Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Entwicklung und Umsetzung eines Testkonzepts für die Vernetzung von Steuergeräten im Antriebsstrang

Jakob Benz

Studiengang:	Softwaretechnik B.Sc.
Prüfer/in:	Prof. Dr. Stefan Wagner
Betreuer/in:	Dr. Asim Abdulkhaleq
Beginn am:	2017/04/18
Beendet am:	2017/10/05
CR-Nummer:	B.1.3, B.8, C.2.5, C.3

Abstract

In the last decades the networking of electronic control units in cars underwent extensive changes. Going from basic point-to-point connections to using bus systems like e.g. CAN, not only the amount of cables needed has been reduced drastically, but also there now is a very flexible environment concerning the integration of electronic control units. However the downside of this trend are the incresing requirements towards the networking software and the higher complexity of the networking software's use cases. Consequently the costs for the network test increases likewise.
This work presents a new concept for the network test, which aims to move preferably the whole network test to the test environment at the desk, since the test in an actual car or using the HIL is very expensive and time-consuming.
In order to perform the network test completely automated, the tool *Network-Test-Generator* has been desgined and implemented. It generates the test case description as .xml-file based on one or more network descriptions and the calibration of the electronic control unit currently tested. This test case description then is used by the already existing tool chain for the network test in the test environment at the desk.

Zusammenfassung

Die Vernetzung von Steuergeräten in Fahrzeugen hat sich in den letzten Jahrzehnten stark verändert. Wurden früher noch einfache Punkt-zu-Punkt-Verbindungen eingesetzt, greift man heute fast ausschließlich zu Bussystemen wie z.B. CAN. Neben der Reduktion des benötigten Kabelbaums wird durch den Einsatz von Bussystemen einen äußerst flexible Integrationsumgebung für neue Steuergeräte im Fahrzeug geschaffen. Allerdings steigen mit dieser Entwicklung auch die Anforderungen an die Vernetzungssoftware und die Komplexität derer Aufgaben. Dies wiederum verursacht einen sehr hohen Aufwand beim Test dieser Vernetzungssoftware.
In dieser Arbeit wird nun ein Testkonzept entwickelt, das die Umfänge der Vernetzungstests von Steuergeräten im Antriebsstrang möglichst vollständig an die Testumgebung am Arbeitsplatz verlagern soll, da die Tests im Testfahrzeug bzw. am HIL im Vergleich zum Tischaufbau äußerst aufwendig sind - im Hinblick auf sowohl die entstehenden Kosten als auch den zeitlichen Aufwand.
Für die automatisierte Durchführung der Vernetzungstests am Tischaufbau wird das Tool *Network-Test-Generator* entwickelt, welches die Testfallbeschreibung im Form eines Testvektors als .xml-Datei auf Basis einer oder mehrerer Netzwerkbeschreibungen sowie der Steuergeräte-Parametrisierung generiert. Dieser Testvektor kann dann mit der bereits vorhandenen Toolkette für den automatisierten Vernetzungstest von Steuergeräten am Tischaufbau verwendet werden.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AUTOSAR	Automotive Open System Architecture
CAN	Controller Area Network
CCP	CAN Calibration Protocol
CPC	Central Powertrain Controller
CRC	Cyclic Redundancy Check
ECM	Engine Control Module
ECU	Electronic Control Unit (Steuergerät)
ECUC	ECU Configuration Description
ESP	Elektronisches Stabilitätsprogramm
EVA-Prinzip	Eingabe-Verarbeitung-Ausgabe-Prinzip
LDF	LIN Description File
LIN	Local Interconnect Network
OBD	On-Board-Diagnose
SCI	Serial Communication Interface
XCP	Universal Measurement and Calibration Protocol

Begriffslexikon

.arxml-Datei	XML-Datei in einem von AUTOSAR definiertem Schema; enthält Vorgaben für die Steuergeräte-Konfiguration und die Netzwerkbeschreibung
Black-Box	Ein System, das nur von außen und Vernachlässigung der internen Struktur betrachtet wird / werden kann.
Botschaft	Eine Übertragungseinheit gemäß eines Transportprotokolls.
E/E-Fahrzeugarchitektur	Elektrisch/Elektronische-Fahrzeugarchitektur.
Ethernet	Kabelbasierte Vernetzungstechnik, die sowohl Software- als auch Hardwarestandards spezifiert.
Externes Signal	Ein Signal auf dem Bus.
Glass-Box	Ein System, das unter Berücksichtigung der internen Struktur betrachtet wird / werden kann.
HIL	Hardware in the Loop. Verfahren, bei welchem ein eingebettetes System (z.B. ein Steuergerät) durch die modellbasierte Simulation seiner Umgebung (z.B. restliches Fahrzeug) getestet wird.
Internes Signal	Ein steuergeräteinternes Signal.
OSI-Modell	*Open Systems Interconnection Model*. Referenzarchitektur für ich Schichten angeordnete Netzwerkprotokolle.
Protokollstapel	Architekturmuster, das mehrere Kommunikationsprotokolle hierarchisch in Schichten anordnet.
SNA-Wert	„Signal-not-available“-Wert. Wird verwednet, um zu signalisieren, dass ein Signal nicht verfügbar ist.

Kapitel 1

Einleitung

1.1. Motivation

Die intensive Elektronifizierung der letzten Jahrzehnte hat sich im Automobil besonders stark niedergeschlagen. Seit Mitte der 90er-Jahre werden mechanische Regelungssysteme sukzessive durch elektronische Steuerungssysteme abgelöst. Für das Automobil bedeutet dies eine steigende Anzahl von Steuergeräten. Dieser Trend wird durch steigende Kundenwünsche und den Innovationsdruck, welcher z.B. durch immer schärfere Abgabnormen erzeugt wird, zusätzlich verstärkt.
Wurden Steuergeräte anfangs noch mit einfachen Punk-zu-Punkt-Verbindungen und größtenteils analogen Signalen vernetzt, kommen heutzutage Bussysteme zum Einsatz, welche u.a. mehr Flexibilität und eine hohe Kostenersparnis durch z.B. die Reduzierung der notwendigen Kabelbäume mit sich bringen.
Wie in der Softwareentwicklung spielt auch in der Steuergeräte-Entwicklung der Test des entwickelten Produkts eine große Rolle und ist ebenfalls mit einem hohen Aufwand verbunden. Einen vergleichsweise sehr großen Anteil nimmt hierbei der Vernetzungstest von Steuergeräten ein. Die Vielfalt an Steuergeräten und eingesetzten Bussystemen im Automobil wird - gerade im Hinblick auf die sich in Richtung nachhaltiger Antriebskonzepte ändernden Anforderungen - in den nächsten Jahren im Umfang steigen, weshalb natürlich auch Auswirkungen auf den Vernetzungstest von Steuergeräten zu erwarten sind.

1.2. Problembeschreibung

Der Vernetzungstest im Antriebsstrang umfasst im Hinblick auf zu testende Steuergeräte nur das ECM (Engine Control Module) und das CPC (Central Powertrain Controller), welches den Antriebsstrang mit dem restlichen Fahrzeug verbindet. Da jedoch mehrere Versionen parallel in Entwicklung sind und mehrmals pro Jahr neue Softwarestände entwickelt werden, nimmt der Vernetzungstest enorme Ausmaße an. Dieser durch die aktuell manuell durchgeführten Tests verursachte Aufwand wird irgendwann nicht mehr zu bewältigen und wirtschaftlich zu vertreten sein.
Die Vernetzungstests werden dabei entweder direkt in einem mit Messtechnik ausgestatteten Testfahrzeug, am HIL (Hardware in the Loop) oder an einem Tischaufbau durchgeführt. Der zeitliche und finanzielle Aufwand sinkt zwar in dieser Reihenfolge, jedoch verringern

sich auch Vollständigkeit und Realitätsnähe der Tests. Während im Fahrzeug vollständig reale Daten vorliegen, werden diese am HIL und am Tischaufbau simuliert. Am HIL können im Vergleich zum Tischaufbau z.B. noch einige Szenarien (z.B. Fahrzeugstart, Notbremsung, etc.) durchgespielt werden.

1.3. Ziele

In dieser Arbeit soll nun ein Konzept für den Vernetzungstest der Steuergeräte im Antriebsstrang entwickelt und umgesetzt werden. Hierfür wird zu Beginn eine Auflistung von Testkategorien erstellt, welche anschließend im Hinblick auf die Durchführbarkeit am Tischaufbau evaluiert werden sollen. Die am Tischaufbau dürchführbaren Testkategorien werden anschließend ausgearbeitet und detailliert beschrieben. Darüberhinaus soll für diese Tests eine Software entwickelt werden, welche Testfälle in Form von Testvektoren im XML Format generiert, die dann als Input von der bestehenden Toolkette verwendet werden.
Indirekt wird dadurch v.a. die Reduzierung des zeitlichen und auch finanziellen Aufwands für den Vernetzungstest von Steuergeräten im Antriebsstrang durch einen erhöhten Automatisierungsgrad angestrebt. Darüberhinaus bietet die entwickelte Software eine gute Grundlage, um den Umfang der am Tischaufbau durchgeführten Tests zu erweitern.

1.4. Aufbau der Arbeit

Im Anschluss an diese Einleitung gibt Kapitel 2 einen Überblick über Steuergeräte im technischen Kontext des Fahrzeugs. Ebenfalls wird beschrieben, wie die Kommunikation zwischen Steuergeräten in Fahrzeugen heutzutage mit Bussystemen realisiert wird. Es wird eine Übersicht über die im Rahmen diese Arbeit relevanten Bussysteme gegeben. Darüberhinaus werden die Themen AUTOSAR, XCP und OBD im Hinblick auf die Ziele dieser Arbeit vorgestellt. Abschließend werden die verschiedenen Testumgebungen mit Fokus auf den Testaufbau am Arbeitsplatz beschrieben.

Das Testkonzept für den Vernetzungstest von Steuergeräten im Antriebsstrang wird in Kapitel 3 detailliert vorgestellt. Es wird erörtert, welche Testkategorien benötigt werden, um die Anforderungen an die Vernetzungssoftware vollständig abzudecken. Für die am Tischaufbau durchführbaren Testkategorien werden die notwendigen Testfälle und Testschritte detailliert beschreiben.

Das entwickelte Tool *Network-Test-Generator* zur Erzeugung der Testvektoren für den automatisierten Vernetzungstest am Tischaufbau wird in Kapitel 4 vorgestellt.

Abschließend wird in Kapitel 5 ein Fazit gezogen und ein Ausblick auf zukünftige Arbeiten z.B. im Hinblick auf eine Erweiterung der am Tischaufbau automatisiert durchführbaren Tests gegeben.

Kapitel 2

Grundlagen

2.1. Steuergeräte und deren Vernetzung im Fahrzeug

Im Folgenden wird ein kurzer Überblick über Steuergeräte, wie sie heutzutage im Fahrzeug eingesetzt werden, sowie deren Vernetzung gegeben.

2.1.1. Steuergerät

Ein Steuergerät (engl. *ECU = electronic control unit*) ist in einem Fahrzeug überall dort zu finden, wo es der Steuerung, Regelung und Überwachung von z.B. Prozessen, Funktionen oder mechanischen Komponenten bedarf. Das bekannteste Beispiel ist das Motorsteuergerät. Es ist dafür verantwortlich, alle mit dem Motormanagement in Zusammenhang stehenden Funktionen zu steuern und zu überwachen.
Die Funktionsweise eines Steuergeräts basiert auf dem EVA-Prinzip [12], welches sich wie folgt erklären lässt:

1	Eingabe	z.B. Messwerte von Sensoren (beispielsweise Temperatur).
2	Verarbeitung	z.B. Kontrolle von Messwerten (Temperatur zu hoch/niedrig).
3	Ausgabe	z.B. Senden eines Signals als Reaktion auf Abweichung vom Sollwert.

Tabelle 2.1. EVA-Prinzip mit Beispiel im technischen Kontext des Fahrzeugs

Steuergeräte sind dabei eingebettete Systeme, da sie in den technischen Kontext des Fahrzeugs eingebunden sind. Im Hinblick auf die Hardwareauswahl werden Steuergeräte immer für einen fest definierten Aufgabenbereich und Funktionsumfang entwickelt. Daher unterscheiden sich Steuergeräte in der Größe und Leistungsfähigkeit teilweise stark. Es gibt sowohl Ein-Chip-Geräte für simplere Aufgaben wie z.B. die Ambientenlicht-Steuerung als auch Mehrprozessorsysteme mit Grafik wie z.B. ein Display-Steuergerät.
Die Automobilhersteller entwickeln die Steruergeräte-Hardware nur sehr selten selbst. Viel mehr wird diese Entwicklung von Zulieferern wie übernommen.

2.1.2. AUTOSAR

AUTOSAR steht für *Automotive Open System Architecture*. Die AUTOSAR [1] Initiative wurde 2003 durch den Zusammenschluss mehrerer Automobilhersteller sowie verschiedener Partner der Automobilindustrie wie z.B. Zulieferern aber auch Unternehmen der Software- und Elektronikindustrie zu einer Entwicklungspartnerschaft gegründet. Stand Januar 2015 beträgt die Zahl der an der AUTOSAR-Initiative beteiligten Partner 180 Unternehmen.

2.1.2.1. Idee

Das primäre Ziel der AUTOSAR Initiative ist eine standardisierte Softwarearchitektur für die im Fahrzeug zum Einsatz kommenden Steuergeräte. Da die Anforderungen an jene Steuergeräte im Laufe der letzten Jahre stetig zugenommen haben und diese Entwicklung in den nächsten Jahren mit Sicherheit fortschreiten wird, steigt die Komplexität der E/E-Fahrzeugarchitektur ebenfalls an. Die Idee ist, gemeinsam eine standardisierte Softwarearchitektur zu erarbeiten, welche unabhängig von der eingesetzten Hardware ist, sodass problemlos auf unterschiedliche Hersteller- und Fahrzeugasuführungen skaliert werden kann. Es wird jedoch genügend Freiraum in der Entwicklung gewahrt, damit der Wettbewerb unter den Automobilherstellern nicht gestört wird - dieser ist ausdrücklich erwünscht.

2.1.2.2. Konzept

Die von der AUTOSAR-Initiative entwickelte Softwarearchitektur basiert auf einem Schichtenmodell, das aus folgenden Schichten besteht:

- Basissoftware
- Laufzeitumgebung
- Anwendungsschicht

Die Basissoftware besteht aus einigen vordefinierten Softwaremodulen, die die Kommunikation mit der zugrunde liegenden Hardware und die Nutzung der Ressourcen des Steuergeräts ermöglichen. Die Laufzeitumgebung ist eine Middleware, die für die Kommunikation zwischen Applikations-Software-Komponenten und der Basissoftware benötigt wird. Es spielt dabei keine Rolle, ob die Kommunikation zwischen mehreren Steuergeräten (z.B. über einen CAN-Bus) oder lediglich innerhalb eines Steuergeräts erfolgt. In der Anwendungsschicht finden sich die eigentlichen Softwarekomponenten, die die Anwendungsfunktionalität des Steuergeräts realisieren.
Mit diesem Schichtenmodell kann Software einfach auf eine andere oder neue Hardware portiert werden, was vor der Einführung von AUTOSAR nur mit umfangreichen Anpassungen möglich war. Es wird also eine strikte Trennung zwischen technischem

[1] https://www.autosar.org

(hardwarenahem) und funktionalem Code erreicht, die wiederum eine hohe Hardwareunabhängigkeit gewährleistet [13].

2.1.2.3. Methodik

Die AUTOSAR Methodik gibt einige Richtlinien für die Entwicklung von Steuergeräte-Software vor.

1. Zu Beginn wird das System entworfen. Die Fahrzeugfunktionen werden als Softwarekomponenten umgesetzt, welche einen atomaren Teil der Software darstellen. Anschließend findet die Verteilung der Softwarekomponenten auf die Steuergeräte statt. Neben der Systemarchitektur wird auch die Netzwerkkommunikation beschrieben. Dieser Prozess resultiert in der Systembeschreibung (.arxml-Datei).

2. Aus der Beschreibung des (gesamten) Systems wird für jedes Steuergerät ein sogenannter *ECU Extract of System Description* generiert. Darin enthalten ist eine Teilmenge mit für das jeweilige Steuergerät relevanten Informationen der gesamten Systembeschreibung, welche die Beschreibung der auf diesem Steuergerät vorhandenen Softwarekomponenten und den relevanten Teil der Kommunikationsmatrix beinhaltet.

3. Die Basissoftware und die Laufzeitumgebungen werden konfiguriert. So wird z.B. der benötigte Umfang der Basissoftware festgelegt, um eine möglichst optimale Steuergeräte-Komfiguration zu erhalten. Daraus resultiert die *ECU Configuration Description* (ECUC).

4. Abschließend wird die Basissoftware sowie die Laufzeitumgebung durch einen Codegenerator erstellt.

Darüberhinaus wird mit der AUTOSAR Methodik ein standardisierter Datenaustausch ermöglicht [3].

2.1.3. Vernetzung von Steuergeräten

Steuergeräte in aktuellen Fahrzeugen arbeiten nur in Ausnahmefällen als eigenständige Einheiten, weshalb die Kommunikation und der Datenaustausch zwischen den Steuergeräten eine wichtige Rolle bei der Entwicklung spielt. Vor einigen Jahrzehnten wurde der Datenaustausch noch mittels Punkt-zu-Punkt-Verbindungen und größtenteils analogen Signalen realisiert. Heutzutage ist dies nicht mehr denkbar, da Informationen (Signale) in vielen Fällen von mehreren Steuergeräten benötigt werden.
Mit der rasanten Zunahme der mikroprozessorgesteuerten Systeme in heutigen Fahrzeugen (z.B. Fahrerassistenzsysteme oder Infotainment) haben sich im Laufe der Jahre Bussysteme als primäres Transportmedium durchgesetzt. Ein weiterer Vorteil von Bussystemen ist die Kostenersparnis durch die Reduzierung des Leitungssatzes. Die zunehmende Komplexität und das stark wachsende Datenaufkommen sind dafür verantwortlich, dass heutzutage mehrere (miteinander verbundene) Netzwerke im Fahrzeug vorzufinden sind. In einer

aktuellen Mercedes-Benz S-Klasse mit Vollausstattung werden bis zu 175 Steuergeräte verbaut.
Abbildung 2.1 zeigt ein stark vereinfachtes Beispiel einer Netzwerktopologie in einem typischen aktuellen Fahrzeug. Die verschiedenen durch ein Gateway miteinander verbundenen Bussysteme sind hier gut zu erkennen (Antriebstrang, Multimedia, Chassis, Tür, Diagnose). Ein solches Gateway unterstützt mehrere Transportprotokolle und kann daher Botschaften von einem Netzwerk in ein anderes Netzwerk weiterleiten. Man beachte, dass diese Abbildung keine vollständige Topologie wiederspiegelt, sondern viel mehr dem Leser helfen soll, sich ein Bild von der Steuergerätelandschaft und der Vernetzung im Fahrzeug zu machen.

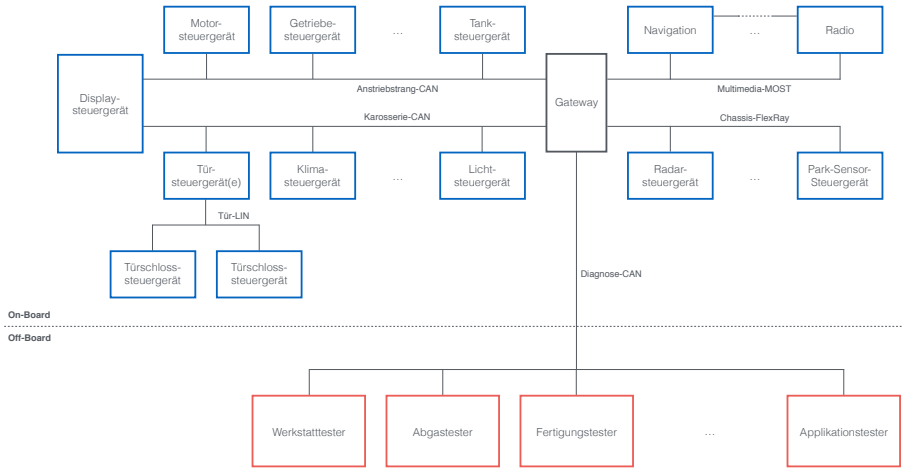

Abbildung 2.1. Beispielhafte Darstellung der Steuergeräte und Bussysteme in einem aktuellen Mittelklassefahrzeug (angelehnt an [1])

Es ist ersichtlich, dass nicht alle Bussysteme auf dem gleichen Protokoll basieren. In diesem Beispiel sind die folgenden Bussysteme vertreten:

- CAN (Controller Area Network)
- LIN (Local Interconnect Network)
- FlexRay
- MOST (Media Oriented Systems Transport)

In Kapitel 2.2 werden die verschiedenen Bussysteme näher vorgestellt. Der Umfang beschränkt sich dabei jedoch auf die im Kontext dieser Arbeit relevanten Protokolle, also jene, die für den automatisierten Vernetzungstest von Steuergeräten im Abtriebstrang benötigt werden. In der Zukunft wird Ethernet mit Sicherheit eine zunehmend große Rolle spielen, da die Anforderungen im Bereich Multimedia / Infotainment und Fahrerassistenzsysteme im Fahrzeug ständig wachsen. Ethernet eignet sich hierfür aufgrund der hohen

Übertragungsraten (100 Mbit/s) und der Flexibilität, die sich durch die Unterstützung verschiedener physikalischer Übertragungsmedien und die Möglichkeit der Verbindung mehrere Ethernet-Netzwerke durch Switches ergibt.

2.2. Bussysteme

Betrachtet man Abbildung 2.1, stellt man fest, dass sich auf einem hohen Abstraktionsniveau zwei Anwendungsgebiete unterscheiden lassen: On-Board-Kommunikation und Off-Board-Kommunikation (vgl. Abbildung 2.1). Diese lassen sich wiederum in weitere Teilgebiete aufteilen, in denen unterschiedliche Anforderungen an die zum Einsatz kommenden Bussysteme gestellt werden. Tabelle 2.2 gibt einen Überblick über die Anwendungsbereiche und die Anforderungen an die in der On-Board-Kommunikation zum Einsatz kommenden Bussysteme.

On-Board-Kommunikation	
High-Speed-Systeme	High-Speed-Systeme werden meist in der Motor-, Fahrwerks- und Bremsensteuerung eingesetzt. Diese Systeme stellen Echtzeit-Anforderungen an die Kommunikation, da sie sicherheitskritische Funktionen umsetzen. Des Weiteren werden die Daten in sehr kurzen Intervallen zyklisch übertragen, weshalb eine hohe Datenrate unabdingbar ist.
Low-Speed-Systeme	Low-Speed-Systeme werden für einfachere Aufgaben wie z.B. die Tür- und Fenstersteuerung oder die Lichtsteuerung eingesetzt. Die benötigten Datenraten sind hier eher niedrig, weshalb kostengünstigere Low-Speed-Systeme zur Vereinfachung des Kabelbaums eingesetzt werden und sogar speziell für diese Zwecke entwickelt werden.
Multimedia-Bussysteme	Multimedia-Bussysteme werden vorrangig im Infotainment-Bereich eingesetzt. Die Menge der zu übertragenden Daten ist hier sehr hoch, z.B. Radardaten / GPS Daten für das Navigationssystem. Allerdings sind die Anforderungen an die Latenzzeit und die Ausfallsicherheit geringer, da keine sicherheitskritische Steuerung übernommen wird und die Kosten für solche Bussysteme in der Regel schon sehr hoch sind.

Tabelle 2.2. Anwendungsbereiche der Bussysteme in der On-Board-Kommunikation

Die Off-Board-Kommunikation ist für den Vernetzungstest von Steuergeräten im Fahrzeug weniger relevant. Ein Beispiel dafür wäre z.B. das Auslesen des Fehlerspeichers mittels eines Diagnosegeräts in einer Kfz-Werkstatt oder das allgemeine Auslesen von steuergeräteinternen Variablen während der Entwicklung.

2.2.1. Kommunikation zwischen Steuergeräten

Steuergeräte kommunizieren über einen sogenannten Protokollstapel. Dieser entsteht, indem Botschaften von der Anwendung bis zur eigentlichen physikalischen Übertragung mehrere Schichten durchlaufen. Analog verläuft der Empfang von Botschaften, nur in entgegengesetzter Richtung. In aktuellen Steuergeräten kommen dabei die Schichten 1, 2, 4 und 7 des OSI-Modells (*engl. Open Systems Interconnection Model*) zum Einsatz; die restlichen Schichten spielen (noch) keine Rolle. Abbildung 2.2 veranschaulicht die Kommunikation zwischen zwei Steuergeräten über einen solchen Protokollstapel. Die einzelnen Schichten sollen dabei den Eindruck vermittelt bekommen, direkt mit den jeweiligen Schichten der anderen Steuergeräte zu kommunizieren (virtuelle Kommunikation), obwohl die reale Kommunikation über ggf. mehrere Schichten und einen Bus abläuft. Dadurch können die Steuergeräte voneinander entkoppelt werden. Man beachte, dass die interne Struktur der Steuergeräte stark vereinfacht dargestellt ist.

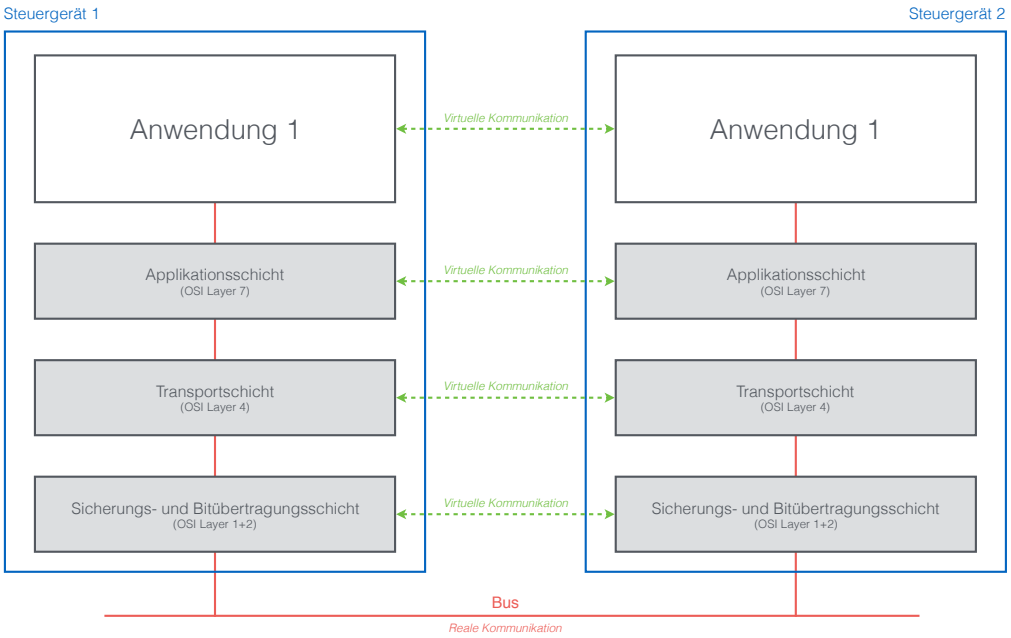

Abbildung 2.2. Kommunikation zwischen Steuergeräten über Protokollstapel (angelehnt an [1])

In den folgenden Kapiteln werden nun - im Rahmen des für diese Arbeit benötigten Umfangs - die Bussysteme CAN, LIN und FlexRay im Hinblick auf die folgenden Aspekte vorgestellt, welche in Tabelle 2.3 beschrieben werden.

Aspekte	
Datenrate	Die Geschwindigkeit, mit welcher einzelne Bits über die physikalischen Leitungen des Bussystems übertragen werden. Formal: Menge der Daten, die über einen Zeitraum über ein physikalisches Übertragungsmedium übertragen werden kann.
Topologie	Beschreibt die (physikalische) Verbindungsstruktur zwischen den Kommunikationsteilnehmern in einem Netzwerk. Beispiele sind die Punkt-Zu-Punkt-Verbindung, die Ring-Topologie oder die Linien-Topologie.
Kommunikationsprinzip	Beschreibt die im Kommunikationsprotokoll festgelegten Regeln und Prinzipien, nach welchen im Netzwerk über das Bussystem kommuniziert wird.
Framing	Die Übertragung von Daten in einem Bussystem erfolgt durch sogenannter Frames (Botschaften). Beschreibt den Aufbau der Frames im Hinblick auf das jeweilige Bussystem.
Datensicherung	Mechanismen, die eine hohe Ausfallsicherheit, eine korrekte Datenübertragung und eine Fehlererkennung garantieren sollen. Im Rahmen dieser Arbeit werden lediglich logische Fehlererkennungstechniken vorgestellt, es wird nicht auf die elektromagnetische Verträglichkeit (EMV) eingegangen.
Kommunikationsbeispiel	Beispielhafte Kommunikation zwischen einigen Knoten in einem kleinen Netzwerk.

Tabelle 2.3. Aspekte für die Vorstellung der Bussysteme

2.2.2. CAN

In heutigen Fahrzeugen wird CAN (Controller Area Network) im Vergleich zu anderen Bussystemen am häufigsten zur Vernetzung eingesetzt. Von Bosch in den 1980er Jahren entwickelt zeichnet es sich bis heute durch eine hohe Geschwindidkeit und Sicherheit bei gleichzeitig relativ geringen Kosten aus, weshalb es sich besonders für den Einsatz im Antriebsstrang eignet.
Standardisiert ist es durch die in Tabelle 2.4 genannten Normen:

ISO 11898-1	Spezifikation des CAN-Protokolls.
ISO 11898-2	High-Speed-CAN.
ISO 11898-3	Low-Speed-CAN.

Tabelle 2.4. ISO Normen für CAN

2.2.2.1. Datenrate

In High-Speed-CAN-Netzwerken können Datenübertragungsraten bis zu 1Mbit/s erreicht werden, jedoch ist Länge der Übertragungsleitungen hierbei auf 40m beschränkt (bei einer maximalen Länge von 100m können 500kbit/s garantiert werden). Die maximale Buslänge kann wie folgt berechnet werden:

$$\text{Buslänge} \leqslant 40 \ldots 50 \cdot \frac{1\text{Mbit/s}}{\text{Bitrate}}$$

Diese Formel ist keine exakte Formel, da mit steigender Bitrate zusätzliche Faktoren wie z.B. die Verzögerungszeiten der Bustranceiver berücksichtigt werden müssen ([1] Seite 46). Das $\leqslant$-Zeichen in der Formel ist als „maximal“ zu interpretieren.
In Low-Speed-CAN-Netzwerken beträgt die maximale Datenübertragungsrate 125 kbit/s bei einer maximalen Länge der Übertragungsleitungen von 500m.

2.2.2.2. Topologie

CAN-Netzwerke basieren immer auf der klassischen Bus-Topologie (vgl. Abbildung 2.3). Gemäß ISO 11898 beträgt die maximale Anzahl an CAN-Knoten in einem Netzwerk 32. Innerhalb der Steuergeräte stellt die aus CAN-Transceiver und CAN-Controller bestehende CAN-Schnittstelle die zur Kommunikation im CAN-Netzwerk benötigte Funktionalität bereit. Da das CAN-Protokoll eine sogenannte Differenzsignalübertragung vorsieht, werden 2 Übertragungsleitungen benötigt: die CAN-High-Leitung (CANH) sowie die CAN-Low-Leitung (CANL).

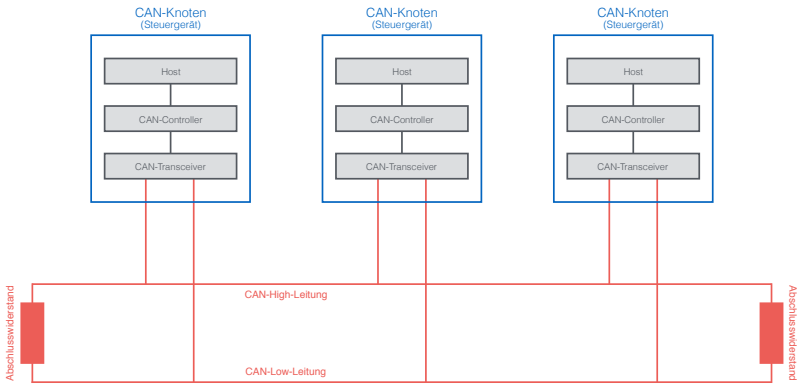

Abbildung 2.3. CAN-Netzwerk mit 3 Knoten (angelehnt an [4])

	High-Speed-CAN	**Low-Speed-CAN**
logische 0	2 Volt	
logische 1	5 Volt	0 Volt

Tabelle 2.5. Zusammenhang zwischen Differenzspannung und logischem Wert (0, 1)

Tabelle 2.5 beschreibt die den logischen Werten jeweils zugeordneten Differenzspannungen. Die logische 0 wird dabei dominant übertragen. Um Reflexionen und infolgedessen Störungen bei der Signalübertragung zu vermeiden, befindet sich an den Enden des CAN-Netzwerkes jeweils ein Abschlusswiderstand mit 120Ω.

2.2.2.3. Kommunikationsprinzip

Die ISO 11898 definiert eine *Multi-Master-Architektur* für CAN-Bussysteme. Dies bedeutet, dass jeder Knoten in einem CAN-Netzwerk grundsätzlich gleichberechtigt ist. Die Kommunikation erfolgt ereignisorientiert und folgt keinem vordefinierten zeitlichen Schema. Daher wird der Übertragungskanal nicht unnötigt belegt, wodurch schnelle Buszugriffszeiten gewährleistet werden können. Prinzipiell darf jeder Knoten eine Datenübertragung initiieren. Auf den Bus abgesetzte Botschaften werden zunächst an alle anderen Knoten im CAN-Netzwerk gesendet. Durch die empfängerselektive Adressierung ermittelt jeder Knoten die für ihn relevanten Botschaften und verwirft die übrigen. Die Adressierung ist weiterhin inhaltsbezogen, da Botschaften durch den in ihnen enthaltenen *Message Identifier (ID)* identifiziert werden. Diese Art der Adressierung hat den Vorteil, dass zusätzliche Knoten ohne umfangreiche Konfiguration in das Netzwerk integriert werden können. Des Weiteren wird kein Protokoll für den Verbindungsaufbau benötigt, da die Kommunikation verbindungslos abläuft. Darüberhinaus dient die ID zur Priorisierung der Botschaften,

sodass eine eindeutige Sendereihenfolge gegeben ist und sich folglich immer nur eine Botschaft auf dem Bus befindet.

2.2.2.4. Framing

In der ISO 11898-1 werden die folgenden Frametypen definiert:

CAN Frames	
Data Frame	Das Data Frame (vgl. Abbildung 2.4) wird für die normale Datenübertragung verwendet. Es können dabei maximal acht Bytes Nutzdaten übertragen werden.
Remote Frame	Das Remote Frame dient zur Anforderung von Daten bei einem beliebigen CAN-Knoten. Es ist bis auf das fehlende Data Field (vgl. Abbildung 2.4) genau wie ein Data Frame aufgebaut.
Error Frame	Das Error Frame ermöglicht das Signalisieren von während dem Kommunikationsbetrieb entdeckten Fehlern. Die aktuelle Botschaftsübertragung wird dabei abgebrochen. Ein Error Frame besteht aus nur 2 Feldern: Error Flag und Error Delimiter.

Tabelle 2.6. CAN Frametypen

Das Remote Frame kommt in Fahrzeugen verbauten CAN-Bussystemen kaum zum Einsatz, da die Kommunikation hier ereignisorientiert abläuft. Für den Umfang dieser Arbeit ist v.a. das Data Frame interessant, welches im Folgenden detailliert vorgestellt wird.
Zu Beginn einer Übertragung eines Data Frames wird immer das SOF-Bit (Start of Frame) gesendet. Da sich das Netzwerk im Zustand Bus Idle (rezessiv) befindet und das SOF-Bit immer dominant übertragen wird, kann auf diese Weise das Netzwerk synchronisiert werden. Im Arbitration Field befindet sich der Identifier für die Priorisierung und Adressierung des Frames, das RTR-Bit gibt den Frametyp (Data oder Remote Frame) an.
Nun folgt in jedem Fall das IDE-Bit, welches Aufschluss über das Format des Identifiers gibt: ein Identifier im Standard-Format ist 11 Bits lang, wohingegen das Extended-Format 29 Bits umfasst. Im Extended-Format wird das RTR-Bit durch das SRR-Bit ersetzt, welches rezessiv übertragen wird. Die beiden Bits r1 und r0 haben keine Bedeutung und werden dominant übertragen.
Die Länge des Data Fields, also die Anzahl der Nutzdatenbytes wird im DLC (Data Length Code) übermittelt. Diese Länge bewegt sich zwischen null und acht Bytes. Die Nutzdaten werden mit Hilfe des Cyclic Redundancy Checks (CRC) gesichert (siehe Kapitel 2.2.2.5). Die dafür benötigte Prüfsumme wird als CRC Sequence übertragen, gefolgt von einem

rezessiv übertragenen CRC Delimiter.

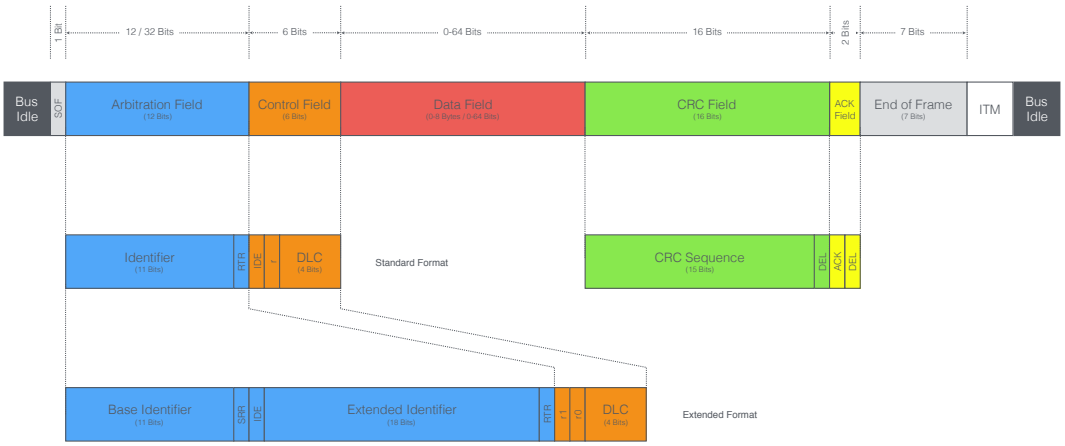

Abbildung 2.4. Interner Aufbau eines CAN Data Frames (angelehnt an [4])

Auf Basis dieser Prüfsumme entscheiden die Empfänger, ob sie jeweils positiv (ACK-Bit dominant) oder negativ (ACK-Bit rezessiv) quittieren. Der ACK-Delimiter wird stets rezessiv übertragen.
Die Übetragung eines Data Frames wird mit 7 rezessiven Bits beendet.

2.2.2.5. Datensicherung

Da CAN-Bussysteme in Fahrzeugen vorrangig für zeit- und sicherheitskritische Steuerungs- und Überwachungsaufgaben eingesetzt wird, spielt die korrekte und reibungslose Übertragung der Daten eine große Rolle. In einem CAN-Netzwerk werden daher 5 verschiedene logische Fehlererkennungsmechanismen verwendet. Tabelle 2.7 erklärt diese 5 Mechanismen zur logischen Fehlererkennung sind:

Logische Fehlererkennung	
Bitmonitoring	Das Bitmonitoring wird von sendenden Knoten durchgeführt. Dabei vergleichen diese jedes einzelne auf den Bus aufgelegte Bit mit dem eigentlichen Buspegel und signalisieren bei einem Unterschied einen Fehler (mit Ausnahme des Arbitration Fields und des ACK Slots).

Bitstuffing	Gemäß des CAN-Protokolls müssen alle Sender nach fünf homogenen Bits ein komplementäres Bit senden. Dieser Resynchronisationsmechanismus sorgt für einen Gleichlauf über die gesamte Übertragungsdauer hinweg. Die Aufgabe der Empfänger ist es, kontinuierlich einen Stuff Check durchzuführen, sodass bei Erkennung von mehr als fünf homogenen Bits ein Fehler signalisiert wird. Der Stuff Check wird auf den Bereich zwischen dem Start of Frame (SOF) bis einschließlich der CRC Sequenz (vgl. Abbildung 2.4) angewandt.
Form Check	Mit dem Form Check validieren die Empfänger das Botschaftsformat. Dies wird durch das Überprüfen von gewissen Komponenten der empfangenen Botschaft realisiert, die stets identisch übertragen werden. So werden der CRC-Delimiter, der ACK-Delimiter und das End of Frame (EOF) (vgl. Abbildung 2.4) immer rezessiv übertragen. Wird an einer solchen Stelle ein dominanter Buspegel detektiert, zieht dies die Signalisierung eines Fehlers nach sich.
CRC	Basierend auf den zu übertragenden Bits zwischen dem Start of Frame (SOF) und dem Data Field (einschließlich) sowie einem in der ISO 11898-1 definierten sogenannten Generatorpolynoms wird die CRC Sequenz berechnet, welche einem Vielfachen der zu übertragenden Bits entspricht. Die Empfänger prüfen diese Anforderung und signalisieren ggf. einen Fehler. Ein Fehler wird nur dann nicht erkannt, wenn die CRC Sequenz selbst fehlerhaft übertragen oder falsch berechnet wurde. Auf Basis dieser Überprüfung setzen die Empfänger das ACK.
ACK Check	Der Sender vergleich den rezessiven Buspegel mit dem Buspegel im ACK-Slot. Wurde dieser nicht von einem Empfänger (dominant) überschrieben, liegt ein ACK-Fehler vor. Es genügt also schon ein positiv quittierender Empfänger, sodass der ACK Check erfolgreich ist.

Tabelle 2.7. Logische Fehlererkennungsmechanismen in einem CAN-Netzwerk

2.2.2.6. Kommunikationsbeispiel

Abbildung 2.5 veranschaulicht eine beispielhafte Kommunikation in einem CAN-Netzwerk mit 4 Knoten. Die Kommunikationsmatrix (hier stark vereinfacht) definiert die IDs der Botschaften sowie die zugehörigen Sender und Empfänger. Im Sendezweig ist die Sendereihenfolge zu erkennen. Die Bits auf dem Bus können von allen Kommunikationsteilnehmern empfangen werden.

Kommunikationsmatrix

Data Frame ID	CAN Knoten A	CAN Knoten B	CAN Knoten C	CAN Knoten D
0x12	Sender	Empfänger		
0x34		Sender	Empfänger	Empfänger
0x52		Empfänger		Sender
0x67	Empfänger	Empfänger	Sender	Empfänger
0xB4	Empfänger		Sender	
0x3A5	Sender	Empfänger	Empfänger	Empfänger

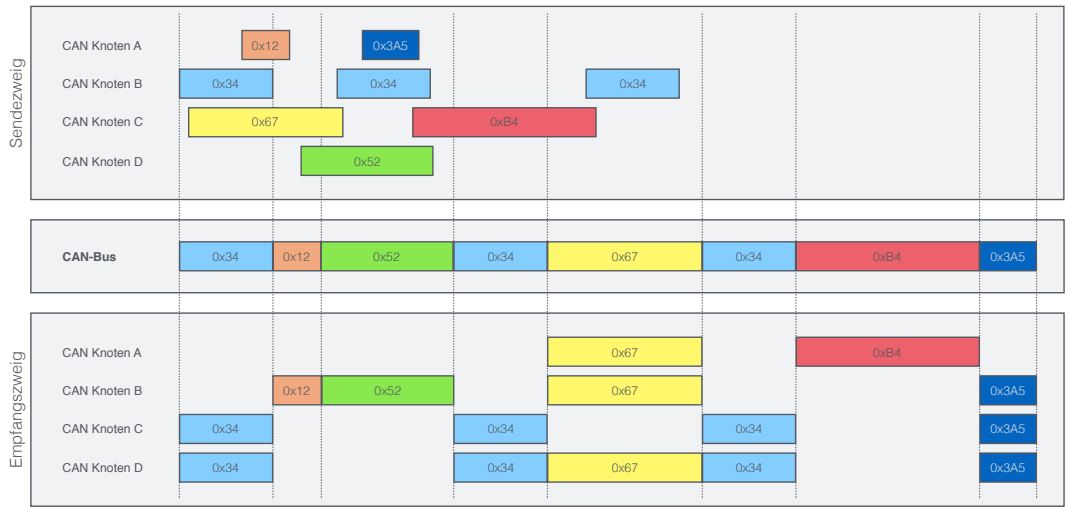

Abbildung 2.5. Beispielhafte Kommunikation in einem CAN-Netzwerk mit vier Knoten (angelehnt an [4])

2.2.3. LIN (Local Interconnect Network)

Ende der 1990er Jahre schlossen sich einige Automobilhersteller (darunter Daimler-Benz und BMW) zu einem Konsortium zusammen, um gemeinsam an einem im Vergleich zu CAN (Low-Speed-CAN) kostengünstigeren Bussystem für Sensoren und Aktoren v.a. im Komfortbereich zu arbeiten. In den davorliegenden Jahren hatten einige Automobilhersteller die Entwicklung bereits in Eigeninitiave begonnen. Eine Preissenkung war jedoch nur bedingt möglich, da die Stückzahlen relativ gering blieben.
Künftig soll der letzte vom LIN Konsortium entwickelte Stand als ISO 17987-1 erscheinen.

2.2.3.1. Datenrate

Die Datenübertragungsrate liegt im Bereich von 1 - 20 kbit/s. Empfohlen werden 2,4 kbit/s, 9,6 kbit/s und 19,2 kbit/s. Die maximale Länge der Übertragungsleitungen liegt bei 40m.

2.2.3.2. Topologie

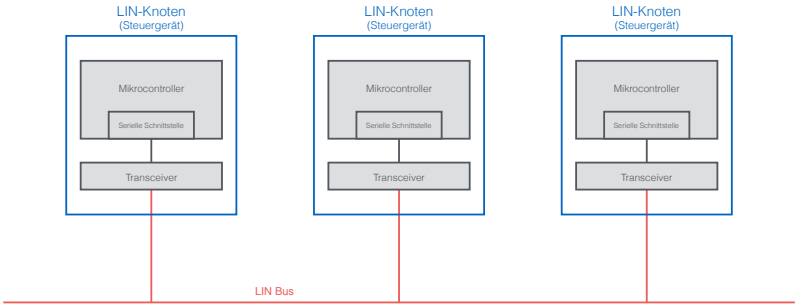

Abbildung 2.6. LIN-Netzwerk mit 3 Knoten (angelehnt an [5])

LIN-Netzwerke sind wie CAN-Netzwerke als Linien-Tpologie aufgebaut. Es wird eine maximale Anzahl von 16 Knoten empfohlen. Die LIN-Spezifikation sieht eine bidirektionale Eindrahtleitung vor, wobei für die logische 0 und die logische 1 jeweils Spannungspegel definiert werden, wobei diese Pegel eher als Bereiche zu deuten sind, wie Tabelle 2.8 verdeutlicht. Die angegebenen Werte beziehen sich dabei auf die Versorgungsspannung. Die logische 0 wird dominant übertragen.

	Sender	**Empfänger**
logische 0	$\leqslant 20\%$	$\leqslant 40\%$
logische 1	$\geqslant 80\%$	$\geqslant 60\%$

Tabelle 2.8. Zusammenhang zwischen Spannungspegel und logischem Wert (0, 1)

2.2.3.3. Kommunikationsprinzip

LIN-Bussysteme weisen eine *Master-Slave-Architektur* auf. In einer solchen Kommunikationsarchitektur existiert ein Master-Knoten, der für die Kommunikationssteuerung innerhalb des Netzwerkes zuständig ist. Slave-Knoten dürfen nicht ohne Weiteres senden. Erst nach einer Aufforderung durch den Master-Knoten sind diese sendeberechtigt. Der Master-Knoten umfasst neben einem Slave Task im Gegensatz zu den Slave-Knoten auch noch einen sogenannten Master Task. Dieser verfügt über einen vom Systemdesigner vordefinierten Ablaufplan, der eine in Slots organisierte feste zeitliche Abfolge zu sendenen Botschaften beschreibt. Gemäß dieses Ablaufplans sendet der Master-Knoten pro Slot einen Frame Header, welchen alle Knoten erhalten (auch der Slave Task im Master-Knoten). Dieser Frame Header beinhaltet eine ID, sodass genau ein Slave-Knoten mit einer Frame Response antwortet, bzw. den Frame Header um die Frame Response ergänzt, woraus ein Frame entsteht, welches allen Knoten im Netzwerk zur Verfügung steht. Anhand der

ID entscheiden die empfangenden Knoten, ob der Inhalt verworfen wird oder für diese relevant ist. Wie CAN ist auch die Kommunikation bei LIN verbindungslos, es wird also kein Protokoll für den Verbindungsaufbau benötigt.

2.2.3.4. Framing

Ein LIN Standard Frame setzt sich aus einem Frame Header und einer Frame Response zusammen (vgl. Abbildung 2.7). Mit dem Senden von Frame Headern fordert der Master-Knoten Daten (Frame Responses) von den Slave-Knoten an.
Der Frame Header beginnt mit dem SYNC Break, welcher sich aus mindestens 13 dominanten und einem rezessiven Bit zusammensetzt, wodurch jeder Slave-Knoten den Beginn einer Übertragung feststellen kann. Um den Bittakt der Empfänger zu synchronisieren, wird anschließend das SYNC Byte (0x55 bzw. 0b01010101) (alternierende Bitfolge) gesendet. Der Frame Header endet mit der Übertragung des Protected Identifiers, der wie bei CAN zur inhaltsbezogenen Adressierung dient, sodass genau 1 Knoten mit einer Antwort reagiert. Innerhalb des PIDs stehen 6 Bits zur eigentlichen Adressierung zur Verfügung, die restlichen 2 Bits werden für die Paritätsprüfung (XOR-Verknüpfung) verwendet (P0 steht für gerade, P1 für ungerade Parität). Man beachte, dass diese Prüfung im Vergleich mit anderen Datensicherungsverfahren verhältnismäßig unsicher ist, so bleibt jeder einhundertste Bitfehler unerkannt ([5]).

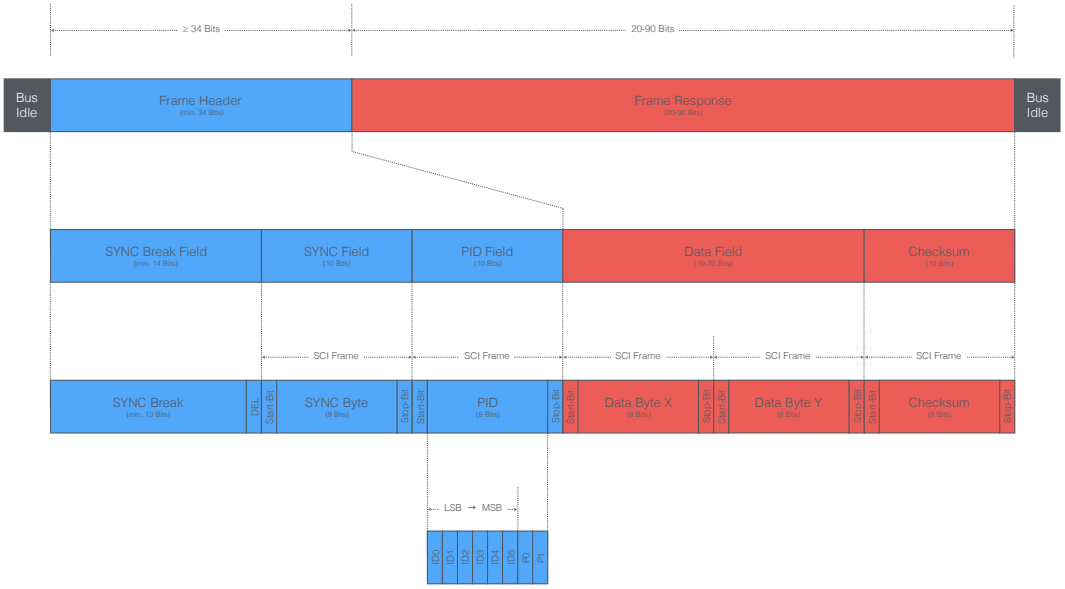

Abbildung 2.7. Interner Aufbau eines LIN Standard Frames (angelehnt an [5])

In der Frame Response werden die als Antwort von einem Slave-Knoten bereitgestellten Daten übertragen. Die Länge der Nutzdaten ist dabei auf maximal acht Bytes beschränkt.

Jeder Knoten im Netzwerk hat Zugriff auf Frame Responses und kann anhand des LIN Description File (LDF) entscheiden, ob die jeweiligen Informationen relevant sind. Zur Sicherung der Nutzdaten enthält eine Frame Response eine Prüfsumme.
Bei der Datenübertragung kommen sogenannte SCI Frames (Serial Communication Interface Frames) zum Einsatz, da aufgrund des nicht vorhandenen Kommunikationscontrollers diese direkt über das SCI des Mikrocontrollers umgesetzt wird. Dabei werden die bis zu acht Nutzdatenbits von einem dominantem Start-Bit und einem rezessiven Stop-Bit eingerahmt (vgl. Abbildung 2.7).
In einem LIN-Netzwerk werden die verschiedene Botschaftstypen verwendet. Diese werden in Tabelle 2.9 vorgestellt.

LIN Frames	
Unconditional Frame	Ein Unconditional Frame entspricht einem Standard Frame und kann ohne Weiteres für die Datenübertragung verwendet werden.
Event Triggered Frame	Ein Event Triggered Frame ermöglicht das Abfragen von Daten von mehreren Knoten. Im LDF werden dabei mehrere Knoten für die Antwort auf einen Frame Header konfiguriert. Eine Antwort erfolgt jedoch nur gemäß dem Fall, dass sich die angeforderten Daten seit der letzten Abfrage geändert haben. Ein solches Verhalten bietet sich für Werte an, die sich nicht gleichzeitig in mehreren Knoten ändern (z.B. bei Türkontaktsensoren). Falls jedoch eine Kollision entsteht, weil mehr als Knoten antwortet, wird diese Kollision durch den Master-Knoten aufgelöst, indem die Werte der Reihe nach angefordert werden (im nächsten Slot).
Sporadic Frame	Ein Sporadic Frame steht für die dynamische Übertragung von Daten im Master-Knoten oder die Anforderung von Slave-Daten zur Verfügung. Auch für diese Frames muss ein Slot reserviert werden, da der Master-Knoten aber nur bei Bedarf einen Frame-Header absetzt, kann es passieren, dass der entsprechende Slot ungenutzt bleibt.

Diagnostic Frame	Diagnostic Frames werden als Master Request Frames oder Slave Response Frames verwendet. Bei einem Master Request Frame sendet der Master-Knoten sowohl den Frame-Header als auch die Frame-Reponse, bei einem Slave Response Frame erfolgt die Übertragung wie gewohnt. Master Request Frames dienen zur Initiierung eines Diagnose-Vorgangs (Diagnose Request), Slave Response Frames zur Beantwortung eines Diagnose Requests.

Tabelle 2.9. LIN Frametypen

2.2.3.5. Datensicherung

Für LIN-Bussysteme sind verschiedene Fehlererkennungsmechanismen definiert, die Mechanismen zur Fehlerbehandlung müssen jedoch bei der Implementierung festgelegt werden. Wie bei CAN werden fünf Mechanismen zur logischen Fehlererkennung definiert. Tabelle 2.10 gibt einen Überblick über diese Mechanismen.

Logische Fehlererkennung	
Bitmonitoring	Das Bitmonitoring wird von sendenden Knoten durchgeführt. Dabei vergleicht dieser jedes einzelne auf den Bus aufgelegte Bit mit dem eigentlichen Buspegel. Es wird byteweise verglichen.
Checksum Check	Die Empfänger prüfen die Korrektheit der empfangenen Daten mit Hilfe der in der Frame Response übertragenen Checksum. Dabei stehen zwei Varianten zur Verfügung: Classic Checksum und Enhanced Checksum. Bei Verwendung der Enhanced Checkum wird neben den ankommenden Bytes auch der Protected Identifier (PID) zur Prüfung herangezogen. Die Classic Checksum werden dagegen nur die ankommenden Bytes berücksichtigt. Ein Fehler liegt vor, falls die Summe aus Daten und Checksum nicht 0xFF ergibt.
Parity Check	Die Empfänger überprüfen gemäß den im Protokoll spezifierten Regeln die im Frame Header zur Sicherung des Identifiers übertragenen beiden Paritätsbits.

Slave Responding Check	Die Empfänger überprüfen, ob im Anschluss an einen Frame Header auch einen Frame Response übertragen wird. Die einzige Ausnahme stellt hier das Event Triggeres Frame dar.
Sync Field Check	Die Empfänger überprüfen, ob sich der Datenübertragungstakt des Master-Knotens innerhalb der Toleranz befindet.

Tabelle 2.10. Logische Fehlererkennungsmechanismen in einem CAN-Netzwerk

2.2.3.6. Kommunikationsbeispiel

Eine beispielhafte Kommunikation in einem LIN-Netzwerk mit einem Master-Knoten und zwei Slave-Knoten wird in Abbildung 2.8 veranschaulicht. Die Kommunikationsmatrix definiert die IDs der Botschaften sowie die Sender-Empfänger-Beziehungen. Betrachtet man den Sendezweig ist das Kommunikationsprinzip leicht ersichtlich.

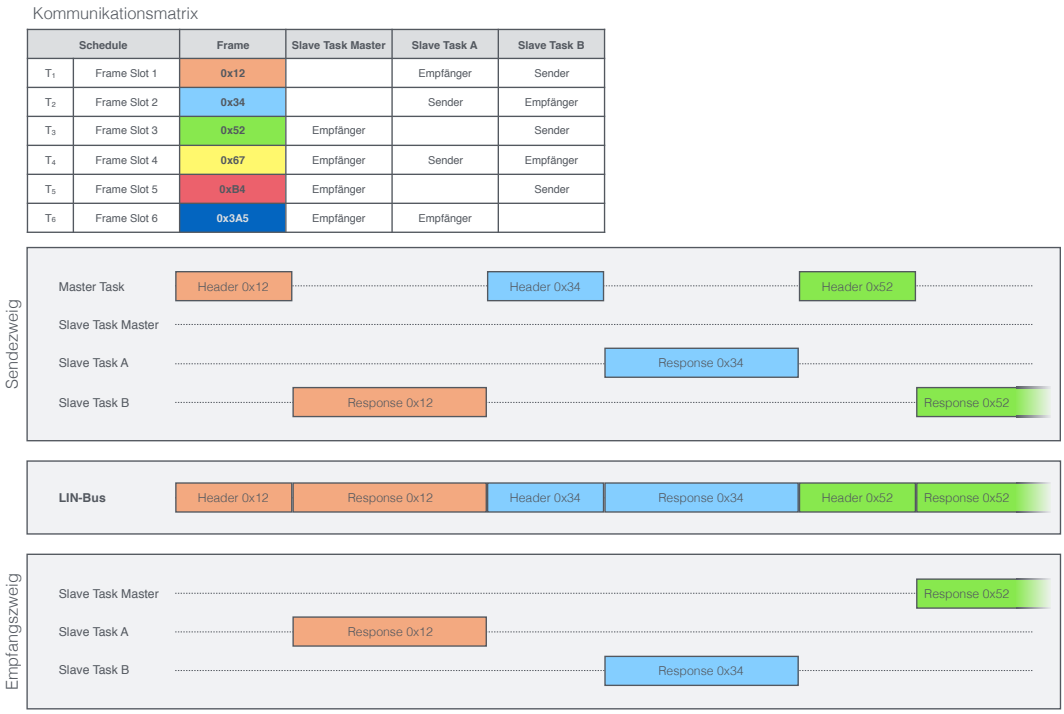

Abbildung 2.8. Beispielhafte Kommunikation in einem LIN-Netzwerk mit drei Knoten (angelehnt an [5])

2.2.4. FlexRay

Für besonders sicherheitskritische Fahrzeugfunktionen und Systeme wie z.B. das ESP oder das Tempomat erfüllen Bussysteme wie CAN nicht die notwendigen Anforderungen wie eine hohe Datenübertragungsrate und Echtzeitfähigkeit. Ähnlich wie das LIN Konsortium enstand im Jahre 2000 das FlexRay Konsortium mit dem Ziel, ein herstellerübergreifendes Kommunikationssystem mit den im Vorigen genannten Anforderungen zu schaffen.
Das FlexRay-Protokoll ist heute unter der ISO 17458 verfügbar.

2.2.4.1. Datenrate

Die maximale Datenübertragungsrate in einem FlexRay-Netzwerk ist auf 10Mbit/s begrenzt. Der größtmögliche Abstand zwischen zwei Knoten beträgt dabei 24m.

2.2.4.2. Topologie

FlexRay unetrstützt sowohl einkanalige als auch zweikanalige Strukturen. Man beachte aber, dass alle zur Zeit verfügbaren Kommunikationskontroller zwei Kanäle aufweisen. Ein FlexRay-Netzwerk, auch FlexRay-Cluster genannt, kann durch verschiedene Topologien realisiert werden. Dies ist unter anderem durch die Anzahl der sich im Netzwerk befindenen Knoten bedingt. So sind Punkt-zu-Punkt-Verbindungen, Linien-Topologien wie bei CAN und LIN, aber z.B. auch Stern-Topologien möglich. Es wird in FlexRay-Clustern zwischen aktiven und passiven Topologien unterschieden.

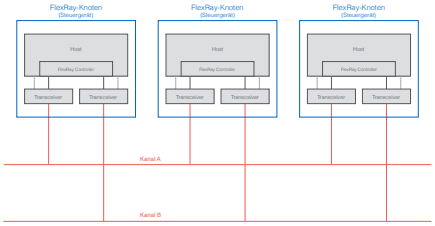

Abbildung 2.9. FlexRay-Cluster mit 3 Knoten in einer passiven Linientopologie (angelehnt an [6])

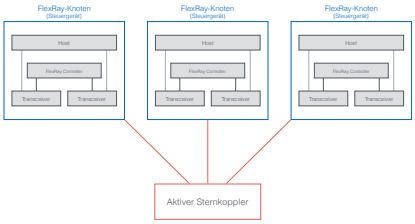

Abbildung 2.10. FlexRay-Cluster mit 3 Knoten mit aktivem Sternkoppler (angelehnt an [6])

Abbildung 2.9 zeigt eine passive Linientopologie mit zwei Kanälen, wohingegen in Abbildung 2.10 eine Sterntopologie mit aktivem Koppler zu sehen ist, der die einzelnen FlexRay-Knoten miteinander verbindet. Dies hat den Vorteil, dass Übertragungsfehler reduziert werden, indem defekte Kommunikationszweige vom restlichen Netzwerk abgetrennt werden können. Eine solche Topologie kann auch mit zwei Kanälen umgesetzt werden, es bedarf dann allerdings eines zweiten Sternkopplers. In diesme Fall steht ein redunanter Kommunikationskanal zur Verfügung, wodurch die Ausfallsicherheit erhöht werden

kann. Es ist ebenso möglich, den zweiten Kanal für eine Verdopplung der maximalen Übertragungsrate auf 20 Mbits/s zu verwenden. Durch das Hintereinanderschalten zweier aktiver Sternkoppler kann die maximale Entfernung zweier FlexRay-Knoten auf 72m im Verleich zu 24m erhöht werden.

2.2.4.3. Kommunikationsprinzip

Die Kommunikationsarchitektur in einem FlexRay-Netzwerk ist zeitgesteuert und basiert auf einer Multi-Master-Architektur. D.h., dass der Ablauf und die Aktivierung von Aktionen statisch und zeitlich fest definiert ist und es keinen, den Kommunikationsablauf zentral koordinierenden, Knoten gibt. Somit kann eine deterministische Datenkommunikation garantiert werden. FlexRay-Knoten senden also nach einem exakt festgelegten zeitlichen Ablaufplan. In diesem Ablaufplan ist für jede Botschaft ein Slot innerhalb eines Kommunikationszyklus festgehalten, sodass der Sendezeitpunkt jeder Botschaft genau bestimmt werden kann. Die Adressierung ist inhaltsbezogen, die Empfänger entscheiden anhand der Frame ID, ob der Inhalt der Botschaft für sie relevant ist oder nicht.
Der Kommunikationsplan kann neben statischen Segmenten auch dynamische Segmente aufweisen. Diese dynamischen Segmente werden für die eireignisgesteuerte / bedarfsorientierte Übertragung von Botschaften und für asynchrone Vorgänge genutzt. Dynamische Segmente sind als eine Erweiterung des Kommunikationszyklus anzusehen, welcher sich dann aus einem statischen und einem dynamischen Segment zusammensetzt.

2.2.4.4. Framing

Das FlexRay-Protokoll definiert eine einheitliche Botschaftsstruktur. Ein Frame (vgl. Abbildung 2.11) setzt sich hier aus den folgenden Teilen zusammen:

- Header
- Payload
- Trailer

Es wird zwar nur ein Botschaftstyp verwendet, jedoch befinden sich zu Beginn des Headers 5 Indikatorbits, die den Typ der zu übertragenden Botschaft näher spezifizieren. Diese sind die Folgenden:

1. Reserved Bit
   Für zukünftige Verwendung reserviert.
2. Payload Preamble Indicator
   Botschaft im statischen / dynamischen Segment: Netzwerkmanagement-Vektor / Message-ID zu Beginn der Payload.
3. Null Frame Indicator
   Wenn gesetzt: Payload enthält gültige Daten.

4. Snyc Frame Indicator
   Wenn gesetzt: Botschaft wird für netzwerkweite Synchronisation verwendet.

5. Startup Frame Indicator
   Wenn gesetzt: Botschaft wird für den Startup-Prozess des Netzwerks benötigt.

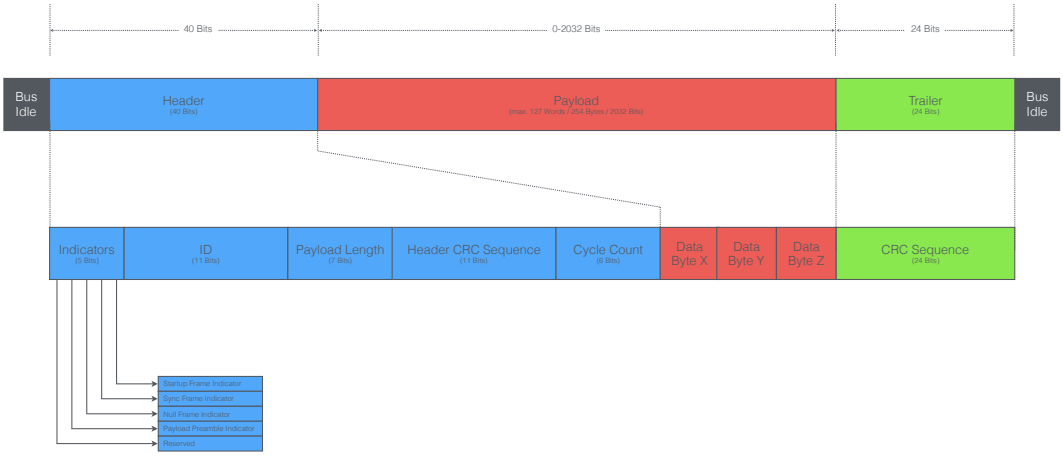

Abbildung 2.11. Interner Aufbau eines FlexRay Frames (angelehnt an [6])

Darauf folgt die ID, welche eine Botschaft einem Slot zuordnet. Die ID 0x00 steht für ungültige Botschaften. Die nächsten 7 Bits geben die Länge der Nutzdaten in Worten an, es können also maximal 254 Nutzdatenbytes übertragen werden. Die ID wird mit dem CRC-Verfahren gesichert. Im Header abschließenden Cycle Counter wird die fortlaufende Nummer modulo 63 des Kommunikationszyklus, in welchem die jeweilige Botschaft gesendet wird, übertragen.
Die Zusammensetzung der Payload hängt davon ab, ob es sich um ein statisches oder dynamisches Segment handelt. Bei einem statischem Segment können die ersten zwölf Nutzdatenbytes für die Übertragung des Network Management Vectors (Realisierung des Netzwerkmanagements) genutzt werden, es muss dann allerdings der Payload Preamble Indicator im Header gesetzt werden. Ist dieser Indikator bei einer dynamischen Botschaft gesetzt, bedeutet dies, dass die ersten beiden Nutzdatenbytes den Message Identifier beinhalten, welcher z.B. für eine feinere Akzeptanzfilterung herangezogen werden kann.
Zur Sicherung der Nutzdaten kommt wieder das CRC-Verfahren zum Einsatz, mit Hilfe wessen eine CRC-Sequenz als Trailer übertragen wird.

2.2.4.5. Datensicherung

In einem FlexRay-Netzwerk werden wie bei CAN und LIN Mechanismen zur logischen Fehlererkennung eingesetzt. In Tabelle 2.11 werden diese erklärt.

Logische Fehlererkennung	
CRC	Ein FlexRay Frame weist 2 CRC Sequenzen auf (vgl. Abbildung 2.11). Die Header CRC Sequence dient zur Sicherung der ihr vorangehenden Teile des Headers. Die im Trailer übertragene CRC Sequenz sichert die Nutzdaten (Payload).
Bus Guardian	Jeder Knoten in einem FlexRay-Netzwerk verfügt über einen sogenannten Bus Guardian (Buswächter). Dieser stellt sicher, dass der zugehörige Knoten nur auf den Bus zugreifen kann, wenn es laut Kommunikationsablaufplan auch vorgesehen ist. So kann ein deterministischer Ablauf der Kommunikation sichergestellt werden.

Tabelle 2.11. Logische Fehlererkennungsmechanismen in einem CAN-Netzwerk

In aktiven Topologien, z.B. einer Topologie mit aktivem Sternkoppler, kann dieser Sternkoppler Fehler in Form von defekten Kommunikationszweigen entdecken und vom restlichen Netzwerk isolieren.
Ebenso kann auch die redundante Auslegegung des physikalischen Übertragungskanals als ein Mittel der Datensicherung angesehen werden, da bei Ausfall eines Kanals nicht das gesamte Netzwerk ausfällt.

2.2.4.6. Kommunikationsbeispiel

Abbildung 2.12 zeigt eine beispielhafte Kommunikation in einem FlexRay-Cluster mit fünf Knoten, welchem eine zweikanalige Linientopologie zugrunde liegt. In der Kommunikationsmatrix werden die Botschaften den entsprechenden Slots zugeordnet sowie einem Kanal zugewiesen. Anhand des Kommunikationsablaufs ist es leicht ersichtlich, wie die Aufteilung in ein statisches und dynamisches Segment sowie die Aufteilung innerhalb des dynamischen Segments erfolgt.

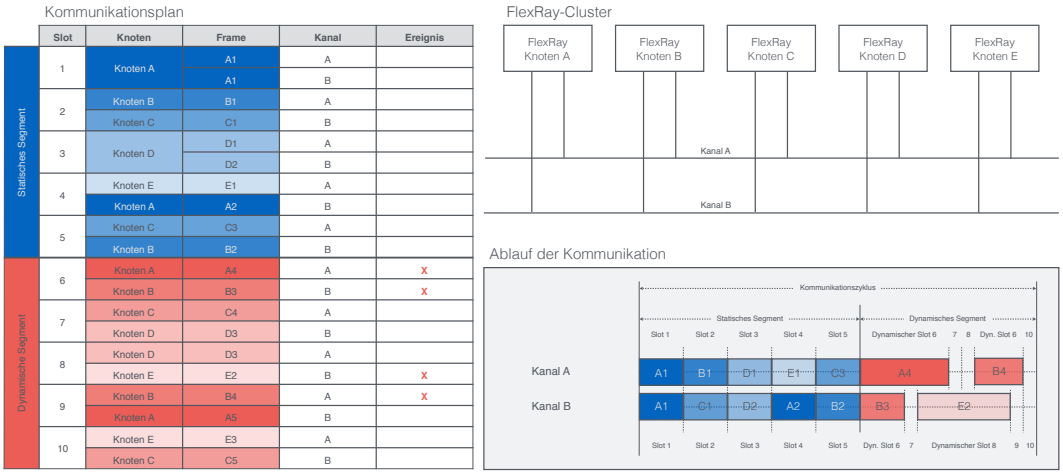

Abbildung 2.12. Beispielhafte Kommunikation in einem FlexRay-Cluster mit fünf Knoten (angelehnt an [6])

2.3. XCP

XCP steht für *Universal Measurement and Calibration Protocol* und ist ein Netzwerkprotokoll. Standardisiert ist XCP derzeit durch den ASAM e.V. [2] als ASAM MCD-1 XCP [3]. Es ist der Nachfolger von CCP (*CAN Calibration Protocol*), bei dessen Entwicklung der lesende und schreibene Zugriff auf steuergeräteinterne Daten mit Hilfe des CAN-Protokolls im Vordergrund stand. Als sich neben CAN auch Bussysteme wie FlexRay und Ethernet etablierten, wurde ein neues Protokoll benötigt, welches die Messung und Manipulation von steuergeräteinternen Daten und Parametern ermöglicht. Das „X“ in XCP steht für eXtended, was in diesem Fall Erweiterung sowie auch Generalisierung bedeutet. Generalisierung insofern, als dass XCP als Zweischichtenprotokoll nun die folgenden Transportschichten definiert:

- XCP-on-CAN
- XCP-on-CAN FD
- XCP-on-FlexRay
- XCP-on-Ethernet
- XCP-on-Sxl
- XCP-on-USB

[2] https://www.asam.net
[3] https://www.asam.net/nc/home/asam-standards.html

Man beachte, dass durch ASAM derzeit kein XCP-on-LIN-Standard definiert wird. Die Vector Informatik GmbH bietet dafür jedoch eine Lösung an, die bereits in einigen Projekten verwendet wird ([7]).

2.3.1. Einsatzgebiet und Funktionen

XCP wird vor allem für die Entwicklung von Steuergeräten verwendet, insbesondere für die Kalibrierung. Ohne XCP muss das Steuergerät als Black-Box betrachtet werden. Der Entwickler kann nicht nachvollziehen, wie sich die Algorithmen in der Steuergeräte-Software verhalten. Durch XCP erhält der Entwickler jedoch zur Laufzeit einen Einblick in Variablen, Parameter und Speicherinhalte. Dies geschieht dabei eventsynchron zu steuergeräteinternen Vorgängen oder auch durch einzelne Anfragen. So können z.B. Schwachstellen in Algorithmen entdeckt oder die Parametrisierung optimiert werden. Das Steuergerät wird somit also zur Glass-Box.
Im Rahmen dieser Arbeit ist vor allem die Möglichkeit der Messung von steuergeräteinternen Daten von Nutzem. Somit können z.B. einzelne Variablen auf korrekte Werte und damit auch indirekt die Fehlerkennungmechanismen der Vernetzungssoftware überprüft werden.

2.4. OBD

OBD steht für *On-Board-Diagnose* und ist ein fahrzeugeigenes System zur Diagnose von während des Fahrbetriebs auftretenden Fehlern v.a. im Zusammenhang mit der Abgas-Steuerung. Zu Beginn war OBD noch nicht standardisiert und es gab teils sogar Unterschiede zwischen verschiedenen Modellen des gleichen Herstellers. Mit der Normierung durch die folgenden Normen, besteht mit OBD-2 nun ein standardisiertes Diagnosesystem, das über den OBD-2-Stecker, welcher in jedem Fahrzeug vorgeschrieben ist, mit der passenden Software ausgelesen werden kann (z.B. in der Werkstatt). Mit dem OBD-System ist z.B. die Motorkontrollleuchte verbunden [16].

2.5. Testumgebungen

Der Vernetzungstest von Steuergeräten wird in verschiedenen Testumgebungen durchgeführt, da je nach Testkategorie unterschiedliche Anforderungen an die Testumgebung existieren. Diese Testumgebungen werden in den folgenden Abschnitten vorgestellt.

2.5.1. Im Fahrzeug

In einem mit Messtechnik ausgestatteten Testfahrzeug kann der Vernetzungstest unter realen Bedingungen durchgeführt werden. Die Kommunikation zwischen den Steuergeräten findet unter den gleichen Bedingungen wie in einem ausgelieferten Kundenfahrzeug statt. Dem Realitätsgrad stehen der vergleichsweise hohe Aufwand für die Vorbereitung und Durchführung sowie der Kostenfaktor als Nachteil gegenüber.

2.5.2. HIL

Hardware-in-the-Loop-Simulatoren wurden hauptsächlich entwickelt, um Kosten beim Test von Steuergeräten einsparen und die Testumfänge erhöhen zu können. Mit einem HIL-Simulator kann ein einzelnes Steuergerät mit Hilfe einer modellbasierten Simulation des restlichen Netzwerks getestet werden. Es ist dabei ausreichend Rechenleistung verfügbar, um den Test in Echtzeit durchführen zu können. Es können Szenarien wie z.B. ein Motorstart simuliert werden.

2.5.3. Tischaufbau

Im Rahmen dieser Arbeit ist die Testumgebung „Tischaufbau“ am interessantesten. Hier wird ein einzelnes Steuergerät mit Hilfe einer softwarebasierten Restbussimulation getestet. Das Automatisierungspotential ist hier am höchsten. Darüberhinaus sind die Kosten und der Aufwand bei Verwendung des *Network-Test-Generators*, welcher im Rahmen dieser Arbeit entwickelt wird, sehr gering. Im Folgenden wird der Testaufbau wie in Abbildung 2.13 dargestellt, erklärt.
Das zu testende Steuergerät ist über eine USB-Schnittstelle mit einem Computer verbunden und an eine externe Stromquelle angeschlossen. Zusärtlich kann über eine mit dem Steuergerät verbundene Schalterbox die Zündung ein- und ausgeschaltet werden.
Die Kommunikation mit dem Steuergerät erfolgt über CANoe. CANoe ist ein von der Vector Informatik GmbH entwickeltes Tool für die Entwicklung und den Test von Steuergeräten. Mit Hilfe von CANoe können u.a. Signale manipuliert und gemessen werden. Es werden dabei alle im Antriebsstrang zum Einsatz kommenden Bussysteme (Protokolle) unterstützt. Als Input dient für CANoe eine sogenannte CANoe-Konfiguration. Diese bietet die Basis für die Restbussimulation, welche das gesamte Netzwerk (oder auch mehrere Netzwerke) exklusive des (realen) zu testenden Steuergeräts simuliert.

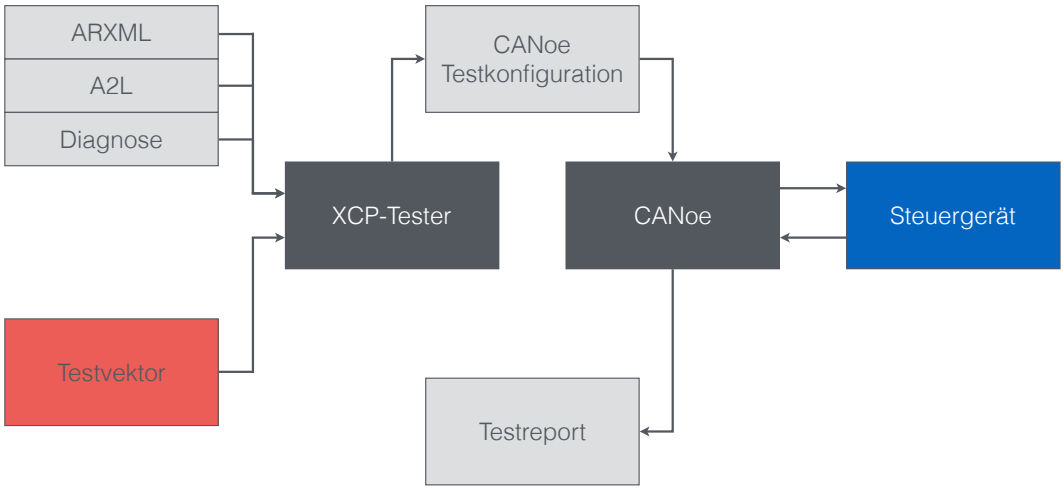

Abbildung 2.13. Toolkette für den Vernetzungstest am Tischaufbau

Die jeweilige CANoe-Konfiguration wird vom XCP-Tester erzeugt. Dieses Tool bekommt folgenden Input:

- Eine / oder mehrere Netzwerkbeschreibung(en) in einem von AUTOSAR definierten Schema (.arxml-Datei).
- Eine .a2l-Datei, welche für steuergeräteinterne Datenobjekte Informationen wie z.B. die Speicheradresse, Datentyp, Umrechnungsvorschriften in physikalische Größen und Ablagestruktur enthält.
- Eine .cdd-Datei, welche eine Beschreibung der internen Diagnose-Struktur enthält.
- Eine Beschreibung der durchzuführenden Testfälle in Form einer .xml-Datei (Testvektor) basierend auf dem Testkonzept, das in Kapitel 3 vorgestellt wird.

Die ersten drei Dateien stehen für jedes Steuergerät und jeden Softwarestand zur Verfügung. Für die Generierung des Testvektoren wird das im Rahmen dieser Arbeit entwickelte Tool *Network-Test-Generator* verwendet. Als Input für den *Network-Test-Generator* dienen die folgenden Dateien (vgl. Abbildung 2.14):

- Eine / oder mehrere Netzwerkbeschreibung(en) (.nwmx-Datei(en)).
- Eine .dcm-Datei, welche sämtliche Bedatungsgrößen für einen Softwarestand enthält.

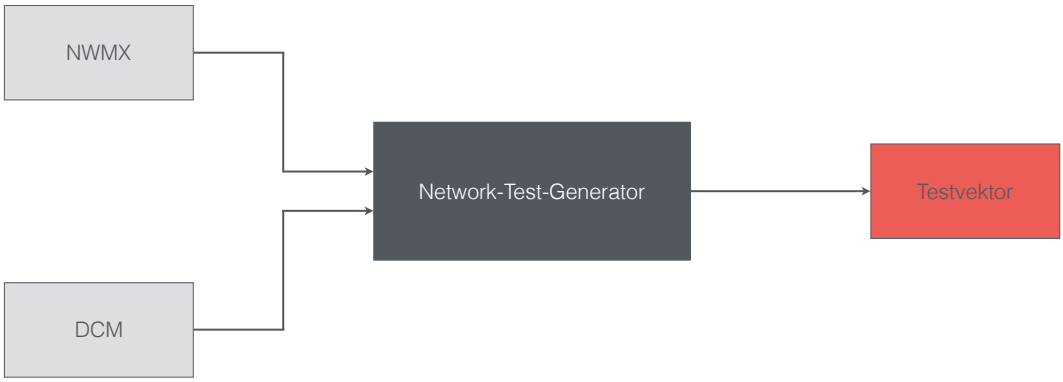

Abbildung 2.14. Testvektor-Generierung mit dem *Network-Test-Generator*

Die Funktionsweise des *Network-Test-Generator* wird in Kapitel 4 detailliert beschrieben.

Kapitel 3

Testkonzept für den Vernetzungstest von Steuergeräten

In diesem Kapitel wird das Testkonzept für den Vernetzungstest der Steuergeräte im Antriebsstrang vorgestellt. Dazu werden als erstes die Testkategorien für den Vernetzungstest aufgelistet. Diese Testkategorien werden auf Basis des Funktionslastenheft Vernetzung erarbeitet. In diesem Lastenheft sind die Anforderungen an die Vernetzungs-Funktionen, welche die Kommunikation zwischen Anwendungssoftware auf über ein Netzwerk verbundenen Steuergeräten ermöglichen, festgehalten. Anschließend werden die Testkategorien in die in Tabelle 3.1 beschriebenen Gruppen unterteilt:

Bewertung der Testkategorien	
Am Tischaufbau automatisiert durchführbar	Testfälle dieser Kategorie können mit der aktuell verfügbaren Toolkette (vgl. Kapitel 4) automatisiert am Tischaufbau durchgeführt werden.
Am Tischaufbau manuell durchführbar	Testfälle dieser Kategorie können theoretisch manuell am Tischaufbau durchgeführt werden, sind jedoch entweder bereits durch eine andere Toolkette realisiert oder es fehlt die notwendige Hardware und die Unterstützung seitens des XCP-Testers (vgl. 2.5.3).
Nicht am Tischaufbau durchführbar	Testfälle dieser Kategorie können nicht am Tischaufbau durchgeführt werden, müssen also im Fahrzeug oder am HIL durchgeführt werden.

Tabelle 3.1. Bewertung der Testkategorien hinsichtlich (automatisierter) Durchführbarkeit am Tischaufbau

Für die am Tischaufbau automatisiert durchführbaren Testkategorien wird der Test-Ablauf mit allen Testschritten detailliert erörtert. Das zu entwickelnde Tool, welches in Kapitel 4 vorgestellt wird, unterstützt die Testfall-Generierung für eben diese Testkategorien.

3.1. Testkategorien für den Vernetzungstest

Die Testkategorien für den Vernetzungstest müssen die Anwendungsfälle der Vernetzungs-Funktionen abdecken. Diese lassen sich wie folgt kategorisieren.

Anwendungsfälle der Vernetzungs-Funktionen	
Anwendungssignale übertragen	Die Vernetzungssoftware muss die Signale vom Netzwerk (Empfangsrichtung) und an das Netzwerk (Senderichtung) fehlerfrei bereitstellen, d.h. gesendete Werte dürfen während der Signalübertragung nicht verfälscht werden.
Botschaftsrouting & Signalrouting	Die Vernetzungssoftware muss Botschaften und Signale, die von Steuergerät A in Netzwerk X an Steuergerät B in Netzwerk Y gesendet werden, über den korrekten Routingpfad fehlerfrei übertragen.
Fehlererkennung	Die Vernetzungssoftware muss Fehler, die während der Kommunikation im Netzwerk entstehen, erkennen.

Tabelle 3.2. Anwendungsfälle der Vernetzungs-Funktionen

Im Folgenden werden für die in Tabelle 3.2 genannten Anwendungsfälle jeweils die notwendigen Test-Kategorien vorgestellt und hinsichtlich der in Tabelle 3.1 aufgeführten Gruppen bewertet.

3.1.1. Anwendungssignale übertragen

Die Vernetzungssoftware muss Anwendungssignale sowohl empfangsseitig (Signale vom Netzwerk dem Steuergerät bereitstellen) als auch sendeseitig (Signale vom Steuergerät an das Netzwerk bereitstellen) übertragen. Um diese funktionale Anforderung im Vernetzungstest abdecken zu können, werden die folgenden Testkategorien benötigt.

3.1.1.1. Empfangssignaltest

Mit diesem Test soll das Bereitstellen von Anwendungssignalen vom Netzwerk an das Steuergerät getestet werden. Der Empfangssignaltest ist mit dem in Kapitel 2.5.3 beschriebenen Testaufbau durchführbar.
Die Menge der zu prüfenden Signale entspricht allen externen Signalen (Bussignalen), für die ein internes Signal (Applikationssignal) im zu testenden Steuergerät definiert ist. Das externe Signal muss für den Test manipuliert werden können, um anschließend durch eine Messung des internen Signals und einem Wertevergleich feststellen zu können, ob das Signal über dem gesamten Wertebreich korrekt empfangen und eingelesen wurde.
Um die empfangsseitige Signalübertragung auf Korrektheit prüfen zu können, müssen die folgende Aspekte im Hinblick auf die Auswahl der Testfälle berücksichtigt werden.

- **Datentyp**:
  - Bit: das Signal bildet einen bool'schen Wert ab und hat Länge $n = 1$, z.B. ist die Fahrertür geschlossen oder offen.
  - Zustand: das Signal hat eine fest definierte Menge von vordefinierten Werten und Länge n, z.B. der Zündstatus.
  - Physikalisches Muster: das Signal hat einen festen Wertebereich, eine Skalierung (siehe unten), eine physikalische Einheit und Länge n, z.B. die Öltemperatur.
- **Skalierung**: Signale können auf dem Bus ausschließlich binär übertragen werden. Die binär übertragenen Werte werden auch als Rohwerte bezeichnet. Der Wertebereich eines Rohwertes ist durch die Anzahl der für das Signal in der Botschaft reservierten Bits gegeben. Sind für ein Signal beispielsweise n Bits vorgesehen, gilt für einen Rohwert x_{Raw}:
$$x_{Raw} \in \{0, 1, ..., 2^n - 1\} \subset \mathbb{N}$$
Um diese Rohwerte in physikalische Werte umrechnen zu können, bedarf es einer Umrechnungsvorschrift. Diese ist durch die Skalierung
$$Scal = (F, O), \quad F, O \in \mathbb{Q}$$
eines Signals definiert. Eine Skalierung wird durch ein 2-Tupel (F, O) bestehend aus einem Faktor $F \in \mathbb{Q}$ und einem Offset $O \in \mathbb{Q}$ definiert und kann als eine bijektive Abbildung von der Menge der Rohwerte R_{Raw} in die Menge der physikalischen Werte $R_{Physical}$ angesehen werden:
$$f_{Scal} : R_{Raw} \to R_{Physical}, x \mapsto F \cdot x + O$$
Für Bit- und Zustandssignale gilt immer:
$$F = 1 \wedge O = 0$$
Die Werte werden folglich eins-zu-eins abgebildet. Für Signale mit physikalischem Muster können der Faktor und der Offset beliebig definiert werden, sodass beispiels-

weise auch negative Werte möglich werden (z.B. für die Temperaturanzeige). Die Skalierung kann für das externe und interne Signal jeweils unterschiedlich definiert sein. Die Vernetzungssoftware muss ggf. eine Umrechnung durchführen, falls sich die externe Skalierung $Scal_{Ext}$ von der internen Skalierung $Scal_{Int}$ unterscheidet. Mit der internen Skalierung nicht darstellbare Werte werden dabei immer auf den nächst kleineren Wert abgerundet. Um zwei Skalierungen vergleichen zu können, definiert man die Gleichheitsrealtion $=$ und die Äquivalenzrelation $\equiv$ auf der Menge M_{Scal} aller möglichen Skalierungen wie folgt:

- Zwei Skalierungen $Scal_1$ und $Scal_2$ heißen faktorgleich, falls ihre Faktoren identisch sind. Es gilt:
$$Scal_1 =_F Scal_2 \Leftrightarrow F_1 = F_2$$
- Zwei Skalierungen $Scal_1$ und $Scal_2$ heißen offsetgleich, falls ihre Offsets identisch sind. Es gilt:
$$Scal_1 =_O Scal_2 \Leftrightarrow O_1 = O_2$$
- Zwei Skalierungen $Scal_1$ und $Scal_2$ heißen gleich, falls sie faktor- und offsetgleich sind. Es gilt:
$$Scal_1 = Scal_2 \Leftrightarrow Scal_1 =_F Scal_2 \wedge Scal_1 =_O Scal_2 \Leftrightarrow F_1 = F_2 \wedge O_1 = O_2$$

Nun kann man für einen Rohwert x_{Raw} den physikalischen Wert x_{Phys} wie folgt bestimmen:
$$x_{Phys} = f_{Scal}(x_{Raw})$$

Im Folgenden wird x_{Phys} mit x abgekürzt, handelt es sich um einen Rohwert, schreibt man x_{Raw}.

Ein Wert x heißt darstellbar mit der Skalierung $Scal$, falls gilt:
$$(x - O_{Scal}) \mod F_{Scal} = 0$$

- Möglicher (physikalischer) **Wertebereich** R:
  - Bit: $R = \{0, 1\}$
  - Zustand: $R = \{SV_0,\ SV_1,\ ..., SV_i\}, \quad i \in \mathbb{N}$
  - Physikalisches Muster: $R \in \{f_{Scal}(x_0),\ f_{Scal}(x_1), ..., f_{Scal}(x_{n^2-1})\}$

  Die Wertebereiche des externen (Bus) und des internen (Steuergerät) Signals R_{Ext} und R_{Int} müssen theoretisch gesehen nicht identisch sein. Ist der Wertebereich des internen Signals größer als der des externen Signals, muss die Vernetzungssoftware eine Begrenzung des internen Wertebereichs durchführen. Analog gilt für den umgekehrten Fall, dass die Vernetzungssoftware keinen ungültigen Wert $x \notin R_{Int}$ einlesen darf. Das interne Signal bleibt in diesen Fall auf dem letzten gültigen Wert stehen.

  Die in den folgenden Abbildungen (3.1, (3.2, 3.3, 3.4, 3.5) gezeigten Konstellationen können dabei auftreten. Der blau hinterlegte Bereich stellt den (absolut) gültigen Wertebereich R dar.

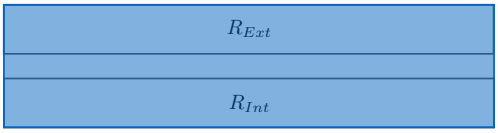

Abbildung 3.1. $R = R_{Ext} = R_{Int} : Min_{Ext} = Min_{Int} < Max_{Int} = Max_{Ext}$

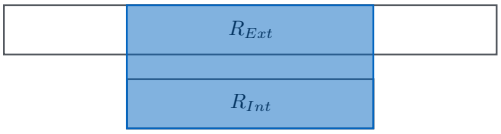

Abbildung 3.2. $R = R_{Ext} \supseteq R_{Int} : Min_{Ext} \leqslant Min_{Int} < Max_{Int} \leqslant Max_{Ext}$

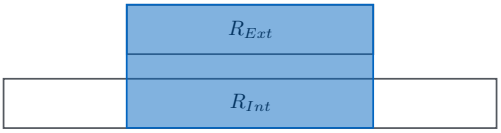

Abbildung 3.3. $R = R_{Ext} \supseteq R_{Int} : Min_{Int} \leqslant Min_{Ext} < Max_{Ext} \leqslant Max_{Int}$

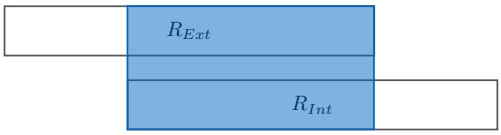

Abbildung 3.4. $R \subset R_{Ext} \subset R_{Int} : Min_{Ext} < Min_{Int} < Max_{Ext} < Max_{Int}$

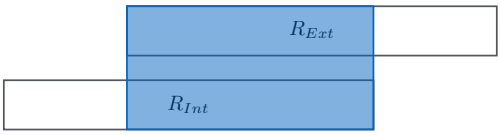

Abbildung 3.5. $R \subset R_{Ext} \subset R_{Int} : Min_{Int} < Min_{Ext} < Max_{Int} < Max_{Ext}$

Aus dem Wertebereich eines Signals lassen sich auf das jeweilige Minimum und Maximum ablesen.

- **Initialisierungswert(e)**: für das externe und interne Signal ist jeweils ein Initialisierungswert in der jeweiligen Kommunikationsmatrix definiert. Diese können, müssen aber nicht identisch sein, so kann beispielsweise der externe Initialisierungswert außerhalb des gültigen Wertebereichs des internen Signals liegen.
- **SNA-Wert**: Für ein externes Signal kann ein sogenannter SNA-Wert (engl. signal not available) definiert werden. Dieser wird verwendet, um zu signalisieren, dass

ein Signal (bzw. ein Wert) nicht verfügbar ist. Der SNA-Wert kann ggf. gleich dem Initialisierungwert sein.

Auf dieser Basis lassen sich nun die zu testenden Werte ableiten. Für jeden dieser Werte muss ein Testfall durchgeführt werden. Ein Testfall beinhaltet dabei die Manipulation (das Setzen des Setzwerts x_{Set}) des Signals auf dem Bus und die anschließende Messung des internen Signals sowie der Vergleich des Sollwerts x_{Ev} mit dem gemessenen Istwert x_{Ac} Die Unterscheidung zwischen dem Setzwert x_{Set} und Sollwert x_{Ev} ist notwendig, da diese nicht immer identisch sind, wenn z.B. das Maximum des externen Signals größer als das interne Maximum ist, darf (bzw. kann) dieser Wert von der Vernetzungssoftware nicht eingelesen werden - es muss also das interne Maximum abgefragt werden.

Für Signale vom Datentyp **Bit** werden die folgenden Werte getestet:

1. **Nicht gesetztes Bit**:
$$x_{Set} = 0, \quad x_{Ev} = \begin{cases} 0 & \text{, falls } SNA = 1 \\ Init_{Int} & \text{, sonst} \end{cases}$$
2. **Gesetztes Bit**:
$$x_{Set} = 1, \quad x_{Ev} = \begin{cases} 1 & \text{, falls } SNA = 0 \\ 0 & \text{, sonst} \end{cases}$$

Für Signale vom Datentyp **Zustand** wird jeder mögliche Wert SV_i aus der Menge der definierten Zustandswerte SV mit Länge $n = |SV| > 0$ getestet. Die Setzwerte x_{set_0}, x_{set_1}, ..., $x_{set_{n-1}}$ lassen sich wie folgt berechnen:

$$\forall i \in \{0, 1, ..., n-1\} : x_{set_i} = SV_i$$

Die zugehörigen Sollwerte x_{ev_0}, x_{ev_1}, ..., $x_{ev_{n-1}}$ lassen sich wie folgt berechnen:

$$\forall i \in \{0, 1, ..., n-1\} : x_{ev_i} = \begin{cases} Init_{Int} & \text{, falls } SV_i = SNA \wedge i = 0 \\ SV_{i-1} & \text{, } fallsSV_i = SNA \wedge i > 0 \\ SV_i & \text{, sonst} \end{cases}$$

Für Signals mit einem physikalisch Muster müssen die folgenden Werte getestet werden.

1. **Externer Initialisierungswert**: für jedes externe Signal ist ein Initialisierungswert in der jeweiligen Kommunikationsmatrix definiert. In diesem Testfall wird also überprüft, ob das externe Signal vom entsprechenden Steuergerät korrekt gesendet wird. Dieser Testfall ist dabei ein Sonderfall. Das externe Signal wird hier nicht manipuliert, es wird lediglich gemessen. Für den Sollwert gilt:
$$x_{Ev} = Init_{Ext}$$
2. **Interner Initialisierungswert**: auch für interne Signale ist jeweils ein Initialisierungswert (intern) definiert. In diesem Testfall wird also überprüft, ob das interne Signal

korrekt initialisiert wird. Dieser Testfall ist dabei ein Sonderfall. Es wird keine Manipulation des externen Signals vorgenommen - sondern lediglich das interne Signal gemessen. Bei der Berechnung des Sollwerts x_{Ev} sind die folgenden Größen zu beachten (vgl. Abbildungen 3.2 - 3.5):

- $Init_{Ext}$ - Initialisierungswert des externen Signals
- SNA - SNA-Wert des externen Signals
- Min_{Int} - Minimum des internen Signals
- Max_{Int} - Maxmium des internen Signals
- $Init_{Int}$ - Initialisierungswert des internen Signals

Es sind die folgenden Fälle zu unterscheiden:

$$x_{Ev} = \begin{cases} f_{Scal_{Int}}\left(\left\lfloor \frac{InitExt - O_{Scal_{Int}}}{F_{Scal_{Int}}} \right\rfloor\right) & \text{, falls } Min_{Int} \leqslant Init_{Ext} \leqslant Max_{Int} \wedge Init_{Ext} \neq SNA \\ Init_{Int} & \text{, falls } Min_{Int} \leqslant Init_{Ext} \leqslant Max_{Int} \wedge Init_{Ext} = SNA \\ Init_{Int} & \text{, falls } Init_{Ext} < Min_{Int} \wedge Max_{Int} < Init_{Ext} \end{cases}$$

Tritt der erste der 3 beschriebenen Fälle ein, kann die Initialisierung des internen Signals nicht auf Korrektheit geprüft werden, da die Vernetzungssoftware einen (gemäß des internen Wertebereichs R_{Int}) gültigen Wert feststellt, diesen einliest und infolgedessen den internen Initialisierungswert überschreibt. Da der XCP-Verbindungsaufbau mehr Zeit als eben jene Überschreibung in Anspruch nimmt, kann der interne Initialisierungswert in diesem Fall nicht gemessen werden. Um auch in diesem Fall die interne Initialisierung testen zu können, müsste eine gesonderte Kommunikationsmatrix zu Verfügung stehen, die jedes externe Signals mit einem gemäß des internen Wertebereichs ungültigen Wert initialisiert.

3. **Minimum**: um den absolut minimalen gültigen Wert testen zu können, müssen zuerst die folgenden Größen unterschieden werden:

   - Min_{Ext} - Minimum des Wertebereichs des externen Signals
   - Min_{Int} - Minimum des Wertebereichs des internen Signals
   - $Limit_{Lower}$ - Optionale zusätzliche untere Grenze des Wertebereichs des internen Signals, z.B. bei Signalen, die den Datentyp „Prozent“ haben (Begrenzung auf $0 - 100$).

   Das absolut gültige Minimum Min wird dann wie folgt berechnet:

   $$Min = max\{Min_{Ext},\ Min_{Int},\ Limit_{Lower}\}$$

Der Setzwert x_{Set} lässt sich wie folgt berechnen:

$$x_{Set} = \begin{cases} Min & \text{, falls } Min = Min_{Ext} \vee Scal_{Ext} = Scal_{Int} \\ f_{Scal_{Ext}}\left(\left\lceil \frac{Min_{Int} - O_{Scal_{Ext}}}{F_{Scal_{Ext}}} \right\rceil\right) & \text{, sonst} \end{cases}$$

Im Folgenden werden zwei Beispiele gezeigt, die den zweiten Fall (das absolute Minimum ist das Minimum des internen Signals oder die untere Grenze des internen Wertebereichs) für die Berechnung von x_{Set} veranschaulichen. Es ist ersichtlich, dass es keine Rolle spielt, welches Signal eine höhere Präzision (kleineren Skalierungsfaktor) hat. Man beachte, dass mit der internen Skalierung nicht darstellbare Werte von der Vernetzungssoftware immer abgerundet werden.

1. Das externe Signal hat eine geringere Präzision als das interne Signal (vgl. Abbilung 3.6):

$$Scal_{Ext} = (7, -4), \quad Scal_{Int} = (4, 0), \quad Min = Min_{Int} = 0$$

$$x_{Set} = f_{Scal_{Ext}}\left(\left\lceil \frac{0 - (-4)}{7} \right\rceil\right) = f_{Scal_{Ext}}\left(\left\lceil \frac{4}{7} \right\rceil\right) = f_{Scal_{Ext}}(1) = 3$$

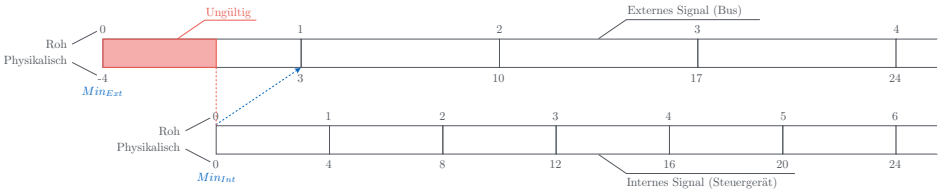

Abbildung 3.6. Berechnung von x_{Set}, falls $F_{Scal_{Ext}} > F_{Scal_{Int}}$

2. Das externe Signal hat eine höhere Präzision als das interne Signal (vgl. Abbilung 3.7):

$$Scal_{Ext} = (4, -4), \quad Scal_{Int} = (7, -2), \quad Min = Min_{Int} = -2$$

$$x_{Set} = f_{Scal_{Ext}}\left(\left\lceil \frac{-2 - (-4)}{4} \right\rceil\right) = f_{Scal_{Ext}}\left(\left\lceil \frac{2}{4} \right\rceil\right) = f_{Scal_{Ext}}(1) = 0$$

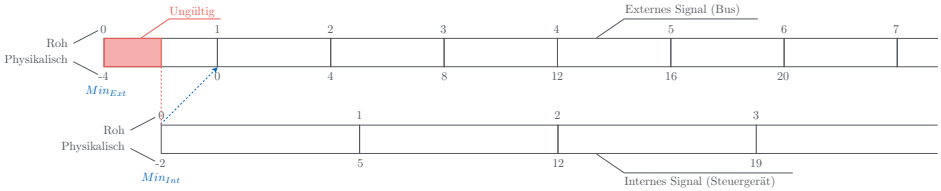

Abbildung 3.7. Berechnung von x_{Set}, falls $F_{Scal_{Ext}} < F_{Scal_{Int}}$

Der Sollwert x_{Ev} lässt sich wie folgt berechnen:

$$x_{Ev} = \begin{cases} f_{Scal_{Int}}\left(\left\lfloor \frac{Min_{Ext} - O_{Scal_{Int}}}{F_{Scal_{Int}}} \right\rfloor\right) & \text{, falls } Min = Min_{Ext} \wedge Scal_{Ext} \neq_{R_{Scal}} Scal_{Int} \\ Min & \text{, sonst} \end{cases}$$

Auch für den Sollwert x_{Ev} werden im Folgenden 2 Beispiele für die Berechnung im ersten Fall gezeigt.

1. Das externe Signal hat eine geringere Präzision als das interne Signal (vgl. Abbilung 3.8):

$$Scal_{Ext} = (7,2), \quad Scal_{Int} = (4,0), \quad Min = Min_{Ext} = 2$$

$$x_{Ev} = f_{Scal_{Ext}}\left(\left\lfloor \frac{2-0}{4} \right\rfloor\right) = f_{Scal_{Ext}}\left(\left\lfloor \frac{2}{4} \right\rfloor\right) = f_{Scal_{Ext}}(0) = 0$$

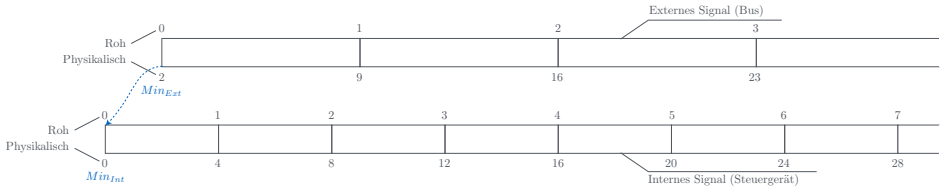

Abbildung 3.8. Berechnung von x_{Ev}, falls $F_{Scal_{Ext}} > F_{Scal_{Int}}$

2. Das externe Signal hat eine höhere Präzision als das interne Signal (vgl. Abbilung 3.9):

$$Scal_{Ext} = (4,6), \quad Scal_{Int} = (7,-2), \quad Min = Min_{Ext} = 6$$

$$x_{Ev} = f_{Scal_{Ext}}\left(\left\lfloor \frac{6-(-2)}{7} \right\rfloor\right) = f_{Scal_{Ext}}\left(\left\lfloor \frac{8}{7} \right\rfloor\right) = f_{Scal_{Ext}}(1) = 5$$

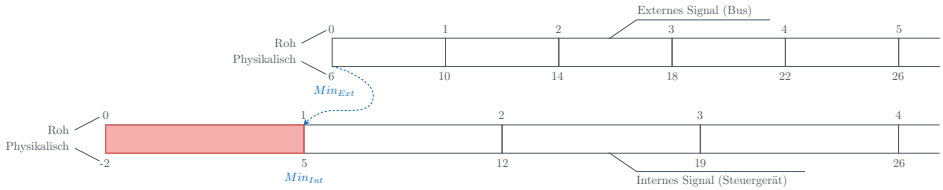

Abbildung 3.9. Berechnung von x_{Ev}, falls $F_{Scal_{Ext}} < F_{Scal_{Int}}$

4. **Externes Minimum**: um den Test des Minimums zu vervollständigen, muss noch ein weiterer Testfall durchgeführt werden. Falls gilt:

$$Min = Min_{Int} \vee Min = Limit_{Lower}$$

wird das externe Minimum nicht getestet. Der Setzwert x_{Set} und der Sollwert x_{Ev} lassen sich wie folgt berechnen:

$$x_{Set} = Min_{Ext}, \quad x_{Ev} = Min$$

Damit wird getestet, ob der letzte gültige Wert des internen Signals (hier: das absolut gültige Minimum) nicht von der Vernetzungssoftware überschrieben wird, da ein gemäß des Wertebereichs des internen Signals nicht darstellbarer und damit ungültiger Wert empfangen wird. Dieser Testfall deckt auch den Test des Minimums des externen Signals ab.

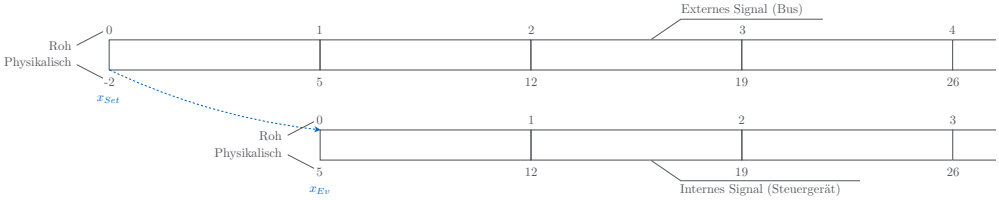

Abbildung 3.10. Test des externen Maximum, falls $Min = Min_{Int} \vee Min = Limit_{Lower}$

Abbildung 3.10 veranschaulicht diesen Fall. Man vergleiche auch Abbildung 3.6 und 3.7.

5. **Maximum**: das Maximum wird analog zum in Testfall 3 beschrieben Minimum berechnet.

6. **Externes Minimum**: auch dieser Testfall erfolgt analog zu Testfall 4.

7. **Sprungweite**: für einen Wertebereich $R = \{x_0, x_1, ..., x_n\}$ berechnet sich die Sprungweite s_R wie folgt:
$$s_R = x_i - x_{i-1} \text{ mit } 0 < i \leqslant n$$
Die Sprungweite wird getestet, um die korrekte Auflösung der Signale (gegeben durch ihre Skalierung) überprüfen zu können.

   7.1 **Minimum + 1x Sprungweite**. Der Setzwert x_{Set} und der Sollwert x_{Ev} lassen sich wie folgt berechnen:

$$x_{Set} = x_{Ev} = \begin{cases} f_{Scal_{Ext}}\left(\left\lceil \frac{Min_{Int} - O_{Scal_{Ext}}}{F_{Scal_{Ext}}} \right\rceil\right) + s_{R_{Ext}} & \text{, falls } Min \neq Min_{Ext} \\ Min + s_{R_{Ext}} & \text{, sonst} \end{cases}$$

7.2 **Minimum + 2x Sprungweite**. Der Setzwert x_{Set} und der Sollwert x_{Ev} lassen sich wie folgt berechnen:

$$x_{Set} = x_{Ev} = \begin{cases} f_{Scal_{Ext}}\left(\left\lceil \frac{Min_{Int} - O_{Scal_{Ext}}}{F_{Scal_{Ext}}} \right\rceil\right) + 2 \cdot s_{R_{Ext}} & \text{, falls } Min \neq Min_{Ext} \\ Min + 2 \cdot s_{R_{Ext}} & \text{, sonst} \end{cases}$$

7.3 **Maximum - 1x Sprungweite**. Der Setzwert x_{Set} und der Sollwert x_{Ev} lassen sich wie folgt berechnen:

$$x_{Set} = x_{Ev} = \begin{cases} f_{Scal_{Ext}}\left(\left\lfloor \frac{Max_{Int} - O_{Scal_{Ext}}}{F_{Scal_{Ext}}} \right\rfloor\right) - s_{R_{Ext}} & \text{, falls } Max \neq Max_{Ext} \\ Max - s_{R_{Ext}} & \text{, sonst} \end{cases}$$

7.4 **Maximum - 2x Sprungweite**. Der Setzwert x_{Set} und der Sollwert x_{Ev} lassen sich wie folgt berechnen:

$$x_{Set} = x_{Ev} = \begin{cases} f_{Scal_{Ext}}\left(\left\lfloor \frac{Max_{Int} - O_{Scal_{Ext}}}{F_{Scal_{Ext}}} \right\rfloor\right) - 2 \cdot s_{R_{Ext}} & \text{, falls } Max \neq Max_{Ext} \\ Max - 2 \cdot s_{R_{Ext}} & \text{, sonst} \end{cases}$$

Bei der Berechnung dieser Werte sind wiederum die Skalierungen des externen und internen Signals zu beachten. Die Sprungweiten sind genau dann verschieden, wenn die beiden Skalierungen nicht faktorgleich sind, da der Skalierungsfaktor die Sprungweite bestimmt. Es gilt also:

$$s_{R_{Ext}} \neq s_{R_{Int}} \Leftrightarrow Scal_{Ext} \neq_F Scal_{Int}$$

Tritt dieser Fall ein, ist der Setzwert x_{Set} ggf. nicht mit der internen Skalierung darstellbar. Im Gegensatz zum Test des Minimums und Maximums wird der Sollwert x_{Ev} hier jedoch nicht gesondert berechnet. Stattdessen wird eine Toleranz für die Evaluierung des gemessenen Werts spezifiziert. Bei der Berechnung dieser Toleranz spielt es keine Rolle, ob die externe oder interne Skalierung eine höhere Präzision (kleinere Sprungweite) als die jeweils andere aufweist (vgl. Abbildung 3.11 und 3.12). Da nicht darstellbare Werte von der Vernetzungssoftware immer abgerundet werden, ist diese Toleranz ausschließlich für Abweichungen nach unten zu verwenden. Damit immer (genau) ein interner Wert innerhalb der Toleranz liegt, wird die Toleranz t wie folgt berechnet:

$$t = s_{R_{Int}}$$

Für den Vergleich von Sollwert x_{Ev} und gemessenem Wert x_{Ac} gilt:

$$\text{Ergebnis positiv} \Leftrightarrow (x_{Ev} - t) \leqslant x_{Ac} \leqslant x_{Ev}$$

Mit der Methode der Spezfikation einer Toleranz ist es besonders einfach, den Test auf

andere (z.B. neue) Versionen der Vernetzungssoftware, die ggf. alle nicht darstellbaren Werte aufrunden, anzupassen. Die Toleranz wäre in diesem Fall als eine Toleranz nach oben zu verstehen. Für den Vergleich von Sollwert x_{Ev} und gemessenem Wert x_{Ac} würde in diesem Fall dann gelten:

$$\text{Ergebnis positiv} \Leftrightarrow (x_{Ev} + t) \leqslant x_{Ac} \leqslant x_{Ev}$$

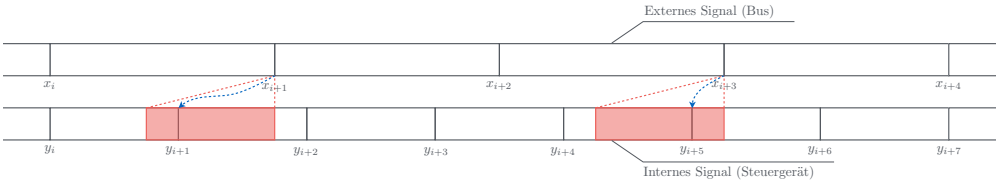

Abbildung 3.11. Toleranz t, falls das externe Signal präziser ist

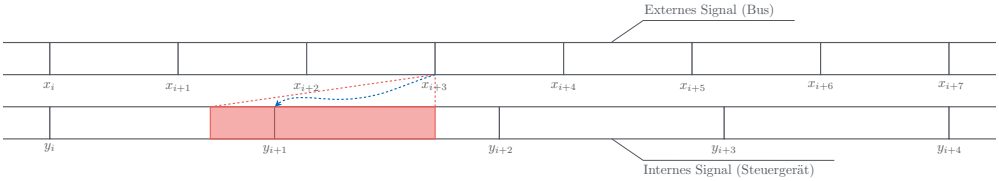

Abbildung 3.12. Toleranz t, falls das interne Signal präziser ist

8. **SNA-Wert**. Unabhängig davon, ob der SNA-Wert indirekt beim Test des Initialisierungswerts (1.) mit getestet wurde, wird noch ein weiterer Testfall für den SNA-Wert benötigt. Es muss der Fall abgedeckt werden, dass ein gültiger Wert gesetzt wurde, anschließend ein ungültiger Wert gesetzt wird (der SNA-Wert), die Vernetzungssoftware diesen ungültigen Wert erkennt und den letzten gültigen Wert nicht überschreibt. Für den Setzwert x_{Set} gilt dann:

   $$x_{Set} = SNA$$

   Der Sollwert x_{Ev} ist der letzte gültige Wert des internen Signals, welcher bei Einhaltung der hier beschriebenen Reihenfolge der Testfälle der letzte Sprungweiten-Wert ist (s. Abschnitt 7.4).

Bei der Messung des internen Signals und dem Vergleich von Sollwert x_{Ev} und gemessenen Istwert x_{Ac} muss noch eine weitere Metrik berücksichtigt werden: die maximale Zeitspanne (Timeout), bis dieser Vergleich erfolgreich ist oder der Testschritt als nicht bestanden gilt. Um diesen Timeout berechnen zu können, müssen wiederum die folgenden Punkte in Betracht gezogen werden:

- die Zykluszeit CT_{Ext}, mit welcher das jeweilige externe Signal gesendet wird
- die maximale Zeitspanne CT_{Int}, bis das Signal vom Steuergerät abgerufen wird

- eine hardware- und steuergeräteabhängige Toleranz T, welche auf Basis der Buslast und den Erfahrungswerten aus Messungen mit dem jeweiligen Steuergerät definiert wird (benötigt, da die Testfälle an sich steuergeräteunabhängig sein sollen). In dieser Toleranz ist auch die Zeit für die XCP-Übertragung sowie die Auswertung durch CANoe berücksichtigt.

Es sind nun 2 Fälle zu unterscheiden. Wir nehmen für beide Fälle an, dass gilt:

$$CT_{Ext} = CT_{Int}$$

Im Worst-Case-Szenario (vgl. Abbildung 3.13) fällt die Manipulation des externen Signals auf den Zeitpunkt unmittelbar nach dem das externe Signal gesendet wurde und die Evaluation des internen Signals benötigt die maximale Zeitspanne CT_{Int}.
Im Best-Case-Szenario (vgl. Abbildung 3.14) fällt die Manipulation des externen Signals auf den Zeitpunkt unmittelbar bevor das externe Signal gesendet wird und die Evaluation des internen Signals führt unmittelbar nachdem Senden des externen Signals zu einem positiven Ergebnis.

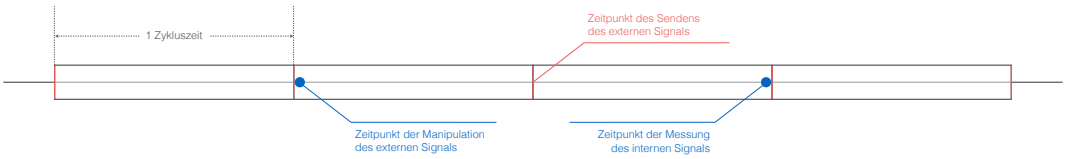

Abbildung 3.13. Worst-Case-Szenario für Manipulation und Messung

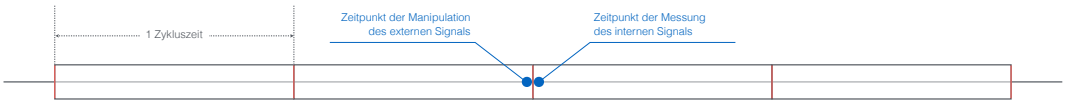

Abbildung 3.14. Best-Case-Szenario für Manipulation und Messung

Der Timeout lässt sich also wie folgt berechnen:

$$Timeout = CT_{Ext} + CT_{Int} + T$$

3.1.1.2. Sendesignaltest

Der Sendesignaltest wird in der Theorie analog zum Empfangssignaltest durchgeführt. Allerdings ist es nicht möglich, diesen Test mit dem in Kapitel 2.5.3 beschriebenen Testaufbau durchzuführen.
Das Problem ist die notwendige Manipulation des internen Signals. Interne Signale werden durch steuergeräteinterne Variablen abgebildet, für welche im Gegensatz zu steuergeräteinternen Parametern ausschließlich ein lesender Zugriff möglich ist. Nur wenn am Anfang der Signalkette eine Eingabe durch beispielsweise einen Sensor gegeben ist, kann eine Signalmanipulation durchgeführt werden. Im gegebenen Testaufbau sind keine realen Sensoren über XCP angeschlossen. Selbst wenn man reale Sensoren an das Steuergerät anschließen würde, wäre es nicht möglich konkrete Werte wie z.B. Grenzen oder Sprungweiten zu testen, da nicht sichergestellt werden kann, dass diese Werte für jedes Signal auch vom jeweiligen Steuergerät gesendet werden. Für eine (sehr begrenzte) Teilmenge der Sensoren wäre dies in gewissem Umfang noch möglich - z.B. für einen Spannungssensor durch manuelles Anlegen verschiedener Spannungen. Dem steht jedoch der Aufwand für die manuelle Durchführung bzw. die Integration der zusätzlichen Hardware und Änderung der Steuergerätesoftware gegenüber, sodass hier die Durchführung am HIL (vgl. Kapitel 2.5.2) bzw. im Fahrzeug (vgl. Kapitel 2.5.1) deutlich mehr Sinn macht.

3.1.2. Botschaftsrouting und Signalrouting

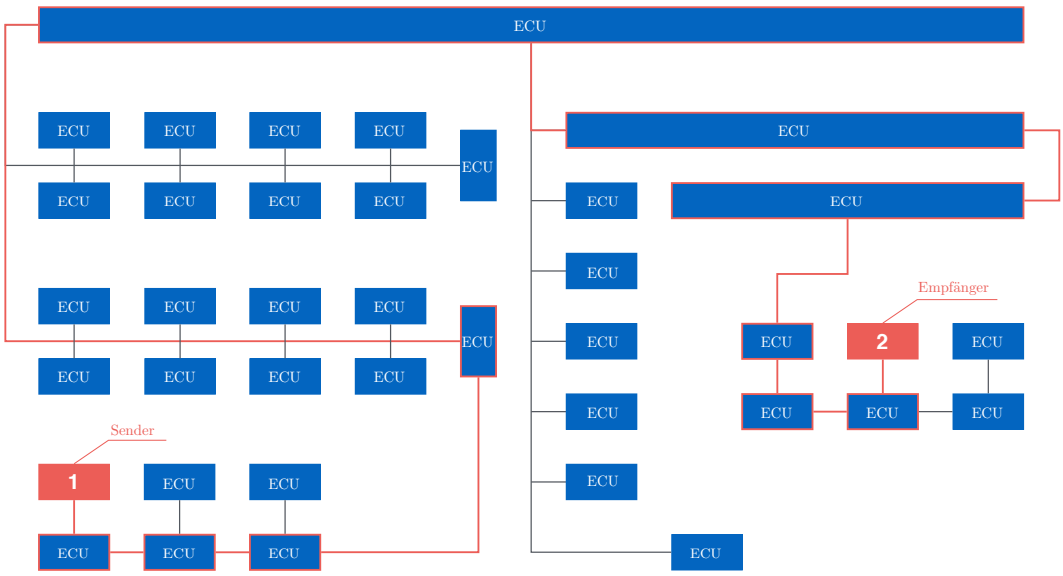

Abbildung 3.15. Routing einer Botschaft über mehrere Netzwerke

Ein weiterer Anwendungsfall der Vernetzungssoftware ist das Botschaftsrouting (vgl. Ta-

belle 3.2) und das damit einhergehende Signalrouting. Nicht alle Botschaften werden von einem Steuergerät zu einem anderen Steruergät im selben Netzwerk übertragen. Es gibt auch Botschaften die von Steuergerät 1 in Netzwerk *A* an Steuergerät 2 in Netzwerk *B* übertragen werden müssen. Der Routingpfad kann dabei mehrere Steuergeräte und Netzwerke umfassen (vgl. Abbildung 3.15). Für den Routingtest können die Testfälle aus Kapitel 3.1.1.1 verwendet werden. Die Berechnung der Testwerte verliert hier sogar an Komplexität, da lediglich Werte auf dem Bus gesetzt und auch auf einem anderen Bus abgefragt werden, es müssen also keine Umrechnungen aufgrund von z.B. unterschiedlichen Skalierungen durchgeführt werden, da das manipulierte und das evaluierte Signal identisch sind. Dennoch wird dieser Test im Rahmen dieser Arbeit nicht mit dem *Network-Test-Generator* (vgl. Kapitel 4) automatisiert, da er bereits mit dem Tool ComSpector durchgeführt wird [17] und eine entsprechende Datenbasis mit Informationen über Botschafts- und Signalrouting aktuell in keinem Format, welches mit vertretbarem Aufwand eingelesen werden kann, vorliegt.

3.1.3. Fehlererkennung

Die Vernetzungssoftware muss Fehler, die während der Signalübertragung entstehen, erkennen und signalisieren. Dies beinhaltet die in den folgenden Kapiteln beschriebenen Fehlerklassen. Der Test der Fehlererkennungsmechanismen kann automatisiert mit dem in Kapitel 2.5.3 Testaufbau durchgeführt werden.

3.1.3.1. Timeout

Wird eine Botschaft nicht im dafür vorgesehenen Zeitfenster empfangen, kann dies mehrere Gründe haben:

1. das Senden der Botschaft hat sich verzögert,
2. die Botschaft wurde gar nicht gesendet, oder
3. die Botschaft wurde zwar (rechtzeitig) gesendet aber die Übertragung war nicht erfolgreich.

In beiden Fällen kann die Botschaft nicht rechtzeitig empfangen werden. Man spricht dann von einem Timeout-Fehler bzw. einem Botschaftsausfall.

3.1.3.2. SQC (Botschaftszähler)

Ein Botschaftsausfall kann mittels Timeout nur genau dann erkannt werden, wenn die darauffolgende Botschaft ebenfalls nicht rechtzeitig empfangen wird. Gesetz dem Fall, dass die darauffolgende Botschaft jedoch innerhalb des Zeitfensters, in welchem die ursprüngliche Botschaft empfangen werden sollte, ankommt, kann ein Timeout-Fehler nicht erkannt werden, da „eine“ Botschaft rechtzeitig empfangen wurde. Daher kann eine Botschaft /

Signalgruppe über einen sogenannten Botschaftszähler (engl. sequence counter) abgesichert werden. Mit Hilfe eines Botschaftszählers können ausgefallene Botschaften auch dann erkannt werden, wenn die darauffolgende Botschaft rechtzeitig gesendet und empfangen wurde, also kein Timeout-Fehler erkannt werden kann.
Ein Botschaftszähler ist typischerweise als ein 4-Bit-Signal realisiert, hat also einen Wertebereich $R_{SQC} = \{0, ..., 15\}$. Er wird mit dem Senden der jeweiligen Botschaft inkrementiert und beim Empfangen der Botschaft evaluiert (mit dem letzten empfangenen Wert verglichen). Dabei können die folgenden Fälle auftreten:

$$SQC_i = (SQC_{i-1} \mod SQC_{max}) + 1 \quad \text{mit } i > 1,\ SQC_{max} = max(R_{SQC})$$

In diesem Fall liegt kein Fehler vor, da der Wert des Botschaftszählers genau dem erwarteten Wert entspricht.

$$SQC_i = ((SQC_{i-1} + x) \mod SQC_{max}) \quad \text{mit } i > 1,\ SQC_{max} = max(R_{SQC})$$

In diesem Fall sind Botschaften ausgefallen. Die Anzahl der ausgefallenen Botschaften $|ML|$ lässt sich wie folgt berechnen:

$$|ML| = \begin{cases} SQC_i - SQC_{i-1} - 1 & \text{, falls } SQC_i > SQC_{i-1} \\ k \cdot SQC_{max},\ k > 0 & \text{, falls } SQC_i = SQC_{i-1} \\ SQC_{max} + (SQC_i - SQC_{i-1}) & \text{, falls } SQC_i < SQC_{i-1} \end{cases}$$

Folgender Sonderfall existiert und wird nicht als Fehler interpretiert:

$$SQC_i = (SQC_{i-1} \mod SQC_{max}) + 2 \quad \text{mit } i > 1,\ SQC_{max} = max(R_{SQC})$$

3.1.3.3. CRC (Cyclic Redundancy Check)

Botschaften können zusätzlich noch über eine Prüfsumme abgesichert werden. Hierbei wird das Verfahren der zyklischen Redundanzprüfung verwendet (engl. cyclic redundancy check). Diesem Verfahren liegt die Polynomdivision zu Grunde. Zusammen mit dem Botschaftszähler (SQC) stellt dies die sogenannte End-to-End-Absicherung dar. Um die CRC-Prüfsumme zu berechnen, wird folgender Input benötigt:

- die Nutzdaten der Botschaft (vgl. Abbildungen 2.4, 2.7 und 2.11)
- eines von 16 von AUTOSAR definierten sogenannten Generatorpolynomen, auch CRC-Polynom genannt, welches auf Basis des aktuellen Werts des Botschaftszählers (Wert entspricht dem 0-basierten Index) ausgewählt wird.

Beim Versenden der Botschaft wird die CRC-Prüfsumme dann wie im folgenden Beispiel berechnet. Man beachte, dass die Nutzdaten in diesem Beispiel nur 2 Bytes lang sind, um das Berechnungsverfahren einfacher veranschaulichen zu können. Es wird das folgende Generatorpolynom verwendet:

$$x^8 + x^4 + x^3 + x^2 + 1$$
$$= 1 \cdot x^8 + 0 \cdot x^7 + 0 \cdot x^6 + 0 \cdot x^5 + 1 \cdot x^4 + 1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$$

Überführt man das Generatorpolynom in Binärpräsentation, erhält man folgenden Input:

Nutzdaten:	1	0	0	1	1	1	0	1	0	0	0	0	1	1	1	1
CRC-Polynom:	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**							

Im ersten Schritt wird die Bitfolge der Nutzdaten durch die binäre Repräsentation der CRC-Polynoms modulo 2 geteilt (vgl. Abbildung 3.16.) (durch Anwendung der exklusiven Oder-Verknüpfung $\oplus$). Dafür werden n Nullen an die Nutzdaten angehängt, wobei n dem Grad des Generatorpolynoms entspricht - hier $n = 8$.

1	0	0	1	1	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**														
0	0	0	1	0	0	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0
			1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**											
			0	0	0	1	0	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0
						1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**								
						0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0
									1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**					
									0	1	0	0	0	1	0	0	1	0	0	0	0	0
										1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**				
										0	0	0	0	0	1	1	1	1	0	0	0	0	0
															1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**
															0	1	1	1	1	1	1	0	1

Abbildung 3.16. Berechnung der CRC-Prüfsumme

1	0	0	1	1	1	0	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**														
0	0	0	1	0	0	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1
			1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**											
			0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1
						1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**								
						0	0	0	1	1	0	0	1	0	1	1	1	1	1	1	1	1
									1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**					
									0	1	0	0	0	1	0	1	0	1	1	1	1	1
										1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**				
										0	0	0	0	0	1	0	0	0	1	1	1	0	1
															1	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**
																							0

Abbildung 3.17. CRC-Check mit positivem Ergebnis

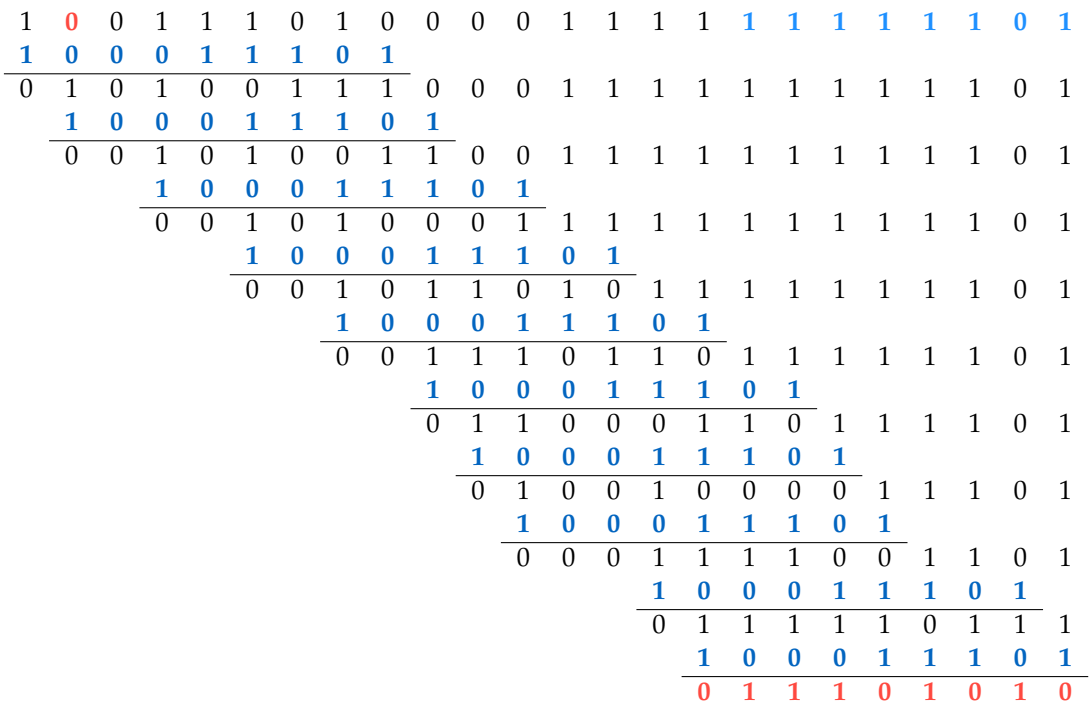

Abbildung 3.18. CRC-Check mit negativem Ergebnis

Der entstandene Rest r_1 dieser Rechnung ist die CRC-Prüfsumme, welche ebenfalls Länge n haben muss. Diese CRC-Prüfsumme wird in einem dafür vorgesehenen Signal in der Botschaft übertragen.

Beim Empfang der Botschaft wird nun der CRC-Check durchgeführt. Dafür wird die CRC-Prüfsumme an die Nutzdaten angehängt und erneut durch das CRC-Polynom modulo 2 geteilt. Abbildung 3.17 zeigt ein Beispiel, in welchem die Nutzdaten inkl. der CRC-Prüfsumme fehlerfrei übertragen wurden. Abbildung 3.18 dagegen zeigt den Fall der fehlerhaften Übertragung.
Bei der Auswertung des Prüfresultate können die in Tabelle 3.19 beschriebenen Fälle auftreten [14].

	Richtig	Falsch
Positiv	$r_2 \neq 0$ Übertragung fehlerhaft	$r_2 \neq 0$ Übertragung fehlerfrei
Negativ	$r_2 = 0$ Übertragung fehlerfrei	$r_2 = 0$ Übertragung fehlerhaft

Abbildung 3.19. Mögliche Prüfresultate des CRC-Checks

Ein falsch positives Prüfresultat tritt genau dann auf, wenn (nur) die CRC-Prüfsumme fehlerhaft übertragen wurde. Ein falsch negatives Prüfresultat tritt genau dann auf, wenn entweder die Nutzdaten und die CRC-Prüfsumme fehlerhaft übertragen wurden, oder die Nutzdaten durch fehlerhafte Übertragung nun ein Vielfaches des CRC-Polynoms sind [15].

3.1.3.4. Ablauf und Bewertung der Fehlerdiagnose

Die Diagnose für die in den vorigen Kapiteln beschriebenen Fehlerklassen läuft in mehreren Phasen ab, welche im Folgenden für die in den Kapiteln 3.1.3.1, 3.1.3.2 und 3.1.3.3 vorgestellten möglichen Kommunikationsfehler beschrieben werden. Die einzelnen Phasen, die die Diagnose eines Fehlers E durchlaufen kann, sind die folgenden:

- **Entprellung**. Die Entprellphase stellt eine Art Toleranz dar, um z.B. Fehler, die durch kurze und unkritische Störungen verursacht werden, abzufangen. Ein Beispiel dafür sind die sogenannten Jitter-Effekte [1].
- **Fehlerspeichereintrag**. In dieser Phase wird der erkannte Fehler im Fehlerspeicher eingetragen.
- **OBD-Fehlerspeichereintrag**. In dieser Phase wird ein OBD (vgl. 2.4) relevanter Fehler im Fehlerspeicher eingetragen.

Sei

$$EC_E, \quad E \in \{\text{Timeout}, \quad \text{SQC}, \quad \text{CRC}\}$$

der Zähler für die Anzahl der Übertragungsfehler. Für jede der oben genannten Phasen wird pro Botschaft ein Schwellwert definiert, der angibt, wann die jeweilige Phase gestartet wird:

[1] Differenz zwischen nominellem und tatsächlichem Empfangszeitpunkt eines Signals.

- Entprellschwelle: $DebLim_E \in \mathbb{N}$
- Fehlerschwelle: $ErrLim_E \in \mathbb{N}$
- OBD-Fehlerschwelle: $ObdErrLim_E \in \mathbb{N}$

Dabei gilt:

$$DebLim_E < ErrLim_E \leqslant ObdErrLim_E$$

Soll die Diagnose eine bestimmte Phase nicht durchlaufen, kann dies durch eine entsprechende Parametrisierung des jeweiligen Schwellwerts erreicht werden. Diese Methode kommt z.B. dann zum Einsatz, wenn im einem Fahrzeug aufgrund seiner Ausstattung ein bestimmtes Steuergerät nicht verbaut ist.

Die folgende Auflistung zeigt die standardmäßigen Phasen der Diagnose für die einzelnen Fehlerkategorien:

- **Timeout**:
  - Entprellung
  - Fehlerspeichereintrag
  - OBD-Fehlerspeichereintrag
- **SQC**:
  - Entprellung
  - Fehlerspeichereintrag
  - OBD-Fehlerspeichereintrag
- **CRC**:
  - Fehlerspeichereintrag
  - OBD-Fehlerspeichereintrag

Aus diesen Informationen lassen sich nun die durchzuführenden Testphasen ableiten. Diese sind Tabelle 3.3 zu entnehmen. In jeder Phase wird jeweils geprüft, ob der Fehler auf Botschaftsebene, Signalebene und im internen Fehlerspeicher korrekt signalisiert wird.

	Timeout	**SQC**	**CRC**
1. Vorbedingung Sicherstellung, dass zum aktuellen Zeitpunkt keine Kommunikationsfehler vorhanden sind. Darüberhinaus wird für jedes Signal auf dem Bus ein gültiger Wert gesetzt, um jegliche Verfälschungen durch unberechenbare Nebenwirkungen ausschließen zu können.	✓	✓	✓

2. Fehler Simulation Simluation eines Botschaftsausfalls (Timeout-Test), eines verfälschten SQC-Werts (SQC-Test) oder einer einer verfälschten CRC-Sequenz (CRC-Test).	✓	✓	✓
3. Entprellung Nach Ablauf der Entprellschwelle wird überprüft, ob sich die Diagnose des jeweiligen Fehlers auch in der Entprellung befindet, es darf also noch kein Fehler im Fehlerspeicher eingetragen worden sein.	✓	✓	
4. Heilung Die Simulation des jeweiligen Fehlers wird gestoppt, sodass die Kommunikation wieder fehlerfrei abläuft.	✓	✓	
5. Sleep Der Test wird für eine kurze (netzwerk- und steuergeräteabhängige) Zeitspanne unterbrochen.	✓	✓	
6. Fehler Simulation Simluation eines Botschaftsausfalls (Timeout-Test), eines verfälschten SQC-Werts (SQC-Test) oder einer einer verfälschten CRC-Sequenz (CRC-Test).	✓	✓	
7. Fehlereintrag Nach Ablauf der Fehlerschwelle wird überprüft, ob der Fehler korrekt diagnostiziert wurde, d.h. es muss ein Eintrag im Fehlerspeicher abgelegt sein. Gleichzeitig darf zu diesem Zeitpunkt (noch) kein OBD relevanter Fehlereintrag vorhanden sein.	✓	✓	✓

8. OBD-Fehlereintrag Nach Ablauf der OBD-Fehlerschwelle wird für alle hinsichtlich des jeweiligen Fehlers OBD relevanten Botschaften überprüft, ob der Fehler korrekt diagnostiziert wurde, d.h. es muss ein OBD relevanter Eintrag im Fehlerspeicher abgelegt sein.	✓	✓	✓
9. Heilung Die Simulation des jeweiligen Fehlers wird gestoppt, sodass die Kommunikation wieder fehlerfrei abläuft.	✓	✓	✓
10. Nachbedingung Sicherstellung, dass zum aktuellen Zeitpunkt keine Kommunikationsfehler (mehr) vorhanden sind. Die Diagnose für Timeout-, SQC- und CRC-Fehler darf sich nicht in der Entprellung befinden, und es darf kein Fehlereintrag vorhanden sein.	✓	✓	✓

Tabelle 3.3. Testphasen der Fehlerdiagnose

Nachfolgend einige Erklärungen zu Tabelle 3.3. Für CRC-Fehler wird keine Entprellung realisiert, daher entfallen die Testphasen 3-6.

Die Berechnung der Zeitpunkte

$$T_{E,S} \quad E \in \{Timeout,\ SQC,\ CRC\},\ L \in \{DebLim,\ ErrLim,\ ObdErrLim\}$$

der jeweiligen Überprüfungen basiert auf der Zykluszeit CT des (externen) Signals auf dem Bus, der Fehlerart E und dem Schwellwert L. Diese Zeitpunkte sind als jeweils späteste Zeitpunke, zu denen z.B. ein Fehler im Fehlerspeicher eingetragen werden sein muss. Sie sind also vergleichbar mit den Timeouts, die im Empfangssignaltest verwendet werden (vgl. 3.1.1.1). Bei der Berechnung dieser Zeitpunkte muss $T_{E,ObdErrLim}$ gesondert behandelt werden. Dies ist notwendig, da für die Überprüfung des Fehlereintrags und des OBD-Fehlereintrags der jeweilige Fehler nicht getrennt simuliert wird. Die Simulation des jeweiligen Fehlers findet für die Überprüfung der Entprellung gertrennt statt, da die Differenz zwischen Entprellschwelle und Fehlerschwelle in der Regel sehr klein ist. Wird eine Botschaft nun noch mit einer sehr geringen Zykluszeit CT gesendet, kann es zu Problemen bei der zeitlich exakten Überprüfung der Fehlersignalisierungsmechanismen kommen. Um dies zu begründen, definiert man die kleinste Zeitspanne Min_{MT}, die mindestens für eine Messung eines internen Signals benötigt wird. Diese Zeitspanne

varriert je nach Steuergerät und Umfang des Tests.
Die folgenden Szenarien können dabei eintreten:

- Es gibt keine Probleme, falls gilt (vgl. Abbildung 3.20):

$$Min_{MT} \leqslant T_{E,DebLim}$$

Abbildung 3.20. $Min_{MT} \leqslant T_{E,DebLim}$

- Es kann zwar überprüft werden, ob sich die Diagnose in der Entprellung befindet, jedoch nicht, ob dies auch rechtzeitig geschieht, falls gilt (vgl. Abbildung 3.21):

$$T_{E,DebLim} < Min_{MT} < T_{E,ErrLim}$$

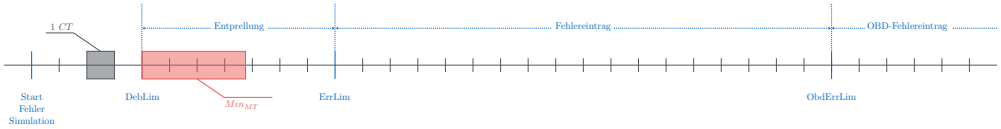

Abbildung 3.21. $T_{E,DebLim} < Min_{MT} < T_{E,ErrLim}$

- Es kann nicht überprüft werden ob die Diagnose die Phase der Entprellung (rechtzeitig) durchläuft; es kann zwar überprüft werden, ob ein Fehler im Fehlerspeicher eingetragen wird, jedoch nicht, ob dies auch rechtzeitig geschieht, falls gilt (vgl. Abbildung 3.22):

$$T_{E,ErrLim} < Min_{MT} < T_{E,ObdErrLim}$$

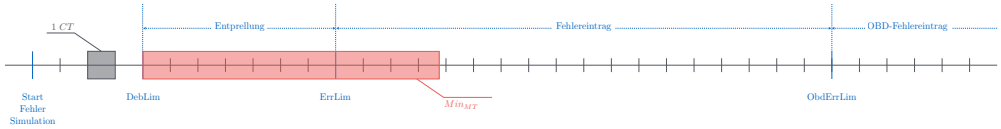

Abbildung 3.22. $T_{E,ErrLim} < Min_{MT} < T_{E,ObdErrLim}$

- Es kann nicht überprüft werden ob die Diagnose die Phasen der Entprellung und des Fehlereintrags (rechtzeitig) durchläuft; es kann zwar überprüft werden, ob ein Fehler und OBD relevanter Fehler im Fehlerspeicher eingetragen werden, jedoch nicht, ob dies auch rechtzeitig geschieht, falls gilt (vgl. Abbildung 3.23):

$$Min_{MT} > T_{E,ObdErrLim}$$

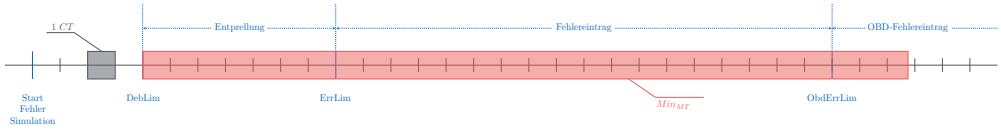

Abbildung 3.23. $Min_{MT} > T_{E,ObdErrLim}$

Es kommt also genau dann zu Problemen, sobald gilt:

$$Min_{MT} > T_{E,DebLim}$$

Daher wird für die Testphase, in der die Entprellung getestet wird, der Fehler gesondert simuliert.

Die Zeitpunke $T_{E,L}$ lassen sich also wie folgt berechnen:

$$T_{E,L} = \begin{cases} (L_E - ErrLim_E) \cdot CT + 1 & \text{, falls } L = ObdErrLim \\ S_E \cdot CT + 1 & \text{, sonst} \end{cases}$$

Alle im Vorigen beschriebenen Testfälle, die die Anforderung der Fehlererkennung abdecken, sind automatisiert mit dem in Kapitel 2.5.3 beschriebenen Testaufbau durchführbar.

3.2. Überblick und Ausblick

Im Folgenden werden ein kurzer Überblick und eine Einstufung gemäß Tabelle 3.1 der Testkategorien gegeben, die für den Vernetzungstest von Steuergeräten im Antriebsstrang benötigt werden.

	Tischaufbau automatisiert	**Tischaufbau manuell**	**Nicht am Tischaufbau**
Empfangssignaltest	✓		
Sendesignaltest			✓
Routingtest		✓	
Timeout-Test	✓		
SQC-Test	✓		
CRC-Test	✓		

Tabelle 3.4. Einstufung der Testkategorien hinsichtlich Durchführbarkeit am Tischaufbau

Tabelle 3.4 zeigt, dass mit der Toolkette des aktuellen Testaufbaus (vgl. Kapitel 2.5.3) bereits der Großteil des hier vorgestellten Testkonzepts zum Test der Vernetzung von Steuergeräten im Antriebsstrang automatisiert durchgeführt werden kann. Um auch noch den Sendesignaltest automatisiert am Tischaufbau durchführen zu können, wird ein separater Softwarestand benötigt, der eine Manipulation des internen Sigals erlaubt, indem interne Signale auf interne Parameter abgebildet werden. Für den Routingtest hingegen bedarf es einer Datenquelle, die Aufschluss über die jeweiligen Botschafts- und Signalroutings eines Steuergeräts gibt. Sind ein solcher Softwarestand und eine solche Datenbasis gegeben, ist die Erweiterung um die aktuell noch nicht automatisiert am Tischaufbau durchführbaren Testkategorien äußerst einfach. Die Testfälle (Testwerte) sind identisch. Beim Sendesignaltest wird statt des externen Signals das interne Signal manipulierte und statt des internen Signals das extrene Signal gemessen. Im Zuge des Routingtests werden nur externe Signals manipuliert und gemessen, das Signal auf dem Bus, über welchen das aktuell getestete Steuergerät das Signal empfängt, wird manipuliert und das Signal, welches das Steuergerät auf einen anderen Bus weiterschickt, wird gemessen.

Kapitel 4

Network-Test-Generator

Nachdem nun das Testkonzept für den Vernetzungstest von Steuergeräten im Antriebsstrang detailliert beschrieben wurde, folgt in diesem Kapitel die Vorstellung des entwickelten Tools *Network-Test-Generator*, welches den für die in Kapitel 2.5.3 gezeigte Toolkette noch fehlenden Input in Form von Testvektoren generiert.

4.1. Grundlegende Entwurfsentscheidungen

Der *Network-Test-Generator* wird als Desktop-Anwendung entwickelt. Als Programmiersprache wird Java 8 [1] verwendet. Für die Entwicklung der graphischen Benutzeroberfläche kommt JavaFX [2] in Verbindung mit controlsFX [3] zum Einsatz. Durch die Verwendung von ausschließlich Java-basierten Technologien wird die Plattformunabhängigkeit des entwickelten Tools gewährleistet.

Der *Network-Test-Generator* ist entsprechend dem EVA-Prinzip [12] aufgebaut. Tabelle 4.1 erklärt dieses Prinzip im Kontext des *Network-Test-Generators*.

1	Eingabe	Eine oder mehrere Netzwerkbeschreibungen (.nwmx-Datei(en)) sowie die Parametrisierung des zu testenden Steuergeräts, welche u.a. die Bedatung der in Kapitel 3.1.3.4 vorgestellten Schwellwerte beinhaltet.
2	Verarbeitung	Erstellung der Testfälle gemäß dem in Kapitel 3 vorgestellten Testkonzept.
3	Ausgabe	Erstellung eines Testvektors in Form einer .xml-Datei, welcher die generierten Testfälle beinhaltet.

Tabelle 4.1. EVA-Prinzip im Kontext des *Network-Test-Generators*

4.1.1. Architekturmuster

Das Tool wird nach dem Model-View-Controller-Pattern entwickelt (vgl. Tabelle 4.2).

Model	Die Datenhaltung bzw. die programmatische Abbildung realer Objekte.
View	Die graphische Benutzeroberfläche.
Controller	Die Implementierung der Anwendungslogik.

Tabelle 4.2. MVC-Pattern im Kontext des *Network-Test-Generators*

[1] `http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`
[2] Framework für die Entwicklung von graphischen Benutzeroberflächen mit Java
[3] controlsFX stellt zusätzliche UI-Elemente für die Entwicklung von graphischen Benutzeroberflächen mit JavaFX bereit (`http://fxexperience.com/controlsfx/`)

4.1.2. Entwurfsmuster

Neben einem objektorientierten Entwurf werden die folgenden Entwurfsmuster bei der Implementierung des *Network-Test-Generators* verwendet:

- **Singleton**: das Singleton-Entwurfsmuster wird auf der Ebene einer Klasse angewandt. Die jeweilige Klasse darf dabei nur genau einmal instantiiert werden - es existiert also zu jedem gegebenen Zeitpunkt der Programmausführung nur maximal eine Instanz dieser Klasse.
- **Observer**: das Observer-Entwurfsmuster wird auf der Ebenes eines Objekts angewandt. Ein Beobachter (Observer) hat dabei die Möglichkeit sich bei einem oder mehreren beobachteten Objekten (Observables) zu registrieren. Er wird dann über Änderungen an den beobachteten Objekten informiert.
- **Builder**: das Builder-Entwurfsmuster wird auf der Ebene einer Klasse angewandt und kommt vor allem bei der Konstruktion komplexer Objekte zum Einsatz (wie z.B. bei Testvektoren). Dabei sollen Konstruktoren mit einer hohen Anzahl von Parametern vermieden werden. Stattdessen wird das Objekt sukzessive erzeugt. Ein weiterer Vorteil ist, dass identische Konstruktionsprozesse wiederverwendet werde können. Die Implementierung des Builder-Entwurfsmusters im *Network-Test-Generator* unterstützt darüberhinaus das Prinzip der sogenannten „Fluent Interfaces“, bei dem (zustandsmanipulierende) Methoden miteinander verkettet werden können, da sie immer die aktuelle Repräsentation der jeweiligen Instanz zurückliefern.

4.1.3. Komponentendiagramm

Abbildung 4.1 zeigt das Komponentendiagramm des *Network-Test-Generators*. Im Folgenden werden die einzelnen Komponenten und deren Aufgaben näher beschrieben.

4.1.3.1. Interne Komponenten

- **Model**: die Komponente Model beinhaltet die objektorientierte Repräsentation der Netzwerkbeschreibung (.nwmx-Datei), der Parametrisierung (.dcm-Datei) und der Testvektoren.
- **Importer**: diese Komponente beinhaltet die Funktionalität zum Einlesen der .nwmx-Dateien und .dcm-Dateien. Die in diesen Dateiene enthaltenen Informationen werden zusammen in einer Datenstruktur gespeichert. Dennoch wird die Komponente Importer in zwei Subkomponenten aufgeteilt, somit ist ein einfaches Austauschen der Komponenten gewährleistet.
- **ValueHelper**: die Komponente ValueHelper stellt die Funktionalität zur Berechnung der Testwerte und die Implementierung der Algorithmen zur Umrechnung (z.B. zwischen zwei Skalierungen) zur Verfügung.

- **Exporter**: diese Komponente beinhaltet die Implementierung des Exports der Testvektoren als .xml-Datei.
- **Benutzeroberfläche**: in dieser Komponente wird die Implementierung der Benutzeroberfläche umgesetzt.
- **Generator**: in dieser (zentralen) Komponente ist die Generierung der Testvektoren realisiert. Außerdem beinhaltet sie die Implementierung der Anwendungslogik. Sie nutzt die bereitgestellten Schnittstellen des Importers, des ValueHelpers, des Exporters und der Benutzeroberfläche.

4.1.3.2. Externe Komponenten

Die externen Komponenten **JavaFX** und **controlsFX** werden für die Implementierung der Benutzeroberfläche verwendet.

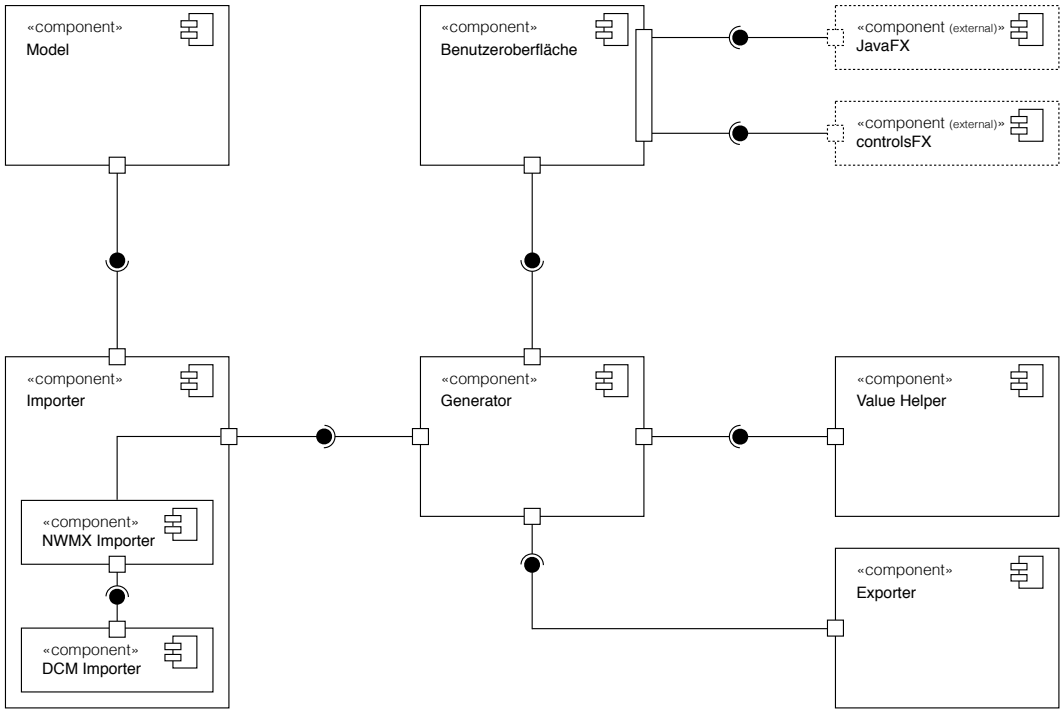

Abbildung 4.1. Komponentendiagramm des *Network-Test-Generators*

4.1.4. Klassendiagramm

Der Feinentwurf des *Network-Test-Generators* wird durch das Klassendiagramm dargestellt. Dieses ist der Übersichtlichkeit halber aufgeteilt und kann den Abbildungen 4.2, 4.3, 4.4 und 4.5 entnommen werden.

Im Folgenden werden die wichtigsten Klassen, deren Funktion nicht ohne Weiteres ersichtlich ist, näher beschrieben.

- der **MasterGenerator** ist die Schnittstelle, welche die Funktionalität zur Generierung der Testvektoren bereitstellt.
- die Klasse **AbstractTestGenerator** beinhaltet grundlegende Funktionen zur Generierung von Testvektoren, die für jede Testkategorie benötigt werden. Für die Generierung des Tests für eine konkrete Testkategorie wird diese Klasse erweitert. In der aktuellen Version sind das die folgenden beiden Klassen:
  - **RXMappingTestGenerator** (Empfangssignaltest)
  - **TimeoutInvalidDataTestGenerator** (Timeout, SQC und CRC Test)

  Diese Klassen beinhalten z.B. Methoden zur Auswahl der zu testenden Botschaften und Signale.
- die Klasse **AbstractTestHelper** realisiert die Generierung der einzelnen Testschritte in den jeweiligen Testphasen. Analog zur Klasse **AbstractTestGenerator** stehen in der aktuellen Version die folgenden Erweiterungen dieser Klasse zur Verfügung:
  - **RXMappingTestHelper** (Empfangssignaltest)
  - **TimeoutInvalidDataTestHelper** (Timeout, SQC und CRC Test)

  Die verschiedenen TestHelper beinhalten z.B. Methoden zur Auswahl der benötigten Testphasen und Testschritte für die von den TestGeneratoren ausgewählten Botschaften und Signale.
- die Klasse **TestReportGenerator** generiert während der Generierung der Testvektoren einen Bericht mit Informationen über z.B. während der Erstellung aufgetrene Fehler sowie eine Auflistung der im Testvektor enthaltenen Netzwerke, Botschaften und Signale.
- die Klasse **Tester** stellt die Repräsentation des Wurzelelements der generierten .xml-Datei dar.
- eine Instanz der Klasse **TestVector** wird für eine Testkategorie verwendet.
- eine Instanz der Klasse **TestGroup** wird für eine Botschaft verwendet.
- eine Instanz der Klasse **TestCase** wird für eine Signal verwendet.

- eine Instanz der Klasse **TestStep** wird für eine Testfall verwendet.
- die Klasse **Value** repräsentiert einen Wert, der als Rohwert und physikalischer Wert dargestellt werden kann (vgl. Kapitel 3.1.1.1).
- die Klasse **Values** stellt verschiedene Hilfsfunktionen für den Umgang mit Instanzen der Klasse Value sowie die Berechnung von Testwerten für ein Signal bereit.
- die Klasse **Scaling** repräsentiert eine Skalierung, wie in Kapitel 3.1.1.1 vorgestellt.
- die Klasse **Scalings** stellt verschiedene Hilfsfunktionen für den Umgang mit Instanzen der Klasse Scaling bereit (z.B. Umrechnung von Rohwert zu physikalischem Wert).
- die Klasse **TestValue** bündelt die Werte für einen Testfall im Empfangssignaltest (Setzwert und Sollwert).

Durch dieses Design ist die Erweiterung um weitere Testkategorien besonders einfach. Für die Unterstützung von Testkategorie X werden die folgenden Implementierungen benötigt:

- XTestGenerator
- XTestHelper
- XTestStage

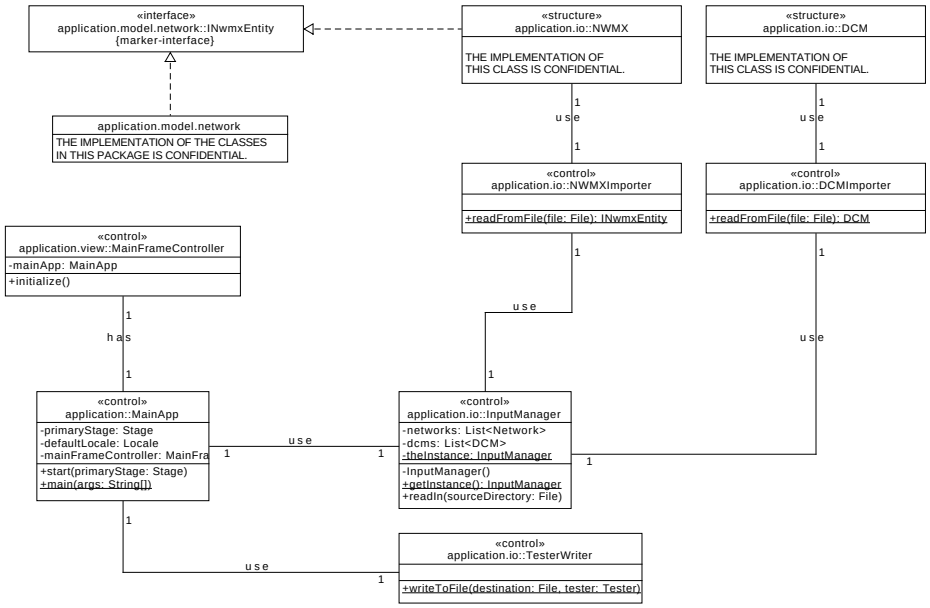

Abbildung 4.2. Klassendiagramm der Komponenten Model (Netzwerkbeschreibung), Importer und Exporter

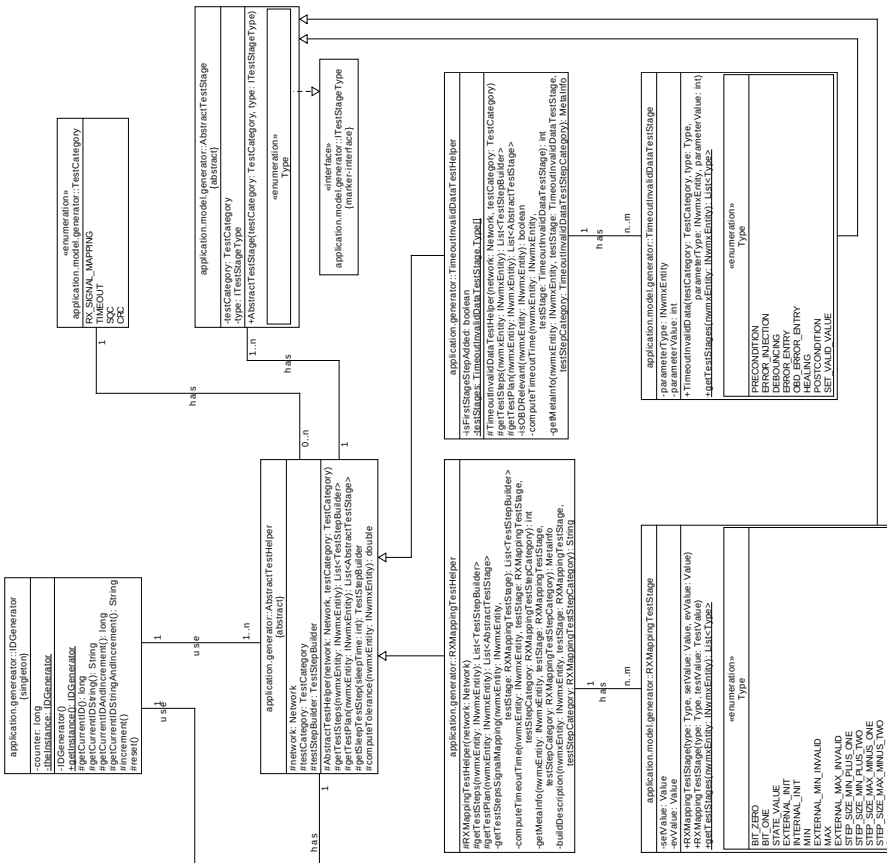

Abbildung 4.3. Klassendiagramm der Komponente Generator

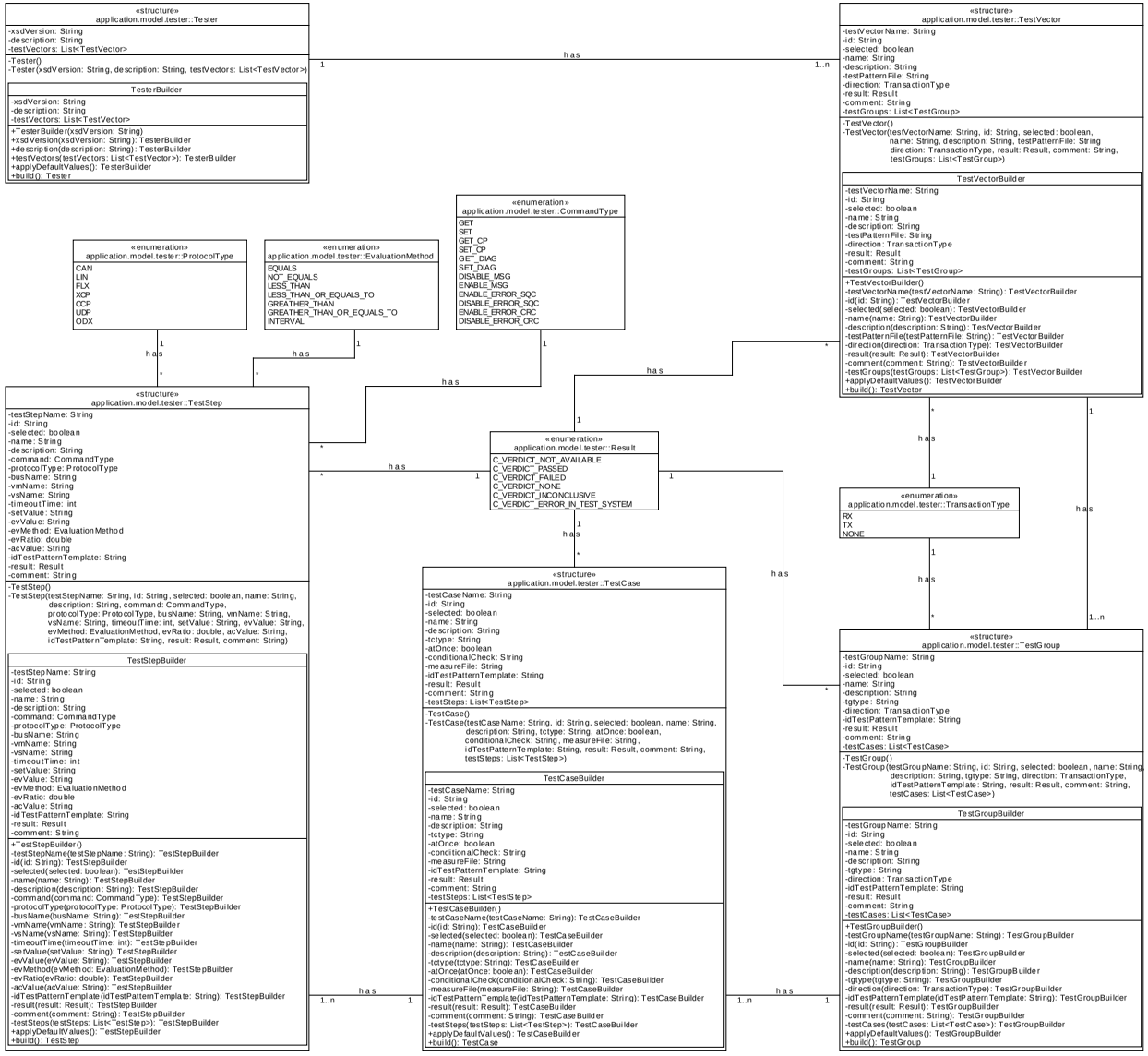

Abbildung 4.4. Klassendiagramm der Komponente Model (Testvektoren)

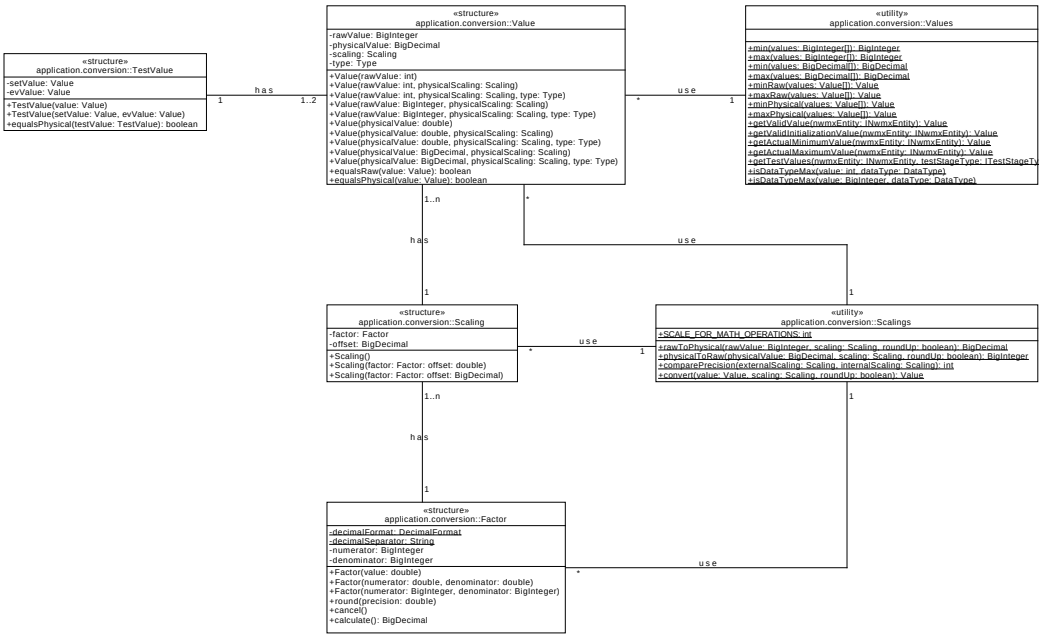

Abbildung 4.5. Klassendiagramm der Komponente ValueHelper

4.2. Anwendungsfälle

Im Folgenden werden die Anwendungsfälle des *Network-Test-Generators* aufgelistet.

4.2.1. UC-1: Quellverzeichnis auswählen

Der Benutzer wählt das Verzeichnis, in welchem das Tool nach Netzwerkbeschreibungen und der Parametrisierung des zu testenden Steuergeräts suchen soll. Dieser Anwendungsfall beinhaltet die folgenden Anwendungsfälle:

- **UC-1-1: Quellverzeichnis überprüfen**: das System überprüft, ob der angegebene Pfad ein nicht leeres lokales Verzeichnis ist.
- **UC-1-2: Datenquellen laden**: das System importiert die im Quellverzeichnis vorhandenen Netzwerkbeschreibung(en) und die Parametrisierung für das zu testende Steuergerät. Dieser Anwendungsfall wird durch die folgenden Anwendungsfälle erweitert:
  - **UC-1-2-1: Netzwerkbeschreibung(en) laden**: das System importiert die im Quellverzeichnis vorhandenen die Netzwerkbeschreibung(en).

- **UC-1-2-2: Netzwerkbeschreibung(en) laden**: das System importiert die im Quellverzeichnis vorhandenen die Parametrisierung des zu testenden Steuergeräts.

4.2.2. UC-2: Datenquellen überprüfen

Der Benutzer überprüft die automatisch für das zu testende Steuergerät aus dem Quellverzeichnis geladenen Datenquellen.

4.2.3. UC-3: Testvektor konfigurieren

Der Benutzer konfiguriert den zu generierenden Testvektor, indem er die zu testenden Netzwerke und die Testkategorien, die im Testvektor berücksichtigt werden sollen, auswählt.

4.2.4. UC-4: Zielverzeichnis auswählen

Der Benutzer wählt das Verzeichnis aus, in welchem der Testvektor als .xml-Datei gespeichert werden soll. Dieser Anwendungsfall beinhaltet die folgenden Anwendungsfälle:

- **UC-4-1: Zielverzeichnis überprüfen**: das System überprüft, ob der angegebene Pfad zu einem lokalen Verzeichnis führt.

4.2.5. UC-5: Generierung starten

Der Benutzer startet die Generierung des Testvektors durch einen Klick auf den Start-Button. Dieser Anwendungsfall beinhaltet die folgenden Anwendungsfälle:

- **UC-5-1: Testvektor generieren**: das System generiert den Testvektor für die ausgewählten Netzwerke und Testkategorien
- **UC-5-2: Testvektor exportieren**: das System exportiert den generierten Testvektor als .xml-Datei in das vom Benutzer ausgewählte Verzeichnis.

Die Anwendungsfälle des *Network-Test-Generators* werden im Use-Case-Diagramm in Abbildung 4.6 veranschaulicht.

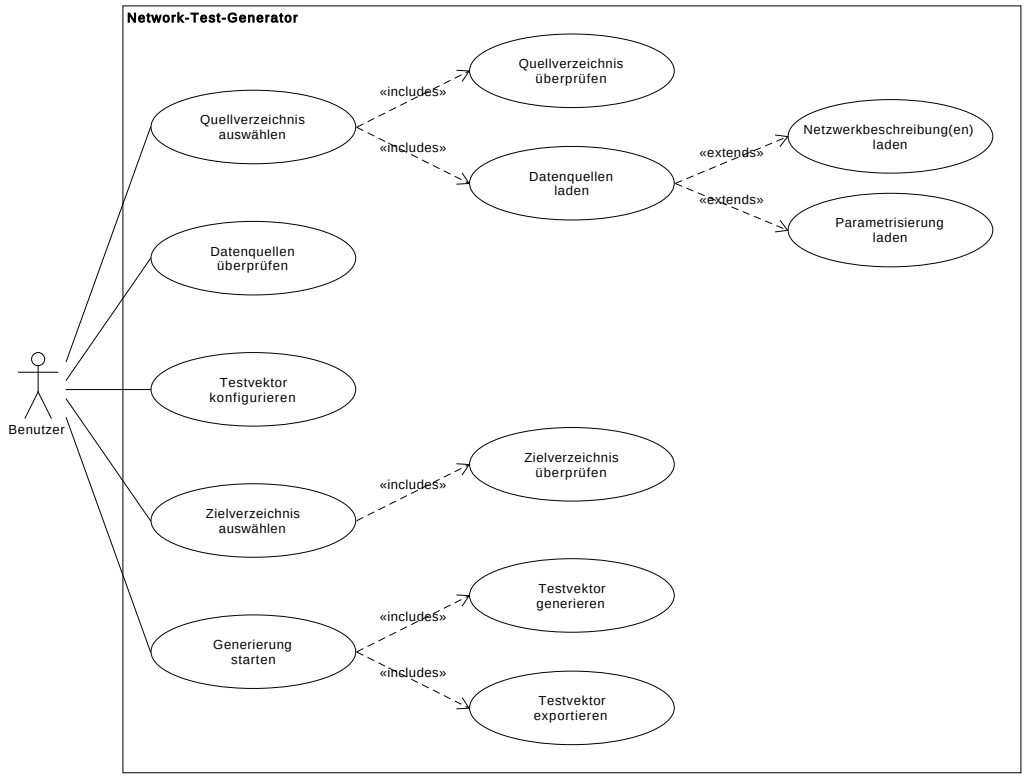

Abbildung 4.6. Use-Case-Diagramm des *Network-Test-Generators*

Zusätzlich zeigt Abbildung 4.7 das Aktivitätsdiagramm des *Network-Test-Generators*.

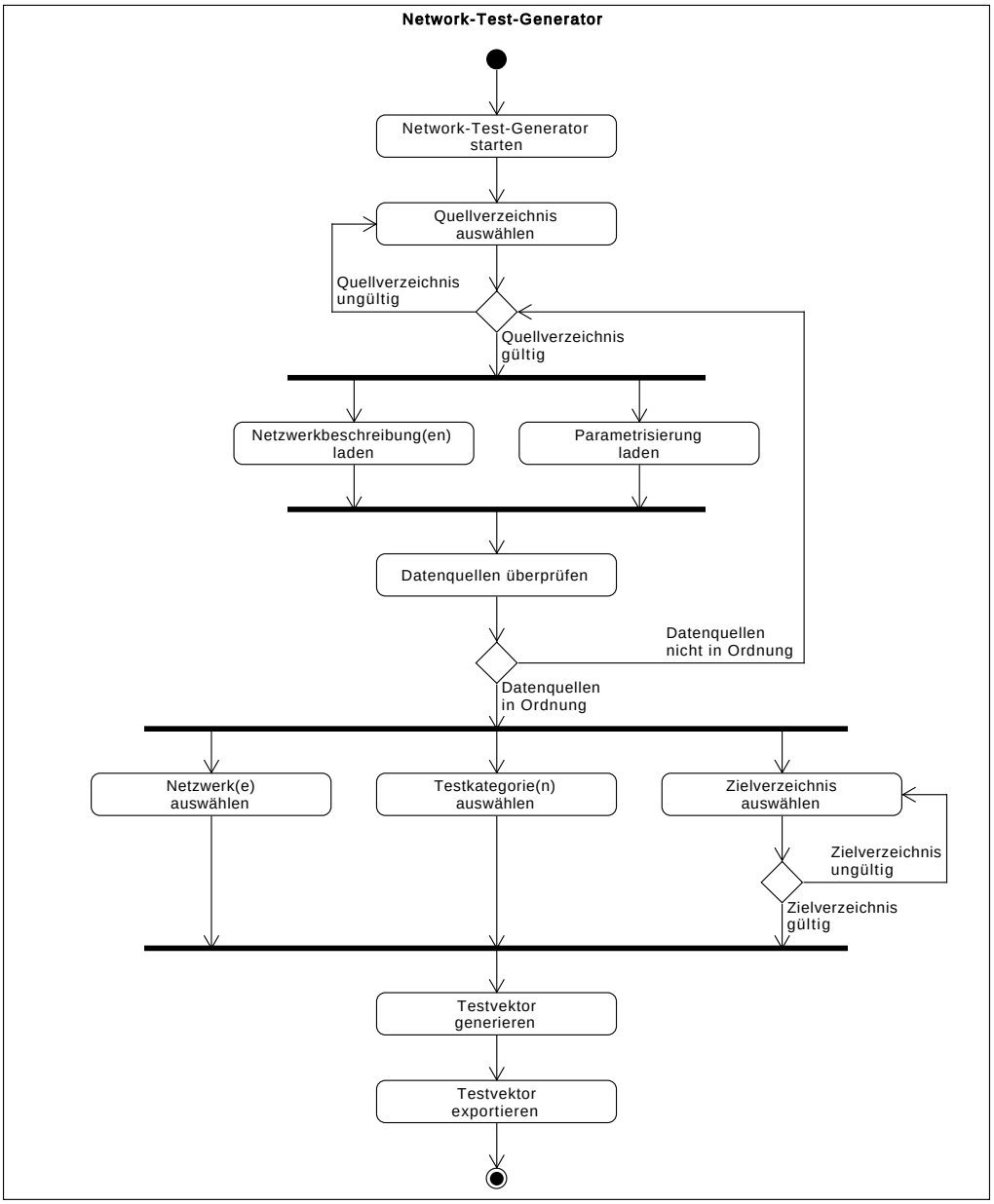

Abbildung 4.7. Aktivitätsdiagramm des *Network-Test-Generators*

4.3. Benutzeroberfläche

Die Benutzeroberfläche des Network-Test-Generators wird in Abbildung 4.8 gezeigt.

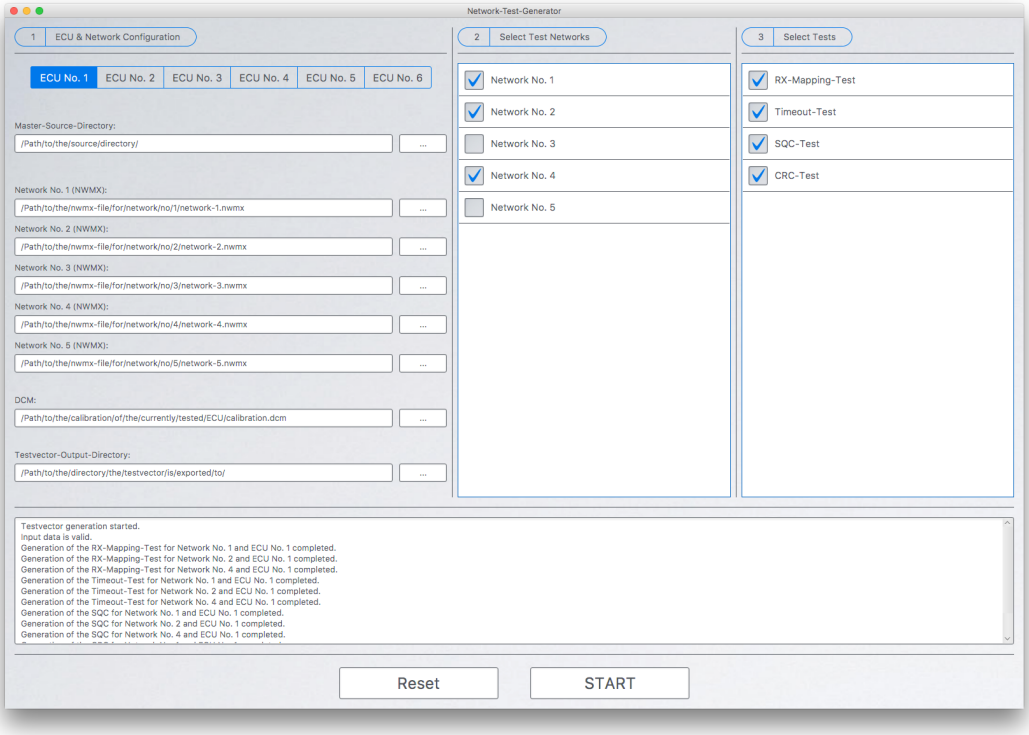

Abbildung 4.8. Benutzeroberfläche des *Network-Test-Generators*

Das Design der Benutzeroberfläche wurde basierend auf den Anwendungsfällen des Network-Test-Generators (vgl. Abbildungen 4.6 und 4.7) aufgebaut. Der Benutzer wählt im ersten Schritt das zu testende Steuergerät aus. Im „Master-Source-Directory“ wird automatisch nach passenden Netzwerkbeschreibungen und Steuergeräteparametrisierungen gesucht. Falls nicht alle benötigten Dateien gefunden werden können, muss der Benutzer diese manuell angeben. Im zweiten Schritt werden die in den Test einzuschließenden Netzwerke ausgewählt. Abschließend werden im dritten Schritt die durchzuführenden Testkategorien ausgewählt. Nach einem Klick auf den Start-Button wird die Testvektor-Generierung gestartet und der Benutzer erhält Informationen über den Fortschritt durch das im Log-Fenster über dem Start-Button.

Kapitel 5

Fazit und Ausblick

Als Abschluss dieser Arbeit gibt dieses Kapitel ein Fazit und einen Ausblick auf mögliche zukünftige Arbeiten im Hinblick auf Erweiterungen des Testkonzepts und des *Network-Test-Generators*.

5.1. Fazit

Im Rahmen dieser Arbeit wurde erörtert, welche Testkategorien für eine vollständige Abdeckung der Anforderungen an die Vernetzungssoftware benötigt werden. Im Anschluss wurde evauliert, welche dieser Testkategorien automatisiert am Tischaufbau durchführbar sind. Mit dem für diese Testkategorien entwickelten Testkonzept und dem entwickelten Tool *Network-Test-Generator* kann nun ein Großteil der Testkategorien automatisiert am Tischaufbau durchgeführt werden. Dadurch können sowohl die Kosten als auch der zeitliche Aufwand, der für den Vernetzungstest von Steuergeräten im Antriebsstrang anfällt, drastisch reduziert werden.

5.2. Ausblick

Sowohl das Testkonzept als auch der *Network-Test-Generator* wurden im Hinblick auf die Möglichkeit einfacher Erweiterungen und Änderungen entwickelt. Der erste Schritt bei der Weiterentwicklung des Testkonzepts und des Tools ist die Erweiteurng um die Unterstützung des Sendesignaltests und des Routingstests. Um auch noch den Sendesignaltest mit dem aktuellen Testaufbau durchführen zu können, müsste ein entsprechende Softwarestand zur Verfügung stehen, der Signale intern durch Parameter, für welche ein schreibender Zugriff über XCP möglich ist, abbildet. Für den Routingtest wird eine Datenbasis mit Informationen über die jeweiligen Botschafts- und Signalroutings sowie eine weitere Importer-Komponente benötigt, die eben jene Datenbasis für die Generierung der Testfälle zur Verfügung stellen kann.
Im Hinblick auf die Steuergeräte und deren Vernetzung außerhalb des Antriebsstrang stellt das entwickelte Konzept eine gute Basis dar, um auch hier die Umfänge der Vernetzungstests möglichst umfassend an den Tischaufbau zu verlagern, und damit die Möglichkeiten der Kosteneinsparung und Aufwands-Reduktion zu erweitern.

Literaturverzeichnis

[1] **Bussysteme in der Fahrzeugtechnik - Protokolle, Standards und Softwarearchitektur**
Autoren: Werner Zimmermann, Ralf Schmidgall
4. Auflage
Vieweg+Teubner
2007
ISBN: 978-3-8348-0907-0

[2] **Kraftfahrtechnisches Taschenbuch**
Autoren: Karl-Heinz Dietsche, Thomas Jäger
Robert Bosch GmbH
28. Auflage
Friedr. Vieweg & Sohn Verlag
2014
ISBN: 978-3-6580-3800-7

[3] **AUTOSAR Technical Overview**
http://www.autosar.org/about/technical-overview/

[4] **Vector-E-Learning „CAN“**
https://elearning.vector.com/vl_can_introduction_de.html

[5] **Vector-E-Learning „LIN“**
https://elearning.vector.com/vl_lin_introduction_de.html

[6] **Vector-E-Learning „FlexRay“**
https://elearning.vector.com/vl_flexray_introduction_de.html

[7] **XCP - Das Standardprotokoll für die Steuergeräte-Entwicklung**
Autoren: Andreas Patzer, Rainer Zaiser
Vector Informatik GmbH
2016

[8] **Ethernet revolutioniert die Fahrzeug-Vernetzung**
http://www.elektroniknet.de/markt-technik/automotive/ethernet-revolutioniert-die-fahrzeug-vernetzung-102613.html

[9] **CAN Specification Version 2.0**
Robert Bosch GmbH
1991
http://esd.cs.ucr.edu/webres/can20.pdf

[10] **LIN Specification Revision 2.2A**
LIN Consortium
2010
https://www.cs-group.de/wp-content/uploads/2016/11/LIN_Specification_Package_2.2A.pdf

[11] **FlexRay Communications System Protocol Specification Version 3.0.1**
FlexRay Consortium
2010
https://de.scribd.com/document/77021636/FlexRay-Protocol-Specification-V3-0-1

[12] **Einfache IT-Systeme**
Autoren: Klaas Gettner, Franz-Josef Lintermann, Udo Schaefer, Walter Schulte-Göcking
7. Auflage
Stam Verlag GmbH
2012
ISBN: 978-3-8237-1140-7
S. 16 ff.

[13] **Vector-E-Learning „AUTOSAR“**
https://elearning.vector.com/vl_autosar_introduction_de.html

[14] **Softwaretechnik - Grundlagen, Menschen, Prozesse, Techniken**
Autoren: Jochen Ludewig, Horst Lichter
3., korrigierte Auflage
dpunkt.verlag GmbH
2013
ISBN: 978-3-86490-092-1
S. 278 ff.

[15] **Error Correction Coding: Mathematical Methods and Algorithms**
Autor: Todd K. Moon
John Wiley & Sons, INC
2005
ISBN: 978-0-471-64800-0

[16] **Fahrzeugdiagnose mit OBD**
Autor: Florian Schäffer
3. neue und überarbeitete Auflage
Elektor Verlag
2015
ISBN: 978-3-89576-303-8

[17] **Mit minimalen Konfigurationsaufwand zur maximalen Testtiefe: Netzwerktests für jedermann**
Autoren: Viola Misch, Katja Hahmann
Elektronik automotive 8/9.2016
S. 35-39
http://www.elektroniknet.de/elektronik-automotive/software-tools/mit-minimalem-konfigurationsaufwand-zur-maximalen-testtiefe-134626.html

Alle URLs wurden zuletzt am 2. Oktober 2017 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift