es weiterhin möglich, flexibel verschiedene Verhaltensmuster des Restbussimulators zu erstellen, da diese durch die dynamische Konfiguration beschrieben wird.

5.1.4 Konzept der Simulationsumgebung

Die Simulationsumgebung muss verschiedene Aufgaben ausführen, damit Nachrichten mit Inhalten versehen werden können und diese anschließend präzise zu definierten Zeitpunkten zu versenden, um das Verhalten des Restnetzwerks zu simulieren. So muss ein Scheduler vorhanden sein, der dafür sorgt, dass die Daten zum richtigen Zeitpunkt bereitstehen, um diese an den Mikrocontroller zu übertragen. Damit die abstrakten Testfälle als Simulationsmodell verwendet werden können, muss die Simulationsplattform in der Lage sein, diese in ausführbareTestfälle umzuwandeln. Des Weiteren muss der Datenverkehr zwischen dem SUT und dem Restbussimulator aufgezeichnet werden, damit eine Analyse der empfangenen Ergebnisse erfolgen kann.

Ausführung des Simulationsmodells

In den Grundlagen wurde bereits beschrieben, dass es verschiedene Arten zur Ausführung der Simulationsmodelle (siehe Abschnitt 2.2.1 auf Seite 12) gibt. Eine Restbussimulation muss das Simulationsmodell in Echtzeit ausführen, damit ein deterministisches Verhalten erzeugt wird und Nachrichten zum richtigen, vorkonfigurierten Zeitpunkt versendet werden. Dieses Verhalten wird mittels diskreter, zeitbasierter Simulationsmethodik erreicht, bei dem der Simulationstakt vorgegeben ist, der die Simulationszeit kontinuierlich inkrementiert. Aktionen werden nur dann ausgeführt, wenn ihr Aktionszeitpunkt der aktuellen Simulationszeit

entspricht. Bei einer variablen Taktlänge diskreter, ereignisbasierter Simulationen ist der Zeitpunkt der Ausführung abhängig vom vorherigen Ereignis und damit nicht konstant. Aussagen zum Ausführungsverhalten können mit dieser Art nicht durchgeführt werden. Daraus folgt, dass die zu realisierende Simulationsumgebung einen diskreten, zeitbasierten Simulationsscheduler implementieren muss. Dieses lässt sich durch die Verwendung eines periodischen Timers realisieren, dessen Frequenz zu konfigurieren und abhängig von der geforderten Ausführungspräzision der Ereignisse ist. Da die Simulationsumgebung auf einer Linux-Plattform mit Real-time-Kernel aufsetzt, kann auf POSIX-Timer zurückgegriffen werden (vgl. Ts'o / Hart / Kacur 2010). Sie basieren auf High-Resolution-Timern und sind unabhängig vom Kernel Takt. Gegenüber den Standard-Linux-Timern weisen sie eine Auflösung im Nanosekundenbereich auf (vgl. Gleixner / Molnar).

Umwandlung abstrakter Testfälle

Weiterhin muss die Simulation die benötigte Ereignisliste erstellen, die das Verhalten der Restbussimulation bestimmt. Da die abstrakten Testfälle als Simulationsmodell verwendet werden, müssen diese in das Aktionzeitpunkt-Ereignis-Paar umgewandelt und anschließend in die Ereignisliste eingetragen werden. Das Simulationsereignis entspricht einem ausführbaren Testfall, der ein vorgefertigtes Nachrichtenpaket enthält. Zum Aktionszeitpunkt wird das Nachrichtenpaket anschließend auf die Hardware-Plattform übertragen und gemäß der konfigurierten Übertragungsstrategie versendet.
Die Umwandlung eines abstrakten Testfalls in das ausführbare Pendant muss vor der eigentlichen Simulation erfolgen und lässt sich in Form einer Zuordnungstabelle realisieren. Diese Zuordnung muss es erlauben, die im Eingangsvektor U definierten Einträge an die erstellten Nachrichtengerüste der statischen Konfiguration zu binden. Weiterhin müssen die abstrakten Werte des Testfalls in reale Daten umgewandelt werden, sodass auch hier eine Zuordnung erfolgen muss. Ist sowohl eine Zuordnung der Nachrichten als auch der Daten möglich, kann das Nachrichtengerüst mit Inhalt versehen und als Simulationsereignis abgelegt werden.

Aufzeichnen des Datenverkehrs

Damit eine Analyse des Testfalls im Nachhinein durchgeführt werden kann, muss die Simulationsplattform in der Lage sein, den gesamten Nachrichtenverkehr aufzuzeichnen. Dabei ist es wichtig, dass die Simulationsumgebung auch die Zeitpunkte des Sendens und Empfangs mit aufzeichnet, da sonst eine Überprüfung der zeitlichen Leistungsanforderungen nicht möglich ist. Für die Aufzeichnung bieten sich deshalb zwei Varianten an: Die erste Variante entspricht der Umsetzung eines eigenen textbasierten Formats, das die Nachrichten in einer Datei speichert. Der Vorteil dieser Variante liegt darin, dass der Datenverkehr nach eigenen

Darstellungskriterien aufgezeichnet wird, die das Analysieren vereinfachen können. Der Entwickler bestimmt dabei die Merkmale, die aufgezeichnet werden.
Die zweite Variante speichert den aufgezeichneten Verkehr in einem speziellen Format (z. B. PCAP-Format (vgl. Jacobson / Leres / McCanne)), das von einem Netzwerkanlysewerkzeug (z. B. Wireshark (vgl. Combs)) gelesen werden kann. Diese Werkzeuge erleichtern die Analyse dahingehend, dass sie bereits viele Möglichkeiten zur Darstellung und Abfragefunktionen zum aufgezeichneten Datenverkehr bereitstellen.
Beide Aufzeichnungsvarianten unterstützen die Analyse des Datenverkehrs auf unterschiedliche Weise, sodass beide Varianten in dieser Arbeit umgesetzt wurden.

Der Vorteil das Modell der abstrakten Testfälle als Simulationsmodell zu verwenden liegt darin, dass nur wenig Rechenzeit zur Simulation benötigt wird. Lediglich die generierten Nachrichtengerüste müssen mit realen Daten versehen werden. Verschiedene Verhaltensmuster werden in dieser Variante der Restbussimulation offline berechnet, sodass keine komplexen Verhaltensmodelle ausgeführt werden. Die Verhaltensmuster können vorher auf einer unterschiedlichen, leistungsfähigeren Plattform erstellt werden.

Auf der anderen Seite bedeutet dieser Ansatz auch Einschränkungen. So lässt sich in dieser Form das Simulationsmodell nur für das Testen von Systemen verwenden, bei denen von vornherein das Verhalten und die versendeten Daten definiert werden. Reaktives Verhalten zur Laufzeit kann nicht bestimmt werden, sodass sich diese Variante nicht zur Verwendung einer rapid-Prototypplattform eignet.

5.1.5 Abbildung abstrakter Testfälle als XML-Dokument

Bereits zu Beginn dieses Kapitels wurde erwähnt, dass das formale Modell der abstrakten Testfälle in eine maschinenlesbare Form überführt werden muss, damit der Simulator es verarbeiten kann. Bei der Auswahl eines geeigneten Formats muss darauf geachtet werden, dass es auch gut vom Menschen gelesen, geschrieben und verstanden werden kann, da der Fokus der Arbeit nicht das automatische Generieren von Testfällen ist. Eine automatische Generierung ist z. B. durch die Analyse von Zustandsautomaten oder Sequenzdiagrammen möglich (vgl. Weißleder / Schlingloff 2011; Belli / Hollmann / Padberg 2011). Weiterhin muss das Modell eine Basis für zukünftige Erweiterungen bereitstellen, falls andere Aspekte zu untersuchen sind.

Die einfachste Möglichkeit ist die Abbildung in einer *Textdatei*, die die Informationen ohne Format in Textform abgespeichert. Diese Möglichkeit ist allerdings zu vernachlässigen, da keine einheitliche Dateistruktur vorgegeben wird. Die Unterscheidung von Strukturinformationen zum eigentlichen Inhalt müsste zusätzlich implementiert werden.
Die zweite und zu bevorzugende Variante ist die Umsetzung der Abbildung in einer *Markup Language*, die ein bestimmtes Format zur Beschreibung des Inhalts vorgibt. So wird die

Extensible Markup Language (XML), insbesondere in der Form von FIBEX-Dokumenten, in der Werkzeugkette und in anderen Projekten aus dem CoRE-Umfeld bereits ausgiebig verwendet.

XML (vgl. W3C XML Working Group 2008) ist ein standardisiertes Dateiformat, das es erlaubt, eigene Datenstrukturen und Typen in einem fest definierten Format zu beschreiben und ist flexibel einsetzbar. Durch dessen breite Verwendung stehen standardisierte Softwarebibliotheken zur Verfügung, die das Einlesen und Extrahieren der Daten aus XML-Dokumenten ermöglichen. Außerdem kann durch die Verwendung eines XML-Schemas (*XML Schema Definition – XSD*) der Aufbau eines XML-Dokumentes beschrieben werden. Ein XSD beschreibt ähnlich zu einer Klassendefinition, den Inhalt eines XML-Dokumentes, sodass nicht nur der Syntax eines Dokumentes, sondern auch dessen Aufbau auf Konformität überprüft werden kann. Ein Ausschnitt eines abstrakten Testfalls als XML-Dokument ist im nachfolgenden Code-Listing 5.1 auf der nächsten Seite dargestellt.

Die Verwendung von alternativen Markup Languages (z. B. JSON (vgl. Crockford 2006)) ist ebenfalls möglich, jedoch ist die Entscheidung aufgrund der ausgiebigen Verwendung auf eine XML-basierte Lösung gefallen. Der Entwicklungsprozess innerhalb des CoRE-Umfeldes kann auf diese Weise einheitlich mit einer Markup Language durchgeführt werden.

5.2 Implementierung der Komponenten

Nachdem Architekturkonzepte zu den verschiedenen Komponenten und Entscheidungen zur geplanten Umsetzung präsentiert wurden, beschreibt dieser Abschnitt die eigentliche Implementierung der Komponenten.

5.2.1 Code-Generator zur Konfiguration des Simulators

Die Implementierung des Quellcode-Generators ist in einem separaten Programm umgesetzt worden, das auf dem Konfigurations-PC ausgeführt werden kann. Dazu ist das in der Vorleistung entwickelte Programm, welches die Berechnung von möglichen Scheduling- und Bandbreitenkonflikte ermöglicht, mit einem Codegenerator erweitert worden. Das Einlesen eines vollständigen FIBEX-Dokumentes wurde bereits umgesetzt, sodass zur Generierung der Konfiguration ein Modell entwickelt worden ist, welches Quellcode-Dateien erzeugt.
Mit dem implementierten generischen Quellcode-Modell, das Attribute der Sprachen C und C++ abbildet, lässt sich der Echtzeit-Ethernet-Netzwerkstack vollständig konfigurieren. Dabei werden die benötigten Scheduling-, Netzwerk- und Cluster- sowie Synchronisationsinformationen aus dem FIBEX-Dokument extrahiert und in entsprechende Strukturen des Konfigurationsinterfaces umgewandelt.

Listing 5.1: *Ausschnitt eines abstrakten Testfalls in XML. In diesem Listing sind die zeitlichen Leistungsanforderungen aufgrund der Übersichtlichkeit nicht dargestellt.*

```
<?xml version="1.0" encoding="UTF-8" ?>
<TestCase>
  <Name>Testfallname</Name>
  <Description>Kurze Beschreibung des Testfalls</Description>
  <Inputs>
    <Input ID="input1">
      <InputName>Name des Eingangs</InputName>
      <Frame-ID>Bezeichner des Frames aus FIBEX</Frame-ID>
    </Input>
  </Inputs>
  <ExpectedOutputs>
    <ExpectedOutput ID="expectedOutput1">
      <OutputName>Name des Ausgangs</OutputName>
      <Frame-ID>Bezeichner des Frames aus FIBEX</Frame-ID>
    </ExpectedOutput>
  </ExpectedOutputs>
  <TestSteps>
    <TestStep ActionPoint="1">
      <TestStepInfo>Informationen zum aktuellen Schritt</TestStepInfo>
      <TestStepInputs>
        <TestStepInput>
          <Input-Ref ID-REF="input1"/>
          <TestStepValues>
            <TestStepValue>HEADLIGHTS_OFF</TestStepValue>
            <TestStepValue>0</TestStepValue>
          </TestStepValues>
        </TestStepInput>
      </TestStepInputs>
      <ExpectedOutputs>
        <ExpectedOutput ID-REF="expectedOutput1">
          <TestStepValues>
            <TestStepValue>HEADLIGHTS_OFF</TestStepValue>
            <TestStepValue>0</TestStepValue>
          </TestStepValues>
        </ExpectedOutput>
      </ExpectedOutputs>
    </TestStep>
        ...
  </TestSteps>
  ...
</TestCase>
```

Für die Konfiguration der Nachrichtengerüste, die von der Simulationsumgebung benötigt werden, wird ebenfalls das eingelesene FIBEX-Dokument verwendet. Das Quellcode-Modell

generiert die Strukturen für die Simulationsumgebung, sodass für jede Nachricht, die im FIBEX definiert wurde, ein entsprechendes Quellcode-Pendant erstellt und als Nachrichtengerüst verwendet wird.

5.2.2 Datenmodell der abstrakten Testfälle

Damit die abstrakten Testfälle von der Simulationsumgebung eingelesen und als Simulationsmodell verwendet werden können, muss ein Datenmodell entwickelt werden, das alle Attribute der Testfälle implementiert. Nachfolgend ist das Datenmodell im UML-Klassendiagramm 5.5 auf der nächsten Seite dargestellt, welches anschließend in XML-Dokumenten umgesetzt worden ist.

Hierbei entsprechen die Klassen `Input` und `ExpectedOutput` den Vektoren U und Y_{soll}. Damit eine Zuordnung der generierten Nachrichtengerüste mit den Inhalten dieses Modells möglich ist, besitzen die beiden Klassen das Attribut `Frame-ID`. Es ermöglicht die Umwandlung der abstrakten Testfälle in ihr ausführbares Pendant innerhalb der Simulationsumgebung, da die Zuordnung der Inhalte zu den Nachrichtengerüsten eindeutig ist. Die Abhängigkeit der Eingänge und erwarteten Ergebnisse der Ausgänge wird mittels der `Teststep`-Klasse durchgeführt. Sie beinhaltet das Attribut `ActionPoint`, das den Zeitvektor T des formalen Modells repräsentiert. Die eigentlichen Inhalte werden durch die Klassen `TestStepInput`, `TestStepExpectedOutput` und dessen dazugehörigen `TestStepValues` repräsentiert. Die Vektoren, die zur Beschreibung der zeitlichen Leistungsanforderungen ($J_{soll}, R_{soll}, J_{L_{soll}}, J_{R_{soll}}$) eingeführt wurden, werden durch die Klassen `Latency` und `Rate` sowie `JitterLatency` und `JitterRate` abgebildet.
Damit eine insgesamt eindeutige Struktur ohne Mehrfachdeklarierung derselben Informationen durchgeführt werden kann, referenzieren die Klassen `TestStepInput` und `TestStepExpectedOutput` die in `Input` und `ExpectedOutput` deklarierten Daten. Insbesondere lassen sich, durch die Verwendung von Referenzen, die Assoziationen der zeitlichen Leistungsanforderungen zu den jeweiligen zugehörigen Nachrichtenein- bzw.-ausgängen am SUT, gut darstellen. Das Datenmodell kann auf diese Weise komplett in einem XML-Dokument dargestellt werden.

5.2.3 Umsetzung der Restbussimulationsarchitektur

Die Restbussimulationsarchitektur ist, wie im Konzept (siehe Abschnitt 5.1.2 auf Seite 79) diskutiert, durch zwei Komponenten implementiert worden. Der Mikrocontroller führt den Echtzeit-Ethernet-Netzwerkstack aus und versieht gesendete und empfangene Nachrichten mit einem Zeitstempel. Die Nachrichten werden gemäß ihrer konfigurierten Übertragungsstrategie an das SUT übertragen sowie empfangen. Der Host führt das Simulationsmodell

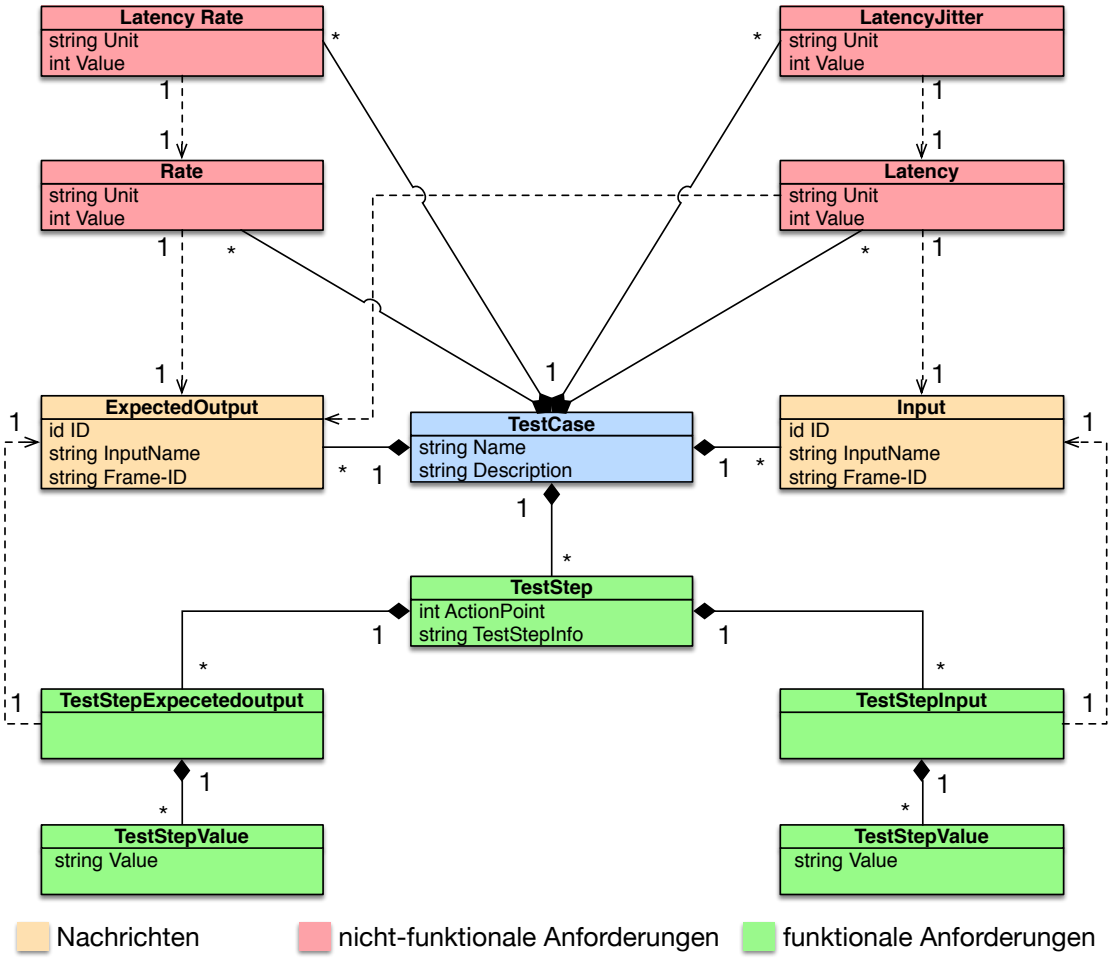


Abbildung 5.5: *XML-Datenmodell des abstrakten Testfallmodells für zeitliche Leistungsanforderungen*

aus, generiert über den Simulationsscheduler die Nachrichten mit denen das SUT getrieben wird und speichert die gesendeten Nachrichten über den Nachrichtenlogger, sodass im Anschluss der Testfallausführung die Ergebnisse überprüft werden können.

5.2.4 Kommunikation zwischen dem Host und dem Mikrocontroller

Die Kommunikation zwischen dem Mikrocontroller und der auf dem Host ausgeführten Simulationsumgebung ist die essentielle Komponente in der Architektur dieser Restbussimulation

und ist, wie in Abbildung 5.6 dargestellt, umgesetzt worden. Sie muss zum einen in der Lage sein, generierte Nachrichten aus der Simulationsumgebung vom Host zum Mikrocontroller zu übertragen. Zum anderen müssen sowohl gesendete als auch empfangene Nachrichten mit dem jeweiligen Zeitstempel vom Mikrocontroller zurück zum Host transferiert werden. Für diese Aufgabe ist die Verwendung einer DPM-Schnittstelle, wie im Konzept beschrieben, gut geeignet. Einerseits erlaubt sie Schreib- und Lesezugriffe vom Host auf dem Mikrocontroller ohne direkte Ausführung von Code auf dem Mikrocontroller, andererseits kann auf einer bestehenden Implementierung aufgebaut werden.

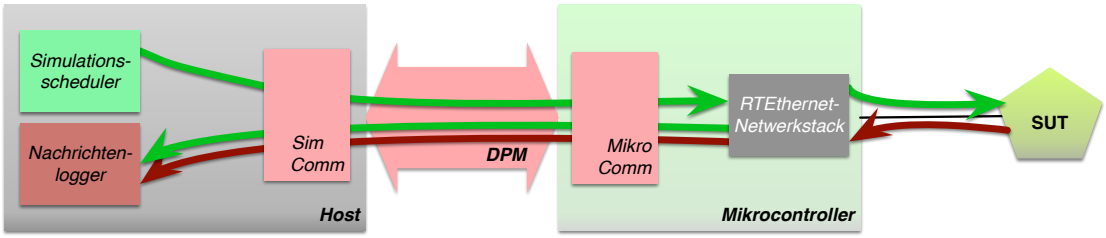


Abbildung 5.6: *Implementierung und Kommunikation zwischen dem Mikrocontroller und dem Host*

Damit die DPM-Schnittstelle sowohl auf dem Mikrocontroller als auch auf dem Hostrechner genutzt werden kann, muss sie mit den richtigen Parametern konfiguriert werden. Insbesondere muss der Speicheradressraum konfiguriert werden, damit eine Kommunikation möglich ist. Hierbei ist darauf zu achten, dass der Host und der Mikrocontroller zwei unterschiedliche Speicheraddressräume besitzen, sodass der Host nicht direkt auf die Speicheradressen des Mikrocontroller zugreifen kann. Addressräume, die nicht explizit für den Zugriff konfiguriert wurden, können vom Host aus nicht angesprochen werden. Des Weiteren erlaubt die Verwendung der DPM-Schnittstelle das Auslösen von Interrupts sowohl vom Host auf dem Mikrocontroller, als auch in umgekehrter Richtung, sodass auch diese Funktionalität mit den richtigen Parametern zu versehen ist und Interrupt-Requests in beide Richtungen ausgeführt werden.

Implementierung des Kommunikationsmoduls auf dem Host

Der Zugriff auf Speicherbereiche via DPM-Schnittstelle ist nur in der Richtung vom Host auf dem Mikrocontroller möglich, sodass die Datenübertragung durch den Host realisiert werden muss. Dazu ist ein Modul entwickelt worden, dass die generierten Interrupts des Mikrocontrollers verarbeitet, Interrupts auf dem Mikrocontroller auslöst und auf diese Weise die Steuerung der Kommunikation ermöglicht. Außerdem initialisiert es die Datenübertragung, sodass Nachrichten gelesen, als auch geschrieben werden können.

Der Zugriff vom Host auf die Speicherbereiche des Mikrocontrollers wird durch das spezielle Software-Framework UIO (vgl. Koch / Tsirkin 2009) ermöglicht, das es erlaubt I/O-Operationen im Userspace durchzuführen. Sie gewährt den Zugriff auf die Speicherbereiche des Mikrocontrollers via Memory-Mapped-IO im Userspace vom Host und erleichtert dadurch die Implementierung, da die wesentlichen Teile der Anwendung im Userspace ausgeführt werden können. Im Kernelspace müssen die generierten Interrupts des Mikrocontrollers in den Userspace weitergeleitet werden.

Implementierung des Kommunikationsmoduls auf dem Mikrocontroller

Da der Speicherzugriff via DPM-Schnittstelle nur unidirektional möglich ist, beschränkt sich die Implementierung auf dem Mikrocontroller auf das Entgegennehmen und Auslösen von Interrupts. So wird die Datenübertragung gesteuert, die Konfiguration der Speicherbereiche durchgeführt und die Weiterleitung von zu sendenden und empfangenen Nachrichten umgesetzt. Interrupts, die vom Host generiert werden, werden verwendet um die Nachrichten nach der Datenübertragung an den Echtzeit-Ethernet-Netzwerkstack weiterzugeben. In Gegenrichtung initialisieren sie das Lesen von Nachrichten, indem der Host nach dem Versand bzw. Empfang einer Nachricht informiert wird.

Beschreibung der Kommunikation beim Senden einer Nachricht

Im nachfolgenden UML-Sequenzdiagramm 5.7 auf der nächsten Seite ist der implementierte Kommunikationsablauf dargestellt, der das Versenden einer generierten Nachricht bis zum Empfang beim SUT darstellt.

Der erste Schritt innerhalb der Kommunikation zwischen Host und Mikrocontroller besteht daraus, dass aus der Simulationsumgebung eine generierte Nachricht zu versenden ist und diese Nachricht an die Kommunikationsschnittstelle `SimComm` weitergereicht wird. Sie ist für die Übertragung der Daten und Nachrichten zwischen Host und Mikrocontroller verantwortlich. Dabei sendet sie per Interrupt einen `SchreibRequest()` an die Kommunikationsschnittstelle des Mikrocontrollers `MikroComm`, um zu überprüfen, ob der Mikrocontroller bereit ist, neue Daten zu empfangen und nicht mit der Verarbeitung einer vorherigen Nachricht beschäftigt ist. Wenn diese mit einem `Reply(OK)` beantwortet wurde, schreibt der `SimComm` die generierte Nachricht in den Speicher des Mikrocontrollers und löst einen Interrupt aus, sobald der Kopiervorgang abgeschlossen ist (`Ready()`). Anschließend leitet `MikroComm` die Nachricht an den Echtzeit-Ethernet-Netzwerkstack `MikroRTEStack` via `VerarbeiteNachricht()` weiter. Dieser ist anschließend für das Versenden gemäß ihrer Übertragungscharakteristik verantwortlich.
Damit die gesendete Nachricht mit dem Sendezeitpunkt in der Simulationsumgebung abgespeichert werden kann, muss sie im nächsten Schritt zurück übertragen werden.

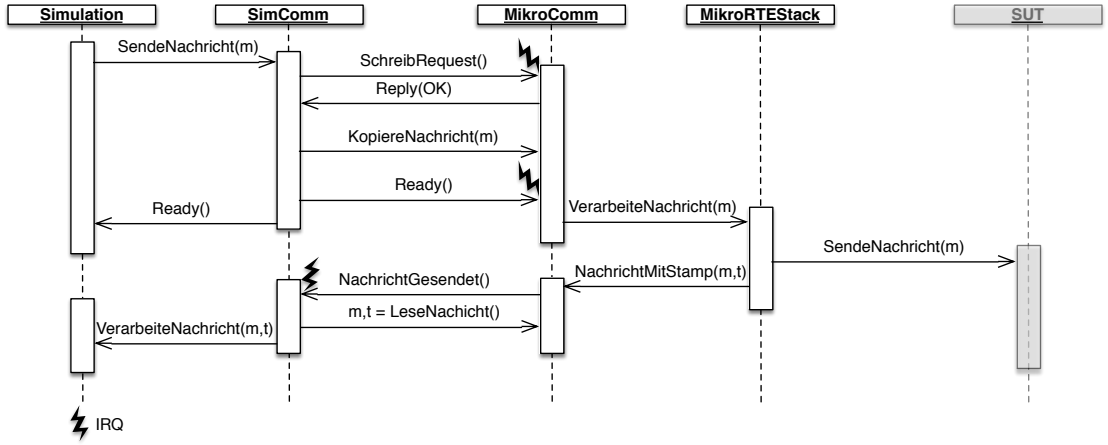


Abbildung 5.7: *Kommunikation zwischen Host und Mikrocontroller beim Senden einer Nachricht*

Der Echtzeit-Ethernet-Netzwerkstack versieht die Nachricht mit einem Zeitstempel und teilt `MikroComm` mit, dass die Nachricht in die Simulation übertragen werden kann (`NachrichtMitStamp(m,t)`). Anschließend teilt `MikroComm` dem Host per Interrupt mit, dass die Nachricht versendet wurde (`NachrichtGesendet()`). Der Host kopiert (`LeseNachricht()`) sodann die Nachricht mit Zeitstempel auf den Host, der diese weiterverarbeiten kann.
Nachrichten, die nicht aus der Simulationsumgebung und autonom vom Echtzeit-Ethernet-Stack versendet werden (PCF Synchronisationsnachrichten), werden ebenfalls mit einem Zeitstempel versehen und auf die gleiche Art zum Host übertragen, damit auch diese Nachrichten zur Analyse hinzugezogen werden können.

Beschreibung der Kommunikation beim Empfang einer Nachricht

Die Kommunikation zwischen Mikrocontroller und Host beim Empfang einer Nachricht ist im UML-Sequenzdiagramm 5.8 auf der nächsten Seite dargestellt. Der Ablauf ist ähnlich zu dem Übertragen einer gesendeten Nachricht. Der Unterschied liegt im Ausgangspunkt der Kommunikation, der in diesem Fall beim Echtzeit-Ethernet-Stack liegt. Dieser empfängt eine Nachricht, versieht sie mit dem Empfangszeitpunkt und teil es dem `MikroComm` mit. Im Anschluss wird ein Interrupt auf dem Host ausgelöst (`NachrichtEmpfangen()`), der den Empfang einer Nachricht signalisiert. Der Host kopiert (`LeseNachricht()`) daraufhin die empfangene Nachricht vom Mikrocontroller zur weiteren Verarbeitung.

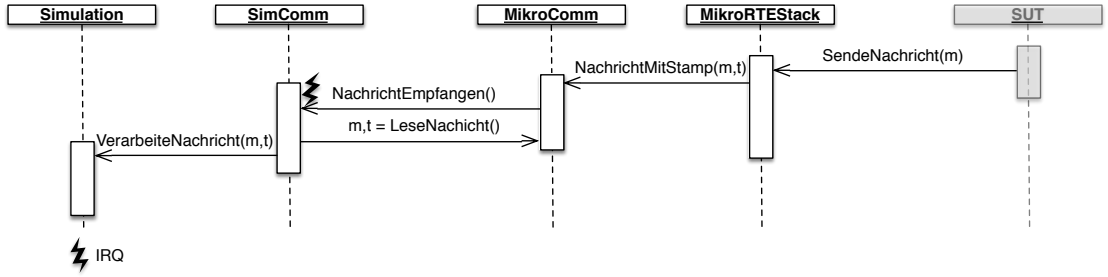


Abbildung 5.8: *Kommunikation zwischen Host und Mikrocontroller beim Empfangen einer Nachricht*

5.2.5 Umsetzung der Simulationsumgebung

Nachdem die Implementierung der Kommunikation zwischen Host und Mikrocontroller beschrieben worden ist, folgt die Beschreibung der Umsetzung der Simulationsumgebung. Sie ist in der Lage, abstrakte Testfälle einzulesen, um daraus die benötigte Ereignisliste zu erstellen und anschließend die Ereignisse zu den geplanten Zeitpunkten auszuführen. Außerdem wird der Datenverkehr zwischen dem Restbussimulator aufgezeichnet. Das Ablaufdiagramm 5.9 auf der nächsten Seite stellt die Funktionsweise der Simulationsumgebung dar und teilt sich in die Phasen *Initialisierung*, *Simulation* und *Auswertung* auf. In der Initialisierungsphase wird die Umwandlung der abstrakten zu ausführbaren Testfällen durchgeführt und die Ereignisliste erstellt. In der Simulationsphase wird der Datenverkehr aufgezeichnet und die Ereignisse solange ausgeführt, bis kein Eintrag mehr in der Ereignisliste vorhanden und das Ende der Simulation bzw. des Testfalls erreicht ist. Der letzte Schritt ist das Abspeichern des aufgezeichneten Datenverkehrs in das gewünschte Dateiformat in der Auswertungsphase.

Implementierung der Initialisierungsphase

Der erste Schritt innerhalb der Simulationsumgebung ist die Erstellung der Ereignisliste, die die Simulation treibt. Das bedeutet, dass die abstrakten Testfälle in ihr ausführbares Pendant umgewandelt und anschließend als Ereignis-/Aktionszeitpaar in die Ereignisliste eingetragen werden. Sie wird nach der Aktionszeit in aufsteigender Weise sortiert, sodass die Reihenfolge der Ereignisausführung festgelegt ist.
Die Umwandlung von abstrakten zu ausführbaren Testfällen wird durch die statische Konfiguration der Simulationsumgebung ermöglicht und bedeutet, dass für jeden zeitabhängigen Eingang (U) eine Nachricht mit Inhalt versehen wird. Da das XML-Modell der abstrakten Testfälle eine Zuordnungsmöglichkeit des Eingangsvektors (U) zu realen Nachrichten implementiert, wird das passende Nachrichtengerüst der statischen Konfiguration verwendet

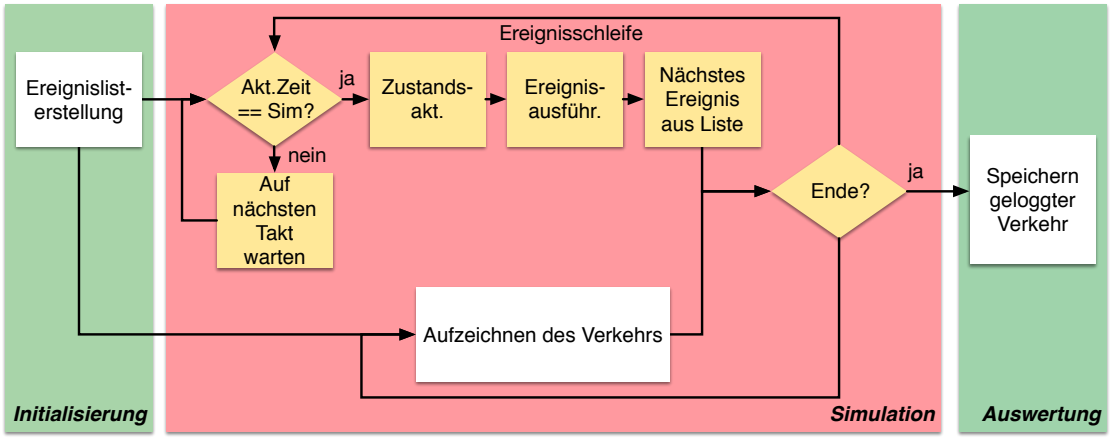


Abbildung 5.9: *Ablauf der Ausführung des Simulationsmodells und parallele Aufzeichnung des Datenverkehrs*

und anschließend mit den definierten Inhalten des Testfalls versehen. Die Zeitpunkte des abstrakten Testfalls werden dementsprechend als Aktionszeitpunkte verwendet, sodass ein Ereignis-/Aktionszeitpaar aus einer Nachricht und dem Zeitpunkt besteht.

Implementierung der Simulationsphase

Nachdem die Ereignisliste in der Initialisierungsphase erstellt worden ist, werden die Ereignisse in der Simulationsphase ausgeführt. Dafür wurde die Ereignisschleife entsprechend des Ablaufdiagramms 5.9 implementiert, die die Ereignisse entsprechend ihres Aktionszeitpunktes ausführt. Da die Aktionszeit der Simulationszeit entspricht, ist ein Taktgeber basierend auf einem POSIX-Timer implementiert worden, der die Simulationszeit entsprechend eines konfigurierten Taktes inkrementiert. Durch die Verwendung der POSIX-Timer wird eine hohe zeitliche Genauigkeit bei der Inkremententierung der Simulationszeit erreicht, sodass auch die Ausführung der Ereignisse genau getimed werden kann.

Implementierung der Analysephase

Das Aufzeichnen des Datenverkehrs wird parallel zur Simulation durchgeführt. Dieses wurde durch die Implementierung eines parallel ausgeführten Threads realisiert, der die empfangenen Nachrichten des Kommunikationsmoduls vom Host entgegennimmt und zusammen mit der Simulationsphase beendet wird. Damit die Aufzeichnung während der Laufzeit wenig Einfluss

auf die Simulation hat, wird der gesamte Datenverkehr gepuffert und am Ende der Simulation in die jeweilige Darstellungsvariante überführt und abgespeichert.

Die Darstellung des aufgezeichneten Datenverkehrs ist durch zwei Varianten realisiert worden um eine möglichst effiziente Analyse durchführen zu können. Dabei wurden die vorgestellten Möglichkeiten umgesetzt, sodass zum einen eine durch den Benutzer konfigurierbare, textbasierte Darstellung umgesetzt worden ist. Zum anderen werden die aufgezeichneten Nachrichten in dem bekannten PCAP-Binärformat (vgl. Jacobson / Leres / McCanne) abgespeichert, das die Möglichkeit bereitstellt, den aufgezeichneten Verkehr mit dem weit verbreiteten Netzwerkanalysetool Wireshark (vgl. Combs) zu analysieren.
Für die Darstellung in einer konfigurierbaren Textform müssen die in der statischen Konfiguration bereitgestellten Nachrichten eine `toString()`-Methode implementieren. Auf diese Weise lässt sich die Darstellung durch den Benutzer steuern, sodass z. B. sprechende Namen für Inhalte der Nachrichten verwendet werden können. Unter Umständen kann dadurch die Analyse der funktionalen Anforderungen vereinfacht werden, wenn auf dem ersten Blick fehlerhaftes Verhalten erkannt wird.
Die Möglichkeit den aufgezeichneten Verkehr in Wireshark zu betrachten hat den Vorteil, bereits auf implementierte Analysemöglichkeiten des Tools zurückzugreifen und den Analyseprozess zu vereinfachen. So erlaubt es Wireshark dem Benutzer, den gesamten Nachrichtenverkehr nach bestimmten Nachrichtenarten zu filtern und bestimmte Informationen aus dem aufgezeichneten Datenverkehr per spezieller Abfragen zu ermitteln. Das Abspeichern im PCAP-Format wurde durch die Verwendung der Softwarebibliothek libpcap (vgl. Jacobson / Leres / McCanne) umgesetzt.

5.2.6 Bereitstellung von Schnittstellen zur statischen Konfiguration

Damit die statische Konfiguration der Restbussimulation durchgeführt werden kann, werden spezielle Schnittstellen in der Software sowohl in der Simulationsumgebung als auch im Echtzeit-Ethernet-Netzwerkstack auf dem Mikrocontroller benötigt. Wie bereits im vorherigen Architekturabschnitt beschrieben (siehe 5.1.3 auf Seite 80), ändert sich die statische Konfiguration nicht zur Laufzeit der Simulation, sodass sich diese Konfiguration entsprechend im Quellcode des Restbussimulators einbinden lässt. Beide Komponenten benötigen unterschiedliche Informationen, sodass diese auch in den Schnittstellen bereitzustellen und in separaten Modulen der jeweiligen Komponenten untergebracht sind.

Auf Seiten des Mikrocontrollers muss der Echtzeit-Ethernet-Netzwerkstack mit den statischen Informationen konfiguriert werden, damit die Vermittlung der zu versendenden und zu empfangenen Nachrichten gemäß der Übertragungsstrategie stattfindet sowie die Ausführung des passenden Synchronisierungsdienst ermöglicht. Da diese Schnittstelle bereits durch den Echtzeit-Ethernet-Netzwerkstack als eigenes Modul in Quellcode-Form vorhanden ist, wird diese für die statische Konfigurierung ohne Veränderung verwendet.

Zusätzlich zu der Konfigurierung des Netzwerkstacks ist ein weiteres Modul implementiert worden, das als Schnittstelle dient, um die vom Host auf den Mikrocontroller übertragenen Nachrichten zu verarbeiten und in die vom Netzwerkstack bereitgestellten Nachrichtenbuffer einzutragen. Dieses ist besonders wichtig, da der Netzwerkstack für jede konfigurierte Echtzeit-Ethernet-Nachricht einen separaten Buffer bereitstellt. Anhand des Critical-Identifiers (siehe Abschnitt 2.3.3 auf Seite 25) werden Nachrichten in TTEthernet unterschieden und in die dazugehörigen Buffer eingetragen.

Die statische Konfiguration der Simulationsumgebung auf dem Host ist ebenfalls in Form eines separaten Moduls im Quellcode implementiert worden. Im Gegensatz zur statischen Konfiguration des Echtzeit-Ethernet-Netzwerkstacks müssen keine Informationen bezüglich des Echtzeit-Ethernet-Protokolls konfiguriert werden. Stattdessen ist eine Möglichkeit geschaffen worden, die zu versendenden und zu empfangenen Nachrichten und deren Gerüste zu konfigurieren. Auf diese Weise ist es möglich, gezielt auf Datenstrukturen innerhalb einer Nachricht zuzugreifen und mit Daten zu versehen. Dazu werden die generierten Datenstrukturen des Code-Generators verwendet, sodass diese Daten in einem eigenen Modul zur Verfügung stehen.
Weiterhin ist eine Abbildungsfunktion zur Umwandlung der abstrakten Testdaten in reale Daten nötig und wurde durch eine 1:1-Relation implementiert. Damit können sprechende Namen innerhalb der abstrakten Testfälle für den Eingang verwendet werden, um in diskreten Systemen Zustandsübergänge innerhalb des Testfalls deutlich zu machen. Diese Daten müssen durch den Benutzer der Testumgebung ebenfalls vor der Compilezeit konfiguriert werden. Für jede Datenstruktur, die in einer Nachricht definiert ist, ist deshalb eine eigene Relation durch den Benutzer zu implementieren.

Auf diese Weise ist der Restbussimulator komplett konfigurierbar und kann zum Testen von funktionalen und zeitlichen Leistungsanforderungen verwendet werden. Dieses ist im Nachfolgenden Kapitel beispielhaft beim Test der Scheinwerferkomponente durchgeführt.

6 Echtzeit-Ethernet Restbussimulation im Anwendungsbeispiel

Dieses Kapitel beschreibt eine Echtzeit-Ethernet Restbussimulation unter der Ausführung von abstrakten Testfällen. Als Anwendungsbeispiel wird die in Kapitel 3.1 auf Seite 36 beschriebene Scheinwerfersteuerung verwendet und ausgewählte Anforderungen überprüft. Zuerst wird das Konfigurieren der Restbussimulation beispielhaft unter der Verwendung eines FIBEX-Dokumentes beschrieben. Anschließend werden ausgewählte Anforderungen durch Ausführung abstrakter Testfälle überprüft und das Verhalten getestet.

6.1 Konfiguration der Restbussimulation

Der erste Schritt bei einer Echtzeit-Ethernet Restbussimulation ist das statische Konfigurieren der Simulationsumgebung und die Auswahl einer passenden Topologie. Damit eine Konfiguration möglich ist, müssen Informationen des realen Netzwerks vorhanden sein und falls erforderlich, mögliche Bandbreite-, bzw. Schedulingkonflikte berechnet werden. Der letzte Schritt beim Aufsetzen der Restbussimulation ist die Generierung der statischen Konfiguration bei dem die RT-Ethernet-Netzwerkstack-typischen Attribute sowie Nachrichtengerüste erstellt werden.

6.1.1 Verwendung von FIBEX als Netzwerkbeschreibung des realen Systems

Damit eine Restbussimulation vollständig konfiguriert werden kann, müssen alle relevanten Informationen des realen Systems vorhanden sein. Dieses wurde in Form eines FIBEX-Dokument realisiert, welches das in Kapitel 3 auf Seite 36 vorgestellte, gesamte Echtzeit-Ethernet Drive-by-Wire-Demonstrator-System darstellt. So wurden alle Teilnehmer im Netz, die versendeten Nachrichten, deren Aufbau, Übertragungsstrategien und Schedulings auf den Endsystemen und Switches sowie alle virtuellen Links und die Rollen im Synchronisierungsprozess definiert. Dadurch ist es möglich die statische Konfiguration der Restbussimulation für jede ausgewählte Komponente als SUT zu erstellen.

6.1.2 Berechnung möglicher Konflikte und Erstellung der statischen Konfiguration

Der Fokus dieser Arbeit liegt auf der Überprüfung von zeitlichen Leistungsanforderungen, die nur auf der Komponententestebene verifiziert werden können, wenn des SUT direkt mit dem Simulator verbunden ist. Damit der Ablauf bei der Erstellung der statischen Konfiguration trotzdem vollständig erwähnt wird, ist die Berechnung möglicher Scheduling- bzw. Bandbreitenkonflikte Konflikte nachfolgend beschrieben, wenn beide Scheinwerfer während der Integrationsphase getestet werden.

Das Scheinwerfersystem besteht aus einem rechten und einem linken Scheinwerfer, die jeweils getrennt voneinander durch das Infotainmentsystem angesteuert werden. Die Nachrichtenübertragung zwischen Komponenten ist aufgrund des ereignisbasierten Verhaltens der Zustandsänderungen per RC-Nachrichtenklasse spezifiziert. Werden beide Scheinwerfer zusammen in der Integrationstestphase als SUT betrachtet, können aufgrund der RC-Charakteristik keine Schedulingkonflikte auftreten, da die Übertragung unabhängig vom Netzwerkzyklus ist. Die zu übertragenden Nachrichten sind als minimale Ethernet-Frames (64Byte) mit einer *Bandwidth-Allocation-Gap* von 500µs definiert. Daraus lässt sich die Worst-Case Bandbreite berechnen, die eine Scheinwerferkomponente beim Senden benötigt. Sie liegt bei 1,344 $^{\text{MByte}}/_{\text{s}}$ und für zwei Scheinwerfer bei 2,688 $^{\text{MByte}}/_{\text{s}}$, sodass auch auf Seite der Bandbreite keine Konflikte möglich sind und eine Restbussimulation für zwei Komponenten mit nur einem Netzwerkinterface durchgeführt werden kann.

Aufbauend auf diesem Ergebnis wird die statische Konfiguration für den Restbussimulator erstellt. Da die Scheinwerfer den *Synchronization-Client* im Synchronisationsprozess ausführen, muss der Restbussimulator den Dienst des *Sychronization-Masters* ausführen. Während der Testphase wird neben der Zustandsänderungsnachricht, auch die autonom versendeten Synchronisationsnachrichten vom Restbussimulator erzeugt. Empfangen werden die Status-Nachrichten der Scheinwerfer.

6.2 Ausführung der Restbussimulation

Nachdem die Restbussimulation statisch konfiguriert wurde, können die Scheinwerfer im Komponententest überprüft werden, um sowohl funktionale, als auch zeitliche Leistungsanforderungen zu überprüfen. Beispielhaft wird die Überprüfung von vier Funktionen der Scheinwerfer und dessen zeitliche Leistungsanforderungen überprüft. Der Aufbau der Restbussimulation für einen Scheinwerfer während des Komponententest ist in Abbildung 6.1 auf der nächsten Seite dargestellt. In diesem Fall ist der Scheinwerfer als einzelne Komponente direkt mit dem Restbussimulator verbunden und empfängt die Synchronisations- und die

generierten `SetzeScheinwerferZustand`-Nachrichten. Der Restbussimulator empfängt in diesem Fall die `SendeAkutellenZustand` des Scheinwerfers.

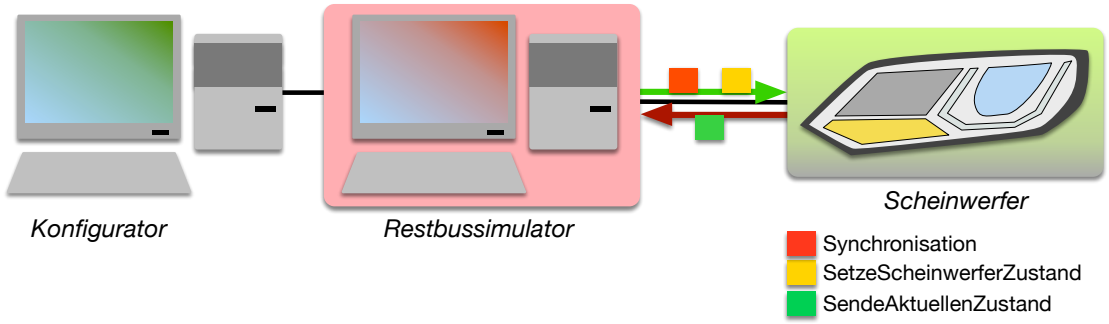


Abbildung 6.1: *Versuchsaufbau beim Komponententest mit einem Scheinwerfer als System-under-Test*

Da das zeitliche Leistungsverhalten ein wichtiges Attribut von der Scheinwerfersteuerung ist und für alle Funktionen des Scheinwerfers gilt, werden diese zuerst überprüft. Anschließend werden funktionale Anforderungen der ausgewählten Funktionen getestet.

6.2.1 Überprüfung der Leistungsanforderungen und Zustandsquittierung

Zuerst werden in einem Testfall die zeitlichen Leistungsanforderungen der Reaktionszeit und der Senderate (vgl. Anforderungen $KS_{LA}1$, $KS_{LA}2$, $KS_{LA}3$ und $KS_{LA}4$) des Scheinwerfers überprüft und ist in der nachfolgenden Tabelle 6.1 auf der nächsten Seite dargestellt. Abgeleitet wurde der Testfall aus den in den Abbildungen 3.3 auf Seite 50, 3.4 auf Seite 51 und 3.5 auf Seite 51 dargestellten Zustandsautomaten und Sequenzdiagrammen. Der Zustandsautomat stellt die funktionalen Anforderungen der Zustandsquittierung dar, während die zeitlichen Leistungsanforderungen innerhalb der Sequenzdiagramme modelliert sind.

In diesem Testfall wird eine alternierende Zustandsänderung an den Scheinwerfer gesendet, um die Reaktionsgeschwindigkeit bei der Übernahme des neuen Scheinwerferzustands und die periodische Senderate des aktuellen Scheinwerferzustands zu überprüfen. Der alternierende Zustand wird in diesem Testfall jede Sekunde gesendet und schaltet den Hauptscheinwerfer abwechselnd über einem Zeitraum von 60 Sekunden ein und aus.
Zu Beginn des Tests wird der Zustand des Scheinwerfers mit der Zustandsnachricht `SW_AUS` zurückgesetzt. Dieser *Reset-Befehl* findet sich in allen durchgeführten Testfällen mit der gleichen Bedeutung wieder. Als nächstes wird die Zustandsänderung `HS_EIN` zum Zeitpunkt T-2s zum Scheinwerfer gesendet und startet mit dem Einschalten des Hauptscheinwerfers

und wird 60 Sekunden lang alternierend wiederholt. Zum Zeitpunkt T-62s wird der Hauptscheinwerfer ausgeschaltet und der Testfall ist beendet.
Die zeitliche Leistungsanforderung der Reaktionszeit ist durch die Latenz und dessen Jitter dargestellt, die in diesem Fall bei 500µs bzw. 50µs liegt. Die Senderate des Scheinwerfers und dessen Jitter betragen in diesem Fall 5000µs sowie 10µs.

T	1s	2s	3s	...	61s	62s
U	$u_1 =$ SW_AUS	$u_1 =$ HS_EIN	$u_1 =$ HS_AUS	...	$u_1 =$ HS_EIN	$u_1 =$ HS_AUS
Y_{soll}	$y_1 =$ SW_AUS	$y_1 =$ HS_EIN	$u_1 =$ HS_AUS	...	$u_1 =$ HS_EIN	$y_1 =$ SW_AUS
Y_{ist}	$y_1 =$ SW_AUS	$y_1 =$ HS_EIN	$y_1 =$ SW_AUS	...	$y_1 =$ HS_EIN	$y_1 =$ SW_AUS
L_{soll}	$l_1(u_1, y_1) =$ 500µs					
J_{Lsoll}	$j_{L1}(l_1) \leq$ 50µs					
L_{ist}	$l_1(u_1, y_1) =$ 517 - 519µs, MEDIAN=518µs, MITTEL=517,84µs					
R_{soll}	$r_1(y_1) =$5000µs					
J_{Rsoll}	$j_{R1}(r_1) \leq$ 10µs					
R_{ist}	$r_1(y_1) =$4997 - 5002µs, MEDIAN=5000µs, MITTEL=5000µs					

Tabelle 6.1: *TF1: Überprüfung zeitlicher Leistungsanforderungen*

Die anschließende Auswertung des aufgezeichneten Verkehrs ist in der gleichen Tabelle 6.1 mit den erwarteten *Soll-* und den tatsächlichen *Istwerten* abgebildet. Der Scheinwerfer quittiert die gesendeten, alternierenden Zustände jeweils mit dem erwarteten Wert. Dieses Verhalten ist deshalb durch grün-markierte Felder in der Tabelle 6.1 dargestellt. Auch die zeitlichen Leistungsanforderungen werden erfüllt und sind dementsprechend grün markiert. Die Reaktionsgeschwindigkeit des Systems liegt auf dem Intervall 517-519µs und im Mittel bei 518µs. Durch die Angabe des maximal zulässigen Jitters ($\leq$ 50µs) werden die die Anforderungen der geforderten 500µs erfüllt. Auch die Anforderungen der periodischen Übertragung der Nachrichten werden erfüllt. Die Senderate liegt im Intervall zwischen 4997 und 5002µs und im Mittel bei 5000µs, sodass die Anforderung der Senderate genau erfüllt wird und unterhalb des zulässigen Jitters von 10µs liegt.

6.2.2 Überprüfung der definierten Wertebereiche der LED-Leuchten

Der nächste Testfall überprüft das Reaktionsverhalten der LED-Leuchten (vgl. Anforderung $\texttt{TS}_{\texttt{es}}8$, $\texttt{KS}_{\texttt{es}}3$ und $\texttt{FS}_{\texttt{la}}2$) und ist in Tabellen 6.2 und 6.3 auf Seite 100 dargestellt. Die Anforderung definiert den Helligkeitsbereich der LED-Leuchten auf dem Intervall von 0 bis 100%. Die Daten für den Testfall wurden aus dem in Abbildung A.1 auf Seite 109 dargestellten Zustandsautomaten abgeleitet und eine Grenzwertanalyse durchgeführt, da die Helligkeit der LEDs für einen definierten Wertebreich angegeben wurde. Bei einer Grenzwertanalyse werden Äquivalenzklassen gebildet, die spezifizierte Wertebereiche in gültige und ungültige Klassen unterteilt, wodurch eine hohe Testabdeckung unter Verwendung von wenigen Testdaten erreicht wird. Repräsentanten einer Klasse sollen die gleichen Auswirkungen auf das

SUT haben, sodass die Auswahl eines Repräsentanten genügt. Hierbei wurden die Transitionsguards betrachtet, die die Bildung der Äquivalenzklassen ermöglichen. Die Grenzwerte spiegeln die Repräsentanten dieser Klassen wieder, die genau auf der Grenze zwischen zwei Klassen liegen. Im Testfall finden sich dementsprechend die Grenzwerte der Äquivalenzklassen wieder, sowie zwei Repräsentanten der gültigen Äquivalenzklasse.

Zuerst wird zum Zeitpunkt T-1s der aktuelle Zustand des Scheinwerfers zurückgesetzt. Anschließend werden ab Zeitpunkt T-2s die Testdaten aus der Grenzwertanalyse versendet. Dabei werden zuerst die Grenzen der gültigen Klassen überprüft (`LED_0` und `LED_100`). Ein Repräsentant aus der Mitte der Klasse (`LED_50`) sorgt dafür, dass der Unterschied zur ungültigen Äquivalenzklasse (`LED_101`) deutlich wird. In diesem Fall soll der Scheinwerfer mit dem letzten gültigen Zustand antworten und damit die Ausführung von ungültigen Werten verhindern. Abgeschlossen wird der Testfall mit einem weiteren Repräsentanten aus der gültigen Äquivalenzklasse, um das geforderte Verhalten zu verifizieren. Zusätzlich zu der funktionalen Anforderung wurden auch die zeitlichen Leistungsanforderungen in diesem Testfall angegeben. Sie finden sich auch in den nachfolgenden Testfällen wieder.

T	1s	2s	5s	7s	9s	11s
U	$u_1 =$ SW_AUS	$u_1 =$ LED_0	$u_1 =$ LED_100	$u_1 =$ LED_50	$u_1 =$ LED_101	$u_1 =$ LED_75
Y_{soll}	$y_1 =$ SW_AUS	$y_1 =$ LED_0	$y_1 =$ LED_100	$y_1 =$ LED_50	$y_1 =$ LED_50	$y_1 =$ LED_75
Y_{ist}	$y_1 =$ SW_AUS	$y_1 =$ SW_AUS	$y_1 =$ SW_AUS	$y_1 =$ SW_AUS	$y_1 =$ SW_AUS	$y_1 =$ SW_AUS
L_{soll}	$l_1(u_1, y_1) = 500\mu s$					
J_{Lsoll}	$j_{L1}(l_1) \leq 50\mu s$					
L_{ist}	$l_1(u_1, y_1) = 518$ - 518µs, MEDIAN=518µs, MITTEL=518µs					
R_{soll}	$r_1(y_1) = 5000\mu s$					
J_{Rsoll}	$j_{R1}(r_1) \leq 10\mu s$					
R_{ist}	$r_1(y_1) =$4998 - 5002µs, MEDIAN=5000µs, MITTEL=5000µs					

Tabelle 6.2: *TF2: Überprüfung der LED-Leuchten mit negativem Ergebnis*

Nach der erstmaligen Ausführung des Testfalls (siehe Tabelle 6.2) konnte beobachtet werden, dass die Zustandsänderung der LED-Leuchten zwar durch eine Nachricht quittiert wurde, allerdings ohne den aktuellen Zustand dabei zurückzugeben. So wurde durchgängig der Zustand des zurückgesetzten Scheinwerfers gesendet, sodass der Testfall ein negatives Ergebnis erzielt und ein Fehler in der Umsetzung aufgedeckt wurde (zu sehen an der roten Markierung). Eine Analyse des Quellcodes der Scheinwerferkomponente bestätigte diesen Fehler. Es stellte sich heraus, dass der Helligkeitszustand der LEDs nicht in die Zustandsnachricht integriert wurde, sodass dieses Fehlverhalten anschließend zielorientiert behoben werden konnte.

Bei der anschließenden zweiten Ausführung des Testfalls (Tabelle 6.3 auf der nächsten Seite) ist zu erkennen, dass die Scheinwerferkomponente den neuen Helligkeitszustand der LEDs mit den erwarteten Daten quittiert. Auf diese Weise kann gezeigt werden, dass der Fehler behoben wurde. So werden auf Daten der ungültigen Äquivalenzklasse (`LED_101`) zum Zeitpunkt T-9s mit dem alten gültigen Zustand (`LED_50`) quittiert. Insgesamt wurden alle erwarteten

Ergebnisse erzielt, sodass die zweite Ausführung des Testfalls ein positives Ergebnis erzeugt hat, bei der alle Anforderungen erfüllt wurden.

T	1s	2s	5s	7s	9s	11s
U	u_1 = SW_AUS	u_1 = LED_0	u_1 = LED_100	u_1 = LED_50	u_1 = LED_101	u_1 = LED_75
Y_{soll}	y_1 = SW_AUS	y_1 = LED_0	y_1 = LED_100	y_1 = LED_50	y_1 = LED_50	y_1 = LED_75
Y_{ist}	y_1 = SW_AUS	y_1 = LED_0	y_1 = LED_100	y_1 = LED_50	y_1 = LED_50	y_1 = LED_75
L_{soll}	$l_1(u_1, y_1)$ = 500µs					
J_{Lsoll}	$j_{L1}(l_1) \leq$ 50µs					
L_{ist}	$l_1(u_1, y_1)$ = 517 - 518µs, MEDIAN=518µs, MITTEL=518µs					
R_{soll}	$r_1(y_1)$ = 5000µs					
J_{Rsoll}	$j_{R1}(r_1) \leq$ 10µs					
R_{ist}	$r_1(y_1)$ =4998 - 5002µs, MEDIAN=5000µs, MITTEL=5000µs					

Tabelle 6.3: *TF2: Überprüfung der LED-Leuchten mit negativem Ergebnis*

6.2.3 Überprüfung der Fahrtrichtungsanzeigerfunktion

Als weiteres Beispiel wurde die Funktion des Fahrtrichtungsanzeigers getestet der durch die Anforderungen $AB_{LA}2$, $FS_{FA}9$, $FS_{LA}1$ und $TS_{ES}6$ beschrieben wird. Die Fahrtrichtungsanzeigerfunktion soll nach dem Erhalt des Zustands Blinker_AN dauerhaft bis zum expliziten Beenden im Fahrtrichtungsanzeiger-Zustand bleiben. In diesem Zustand soll in der festgelegten Frequenz von 1Hz der Zustand des Scheinwerfers zwischen Blinker_AN und Blinker_AUS wechseln. Der Testfall ist in den Tabellen 6.4 und 6.5 auf der nächsten Seite abgebildet. In diesem Fall konnte der Testfall aus den in den Abbildungen A.1 auf Seite 109 und A.5 auf Seite 113 dargestellten Zustandsautomaten abgeleitet werden.

Dabei wird wie in den vorherigen Testfällen zuerst der Scheinwerfer zurückgesetzt, sodass der eigentliche Test zum Zeitpunkt 2s mit der Übermittlung des Zustands Blinker_AN (BL_AN) startet. Da die Scheinwerferkomponente selbständig die Funktion des Fahrtrichtungsanzeigers ausführt, werden in diesem Fall bis zum Zeitpunkt 62s keine weiteren Nachrichten generiert, die den Zustand des SUT beeinflussen. Damit der Wechsel zwischen einer ein- und ausgeschalteter Blinkerleuchte dennoch getestet werden kann, müssen die erwarteten Ausgänge zu bestimmten Zeitpunkten definiert werden. Da der Fahrtrichtungsanzeiger mit einer Frequenz von 1Hz spezifiziert ist, muss der Wechsel zwischen eingeschalteten und ausgeschalteten Blinker innerhalb von 0,5 Sekunden durchgeführt werden. Dementsprechend werden im Abstand von 0,5 Sekunden die alternierenden erwarteten Zustände modelliert. Zum Zeitpunkt 62s wird die Fahrtrichtungsanzeiger-Funktion explizit durch das Übertragen einer Zustandsänderung (SW_AUS) beendet.

T	1s	2s	2,75s	3,25s	3,75s	...	62s
U	$u_1 =$ SW_AUS	$u_1 =$ BL_AN	$u_1 = \emptyset$	$u_1 = \emptyset$	$u_1 = \emptyset$	...	$y_1 =$ SW_AUS
Y_{soll}	$y_1 =$ SW_AUS	$y_1 =$ BL_AN	$y_1 =$ BL_AUS	$u_1 =$ BL_AN	$y_1 =$ BL_AUS	...	$y_1 =$ SW_AUS
Y_{ist}	$y_1 =$ SW_AUS	$y_1 =$ BL_AN	$y_1 =$ BL_AN	$u_1 =$ BL_AUS	$y_1 =$ BL_AUS	...	$y_1 =$ SW_AUS
L_{soll}	$l_1(u_1, y_1) = 500\mu s$						
J_{Lsoll}	$j_{L1}(l_1) \leq 50\mu s$						
L_{ist}	$l_1(u_1, y_1) =$517 - 518µs, MEDIAN=518µs, MITTEL=518µs						
R_{soll}	$r_1(y_1) = 5000\mu s$						
J_{Rsoll}	$j_{R1}(r_1) \leq 10\mu s$						
R_{ist}	$r_1(y_1) =$4998 - 5002µs, MEDIAN=5000µs, MITTEL=5000µs						

Tabelle 6.4: *TF3: Überprüfung der Blinkerfrequenz mit negativem Testergebnis*

Die Ergebnisse der ersten Ausführung des Testfalls ist ebenfalls in der Tabelle 6.4 dargestellt und brachte ein negatives Testergebnis zum Vorschein (zu erkennen an der roten Markierung der *Istwerte*. Zum Zeitpunkt T-2s wurde die Fahrtrichtungsanzeigerfunktion gestartet und wurde mit dem erwarteten Wert quittiert. Allerdings wurde zum Zeitpunkt 2,75s der erwartete Wert einer ausgeschalteten Blinkerleuchte (BL_AUS) nicht erfüllt, da zu diesem Zeitpunkt die Scheinwerferkomponente eine eingeschaltete Blinkerleuchte (BL_EIN) vermeldete. Erst zum Zeitpunkt 3,25s ist diese Leuchte ausgeschaltet. Auf diese Weise konnte eine weitere Fehlfunktion entdeckt werden und durch Analyse der Quellcodes bestätigt werden. In diesem Fall konnte eine falsche Parametrisierung der *Pulsweitenmodulation (PWM)* festgestellt werden, dessen Periode doppelt so lang wie spezifiziert konfiguriert wurde. PWM ist für die Ausführung der Blinklichtperiode verantwortlich.
Nach dem Beheben des Fehlers wurde der Testfall erneut ausgeführt und erfolgreich abgeschlossen. Dessen Ergebnis ist in Tabelle 6.5 abgebildet und die erwarteten Ergebnisse wurden in diesem Fall zu allen Zeitpunkten erfüllt. Wie bereits im vorherigen Testfall erwähnt, wurden die zeitlichen Leistungsanforderungen ebenfalls überprüft und brachten ein positives Endergebnis.

T	1s	2s	2,75s	3,25s	3,75s	...	62s
U	$u_1 =$ SW_AUS	$u_1 =$ BL_AN	$u_1 = \emptyset$	$u_1 = \emptyset$	$u_1 = \emptyset$	...	$y_1 =$ SW_AUS
Y_{soll}	$y_1 =$ SW_AUS	$y_1 =$ BL_AN	$y_1 =$ BL_AUS	$u_1 =$ BL_AN	$y_1 =$ BL_AUS	...	$y_1 =$ SW_AUS
Y_{ist}	$y_1 =$ SW_AUS	$y_1 =$ BL_AN	$y_1 =$ BL_AUS	$u_1 =$ BL_AN	$y_1 =$ BL_AUS	...	$y_1 =$ SW_AUS
L_{soll}	$l_1(u_1, y_1) = 500\mu s$						
J_{Lsoll}	$j_{L1}(l_1) \leq 50\mu s$						
L_{ist}	$l_1(u_1, y_1) =$517 - 518µs, MEDIAN=518µs, MITTEL=518µs						
R_{soll}	$r_1(y_1) = 5000\mu s$						
J_{Rsoll}	$j_{R1}(r_1) \leq 10\mu s$						
R_{ist}	$r_1(y_1) =$4998 - 5002µs, MEDIAN=5000µs, MITTEL=5000µs						

Tabelle 6.5: *TF3: Überprüfung der Blinkerfrequenz mit positivem Testergebnis*

6.2.4 Überprüfung der Überholmanöveranzeigerfunktion

Nachdem die Funktion des Fahrtrichtungsanzeigers erfolgreich überprüft wurde, wird als letztes Beispiel die Funktion des Überholmanöveranzeigers getestet (vgl. Anforderungen $\mathtt{FS}_{\mathtt{FA}}14$, $\mathtt{FS}_{\mathtt{LA}}1$). Im Gegensatz zur Fahrtrichtungsanzeigerfunktion soll die Überholmanöveranzeigerfunktion nach dreimaliger Ausführung einer Blinkerperiode (AN $\rightarrow$ AUS) automatisch beenden werden und dann in den ursprünglichen Zustand zurückkehren. Der Testfall ist in der Tabelle 6.6 dargestellt und wurde aus den in den Abbildungen A.1 auf Seite 109 und A.6 auf Seite 114 dargestellten Zustandsautomaten abgeleitet.

Nachdem die Scheinwerferkomponente zurückgesetzt worden ist, startet der eigentliche Testfall zum Zeitpunkt T-2s mit der Übermittlung der Zustandsänderung zum Starten der Überholmanöverfunktion (ATBL_AN). Anschließend werden, wie im vorherigen Testfall keine Nachrichten mehr definiert und der Testfall endet zum Zeitpunkt T-5s, indem die Scheinwerferkomponente den Ausgangszustand übermitteln soll.

T	1s	2s	2,75s	3,25s	...	4,75s	5s
U	$u_1 =$ SW_AUS	$u_1 =$ ATBL_AN	$u_1 = \emptyset$	$u_1 = \emptyset$	...	$u_1 = \emptyset$	$u_1 = \emptyset$
Y_{soll}	$y_1 =$ SW_AUS	$y_1 =$ ATBL_AN	$y_1 =$ ATBL_AUS	$u_1 =$ ATBL_AN	...	$y_1 =$ ATBL_AUS	$y_1 =$ SW_AUS
Y_{ist}	$y_1 =$ SW_AUS	$y_1 =$ ATBL_AN	$y_1 =$ ATBL_AUS	$u_1 =$ ATBL_AN	...	$y_1 =$ ATBL_AUS	$y_1 =$ SW_AUS
L_{soll}	$l_1(u_1, y_1) = 500\mu s$						
J_{Lsoll}	$j_{L1}(l_1) \leq 50\mu s$						
L_{ist}	$l_1(u_1, y_1) =$ 517 - 518µs, MEDIAN=518µs, MITTEL=518µs						
R_{soll}	$r_1(y_1) = 5000\mu s$						
J_{Rsoll}	$j_{R1}(r_1) \leq 10\mu s$						
R_{ist}	$r_1(y_1) =$4998 - 5002µs, MEDIAN=5000µs, MITTEL=5000µs						

Tabelle 6.6: *TF4: Überprüfung der Überholmanöverfunktion*

Die erste Ausführung des Testfalls endete mit einem positiven Ergebnis (zu sehen an den grünmarkierten Zellen der Tabelle), bei der alle erwarteten Ergebnisse des SUT erfüllt wurden. So wird die geforderte Frequenz des Fahrtrichtungsanzeigers, sowie das dreimalige Ausführen einer Blinkerperiode eingehalten. Der Scheinwerfer befindet sich zum Zeitpunkt 5s wieder im Ausgangszustand.

6.3 Bewertung der Testfallergebnisse

Die Implementierung des Echtzeit-Ethernet Restbussimulators hat das Testen von verteilten Automotiveanwendungen auf der abstrakten Datenebene ermöglicht. So konnten sowohl funktionale, als auch zeitliche Leistungsanforderungen der beispielhaften Scheinwerfersteuerung überprüft werden und Fehler innerhalb der Komponentenimplementierung aufgedeckt und behoben werden.

Die Systementwicklung von Automotiveanwendungen wird, wie in der Grundlagen beschrieben, dezentral durch verschiedene Hersteller durchgeführt, sodass die Entwicklung einer Scheinwerferkomponenten in der Regel unabhängig von der Entwicklung der Bedien- und Informationsschnittstelle ist. Durch den Einsatz des entwickelten Echtzeit-Ethernet Restbussimulators kann das Testen der einzelnen Komponenten ohne das Vorhandensein des Infotainmentsystems dennoch durchgeführt werden. Der Integrationsprozess wird auf diese Weise beschleunigt, da Fehler schon vor der eigentlichen Integration entdeckt wurden.

7 Zusammenfassung, Fazit und Ausblick

Dieses Kapitel schließt diese Arbeit mit einer Zusammenfassung, einem Fazit und einem Ausblick auf künftige Arbeiten ab.

7.1 Zusammenfassung der Arbeit

Ziel dieser Arbeit war es, eine modellbasierte Testmethodik für verteilte Automotiveanwendungen im Fahrzeugnetzwerk der nächsten Generation zu entwickeln. Dafür wurde eine Restbussimulationsplattform für Echtzeit-Ethernet implementiert, die das Verhalten von nicht vorhandenen Knoten simuliert. Auf diese Weise können funktionale und zeitliche Leistungsanforderungen frühzeitig im dezentralen Entwicklungsprozess überprüft werden.

Zuerst wurden Hintergrundinformationen zu den Eigenschaften von Automotivesystemen und zum zugehörigen Entwicklungsprozess gegeben. Heutige Automotivesysteme sind komplexe, verteilte Systeme, die Funktionen zum größten Teil in Software umsetzen und domänenübergreifend kommunizieren. Sie besitzen sowohl kontinuierliche, als auch diskrete Eigenschaften und sind in verschiedenen Anwendungsdomänen anzutreffen. Der Entwicklungsprozess dieser Systeme ist dezentral und durch verschiedene Zulieferer geprägt, sodass Ingenieure verschiedener Fachbereiche zusammenarbeiten müssen. Der Prozess ist als V-Modell umgesetzt, das den Entwicklungsprozess in eine Konstruktions- und Verifikations-/Validierungsphase aufteilt. Dabei findet ein durchgängiger Top-Down-Prozess statt, der das System in den verschiedenen Phasen verfeinert. Die modellbasierte Entwicklung hat dabei einen immer größeren Einfluss auf den Entwicklungsprozess. Sie dient zum einen als gemeinsame Sprache im dezentralen Entwicklungsprozess und zum anderen dient sie dazu frühzeitig fehlerhaftes Verhalten in der Entwicklung zu erkennen. Formale Modelle erlauben es die Systeme analytisch zu betrachten oder Simulationsmodelle aus ihnen zu erstellen. Die erstellten Modelle eignen sich weiterhin dazu, modellbasiert Testfälle abzuleiten und anschließend auf verschiedenen Plattformen auszuführen. Sie sorgen dafür, dass die entwickelten Systeme systematischer auf verschiedenen Ebenen getestet werden können.
Die *Restbussimulation* hat sich dabei als Testmethodik im modellbasierten Entwicklungsprozess etabliert und ermöglicht es in der Entwicklung befindliche verteilte Komponenten und Systeme zu testen, die bereits auf realen Datenverkehr angewiesen sind. Im Gegensatz zur verwandten *Hardware-in-the-Loop-Simulation* kann eine Analyse des System-under-Tests nur

auf der abstrakten Datenebene durchgeführt werden, da ein Restbussimulator keine Verbindung zu den Sensoren und Aktoren des System-under-Tests besitzt. Damit die Komplexität moderner Fahrzeugnetzwerke während der Restbussimulation beherrscht werden kann, werden nur die für das System-under-Test relevanten Nachrichten versendet und so das Verhalten des „Rest“ des Busses simuliert. Das Verhalten der simulierten Knoten kann während der Laufzeit *online* oder vor der Simulation *offline* errechnet werden. Letztere Variante setzt ein weniger komplexes Simulationsmodell zur Ausführung voraus, wodurch weniger Rechenleistung benötigt wird.
Bisherige heterogene Kommunikationsstrukturen im Fahrzeug stoßen in naher Zukunft an ihre Grenzen, da sie aufgrund der Komplexität nicht mehr beherrschbar sind und für zukünftige Anwendungen zu wenig Bandbreite zur Verfügung stellen. Echtzeit-Ethernet ist ein geeigneter Kandidat das Fahrzeugnetzwerk der nächsten Generation zu werden, da sich diese Technologie bereits in verschiedenen verwandten Umgebungen bewiesen hat. *Time-Triggered Ethernet* als echtzeitfähige Ethernet-Variante stellt dabei drei verschiedene Kommunikationsklassen zur Verfügung, die sich in heutigen Netzwerken wiederfinden und verschiedene Übertragungsanforderungen erfüllen. Auf diese Weise lässt sich ein *flaches*, homogenes Fahrzeugnetzwerk mit hoher Bandbreite und Echtzeitfähigkeit realisieren, das ohne komplexe Gateways auskommt und somit die Beherrschbarkeit von Fahrzeugnetzwerken vereinfacht.

Im nachfolgenden Kapitel wurde die modellbasierte Entwicklung von Echtzeit-Ethernet Applikationen anhand einer Scheinwerfersteuerung beispielhaft dargestellt. Dabei wurde zuerst die Anwendung in natürlicher Sprache beschrieben und anschließend die Anforderungstaxonomie erklärt. Im nächsten Schritt wurden die Anforderungen der Scheinwerfersteuerung in funktionale Anforderungen, Leistungs- und spezifische Qualitätsanforderungen und Einschränkungen eingeteilt. Im Anschluss an die Beschreibung wurden Anforderungen an eine Modellierungssprache für Echtzeit-Ethernet basierte Systeme gestellt, um eine passende Modellierungssprache auswählen zu können. Dabei hat sich die UML-Erweiterung *Modeling and Analysis of Real-time Embedded Systems (MARTE)* als passend herausgestellt, sodass ausgewählte Attribute der Scheinwerfersteuerung mit dieser Erweiterung modelliert wurden. Insbesondere die *Value Specific Language* ermöglicht es, zeitliche Leistungsanforderungen zu modellieren, sodass diese in Sequenzdiagrammen dargestellt wurden.
Damit die Modellierung auch aus mathematischer Sicht möglich ist, wurde das Zustandsraummodell dargestellt. Dieses Modell erlaubt es, sowohl kontinuierliche, als auch diskrete Attribute anhand von mathematischer Funktionen zu beschreiben, bei der die Ein- und Ausgänge sowie die internen Zustände als Vektoren dargestellt werden. Das Zustandsraummodell wird als Basis verwendet, um ein abstraktes Testfallmodell zu erweitern.
Damit Testfälle auf verschiedenen Plattformen ausgeführt werden können, müssen sie eine gewisse Abstraktion zur Testumgebung bereitstellen. So wurde ein abstraktes Testfallmodell vorgestellt, mit dem es möglich ist nicht-funktionale Anforderungen zu überprüfen. Für die Überprüfung von zeitlichen Leistungsanforderungen ist dieses Modell allerdings nicht ausrei-

chend und wurde mit vier weiteren Attributen erweitert, die die Überprüfung von der Latenz und Senderate sowie deren Varianzen ermöglicht.

Das anschließende Kapitel befasste sich konkret mit dem Thema der Echtzeit-Ethernet Restbussimulation und dessen Anforderungen. Dabei wurde zuerst ein Vergleich zu bisherigen Feldbus-basierten Restbussimulationsansätzen durchgeführt, der die Unterschiede beider Ansätze erläutert. So unterscheidet sich eine Restbussimulation für Echtzeit-Ethernet von seinem Feldbus-basierten Pendant nicht nur anhand der möglichen Topologien, sondern auch an der Eigenschaft verschiedene Nachrichtenklassen und Synchronisationsprinzipien bereitstellen zu müssen. Die unterschiedlichen Topologien resultieren aus der Tatsache, dass alle Teilnehmer im Echtzeit-Ethernet eine eigene Kollisionsdomäne haben. Eine gleichzeitige Übertragung zweier unterschiedlicher Nachrichten ist möglich, woraus Scheduling-, bzw. Bandbreitenkonflikte resultieren können. Ein weiterer Unterschied der verwendeten Echtzeit-Ethernet-Variante ist die Unterstützung von drei verschiedenen Nachrichtenklassen, welche in etablierten Feldbus-Netzwerken nicht vorkommen und die Übermittlung von expliziten Synchronisationsnachrichten voraussetzt.
Aus diesen Unterschieden und den in den Grundlagen vorgestellten Eigenschaften der verwendeten Echtzeit-Ethernet-Variante haben sich Anforderungen und benötigte Komponenten einer Restbussimulation ergeben, die anschließend dargestellt wurden. Die wichtigste Komponente hierbei ist eine geeignete Hardware-/Softwareplattform, die die Übertragungsstrategien und Synchronisationsrollen der Echtzeit-Ethernet-Variante vollständig unterstützt damit die Kommunikation zwischen dem Simulator und dem System-under-Test ermöglicht wird. Sie muss eine Schnittstelle zur statischen und dynamischen Konfiguration bereitstellen, denn innerhalb der statischen Konfiguration werden die Kommunikationsparameter justiert, die einem FIBEX-Dokument entnommen werden. Die dynamische Konfiguration beschreibt das Verhalten der simulierten Knoten und wird durch das Modell der abstrakten Testfälle umgesetzt, sodass dieses als Simulationsmodell fungieren kann.

Im vorletzten Kapitel wurde das Konzept und die Implementierung der Restbussimulationsumgebung dargestellt. Dabei wurden zuerst verschiedene Ansätze einer geeigneten Systemarchitektur diskutiert und anschließend eine Architektur ausgewählt, die den geforderten Anforderungen entspricht. Eine Kombination aus Mikrocontroller und x86-Workstation ist zum Zeitpunkt dieser Arbeit die geeignetste Alternative, da diese sowohl zeitliche Präzision als auch eine Vielzahl an Softwarebibliotheken bereitstellt. Das anschließende Konzept der Simulationsumgebung erläutert die Ausführung des Simulationsmodells und die Aufzeichnung des Datenverkehrs, damit die Analyse des Verhaltens des System-under-Tests auf abstrakter Datenebene durchgeführt werden kann. Damit das Modell der abstrakten Testfälle als Simulationsmodell verwendet werden kann, muss es in einem geeigneten Datenformat abgebildet werden, welches eine manuelle als auch maschinelle Bearbeitung zulässt, sodass zwei Varianten dargestellt wurden.
Aufbauend auf den vorgestellten Konzepten wurde die Implementierung der Plattform beschrieben. Dabei wurde ein Code-Generator entwickelt, mit dem es möglich ist die stati-

sche Konfiguration auf der Basis eines FIBEX-Datenmodells zu erstellen. Das Simulations- und abstrakte Testfallmodell wurde anschließend als XML-Datenmodell implementiert. Die anschließende Umsetzung der Restbussimulationsarchitektur beschreibt die Kommunikation zwischen Mikrocontroller und Host-PC, damit gesendete und empfangene Pakete in der Simulationsumgebung aufgezeichnet werden können. Die Umsetzung der Simulationsumgebung sowie die Bereitstellung der Schnittstellen zur statischen Konfiguration schließen das Implementationskapitel ab.

Das letzte Kapitel beschreibt die Ausführung einer Restbussimulation zur Überprüfung von nicht-funktionalen Anforderungen in Echtzeit-Ethernet Netzwerken. Hierbei wurde zuerst überprüft, ob eine Restbussimulation durchführbar ist und ob Scheduling, bzw. Bandbreitenkonflikte auftreten. Anschließend wurde die Erstellung der statischen Konfiguration für das Scheinwerfersystem beschrieben.
Ausgewählte funktionale und zeitliche Leistungsanforderungen der Scheinwerfersteuerung wurden nachfolgend getestet, wobei fehlerhaftes Verhalten der Scheinwerfersteuerung festgestellt und abschließend behoben wurden. Die spezifizierten zeitlichen Leistungsanforderungen konnten durch die Ausführung der Testfälle verifiziert werden, sodass sich der Einsatz der Restbussimulation rentiert hat.

7.2 Fazit einer Echtzeit-Ethernet Restbussimulation

Das Testen von Systemen und Funktionen im Entwicklungsprozess ist ein besonders wichtiger Schritt, mit dem zum einen überprüft wird, ob das System richtig entwickelt und zum anderen, ob das richtige System entwickelt wurde. Damit die entwickelten Systeme auf unterschiedlichen Ebenen getestet werden, hat sich das modellbasierte Testen in der Entwicklung von Automotivesystemen etabliert. Auf diese Weise lassen sich systematisch Testfälle anhand von Systemmodellen ableiten, die auf unterschiedlichen Testplattformen ausgeführt werden. Die Restbussimulation gehört zu einer dieser Testplattformen und ist ein wichtiges Werkzeug in der Entwicklung von verteilten Automotivesystemen. Sie erlaubt es frühzeitig in der Entwicklung befindliche Teile des Systems zu testen, in dem das restliche, relevante System simuliert wird. Für verteilte Automotivesysteme, die auf das Fahrzeugnetzwerk der nächsten Generation setzen, muss eine Restbussimulation vorhanden sein, um frühzeitig Fehler in der Implementierung aufdecken zu können.
Da zeitliche Leistungsanforderungen bei Echtzeit-Ethernet eine große Rolle spielen, müssen diese Attribute auch in den Testfällen berücksichtigt werden. Durch die Entwicklung des abstrakten Testfallmodells für zeitliche Leistungsanforderungen können Systeme auf diese Weise systematisch überprüft werden. Des Weiteren kann das abstrakte Testfallmodell auch als Simulationsmodell verwendet werden, wodurch die Konfiguration und Entwicklung einer Restbussimulation erleichtert wird, da kein separates Simulationsmodell entwickelt werden muss.

Die erzielten Ergebnisse der Restbussimulation des Scheinwerfersystems haben gezeigt, dass die entwickelte Methodik im Fahrzeugnetzwerk der nächsten Generation genutzt werden kann, um frühzeitig fehlerhaftes Verhalten aufdecken zu können. Für die Etablierung von Echtzeit-Ethernet im Fahrzeug ist damit ein wichtiger Schritt erreicht worden, da nicht nur die technischen Vorteile einer neuen Technologie betrachtet werden müssen, sondern auch dessen Handhabbarkeit im Entwicklungsprozess.

7.3 Ausblick auf zukünftige Arbeiten

Zukünftige Arbeiten an einer Echtzeit-Ethernet basierten Restbussimulation würden den Bereich reaktives Verhalten innerhalb der Simulation umfassen. In diesem Fall müssen die Daten zur Laufzeit erzeugt werden, sodass komplexere Simulationsmodelle auszuführen sind. Eine Möglichkeit wäre dabei, Matlab/Simulink-Modelle auszuführen, sodass im gleichen Entwicklungsschritt das Testen von kontinuierlichen Aspekten von Automotivesystemen umgesetzt wird.

Des Weiteren ist eine werkzeuggestütze Auswertung der Testfälle wünschenswert, mit der die Soll- und Ist-Werte der aufgezeichneten Daten verglichen werden können. Das XML-basierte Datenmodell der abstrakten Testfälle bietet sich zur Verwendung an, da es bereits durch die Definition der Soll-Werte für diese Art der Auswertung vorbereitet ist. Eine werkzeuggestützte Auswertung würde den Analyseprozess vereinfachen und fehlerhaftes Verhalten könnte effektiver aufgedeckt werden, sodass während des Verifizierungsprozesses Zeit und Kosten eingespart werden.

Eine Verbesserung der Architektur der Restbussimulationsplattform ist geplant. In diesem Fall wird der Mikrocontroller eingespart, wodurch das doppelte Konfigurieren der Plattform entfallen wird. Ebenfalls kann auf zusätzlich Hardware in Form der Dual-Port-Memory-Schnittstellte verzichtet werden, die das Design der Hardware-/Softwareplattform vereinfachen wird. Dabei wird das Prototypenkonzept auf eine Plattform mit nativer Untersützung von Echtzeit-Ethernet portiert. Die Arbeit von (Reidl 2013) könnte hierbei eine geeignete Basis darstellen, da Ethernet Frames mit einem Zeitstempel direkt in der Hardware versehen werden, sodass sich auch die zeitlichen Leistungsanforderungen überprüfen lassen.

A Modelle der Scheinwerfersteuerung

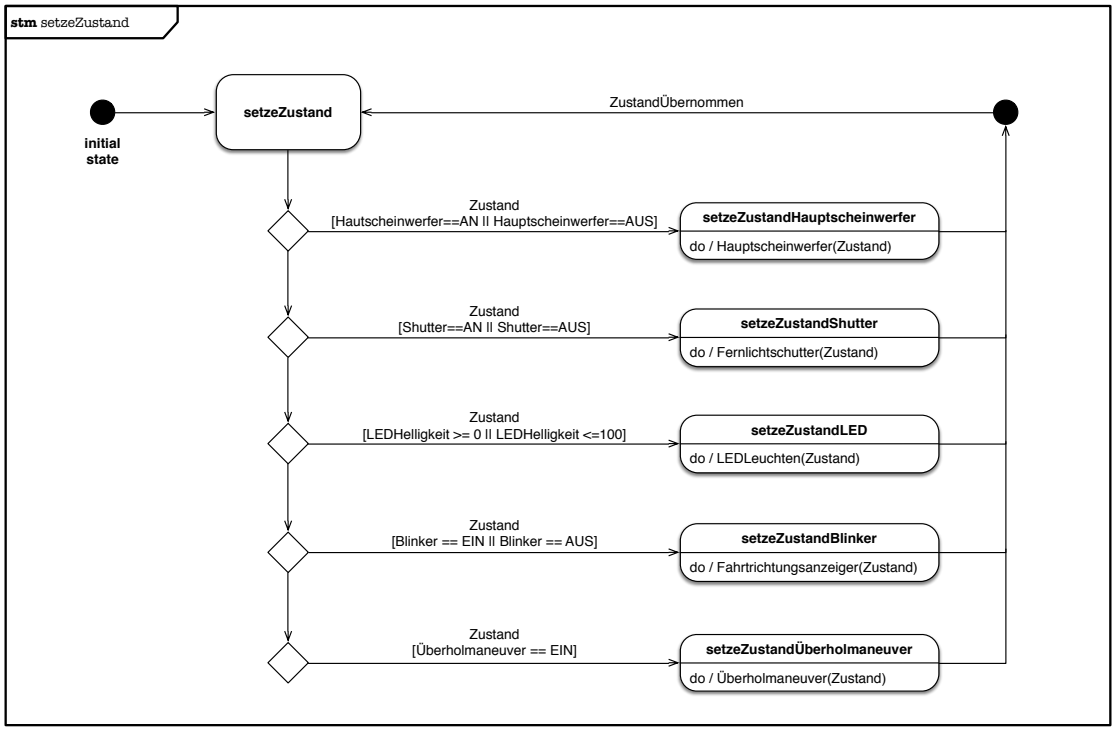


Abbildung A.1: *Zustandsautomat der den eigentlichen Zustand der Scheinwerfer deligiert*

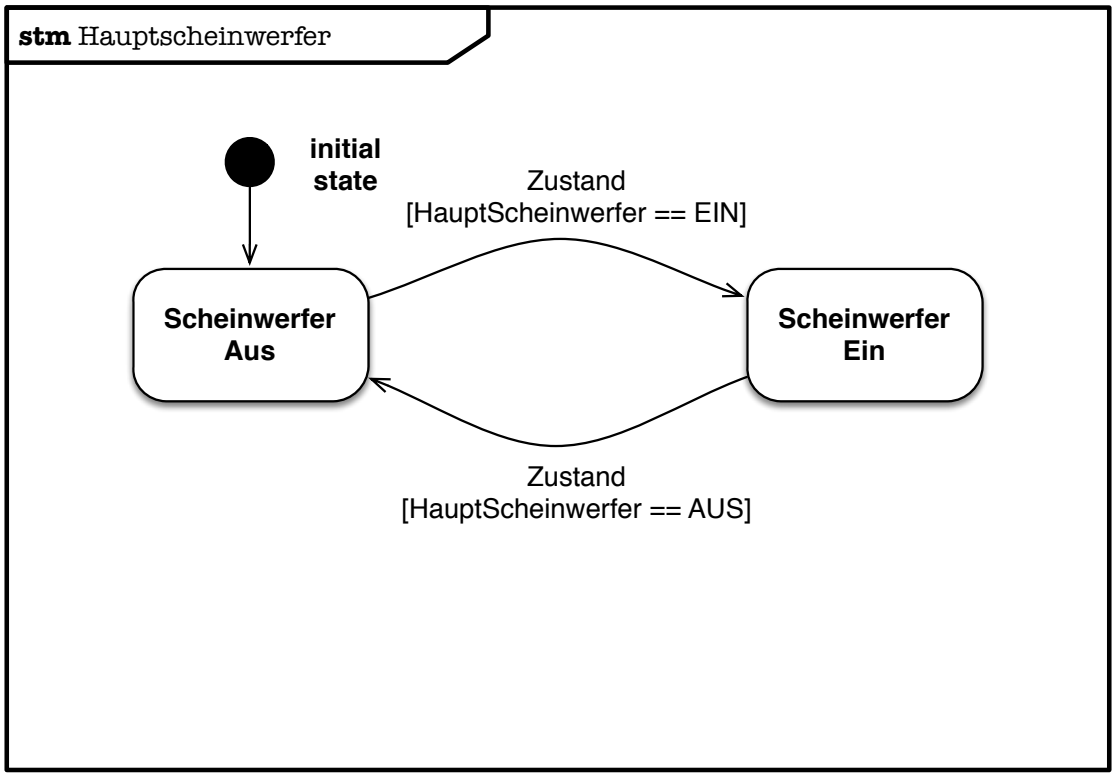


Abbildung A.2: *Zustandsautomat der die Funktion des Hauptscheinwerfers modelliert*

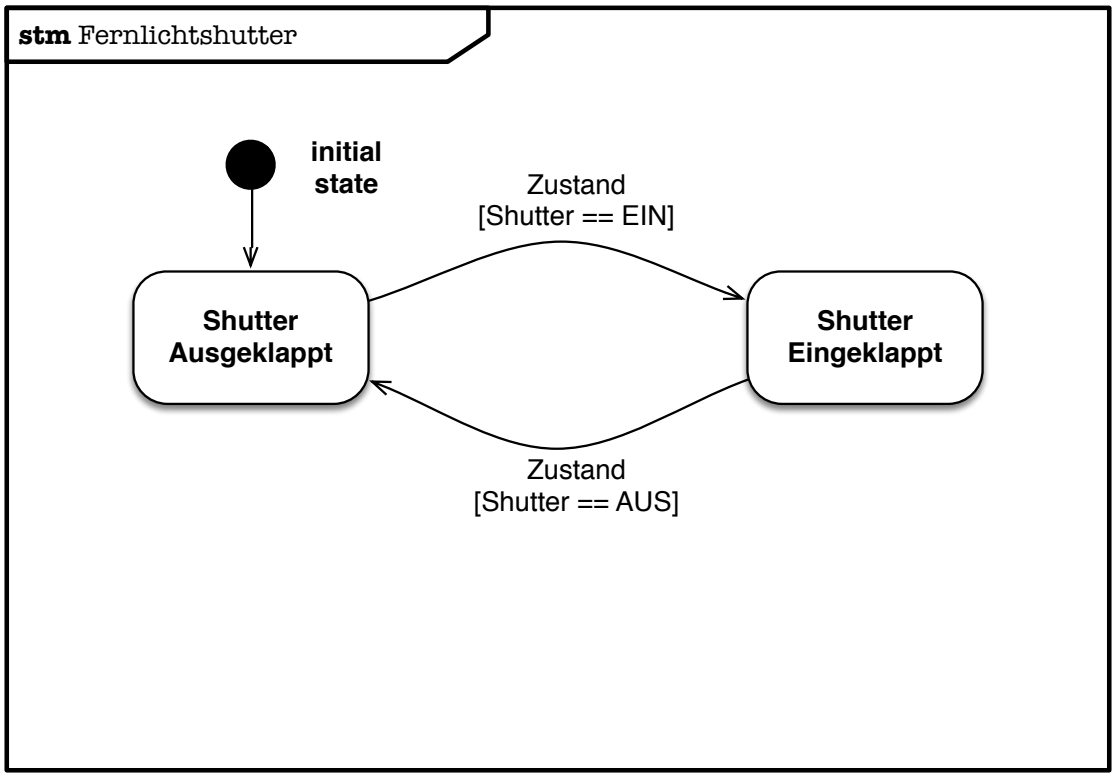


Abbildung A.3: *Zustandsautomat der die Funktion des Fernlichtshutters modelliert*

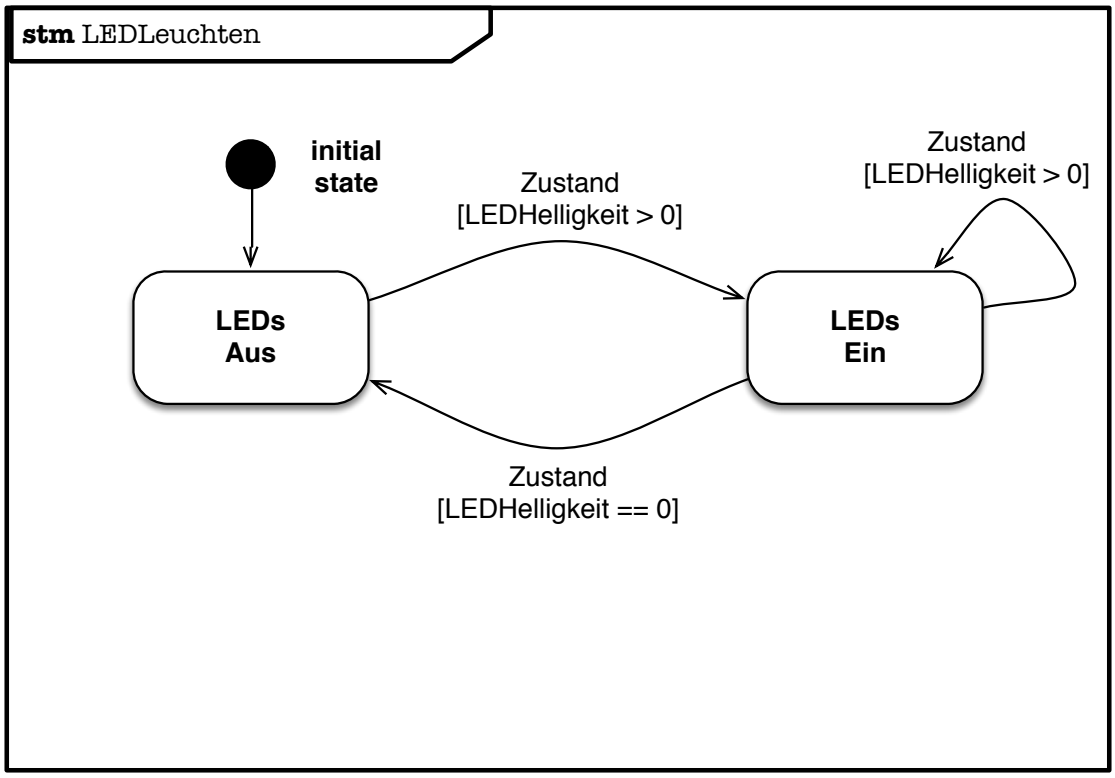


Abbildung A.4: *Zustandsautomat der die Funktion des Hauptscheinwerfers modelliert*

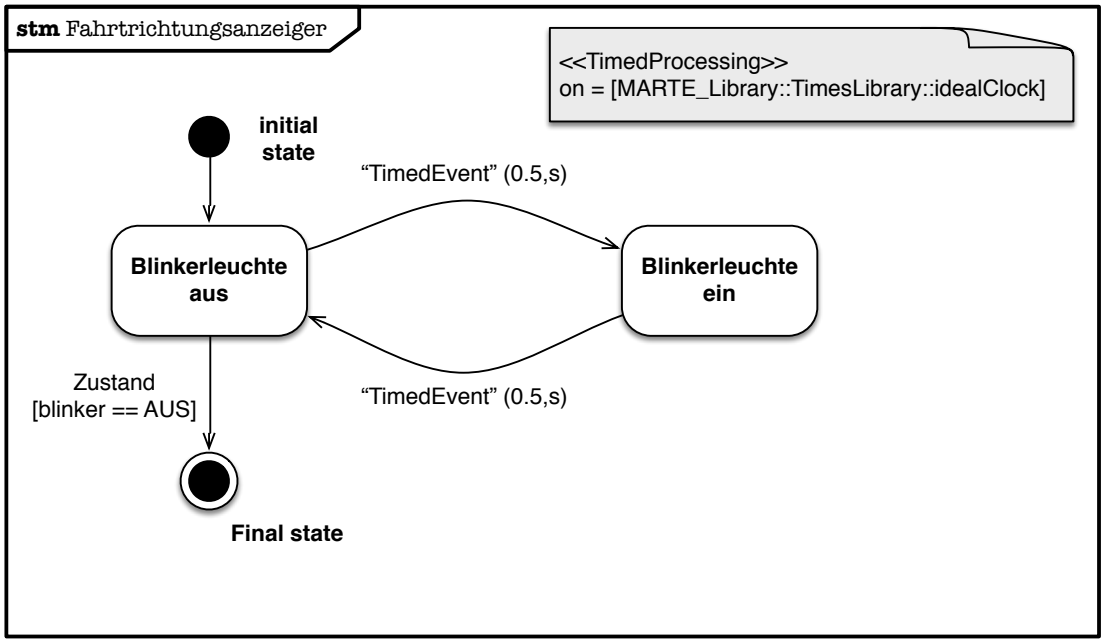


Abbildung A.5: *Zustandsautomat der die Funktion des Fahrtrichtungsanzeigers modelliert*

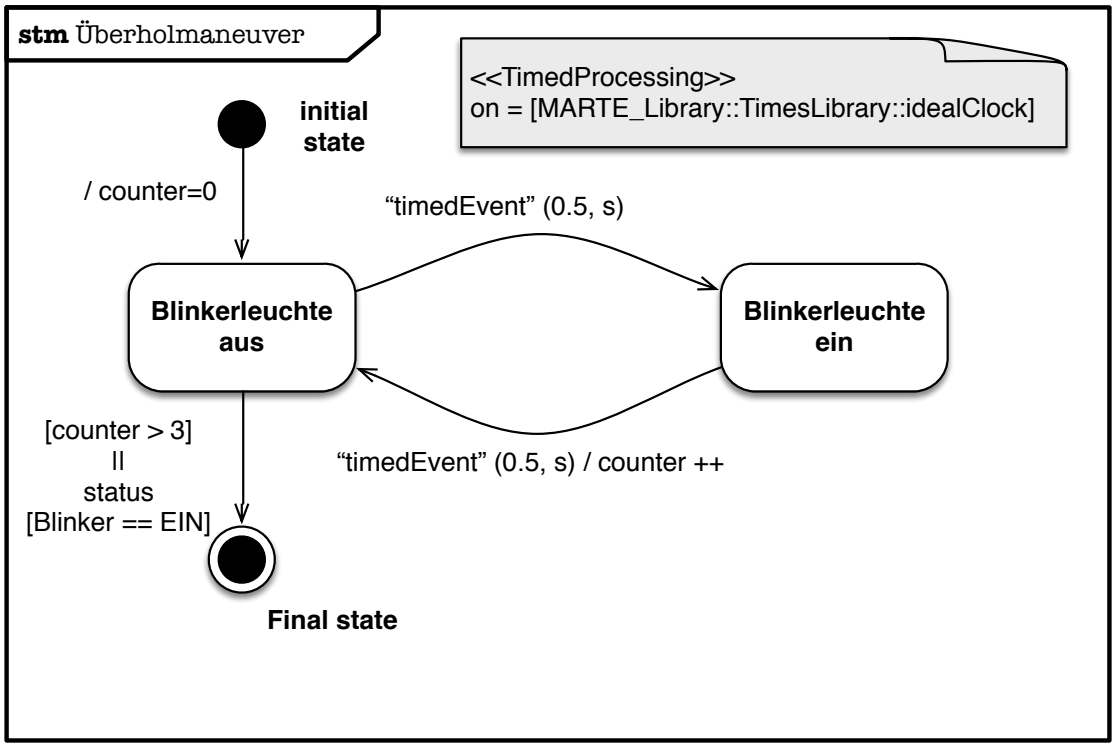


Abbildung A.6: *Zustandsautomat der die Funktion des Überholmaneuvers modelliert*

Literatur

Apvrille / Becoulet 2012 Apvrille, Ludovic ; Becoulet, Alexandre: „Prototyping an Embedded Automotive System from its UML/SysML Models“. In: *Proceedings of Embedded Real-time Software and Systems 2012*. ERTS, 2012-02.

Aeronautical Radio Incorporated 2002 Aeronautical Radio Incorporated: *Aircraft Data Network*. 664. Annapolis, Maryland : ARINC, 2002 – Standard.

Aeronautical Radio Incorporated 2009 Aeronautical Radio Incorporated: *Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network*. ARINC Report 664P7-1. ARINC, 2009 – Standard.

Andrianarison / Piques 2010 Andrianarison, Eric ; Piques, Jean-Denis: „SysML for embedded automotive Systems: a practical approach“. In: *Proceedings of the 2010 Embedded Realtime Software and Systems Conference*. Toulouse, France : ERTS, 2010-05.

ASAM 2013 Association for Standardisation of Automation and Measuring Systems: *Data Model for ECU Network Systems (Field Bus Data Exchange Format)*. 4.1.0. ASAM e.V., 2013-05 – Specification.

Badstübner 2008 Badstübner, Jens: Kollaps im Bordnetz: Schluss mit Can, Lin und Flexray. In: *KFZ-Betrieb* (2008), Nr. 17, S. 68–70.

Bartols 2012 Bartols, Florian: *Cluster Simulation of Real-time Ethernet based Electronic Control Units in Context of Automotive Applications: Extending FIBEX for Real-time Ethernet*. Bericht. 2012-09.

Bartols 2013 Bartols, Florian: *Cluster Simulation of Real-time Ethernet based Electronic Control Units in Context of Automotive Applications: Calculation of Suitable Hardware Setups*. Bericht. 2013-05.

Bauer / Broy / Romberg u. a. 2005 Bauer, Andreas ; Broy, Manfred ; Romberg, Jan u. a.: *AutoMoDe — Notations, Methods, and Tools for Model-Based Development of Automotive Software*. Detroit, MI , 2005-04.

Belli / Hollmann / Padberg 2011 Belli, Fevzi ; Hollmann, Axel ; Padberg, Sascha: Model-Based Integration Testing with Communication Sequence Graphs. In: ZANDER, Justyna (Hrsg.) ; SCHIEFDECKER, Ina (Hrsg.) ; MOSTERMAN, Pieter J. (Hrsg.): *Model-Based Testing for Embedded Systems*. (Computational Analysis, Synthesis and Design of Dynamic Systems Series). Boca Raton, London, New York : CRC Press, 2011-09. Kap. 4, S. 223–244.

Bringmann / Krämer 2008 BRINGMANN, Eckard ; KRÄMER, Andreas: „Model-Based Testing of Automotive Systems“. In: *2008 1st International Conference on Software Testing, Verification and Validation*. 2008-04, S. 485–493. – DOI: 10.1109/ICST.2008.45.

Bossel 2004 BOSSEL, Hartmut: *Systeme Dynamik Simulation: Modellbildung, Analyse und Simulation komplexer Systeme*. Nordersted bei Hamburg : Books on Demand GmbH, 2004. – ISBN: 3-8334-0984-3.

Broy / Krüger / Pretschner u. a. 2007 BROY, Manfred ; KRÜGER, Ingolf h. ; PRETSCHNER, Alexander u. a.: Engineering Automotive Software. In: *Proceedings of the IEEE* 95 (2007-02), Nr. 2, S. 356–373. – ISSN: 0018-9219. – DOI: 10.1109/JPROC.2006.888386.

Charara / Scharbarg / Ermont u. a. 2006 CHARARA, Hussein ; SCHARBARG, Jean-Luc ; ERMONT, Jérôme u. a.: „Methods for bounding end-to-end delays on an AFDX network“. In: *18th Euromicro Conference on Real-Time Systems*. 2006. – DOI: 10.1109/ECRTS.2006.15.

Combs COMBS, Gerald: *Wireshark*. URL: `http://www.wireshark.org` Abruf: 2014-03-26.

CoRE-Arbeitsgruppe (a) CORE-ARBEITSGRUPPE: *Communication over Real-time Ethernet*. Hamburg . URL: `http://core.informatik.haw-hamburg.de` Abruf: 2014-03-24.

CoRE-Arbeitsgruppe (b) CORE-ARBEITSGRUPPE: *CoRE for INET*. URL: `http://core.informatik.haw-hamburg.de/en/projects/simulation/core4inet.html` Abruf: 2014-03-24.

Crockford 2006 CROCKFORD, D.: *The application/json Media Type for JavaScript Object Notation (JSON)*. 4627. IETF, 2006-07 – RFC.

Cuenot / Chen / Gerard u. a. 2007 CUENOT, P. ; CHEN, DeJiu ; GERARD, S. u. a.: „Managing Complexity of Automotive Electronics Using the EAST-ADL“. In: *12th IEEE International Conference on Engineering Complex Computer Systems 2007*. Piscataway, New Jersey : IEEE Press, 2007-07, S. 353–358. – DOI: 10.1109/ICECCS.2007.28.

Dematthieu / Thomas / Andre u. a. 2008 DEMATHIEU, Sebastian ; THOMAS, Frederic ; ANDRE, Charles u. a.: „First Experiments Using the UML Profile for MARTE“. In: *11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC) 2008*. Piscataway, New Jersey : IEEE Press, 2008-05, S. 50–57. – DOI: 10.1109/ISORC.2008.36.

Eberspächer 2014 EBERSPÄCHER: *Restbussimulation für FlexRay und CAN - Eberspächer Elektronics*. Eberspächer. 2014. URL: http://www.eberspaecher-electronics.com/produkte/flexconfig-rbsgateway/restbussimulation.html Abruf: 2014-03-24.

Elmenreich / Ipp 2003 ELMENREICH, Wilfried ; IPP, Richard: „Introduction to TTP/C and TTP/A“. In: *Proceedings of the Workshop on Time-Triggered and Real-Time Communication Systems*. eingeladen; Vortrag: Workshop on Time-Triggered and Real-Time Communication Systems, Manno, Switzerland; 2003-12-02. 2003-12, S. 1–9.

Elektroniknet.de 2008 ELEKTRONIKNET.DE: *Markt für Autoelektronik weiterhin mit ungebrochenem Wachstum*. 2008-02. URL: http://www.elektroniknet.de/automotive/technik-know-how/prozesse-standards-und-qualitaet/article/741/0/Markt_fuer_Autoelektronik_weiterhin_mit_ungebrochenem_Wachstum/ Abruf: 2014-03-24.

Frank / van Laak 2003 Frank, Ulrich ; van Laak, Bodo L.: Anforderungen an Sprachen zur Modellierung von Geschäftsprozessen. In: PROF. DR. ULRICH FRANK (Hrsg.) ; PROF. DR. J. FELIX HAMPE (Hrsg.): *Arbeitsberichte des Instituts für Wirtschaftsinformatik* (2003-01), Nr. 34.

Galla 1999 GALLA, Thomas M.: „Cluster Simulation in Time-Triggered Real-Time Systems“. Diss. Wien : TU Wien, 1999-12.

Glinz 2007 GLINZ, Martin: „On Non-Functional Requirements“. In: *15th IEEE International Requirements Engineering Conference, 2007. RE '07.* 2007, S. 21–26. – DOI: 10.1109/RE.2007.45.

Gleixner / Molnar GLEIXNER, Thomas ; MOLNAR, Ingo: *hrtimers - subsystem for hight-resolution kernel timers*. URL: https://www.kernel.org/doc/Documentation/timers/hrtimers.txt Abruf: 2014-02-04.

Galla / Pallierer 1999 GALLA, Thomas M. ; PALLIERER, Roman: „Cluster simulation-support for distributed development of hard real-time systems using TDMA-based communication“. In: *Proceedings of the 11th Euromicro Conference on Real-Time Systems, 1999.* 1999-06, S. 150–157. – DOI: 10.1109/EMRTS.1999.777461.

Grzemba 2007 GRZEMBA, Andreas (Hrsg.): *MOST: Das Multimedia-Bussystem für den Einsatz im Automobil*. (Elektronik- und Elektrotechnik-Bibliothek). Poing, Deutschland : Franzis, 2007. – ISBN: 978-3-7723-4149-6.

Gugenhan / Schmidt 2011 GUGENHAN, Steffen ; SCHMIDT, Olaf: *Intelligente Tools für RBS, Gateway und Signalmanipulation - Langversion*. Eberspächer, 2011-06 – Techn. Ber. URL: http://www.eberspaecher-electronics.com/fileadmin/data/microsites/electronics/pdf/EBEL_Fachartikel_RBS_GW_SM_D1V0-F.pdf Abruf: 2014-03-24.

Grzemba / Wense 2005 GRZEMBA, Andreas (Hrsg.) ; WENSE, Hans Christian von der (Hrsg.): *LIN-Bus: Systeme, Protokolle, Tests von LIN-Systemen, Tools, Hardware, Applikationen*. (Elektronik). Poing, Deutschland : Franzis, 2005. – ISBN: 3-7723-4009-1.

Helal 2008 HELAL, Magdy: „A Hybrid System Dynamics-Discrete Event Simulation Approach to Simulating the Manufacturing Enterprise“. Diss. Orlando : University of Central Florida, 2008.

Henzinger 2000 Henzinger, Thomas A.: The Theory of Hybrid Automata. In: INAN, M. Kemal (Hrsg.) ; KURSHAN, Robert P. (Hrsg.): *Verification of Digital and Hybrid Systems*. Bd. 170. (NATO ASI Series). Berlin Heidelberg : Springer, 2000-06. – ISBN: 978-3-642-64052-0, S. 265–292.

Häfen / Fries 2008 Patentanmeldung DE 102007002312: *Restbussimulation*. Anmelder: Bayerische Motoren Werke Aktiengesellschaft. Erfinder: HÄFEN, Dr. Robert ; FRIES, Georg. München. 2008-09-04. URL: http://www.freepatentsonline.com/DE102007002312A1.html.

Honeywell International HONEYWELL INTERNATIONAL: URL: http://www.honeywell.com.

IABG 2012 IABG: *Das V-Modell XT Version 1.4*. IABG. 2012-06. URL: http://v-modell.iabg.de/dmdocuments/V-Modell-XT-Gesamt-Deutsch-V1.3.pdf Abruf: 2014-03-24.

IEEE 802.1 AVB Task Group IEEE 802.1 AVB TASK GROUP: *IEEE 802.1 Audio/Video Bridging (AVB)*. URL: http://www.ieee802.org/1/pages/avbridges.html.

IEEE 802.1 TSN Task Group (a) IEEE 802.1 TSN TASK GROUP: *IEEE 802.1Q - Virtual LANs*. URL: http://www.ieee802.org/1/pages/802.1Q.html.

IEEE 802.1 TSN Task Group (b) IEEE 802.1 TSN TASK GROUP: *IEEE 802.1Qbv - Enhancements for Scheduled Traffic*. URL: http://www.ieee802.org/1/pages/802.1bv.html.

International Organization for Standardization 2003 INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *Road vehicles – Controller area network (CAN)*. 11898. Genf : ISO, 2003 – ISO.

Jacobson / Leres / McCanne Jacobson, Van ; Leres, Craig ; McCanne, Steven: *pcap - Packet Capture library*. URL: `http://www.tcpdump.org/` Abruf: 2014-02-05.

Johnson / Paredis / Burkhart u. a. 2007 Johnson, Thomas A. ; Paredis, Christian J.J. ; Burkhart, Roger u. a.: „Modeling Continous System Dynamic in SysML“. In: *Proceedings of the International Mechanical Engineering Congress and Exposition 2007*. 2007-11.

Karfich / Bartols / Steinbach u. a. 2013 Karfich, Oleg ; Bartols, Florian ; Steinbach, Till u. a.: „A Hardware/Software Platform for Real-time Ethernet Cluster Simulation in OMNeT++“. In: *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*. Cannes, France : ACM-DL, 2013-03, S. 334–337. – ISBN: 978-1-4503-2464-9.

Karfich 2013 Karfich, Oleg: „Kopplung einer OMNeT++ basierten Echtzeitsimulation an Real-Time-Ethernet Netzwerke“. Bachelorthesis. Bachelorthesis. Hamburg : HAW Hamburg, 2013-04.

Kaempchen / Fuerstenberg / Dietmayer 2004 Kaempchen, Nico ; Fuerstenberg, Kay Ch. ; Dietmayer, Klaus C.J.: „Ein Sensorfusionssystem für automotive Sicherheits- und Komfortapplikationen“. In: *1. Tagung Aktive Sicherheit durch Fahrerassistenzsysteme*. Techniche Universität München, 2004.

Kopetz / Ademaj / Grillinger u. a. 2005 Kopetz, Hermann ; Ademaj, Astrit ; Grillinger, Petr u. a.: „The time-triggered Ethernet (TTE) design“. In: *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005.* 2005-05, S. 22–33. – DOI: 10.1109/ISORC.2005.56.

Koch / Tsirkin 2009 Koch, Hans-Jürgen ; Tsirkin, Michael S.: *The Userspace I/O HOWTO*. 2009. URL: `https://www.kernel.org/doc/htmldocs/uio-howto/index.html` Abruf: 2014-02-04.

Kuther 2011 Kuther, Thomas: *Echtzeit-Ethernet als Automotive-Netzwerk*. Elektronik Praxis. 2011-11. URL: `http://www.elektronikpraxis.vogel.de/themen/hardwareentwicklung/datenkommunikationsics/articles/335088/` Abruf: 2014-03-24.

Kamke / Walcher 1994 Kamke, Detlef ; Walcher, Wilhelm: *Physik für Mediziner*. 2. Aufl. Wiesbaden : Vieweg und Teubner, 1994. – ISBN: 3-519-13048-3.

Lamberg / Beine / Eschmann u. a. 2004 Lamberg, Klaus ; Beine, Michael ; Eschmann, Mario u. a.: „Model-based testing of embedded automotive software using MTest“. In: *2004 SAE World Congress*. Detroit, U.S. : SAE International, 2004-03.

Lim / Herrscher / Waltl u. a. 2012 LIM, Hyung-Taek ; HERRSCHER, Daniel ; WALTL, Martin Johannes u. a.: „Performance analysis of the IEEE 802.1 ethernet audio/video bridging standard“. In: *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*. Desenzano del Garda, Italy : ACM-DL, 2012-03, S. 27–36. – ISBN: 978-1-4503-1510-4.

Lipfert 2008 LIPFERT, Jan: *Technical Data Reference Guide - netX500/100*. Hilscher GmbH. 2008-12. URL: http://www.hilscher.com.

Lunze 2013 LUNZE, Jan: *Regelungstechnik 1: Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen*. 9. ueberarbeitete Auflage. Berlin, Heidelberg : Springer-Verlag, 2013. – ISBN: 978-3-642-29533-1.

MathWorks 2014a MATHWORKS: *MATLAB*. 2014. URL: http://www.mathworks.de/products/matlab/ Abruf: 2014-03-24.

MathWorks 2014b MATHWORKS: *Simulink*. 2014. URL: http://www.mathworks.de/products/simulink/ Abruf: 2014-03-24.

MathWorks 2014c MATHWORKS: *StateFlow*. 2014. URL: http://www.mathworks.de/products/stateflow/ Abruf: 2014-03-24.

Meier 2008 MEIER, Erich: *V-Modelle in Automotive-Projekten*. Automobil Elektronik. 2008-02. URL: http://www.methodpark.de/ressourcen/download/v-modelle-in-automotive-projekten/open-download/ Abruf: 2011-02-09.

Marscholik / Subke 2007 MARSCHOLIK, Christoph ; SUBKE, Peter: *Datenkommunikation im Automobil: Grundlagen, Bussysteme, Protokolle und Anwendungen*. (Hüthig Praxis). Heidelberg : Hüthig, 2007. – ISBN: 3-7785-2969-2.

Müller / Steinbach / Korf u. a. 2011 MÜLLER, Kai ; STEINBACH, Till ; KORF, Franz u. a.: „A Real-time Ethernet Prototype Platform for Automotive Applications“. In: *2011 IEEE International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. Berlin : IEEE Press, 2011-09, S. 221–225. – ISBN: 978-1-4577-0233-4.

Müller 2011 MÜLLER, Kai: „Time-Triggered Ethernet für eingebettete Systeme: Design, Umsetzung und Validierung einer echtzeitfähigen Netzwerkstack-Architektur“. Bachelorthesis. Hamburg , 2011-08.

Navet / Song / Simonot-Lion u. a. 2005 NAVET, Nicolas ; SONG, Yeqiong ; SIMONOT-LION, Françoise u. a.: Trends in Automotive Communication Systems. In: *Proceedings of the IEEE* 93 (2005-06), Nr. 6, S. 1204–1223. – ISSN: 0018-9219. – DOI: 10.1109/JPROC.2005.849725.

Neuffer / Böke 2010 NEUFFER, Jochen ; BÖKE, Dr. Carsten: Steuergeräte modellbasiert entwickeln. In: *Elektronik automotive* (2010-03), Nr. 3, S. 26–29.

OMG (a) Object Management Group. URL: http://www.omg.org/ Abruf: 2014-03-24.

OMG (b) Object Management Group: *MARTE - Modeling And Analysis Of Real-Time Embedded Systems*. URL: http://www.omg.org/spec/MARTE/ Abruf: 2014-03-24.

OMG (c) Object Management Group: *SysML - Systems Modeling Language*. URL: http://www.omg.org/spec/SysML/ Abruf: 2014-03-24.

OMG (d) Object Management Group: *UML - Unified Modeling Language*. URL: http://www.omg.org/spec/UML/ Abruf: 2014-03-24.

OMG 2010 Object Management Group: *OMG Systems Modeling Language (OMG SysML) - Hybrid SUV Non-Normitative Example*. Object Management Group, 2010-11 – Techn. Ber. URL: http://www.omg.org/ocsmp/HSUV.pdf Abruf: 2014-03-04.

OMG 2011 *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*. 1.1. Object Management Group. Needham, MA, USA , 2011-06.

OMNeT++ Community OMNeT++ Community: *OMNeT++ 4.2.2*. URL: http://www.omnetpp.org.

Piques / Andrianarison 2012 Piques, Jean-Denis ; Andrianarison, Eric: „SysML for embedded automotive Systems: lessons learned“. In: *Proceedings of the 2012 Embedded Realtime Software and Systems Conference*. Toulouse, France : ERTS, 2012-02.

PROFIBUS & PROFINET International PROFIBUS & PROFINET International: *Profinet*. Karlsruhe . URL: http://www.profibus.com/technology/profinet Abruf: 2011-02-07.

Quadri / Sadovykh / Indrusiak 2012 Quadri, Imram Rafiq ; Sadovykh, Andrey ; Indrusiak, Leandro Soares: „MADES: A SysML/MARTE High Level Methodology for Real Time and Embedded Systems“. In: *Proceedings of the 2012 Embedded Realtime Software and Systems Conference*. Toulouse, France : ERTS, 2012-02.

Rausch 2008 Rausch, Mathias: *FlexRay: Grundlagen, Funktionsweise, Anwendung*. München : Carl Hanser Verlag, 2008. – ISBN: 978-3-446-41249-1.

Riegraf / Behh / Kraus 2007 Riegraf, Thomas ; Behh, Siegfried ; Kraus, Stefan: Effizientes Testen in der Automobilelektronik - Von der Simulation bis zur Diagnose. In: *ATZ - Automobiltechnische Zeitschrift* 109 (2007-08), S. 648–655.

Reidl 2013 Reidl, Johannes: „Optimierung der Zeitpräzision eines Linux Ethernet Treibers auf Basis einer PTP Uhr“. Bachelorthesis. Hamburg : HAW Hamburg, 2013-11.

SAE AS-2D Committee 2011 Society of Automotive Engineers - AS-2D Time Triggered Systems and Architecture Committee: *Time-Triggered Ethernet AS6802*. SAE Aerospace. 2011-11. URL: http://standards.sae.org/as6802/.

Schwarz 2002 Schwarz, Peter: „Modellierung und Simulation heterogener technischer Systeme“. In: *66. Physikertagung*. Leipzig, Deutschland , 2002-03. URL: http://publications.eas.iis.fraunhofer.de/papers/2002/016/.

Schlager 2008 Schlager, Martin: *Hardware-in-the-Loop Simulation*. Vdm Verlag Dr. Müller, 2008-04. – ISBN: 978-3-8364-6216-7.

Smithgall / Hall / Varadarajan 2013 Patentanmeldung US 20130058217: *Time Triggered Ethernet System Testing and Method*. Anmelder: Honeywell International Inc. Erfinder: Smithgall, Willian Todd ; Hall, Brendan ; Varadarajan, Srivatsan. Morristown, NJ, USA. 2013-03-07. URL: http://www.freepatentsonline.com/y2013/0058217.html.

Steinbach / Korf / Schmidt 2011 Steinbach, Till ; Korf, Franz ; Schmidt, Thomas C.: „Simulationsbasierte Evaluierung von Metriken in Echtzeit-Ethernet basierten Fahrzeugnetzen“. In: *6ter GI/ITG-Workshop Leistungs-, Zuverlaessigkeits- und Verlaesslichkeitsbewertung von Kommunikationsnetzen und verteilten Systeme (MMBnet 2011)*. Hrsg. von Bernd E. Wolfinger ; Klaus-D. Heidtmann. Hamburg : Universität Hamburg, 2011-08, S. 9–20.

Skruch / Panek / Kowalczyk 2011 Skruch, Pawel ; Panek, Miroslav ; Kowalczyk, Bogdan: Model-Based Testing in Embedded Automotive Systems. In: Zander, Justyna (Hrsg.) ; Schiefdecker, Ina (Hrsg.) ; Mosterman, Pieter J. (Hrsg.): *Model-Based Testing for Embedded Systems*. (Computational Analysis, Synthesis and Design of Dynamic Systems Series). Boca Raton, London, New York : CRC Press, 2011-09. Kap. 19, S. 545–578.

Steinbach / Lim / Korf u. a. 2012 Steinbach, Till ; Lim, Hyung-Taek ; Korf, Franz u. a.: „Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802)“. In: *2012 IEEE Vehicular Technology Conference (VTC Fall)*. Piscataway, New Jersey : IEEE Press, 2012-09.

Steiner 2008 Steiner, Wilfried: *TTEthernet Specification*. TTTech Computertechnik AG. 2008-11. URL: http://www.tttech.com.

Steinbach 2011 Steinbach, Till: „Echtzeit-Ethernet für Anwendungen im Automobil: Metriken und deren simulationsbasierte Evaluierung am Beispiel von TTEthernet“. Masterthesis. Hamburg : Hochschule für Angewandte Wissenschaften Hamburg, 2011-02.

StVZO 2013 *Straßenverkehrs-Zulassungs-Ordnung*. Homepage: Moravia Druck und Verlag. 2013. URL: http://www.stvzo.de Abruf: 2014-01-09.

Schnell / Wiedemann 2008 SCHNELL, Gerhard (Hrsg.) ; WIEDEMANN, Bernhard (Hrsg.): *Bussysteme in der Automatisierungs- und Prozesstechnik - 7. Auflage*. Vieweg + Teubner, 2008. – ISBN: 978-3-8348-0425-9.

Swager 2008 SWAGER, Mark: *Using Matlab with CANoe*. 1.0. Vector Informatik GmbH. 2008-04.

Schäuffele / Zurawka 2013 SCHÄUFFELE, Jörg ; ZURAWKA, Thomas: *Automotive Software Engineering*. Wiesbaden : Vieweg und Teubner, 2013. – ISBN: 978-3-8348-2469-1. – DOI: 10.1007/978-3-8348-2469-1.

Ts'o / Hart / Kacur 2010 TS'O, Theodore ; HART, Darren ; KACUR, John: *RT-Kernel Wiki*. 2010. URL: `https://rt.wiki.kernel.org/index.php/Main_Page` Abruf: 2014-02-04.

TTTech Computertechnik AG TTTECH COMPUTERTECHNIK AG. Wien . URL: `http://www.tttech.com` Abruf: 2014-03-24.

Utter / Pretschner / Legeard 2006 UTTER, Mark ; PRETSCHNER, Alexander ; LEGEARD, Bruno: *A Taxonomy of Modelbased Testing*. Working Paper: 04/2006. Hamilton, New Zealand : University of Waikato, 2006-04 – Techn. Ber.

Vector Informatik VECTOR INFORMATIK: *DBC Communcation Database for CAN*. Vektor Informatik. URL: `http://www.vector.com/vi_candb_de.html` Abruf: 2014-01-30.

Vector Informatik 2013 VECTOR INFORMATIK: *CANoe 8.1 - Steuergeräte-Entwicklung und -Test*. Vektor Informatik. 2013. URL: `http://www.vector.com/vi_canoe_de.html` Abruf: 2014-03-24.

W3C XML Working Group 2008 W3C XML WORKING GROUP: *The XML 1.0 Standard (5th Edition)*. W3C Recommendation. 2008. URL: `http://www.network-theory.co.uk/w3c/xml/`.

Wolfhard / Obermöller 2011 WOLFHARD, Lawrence (Hrsg.) ; OBERMÖLLER, Nils (Hrsg.): *CAN - Controller Area Network: Grundlagen, Design, Anwendungen, Testtechnik*. 5. neu bearbeitete Auflage. Berlin, Offenbach : VDE Verlag, 2011.

Weißleder / Schlingloff 2011 Weißleder, Stephan ; Schlingloff, Holger: Automatic Model-Based Test Generation from UML State Machines. In: ZANDER, Justyna (Hrsg.) ; SCHIEFDECKER, Ina (Hrsg.) ; MOSTERMAN, Pieter J. (Hrsg.): *Model-Based Testing for Embedded Systems*. (Computational Analysis, Synthesis and Design of Dynamic Systems Series). Boca Raton, London, New York : CRC Press, 2011-09. Kap. 4, S. 77–110.

Yaghmour / Masters / Ben-Yoseff u. a. 2008 YAGHMOUR, Karim ; MASTERS, Jon ; BEN-YOSEFF, Gilad u. a.: *Building Embedded Linux Systems - 2nd Edition*. O'Reilly Media, Inc., 2008-08. – ISBN: 978-0-596-52968-0.

Yu / Talpin / Besnard u. a. 2010 Yu, Huafeng ; Talpin, J. ; Besnard, L. u. a.: „Polychronous Analysis of Timing Constraints in UML MARTE“. In: *2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*. 2010-05, S. 145–151. – doi: 10.1109/ISORCW.2010.10.

Zimmermann / Schmidgall 2011 Zimmermann, Werner ; Schmidgall, Ralf: *Bussysteme in der Fahrzeugtechnik - 4. aktualisierte Auflage*. Vieweg + Teubner, 2011. – isbn: 978-3-8348-0907-0.

Zander / Schiefdecker / Mosterman 2011 Zander, Justyna ; Schiefdecker, Ina ; Mosterman, Pieter J.: A Taxonomy of Modelbased Testing for Embedded Systems from Multiple Industry Domains. In: Zander, Justyna (Hrsg.) ; Schiefdecker, Ina (Hrsg.) ; Mosterman, Pieter J. (Hrsg.): *Model-Based Testing for Embedded Systems*. (Computational Analysis, Synthesis and Design of Dynamic Systems Series). Boca Raton, London, New York : CRC Press, 2011-09. Kap. 1, S. 3–22.

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 26. März 2014 Florian Bartols