Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Andrej Wittmann

Ein CAN FD-Modell für eine OMNeT++ basierte Simulationsumgebung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Andrej Wittmann

Ein CAN FD-Modell für eine OMNeT++ basierte Simulationsumgebung

Bachelorarbeit eingereicht im Rahmen einer Bachelorprüfung

im Studiengang TI
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Franz Korf
Zweitgutachter : Prof. Dr. Reinhard Baran

Andrej Wittmann

Thema der Arbeit

Ein CAN FD-Modell für eine OMNeT++ basierte Simulationsumgebung

Stichworte

CAN, CAN FD, CAN Bus, Modell, OMNeT++, Simulation

Kurzzusammenfassung

CAN FD ist eine Weiterentwicklung der CAN Busarchitektur, die den Einsatz von CAN Netzwerken in der Automobil- und Fahrzeugindustrie um einige Jahre verlängert ohne auf die neuen revolutionären Bustechnologien, wie FlexRay, umsteigen zu müssen. Im Vergleich zu CAN bringt CAN FD zwei Verbesserungen mit, die zu einer höheren Datenübertragung führen: eine zusätzliche höhere Übertragungsrate und der Versand eines größeren Datenvolumens pro Nachricht. Um den Einsatz von neuen Busarchitekturen in den Fahrzeugen zu beschleunigen, müssen kostengünstig Modelle entwickelt werden, die reale Szenarien der Datenübertragung nachbilden. Unter OMNeT++ können einfach Netzwerkmodelle erstellt und simuliert werden.

Andrej Wittmann

Title of the paper

CAN FD Model for OMNeT++ based simulation environment

Keywords

CAN, CAN FD, CAN Bus, Modell, OMNeT++, Simulation

Abstract

CAN FD is an improvement of CAN bus architecture, that prolongs the existence of CAN networks in automotive environment and vehicle industry for the next years without to switch to the advanced bus technologies, like FlexRay. Comparing CAN to CAN FD, the CAN FD brings up two new improvements: the usage of possible second bitrate during frame transmission and delivering more data in a message. In order to accelerate the use of new bus architectures in the vehicles, low-cost models have to be developed that simulate real-world data transmission scenarios. OMNeT ++ provides easiest way to create and simulate models of network architecture.

Inhaltsverzeichnis

1 Einleitung

„CAN FD liefert der Automobilindustrie dort die Grundlage für bessere Vernetzungslösungen, wo sich aufgrund des steigenden Datenaufkommens in der Fahrzeugelektronik zunehmend Engpässe offenbaren.“ (vgl. (1)).

Im Vergleich zu den Fahrzeugen vor 30 Jahren, finden heute immer mehr Assistenz- und Infotainment-Systeme, wie z. B. Rückfahrkameras, autonome Bremsassistenten oder Navigation, in die Fahrzeuge Einzug. Daraus folgt, dass die Anzahl von Steuergeräten und das damit verursachte Verkehrsaufkommen kontinuierlich ansteigt. Während es bei Feldbussen, wie FlexRay (vgl. (2)) oder Ethernet (vgl. (3)), die hohe Übertragungsraten erlauben, Engpässe kaum spürbar sind, ist es beim Controller Area Network (CAN) Bus nicht mehr der Fall. Der CAN Bus ist seit vielen Jahren das meist eingesetzte Bussystem in der Automobil- und Fahrzeugindustrie, bei dem Datenübertragungsraten von maximal 1MBit/s erzielen lassen. Die ersten Probleme, die aufgrund der geringen CAN Datenübertragung beobachtet wurden, ist die Übertragung von Softwareupdates in die Steuergeräte, deren Volumen immer größer wird. Die Wartezeit, die daraus resultiert, kann heute Niemanden mehr zugemutet werden. Deshalb sind für den CAN Bus in den letzten Jahren neue Anforderungen definiert worden, weil der Ersatz dieses Bussystems mit revolutionären Bussystemen, wie FlexRay, mit hohen Investitionskosten verbunden ist, die die Fahrzeugindustrie eher zu meiden versucht, weil sie letztendlich die Kunden tragen.

„Darüber hinaus bedeutet ein Umstieg vom ereignisgesteuerten CAN auf das zeitgesteuerte FlexRay für das Gros der Entwickler erhebliche Veränderungen in ihrer gewohnten Arbeits- und Denkweise.“ (vgl. (1))

Im Jahr 2012 stellte Robert Bosch GmbH die neue CAN FD (CAN mit flexibler Datenrate) Nachrichtenspezifikation vor. CAN FD bringt im Vergleich zu älteren CAN Protokollen zwei Verbesserungen mit, die zu einem höheren Durchsatz während der Nachrichtenübertragung führen, und zwar:

(a) die Möglichkeit der Anwendung einer zweiten höheren Datenrate und

(b) in einem Umfeld, wo die Verwendung der höheren Datenrate aufgrund von CAN Bus Eigenschaften nicht möglich ist, wie etwa bei CAN Buslösungen in Großraumfahrzeugen (z. B. LKWs, Omnibusse, usw.), kann der höhere Datendurchsatz aufgrund des größeren Nutzdatenfelds bei einer CAN FD Nachricht im Vergleich zur älteren CAN Nachricht, erreicht werden, das auch zu einer spürbaren Verbesserung bei der Datenübertragung führt.

Der Vorteil von CAN FD liegt nicht nur im höheren Durchsatz. CAN FD bildet die Obermenge von CAN und solange CAN FD Busteilnehmer nicht das neue CAN FD Nachrichtenformat anwenden, ist eine direkte Migration von CAN FD in CAN Netzwerke möglich (vgl. (4)).

Motivation

In den letzten Jahren ist der Einsatz der Simulationssoftware in der Automobil- und Fahrzeugindustrie stark zugenommen. Die Entwicklung von Modellen für neue Systemarchitekturen ist sehr wichtig, um daraus die ersten Erkenntnisse zu sammeln, mögliche Engpässe frühzeitig zu erkennen und den Einsatz dieser Systeme in neuen Fahrzeugen zu beschleunigen. Dabei werden Computersimulationen bevorzugt eingesetzt, mit denen verschiedene Architekturvarianten erprobt werden können, ohne teure Prototypenmodelle aufbauen zu müssen. Um Computersimulationen zu erzeugen, werden oft die frei zur Verfügung stehenden (open source) Simulationswerkzeuge, wie OMNeT++ (siehe Abschnitt 2.3), eingesetzt. OMNeT++ ist besonders für Simulationen von Netzwerkprotokollen und Netzwerkarchitekturen geeignet und bietet u. a. auch Methoden zur stochastischen Modellauswertung unter dem Einsatz von Zufallszahlgeneratoren.

Basierend auf OMNeT++ verwaltet das FiCo4OMNeT-Teilprojekt der CoRE (Communication over Realtime-Ethernet)-Arbeitsgruppe Simulationen zu verschiedenen Busarchitekturen, die heute in die Fahrzeuge eingebaut werden oder allmählich Einzug finden. Beispiele dafür sind Simulationen zu echtzeitfähigen Bussystemen, wie FlexRay, Time-Triggered CAN (TTCAN) oder Echtzeit-Ethernet, bei denen die Zeit ein wichtiger Faktor der Nachrichtenübertragung ist, aber auch ein Simulationsmodell zum ereignisgesteuerten CAN.

Zielsetzung

CAN FD ist die Zukunft des CAN Netzausbaus in den Fahrzeugen. In dieser Arbeit soll zur Ergänzung des CAN Bus-Modells unter OMNeT++, das zuvor für eine Simulationslandschaft aus aktuellen und zukünftigen Fahrzeugbusarchitekturen entwickelt wurde, um CAN FD erweitert werden, so dass ein Mischbetrieb zwischen CAN- und CAN-FD Busteilnehmern simuliert werden kann. Als Grundlage für die Umsetzung von CAN FD wird die CAN FD Nachrichtenspezifikation von Robert Bosch GmbH in der Ursprungsvariante (vgl. (4)) und der aktuelle ISO11898-1:2015 Standard (vgl. (5)) verwendet.

Gliederung der Arbeit

Die Arbeit ist in folgende Kapiteln unterteilt:

Kapitel 2 beginnt mit der Einführung in die CAN Bus Grundlagen. Hier werden Mechanismen der Datenübertragung von CAN und CAN FD erläutert und ihre Nachrichtenformate vorgestellt. Direkt danach werden Begriffe der system-level und ereignisgesteuerten Simulation definiert. Das Kapitel schließt die Einführung in die Simulationsumgebung OMNeT++ ab.

Im Kapitel 3 wird das vorhandene CAN Bus-Simulationsmodell vorgestellt, das unter OMNeT++-Framework entwickelt wurde. Dabei werden der Modellentwurf, ausgetauschte Nachrichtenformate und Funktionen von Modellierungskomponenten beschrieben.

Kapitel 4 geht auf die gestellten Anforderungen für die anschließende Umsetzung von CAN FD ein.

Im Kapitel 5 werden Lösungsansetze für die Umsetzung von CAN FD diskutiert.

Kapitel 6 befasst sich mit der Realisierung von CAN FD im CAN Bus Simulationsmodell. Im Hinblick auf gestellten Anforderungen wird die Implementierung von beschriebenen Konzepten erläutert.

Im Rahmen der Qualitätssicherung geht das Kapitel 7 auf die Methoden zur Überprüfung der Codequalität ein.

Das Kapitel 8 widmet sich der CAN FD-Modellevaluierung.

Die Zusammenfassung und Ausblick werden im Kapitel 9 die Arbeit abschließen.

2 Grundlagen

Dieses Kapitel beginnt mit einer Einführung in die CAN Bus Grundlagen. Hier wird ein Überblick über CAN vermittelt, der sich aus der historischen Entwicklung, Protokollschichten, auf den CAN und CAN FD operieren und Übertragungsmedien zusammensetzt. Anschließend werden Mechanismen der Datenübertragung und Nachrichtenformate erläutert, die für CAN FD und CAN gelten. Den Begriffen der system-level und ereignisgesteuerten Simulation ist der Abschnitt 2.2 gewidmet. Die Einführung in das Simulationswerkzeug OMNeT++ wird dieses Kapitel abschließen.

2.1 CAN Bus

Der CAN Bus gehört zur Familie von Feldbussen. Ein Feldbus verbindet mehrere Feldbusteilnehmer zum Zweck der Kommunikation miteinander.

2.1.1 Entwicklung

Die ersten Steuergeräte vor ca. 40 Jahren, zu denen die elektronische Einspritzung dazugehört, waren über Punkt-zu-Punkt Kommunikationsleitungen miteinander verbunden. Als die Anzahl an mechatronischen Bauteilen zugenommen hat, ist die Art der Verbindung unhandlich geworden. Zudem ist das Fahrzeuggewicht angestiegen, das u.a. zur Erhöhung des Spritverbrauchs beitrug. Aus der Anforderung der Automobil- und Fahrzeugindustrie die Kabellänge und das Kabelgewicht zu reduzieren, hat Robert Bosch GmbH 1983 den CAN Bus zur Vernetzung von Steuergeräten in den Kraftfahrzeugen entwickelt und zusammen mit Intel 1986 vorgestellt. Bis heute verlässt kein Fahrzeug das Fertigungsband nicht ohne ein einziges CAN Bussytem an Board zu haben.

Weitere historische Details (vgl. (6))

- 1987 - erster CAN Chip von INTEL
- seit 1989 gibt Serienbausteine für den Einsatz im Fahrzeug
- 1992 wird CAN in der Mercedes S-Klasse eingesetzt, später ziehen andere Fahrzeughersteller nach

- Unterschiedliche Busprotokolle für Fahrzeuge haben sich herausgebildet, wie z. B. CAN, VAN, J1850, ABUS. VAN und ABUS-Protokolle sind zugunsten von CAN aufgegeben worden
- Seit 1994/95 ist CAN das am meisten verbreitete Protokoll für Fahrzeuganwendungen
- 2001 wird der CAN Bus auch in Kleinwagen eingesetzt (im Karosseriebereich)
- 2012 – Entwicklung der CAN FD Nachrichtenspezifikation 1.0
- 2015 – Überarbeitung des CAN FD Protokolls zur besseren Fehlererkennung und Einführung des ISO11898-1 Standards für CAN und CAN FD

2.1.2 Busrealisierungen

Da die CAN Spezifikation nicht auf die Übertragungsmedien für CAN eingeht und nur auf Teile der Bitübertragung von einem CAN Busteilnehmer zu anderen CAN Busteilnehmern definiert, gibt es mittlerweile unterschiedliche CAN Busrealisierungen, die entsprechend den Anforderungen für unterschiedliche Fahrzeugdomänen entwickelt wurden. Die bekanntesten von ihnen sind

- Highspeed-CAN
- Lowspeed-CAN
- Diagnose CAN

Highspeed-CAN und Lowspeed-CAN haben sich nicht nur in der Automobil- und Fahrzeugindustrie bewährt, sondern werden aufgrund der einfachen und kostengünstigen Bauweise z. B. in der Medizintechnik und Maschinenbau eingesetzt. In den Fahrzeugen wird der Highspeed-CAN im Motorumfeld und der Lowspeed-CAN im Karrosseriebereich (z. B. Türen, Sitze und Schiebedach) eingesetzt. Der CAN Diagnosebus wird zur Verbindung der externen CAN Fahrzeugschnittstelle mit dem Board-Computer vernetzt, um Fahrzeugdiagnose durchführen zu können.

Der CAN Bus wird als eine Zwei-Draht-Variante, bestehend aus zwei verdrillten Kupferkabeln (auch Twisted-Pair genannt), ausgeführt.

Weil der CAN Bus auch im Motorraum verlegt wird, ist er oft äußeren Einflüssen ausgesetzt (z. B. Kurzschlüsse gegen Masse, Batteriespannung, Überschläge aus der Zündanlage oder statische Überschläge bei der Wartung). Durch die sogenannte differentielle Übertragungstechnik, unter der Verwendung der CAN High- und CAN Low-Leitungen, werden Einwirkungen von Störfaktoren weitestgehend eliminiert.

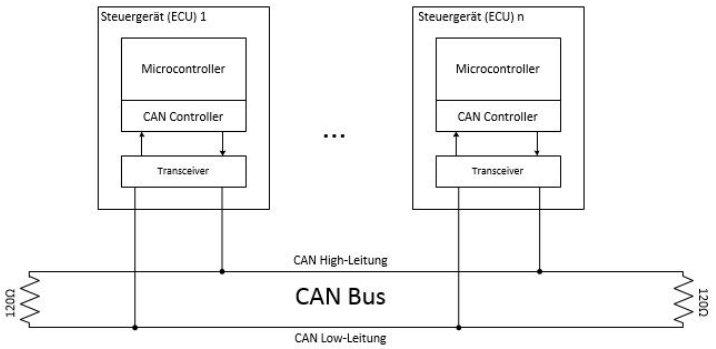

Abbildung 2.1. Highspeed-CAN mit 2 Leitern und 120Ω-Widerständen nach ISO 11898-2

Um Ausgleichsvorgänge (Reflexionen), die mit steigenden Datenraten auf dem Bus zunehmen, zu verhindern, werden die Leiter bei einem Highspeed-CAN Bus an beiden Enden mit 120Ω-Abschlusswiderständen terminiert. Dieser Widerstand ist ausreichend genug, um bei Datenraten bis 1Mbit/s eine sichere Übertragung zu gewährleisten. Beim Lowspeed-CAN Bus sieht die ISO898-3 aufgrund der geringeren maximalen Datenrate von 125kBits/s keine Abschlusswiderstände vor (vgl. CAN Bus, elearning.vector.com). Dadurch kann der Lowspeed-CAN Bus, z. B. bei Kabelbrüchen, immer noch auf eine Ein-Draht-Variante zurückfallen und die Masse dazu nehmen, um weiter zu funktionieren, das dann als Limp-Home-Modus (nach Hause humpeln) bezeichnet wird.

2.1.3 Topologien

Der CAN Bus kommt sehr oft als Linientopologie in den Fahrzeugen vor.

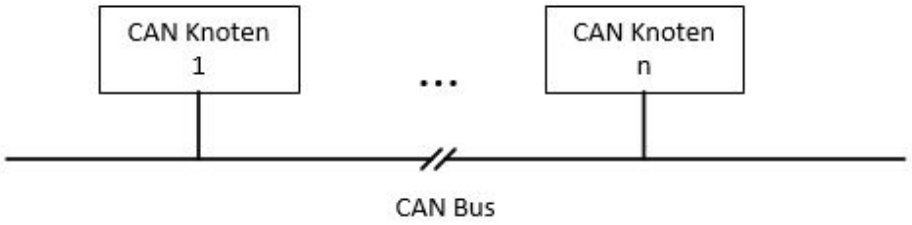

Abbildung 2.2. CAN Bus – Linienarchitektur, vereinfachte Darstellung

Weitere mögliche Topologien sind

- Baumtopologie
- Ringtopologie
- Sterntopologie

Im Vergleich zur Linientopologie kommen diese CAN Architekturen in der Praxis seltener vor (vgl. CAN Bus …ganz einfach!).

2.1.4 Klassen

Weil es mittlerweile mehrere CAN Busrealisierungen gibt und diese sich in Übertragungsraten unterscheiden, werden sie zu Klassen zusammengefasst

CAN Klasse	Datenrate	Leiterlängen	Name	Einsatz
CAN A	$< 10\ kbps$	$< 6{,}7km$	Diagnosebus	Diagnose, Auslesen des Flashspeichers, sicherheitsunkritische Anwendungen
CAN B	$< 125\ kbps$	$< 500m$	Lowspeed-CAN	Beleuchtung, Klimaanlage, Außenspiegel, Verriegelung und Armaturen
CAN C	$< 1\ Mbps$	$< 40m$	Highspeed-CAN	Motorinnenraum, Diagnose

Tabelle 2.1 CAN Klassen

Aufgrund der endlichen Signalausbreitung, die immer kürzer bei steigenden Datenraten wird, wirkt sich die Datenrate direkt auf die Länge der CAN Busleitung aus.

2.2 CAN FD

2.2.1 Eingliederung im ISO/OSI-Referenzmodell

Die internationale Standardisierungsorganisation (International Organisation for Standardisation, kurz ISO), hat u. a. das Schichtenmodell, bekannt als OSI-Modell für Netzwerkprotokolle als Standard definiert. Ein CAN FD Busteilnehmer operiert, genauso wie ein CAN Busteilnehmer, in den untersten Schichten des Referenzmodells (s. Abbildung 2.4), bestehend aus der Sicherungsschicht und Teilen der Bitübertragungsschicht. Die Sicherungsschicht wird zudem in zwei zusätzliche Unterschichten, die Logical Link Control (LLC)- und die Medium Access Control (MAC)-Schicht, aufgeteilt.

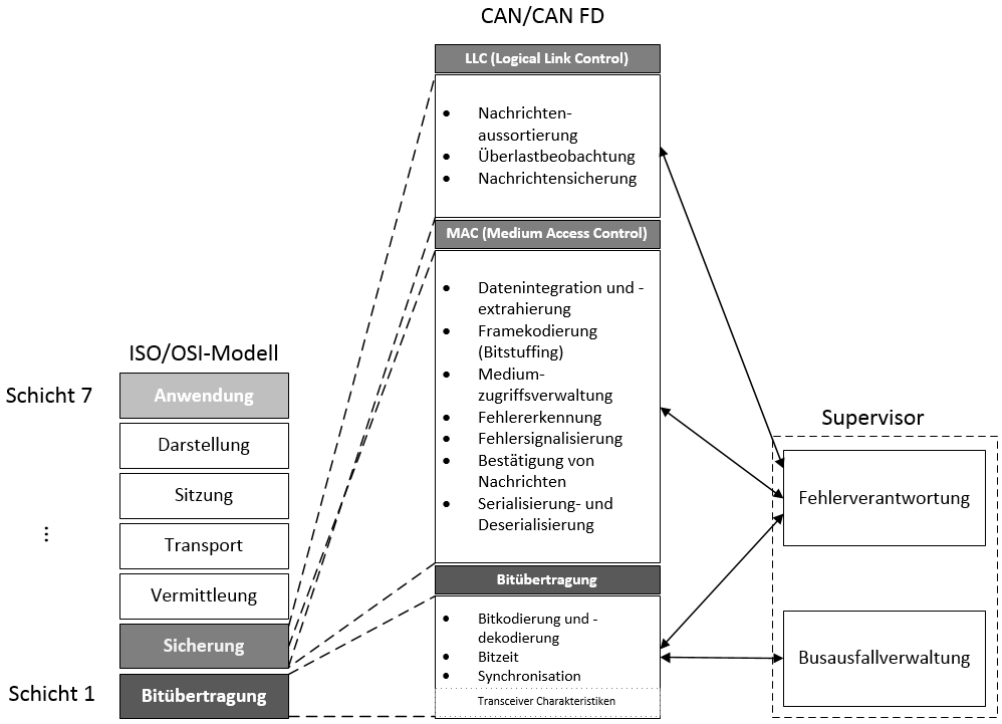

Abbildung 2.3. CAN im ISO/OSI-Modell (vgl. (7) und (4))

2.2.2 Übertragungsmechanismen

Die CAN FD Spezifikation beschreibt Mechanismen der Datenübertragung, die bei CAN FD gelten muss, um mit der Spezifikation konform zu sein. Die Übertragungsmechanismen von CAN FD entsprechen weitestgehend den CAN Übertragungsmechanismen, die in der CAN Spezifikation 2.0 Teil B für CAN festgelegt sind. Diese sind

- Nachrichtenaussortierung
- Überlastbeobachtung
- Nachrichtensicherung
- Datenitegration- und extrahierung
- Nachrichtenkodierung (Bitstuffing)
- Mediumzugriffsverwaltung
- Fehlererkennung und –signalisierung
- Bestätigung von Nachrichten (Acknowledgment)
- Serialisierung und Deserialisierung
- Bitkodierung und –dekodierung
- Bitzeit
- Synchronisation

2.2.2.1 Nachrichtenaussortierung

Nur für den Busteilnehmer relevante Nachrichten werden beim Empfang akzeptiert und alle irrelevanten Nachrichten einfach verworfen.

2.2.2.2 Überlastbeobachtung

Der Empfänger beobachtet die Geschwindigkeit beim Nachrichtenempfang und Nachrichtenverarbeitung. Kommt die Nachrichtenverarbeitung nicht hinterher, wird vom Empfänger eine Überlast gemeldet.

2.2.2.3 Nachrichtensicherung

Eine Kopie der übertragenen Nachricht wird von den Sendern solange gespeichert bis die Nachricht erfolgreich übertragen wird.

2.2.2.4 Serialisierung und Deserialisierung

Nachrichten auf dem Bus werden seriell Bit für Bit übertragen und auf der Empfängerseite wieder zusammengesetzt.

2.2.2.5 Datenintegration und –extrahierung

Jeder CAN FD Busteilnehmer verpackt seine Nutzdaten in ein CAN FD Daten-Frame und verschickt die Nachricht über den CAN Bus. Die Empfänger, für die die Nachricht relevant ist, extrahieren die Nutzdaten für die Weiterverarbeitung aus CAN FD Daten-Frame.

2.2.2.6 Nachrichtenkodierung (Bitstuffing)

CAN oder CAN FD Busteilnehmer senden und empfangen ihre Bits innerhalb einer definierten Bitzeit. Die Bitzeit wird von der lokalen Oszillatorfrequenz abgeleitet. Weil Frequenzen nicht gleichmäßig erzeugt werden und sich auseinander bewegen können (auch als Drift bezeichnet), müssen sich Empfänger mit dem Sender immer wieder neu synchronisieren, wenn während der Nachrichtenübertragung eine Zeit lang keine Spannungsänderung auf dem Bus stattgefunden ist (Unsicherheit des Empfängers, wie viele Bits übertragen wurden). Um eine Spannungsänderung zu erzeugen, fügt der Sender deshalb in den definierten Bitstrombereich während der CAN oder CAN FD Nachrichtenübertragung nach der Stuffbit-Regel Synchronisationsbits, die als Stuffbits (Stopfbits) bezeichnet werden, hinzu. Die Stuffbit-Regel besagt, dass wenn 5 aufeinander folgende Bits gleicher Polarität gesendet werden, muss ein zusätzliches Stuffbit in den Bitstrombereich, das eine inverse Polarität aufweist, eingefügt und gesendet werden. Die Abbildung 2.5 verdeutlicht diesen Sachverhalt, in dem ein übertragener Bitstrom der Stuffbit-Regel unterliegt

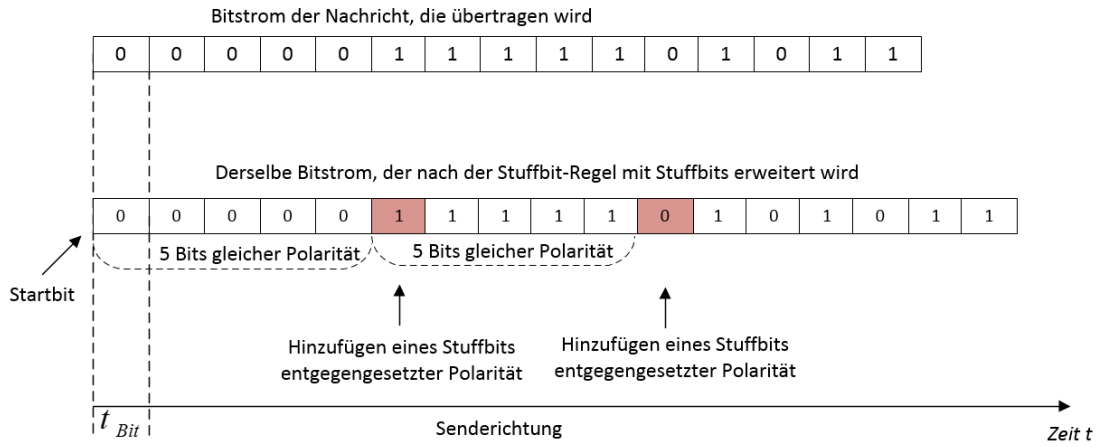

Abbildung 2.4. Einfügen von Stuffbits nach der Stuffbit-Regel

Auf der Seite der Empfänger werden die gesendeten Stuffbits entsprechend der Stuffbit-Regel wieder entfernt, bis schließlich den Empfängern der original gesendete Bitstrom vorliegt.

2.2.2.7 Mediumzugriffsverwaltung

Möchte ein Busteilnehmer eine Nachricht über den CAN Bus senden, so adressiert er nicht einen speziellen Empfänger der Nachricht, sondern kennzeichnet die Nachricht mit einer vorzeichenlosen ganzen Zahl – dem CAN Identifier – die die Priorität der Nachricht angibt. Je kleiner der CAN Identifier ist, desto größer ist die Priorität der Nachricht.

Der Buszugriff erfolgt nach dem Multi-Master-Prinzip. Jedem Busteilnehmer ist es erlaubt sofort mit Übertragung seiner Nachricht zu beginnen, wenn er feststellt, dass keine Anderer Daten über den Bus sendet.

Die gesendete Nachricht wird von allen Busteilnehmern empfangen (Broadcast), auch vom Sender über eine Rückkopplungsschaltung im Transceiver.

Zwischen mehreren gleichzeitig sendenden Busteilnehmern wird während sogenannter Arbitrierungsphase über die gesendeten CAN Identifier zwischen den Busteilnehmern ermittelt, welcher Busteilnehmer letztendlich seine Nachricht weiter übertragen darf. Dabei wird der Buszustand von mit jedem gesendeten Bit beobachtet. Das Verfahren zur Erkennung von Kollisionen auf dem Bus bei mehreren gleichzeitigen Sendern wird als Carrier-Sense-Multiple-Access-Verfahren mit Kollisionserkennungsmechanismus bezeichnet.

Welcher Buszustand sich letztendlich bei mehreren gleichzeitigen Sendern einpendelt, entscheidet das am Bus angewandte Verdrahtungsprinzip. Der CAN Bus arbeitetet nach dem Wired-AND-Prinzip. Beim diesem Prinzip muss nur einer von mehreren gleichzeitig übertragenden Busteilnehmern die 0 übertragen, damit alle Busteilnehmer den Buszustand entsprechend der 0 beobachten. Beim Wired-OR-Prinzip funktioniert das Ganze in entgegengesetzter Richtung. In der CAN Terminologie wird der

Sendezustand, der sich gegenüber dem anderen Sendezustand am Bus durchsetzt, als dominant und dem dominanten Zustand unterlegene Sendezustand als rezessiv bezeichnet.

Derjenige Busteilnehmer, der die Arbitrierung gewinnt (der gesendete Zustand entspricht während der Arbitrierungsphase zu jeder Zeit dem Buszustand), behält das Recht seine Nachricht weiter über den Bus zu übertragen. Alle anderen Busteilnehmer, werden zu den Empfängern dieser Nachricht. Die nachfolgende Abbildung verdeutlicht die Busarbitrierung am Beispiel

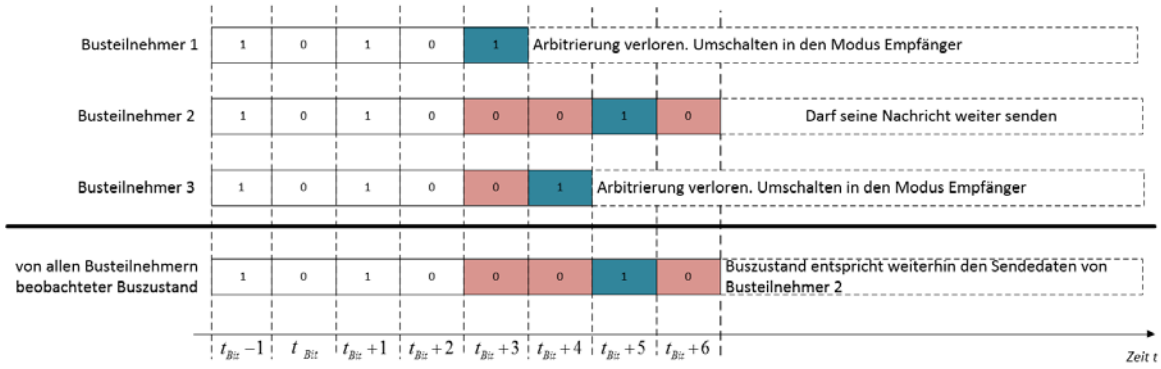

Abbildung 2.5. Busarbitrierung

Alle Busteilnehmer starten simultan während der Zeit t_{Bit} mit einer neuen Nachrichtenübertragung. Dabei ist $k \in Z$ das Zeitintervall zum Übertragen eines Bits, das aus der Datenrate abgeleitet wird. Während die nicht ausgefüllten Kästchen mit Bitzuständen 1 und 0 noch identisch aussehen, unterscheidet sich die Übertragung der Daten des Busteilnehmers 1 ab dem Zeitpunkt $t_{Bit} + 3$ von der Datenübertragung der anderen Busteilnehmer 2 und 3. Die übertragene 0 von Busteilnehmer 2 und Busteilnehmer 3 schlägt auf dem Bus durch und überschreibt die übertragene 1 vom Busteilnehmer 1. Im Zeitpunkt $t_{Bit} + 4$ wird der Busteilnehmer 1 zum Empfänger einer Nachricht, des anderen Busteilnehmers. Zur gleichen Zeit überträgt der Busteilnehmer 2 die 0, die die übertragene 1 des Busteilnehmers 3 auf dem Bus überschreibt, so dass ab dem Zeitpunkt $t_{Bit} + 5$ der Busteilnhmer 3 auch zum Empfänger der Nachricht eines anderen Busteilnehmers wird. Letztendlich beobachtet jeder Busteilnehmer den Sendezustand des Busteilnehmers 2, und der Busteilnehmer 2 hat das Recht erlangt seine Nachricht bis zum Ende zu übertragen.

2.2.2.8 Fehlererkennung und –signalisierung

Fehlerhaft übertragene oder empfangene CAN Daten-, Remote- und CAN FD Daten-Frames werden erkannt und mit einem CAN Error-Frame signalisiert. Dabei werden folgende Fehlerarten unterschieden:

Bit-Fehler

Nachdem ein CAN oder CAN FD Busteilnehmer den Buszugriff erhält, beobachtet der Busteilnehmer die weiteren gesendeten Daten auf dem Bus. Unterscheidet sich der Sendezustand von dem Empfangszustand, wird ein Fehler signalisiert.

Bitstuffing-Fehler

Auf der Bitübertragungsschicht werden auch Regelverletzungen der Stuffbit-Regel überprüft. Liegt eine Verletzung der Stuffbit-Regel vor, wird ein Fehler signalisiert.

Form-Fehler

Einige Felder im CAN Daten-, Remote- oder CAN FD Daten-Frames haben eine festgelegte Form (z. B. das ACK- oder CRC-Feld). Wird diese Form nicht eingehalten, wird ein Fehler signalisiert.

CRC-Fehler (Cyclic Redundancy Check)

Im CRC-Feld eines CAN-Daten, Remote- oder CAN FD Daten-Frames wird eine Prüfsumme auf der Seite des sendenden Busteilnehmers eingefügt. Auf der Seite des Empfängers wird eine eigene Prüfsumme berechnet und mit der eingetragenen Prüfsumme der Nachricht verglichen. Stimmen sie nicht überein, wird ein Fehler signalisiert.

ACK-Fehler (Acknowledgment)

Fällt die Nachrichtenbestätigung aus, wird ein Fehler signalisiert.

2.2.2.9 Bestätigung (Acknowledgment)

Jeder gesendete CAN Daten-, Remote- oder CAN FD Daten-Frame muss von den Empfängern in dem dafür vorhandenen Feld bestätigt werden. Es genügt, wenn nur ein Empfänger die Nachricht quittiert.

2.2.2.10 Bitkodierung- und dekodierung

Zustände von Bitströmen bei einer CAN oder CAN FD Nachrichtenübertragung werden in Spannungswerte kodiert und für eine nominale Bitzeit, die sich aus der Formel (1/Datenrate) ergibt auf dem Bus angelegt. Auf der Emfängerseite wandelt jeder Empfänger während der nominalen Bitzeit die Spannungswerte zurück in Bits um. Das Verfahren zum Übertragen von Bitzuständen in Form von zwei Spannungspegeln, bei dem auch der Ruhezustand der Leitung während einer nominalen Bitzeit der Übertragung eines rezessiven Bitzustands entspricht, wird als Non-Return-To-Zero-Verfahren bezeichnet.

2.2.3 Nachrichtenformate

2.2.3.1 Überblick

Für CAN werden seit der CAN Einführung vier Nachrichtenformate festgelegt. Diese sind

- **Daten-Frame**
- **Remote-Frame**
- **Error-Frame**
- **Overload-Frame**

Daten-Frames werden von den Busteilnehmern zum Versenden von eigenen Nutzdaten verwendet. Ein Busteilnehmer kann Nutzdaten von einem anderen Busteilnehmer mit einem Remote-Frame anfordern. Error-Frames werden zur Fehlersignalisierung benutzt, falls während der Nachrichtenübertragung von Daten- oder Remote-Frames Sender- oder Empfänger Übertragungsfehler erkennen. Kommt ein Busteilnehmer nicht mit der Verarbeitung von empfangenen Nachrichten hinterher, kann er nach dem Empfang eines Daten- oder Remote-Frames seine Überlast mit dem Overload-Frame signalisieren.

CAN FD unterstützt alle CAN Nachrichtenformate und definiert ein neues Daten-Frame Format zur Unterstützung einer schnelleren Datenrate.

In den folgenden Abschnitten wird der Aufbau des jeweiligen Frametyps erläutert. Jedes Frame besteht aus einer Folge von Bits. Die benannten Bitfelder bestehen selbst entweder aus einfachen Bits oder Bitfolgen, deren Anzahl dann neben dem Feldnamen in runden Klammern angegeben ist.

Zur Kennzeichnung, wie die Bits übertragen werden, wird auf die CAN Terminologie zurückgegriffen. Was rezessiv oder dominant bedeutet, kann aus dem Abschnitt 2.1.6.7. nachgelesen werden.

2.2.3.2 CAN Daten- und Remote-Frames

Ein CAN Daten- oder Remote-Frame wird aus folgenden Feldern aufgebaut

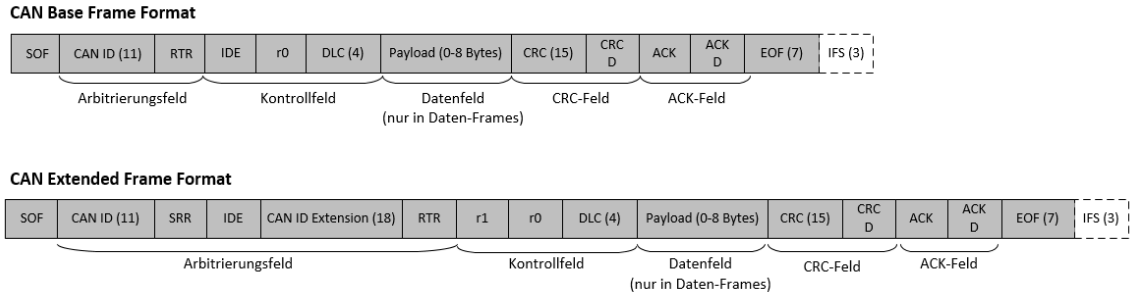

Abbildung 2.6. CAN Daten- und Remote-Frame im Base Frame und Extended Frame Format (vgl. (7))

Ein Remote-Frame unterscheidet sich von einem Daten-Frame nur im fehlenden Nutzdatenfeld (Payload). Die in der Abbildung 2.10 dargestellten Bitfelder werden wie folgt definiert

SOF

Start Of Frame kennzeichnet den Beginn eines Daten- oder Remote-Frames. Das SOF Bit wird stets dominant übertragen.

CAN ID

Vorzeichenlose ganze Zahl, die Priorität der Nachricht während der Busarbitrierung kennzeichnet. CAN Daten- und Remote-Frames existieren im Base Frame und Extended Frame Formaten. Der Unterschied dieser beiden Nachrichtenformate liegt in der Länge des CAN Identifiers und zusätzlichen Bitfelder SRR und r1, die nur im Extended Frame Format vorkommen. Im Base Frame Format ist die CAN ID 11 und im Extended Frame Format 29 Bit lang. Die vorderen 11 Bits, die im Base Frame und im Extended Frame Format nach dem Bit SOF Bit kommen, werden als Base CAN Identifier bezeichnet.

In einem CAN Netzwerk sollte die CAN ID nur von einem Busteilnehmer zum Versenden von Daten-Frames verwendet werden. Ist das nicht der Fall, können die Folgen, die daraus resultieren, unabsehbar werden. Es gibt dennoch zwei Ausnahmen, unter denen die CAN ID identisch zum Versenden von Daten-Frames zwischen zwei Busteilnehmern verwendet werden kann

(a) Derselbe CAN Base Identifier wird von einem Busteilnehmer zum Versenden von Daten-Frames im Base Frame und von einem anderen Busteilnehmer zum Versenden von Daten-Frames im Extended Frame Format verwendet. Die Auflösung wird über das Feld RTR des CAN Daten-Frames im Base Frame Format, das dominant, und das Feld SRR des CAN Daten-Felds im Extended Frame Format, das rezessiv übertragen wird, zugunsten des CAN Daten-Frames im Base Frame Format aufgelöst.

(b) Versenden von Daten- und Remote-Frames mit gleicher CAN ID. Die Auflösung findet im Feld RTR zugunsten des Daten-Frames statt, das von beiden Nachrichtentypen mit unterschiedlicher Polarität gesendet wird.

RTR

Remote Transmission Request. Kennzeichnet ob die Nachricht ein Daten- oder Remote-Frame ist. Bei einem Daten-Frame wird dieses Feld dominant und bei einem Remote-Frame rezessiv gesendet.

SRR

Substitute Remote Request. Dieses Bitfeld kommt ausschließlich in CAN Daten- oder Remote-Frames im Extended Frame Format. Das SRR Bit wird rezessiv übertragen, damit CAN Daten-Frames im Base Frame Format stets eine höhere Priorität gegenüber CAN Daten-Frames im Extended Frame Format mit gleicher Base CAN ID haben.

IDE

IDentifier Extension. Dieses Bitfeld wird dominant in Daten- oder Remote-Frames im Base Frame Format und rezessiv in Daten- oder Remote-Frames im Extended Frame Format zur Unterscheidung zwischen diesen beiden Formaten übertragen.

r1, **r0**

Reserved. Reservierte Felder, die für zukünftige Erweiterungen von CAN Daten- und Remote-Frames vorhanden sind. Sie werden beide dominant übertragen.

DLC

Data Length Code. Kennzeichnet bei Daten-Frames die Länge des nachfolgenden Nutzdatenfelds (Payload) in Bytes. In einem CAN Remote-Frame trägt der Busteilnehmer hier die Größe der Nutzdaten in Bytes ein, die er vom antwortenden Busteilnehmer haben möchte. Das DLC Feld besteht aus 4 Bits. In 4 Bits lassen sich Zahlen im Bereich von 0 bis 15 binär darstellen. In einem CAN Daten-oder Remote-Frame werden darüber Nutzdaten über Zahlenwerte von 0 bis 8 adressiert. Die übrigen Zahlenwerte 9 bis 15 verbleiben bei einem CAN Daten- oder Remote-Frame ohne Bedeutung.

Payload

Datenfeld zum Übertragen von Nutzdaten. In einem CAN Daten-Frame können Nutzdaten in Größen von 0 bis 8 Bytes gesendet werden. Dieses Feld fehlt beim CAN Remote-Frame.

CRC, CRC Delimiter

Cyclic Redundancy Check. Dieses Feld enthält die Prüfsumme, die über eine Polynomdivision berechnet, vom Sender in dieses Feld eingetragen und auf der Seite des Empfängers zur Fehlererkennung verwendet wird. Mit dem speziellen Generatorpolynom, der sowohl dem Sender als auch den Empfängern bekannt ist, wird die Wahrscheinlichkeit der Fehlererkennung mit einer Hamming-Distanz[1] von 6 angegeben. Damit lassen sich 5 beliebig eingestreuten Fehler identifizieren. Zur Berechnung der Prüfsumme wird ein Polynom vom Grad 15 benutzt, aufgrund dessen dieses Feld 15 Bit lang ist.

Das CRC Feld wird anschließend mit einem CRC Trenner (Delimiter, in der Abbildung 2.10 mit CRC D gekennzeichnet), der 1 Bit lang ist und rezessiv gesendet wird, abgeschlossen.

ACK, ACK Delimiter

Acknowledgment. Bestätigung. Das korrekte Empfangen eines Daten- oder Remote-Frames wird dem Sender in diesem Feld quittiert. Dafür sendet der Sender an dieser Stelle ein rezessives Bit. Wird die Nachricht erfolgreich bis zu diesem Feld empfangen, so wird das rezessiv gesendete Bit mit einem

[1] Als eine Hamming-Distanz wird bei einem Vergleich von zwei Binärwörtern die Anzahl sich unterscheidender Bitfelder bezeichnet.

dominanten Bit von den Empfängern überschrieben. Es reicht, wenn nur ein Empfänger dieses Bitfeld dominant überschreibt, damit der Sender das Frame als erfolgreich versendet betrachtet.

Das Acknowledgment-Feld besitzt genauso, wie das CRC Feld, einen Trenner (in der Abbildung 2.10 mit ACK D gekennzeichnet), der 1 Bit lang ist und rezessiv gesendet wird.

EOF

End Of Frame. Besteht aus 7 rezessiv gesendeten Bits und kennzeichnet das Ende eines Daten- oder Remote-Frames.

IFS

Das Interframe Space ist eine Bitfolge bestehend aus 3 rezessiv gesendeten Bits, die nach einer beliebigen Nachricht gesendet werden, um allen anderen Busteilnehmern die Freigabe des Busses anzudeuten.

2.2.3.3 CAN FD Daten-Frame

Das CAN FD Daten-Frame entsprechend der CAN FD Spezifikation (vgl. (4)) wird wie folgt aufgebaut

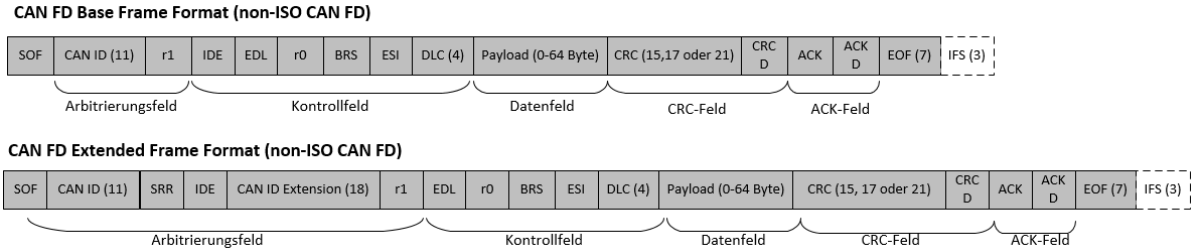

Abbildung 2.7. CAN FD Daten-Frame im Base Frame und Extended Frame Format (vgl. (4))

Nachfolgend werden nur die Bitfelder beschrieben, die sich im Vergleich zum CAN Daten-Frame geändert haben oder neu dazugekommen sind. Alle Bits und Bitfelder, die identisch sind, werden bewusst ausgelassen.

r1

Reserved. Das r1 Feld, das in den CAN Daten-Frames nur im Extended Frame Format vorkommt, befindet sich im CAN FD Base Frame Format an die Stelle des RTR Bits. Die Funktion von r1 ist dieselbe, wie des RTR Bits im CAN Daten-Frame.

EDL

Extended Data Length. Erweiterte Datenlänge. Das Feld EDL wird zur Unterscheidung zwischen CAN FD und CAN Frames verwendet. EDL steht an der Stelle von r0 von CAN Daten- oder Remote-Frames und wird bei CAN FD rezessiv übertragen.

BRS

Bit Rate Switch. Kennzeichnet den Wunsch des Senders zur Umschaltung auf die schnellere Datenrate. Weil ein CAN FD Daten-Frame mit zwei Datenraten versendet werden kann, wird die Nachrichtenübertragung in eine Arbitrierungs- und eine Datenphase eingeteilt.

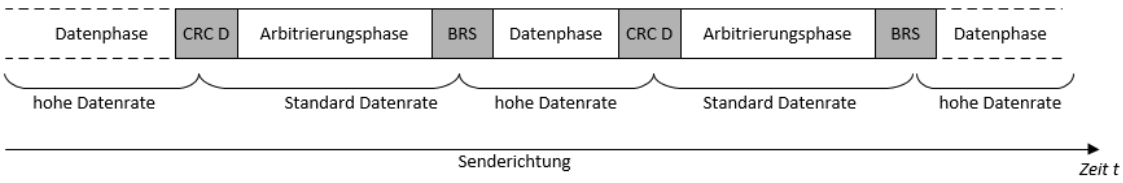

Abbildung 2.8. Übertragungsphasen eines CAN FD Daten-Frames

Die Arbitrierungsphase, die bis zum BRS Feld eines CAN FD Daten-Frames geht, wird mit einer CAN Bus üblichen Standard Datenrate übertragen. In dieser Phase lassen sich auch ältere CAN Nachrichten übertragen, die dann mit CAN FD über die CAN ID um den Buszugriff konkurrieren.

Ist der Buszugriff zugunsten des CAN FD Busteilnehmers entschieden worden, kann er BRS Feld entscheiden, ob er weitere Nachrichtendaten mit einer hohen Datenrate senden möchte. Wenn ja, sendet er in dem BRS Feld ein rezessives Bit. Beim Empfang dieses Bitfelds schalten alle CAN FD-fähigen Empfänger auf den Empfang von weiteren Nachrichtendaten mit einer hohen Datenrate, die allen Empfängern bekannt ist, um. Damit wird die Übertragung nachfolgender Bitfelder des CAN FD Daten-Frames mit der hohen Datenrate initiiert. Die Datenphase verläuft bis zum Ende des CRC Felds. Im CRC Trenner wechseln der CAN FD Sender und alle CAN FD Empfänger wieder zur Standard Datenrate zurück, um die restlichen Nachrichtenbits zu empfangen und erneut über den Buszugriff zu konkurrieren.

ESI

Error State Indicator. Hier trägt der sendende CAN FD Busteilnehmer seinen Fehlererkennungszustand ein. Obwohl der Fehlererkennungszustand auch bei CAN Busteilnehmern verwaltet wird, existiert dieses Feld nur bei CAN FD. Für CAN und CAN FD Busteilnehmer werden zwei Fehlererkennungszustände definiert. Jeder von ihnen kann hiernach sich entweder im Fehler-passiven oder Fehler-aktiven Fehlererkennungszustand befinden. Dieser Zustand hat den entscheidenden Einfluss darauf, welche Bits in den Error-Flags der Busteilnehmer beim Senden eines CAN Error-Frames (s. Abschnitt 2.2.3.4) während einer Fehlererkennung einträgt. Busteilnehmer im aktiven Fehlererkennungszustand senden an den Stellen von Error-Flags dominante Bits, Busteilnehmer im passiven Fehlererkennungszustand senden an den Stellen von Error-Flags rezessive Bits. Jeder CAN

oder CAN FD Busteilnehmer beginnen in einem aktiven Fehlererkennungszustand. Der anschließende Fehlererkennungszustand wird über interne Sende- und Empfangszähler jedes Busteilnehmers verwaltet. Nach bestimmten Regeln werden die Zähler während einer Fehlererkennung, ob nun durch das Senden oder durch das Empfangen eines CAN Daten- oder Remote-Frames oder CAN FD Daten-Frames, erhöht oder verringert. Wird eine bestimmte Schwellengrenze des Sendezählers erreicht, wechselt der Busteilnehmer vom aktiven in den passiven Fehlererkennungszustand. Wird die höchste Schwellengrenze des Sendezählers erreicht, schaltet sich der Busteilnehmer automatisch von der weiteren CAN Kommunikation ab.

DLC, Payload

Wie bei CAN Daten- oder Remote-Frames, adressiert das DLC Feld mit einer Zahlendarstellung von 0 bis 15 in 4 Bits die Länge der Nutzdaten in Bytes. Bei CAN FD ist das Nutzdatenfeld um weitere 7 Nutzdatenübertragungsgrößen erweitert worden. Neben üblichen 0 bis 8 Bytes können zusätzlich Nutzdaten in 12, 16, 20, 24, 32, 48 und 64 Bytes gesendet werden. Während bei CAN Daten-Frames die Zahlenwerte zur Adressierung von Nutzdaten von 9 bis 15 ohne Bedeutung sind, werden sie bei CAN FD vollständig zur Adressierung erweiterter Nutzdatenlängen verwendet.

CRC

Damit auch weiterhin eine Hamming-Distanz von 6 bestehen bleibt und die Wahrscheinlichkeit, dass alle 5 eingestreuten Fehler während einer CAN FD Nachrichtenübertragung von größeren Nutzdatenmengen erkannt werden, sind für CAN FD weitere Polynome definiert worden. Für die Nutzdatenübertragung von 12 oder 16 Byte wird ein Polynom vom Grad 17 und für alle anderen größeren Nutzdatenmengen ein Polynom vom Grad 21 zur Berechnung der Prüfsumme verwendet. Dementsprechend variiert das CRC Feld in der Bit Länge von 15, 17 oder 21 Bit.

Während des Normierungsprozesses wurde bei der ersten CAN FD Spezifikation von 2012 eine Bitfehler-Erkennungsschwäche in dem CRC Feld theoretisch festgestellt (vgl. (8)). Obwohl diese Bitfehler-Erkennungsschwäche in der Praxis kaum oder sehr selten vorkommen kann, wollte Robert Bosch GmbH keine Fehler eingehen und die Zukunft von CAN FD absichern, so dass die zuvor im Jahr 2012 erschienene CAN FD Spezifikation 1.0 einer Revision unterlag und dabei in die CAN FD Spezifikation neue Details eingeflossen sind. Die neue CAN FD Spezifikation, die nun zum ISO11898-1:2015 Standard geworden ist, beseitigt die entdeckte Bitfehler-Erkennungsschwäche. Nichts desto trotz gibt es auf dem Markt, seit der Einführung der ersten CAN FD Spezifikation, die ersten Hersteller der CAN FD-fähigen Controller, die die ursprüngliche CAN FD Spezifikation erfüllen. Deshalb schlägt die CiA für die ursprüngliche CAN FD Variante den Namen non-ISO CAN FD und für den aktuellen ISO11898-1:2015 Standard den Namen ISO CAN FD zu verwenden (vgl. (9)). In der Zukunft wird sich dennoch der ISO Standard durchsetzen. Die Abbildung 2.10 zeigt das veränderte CAN FD Daten-Frame im Base Frame und Extended Frame Format

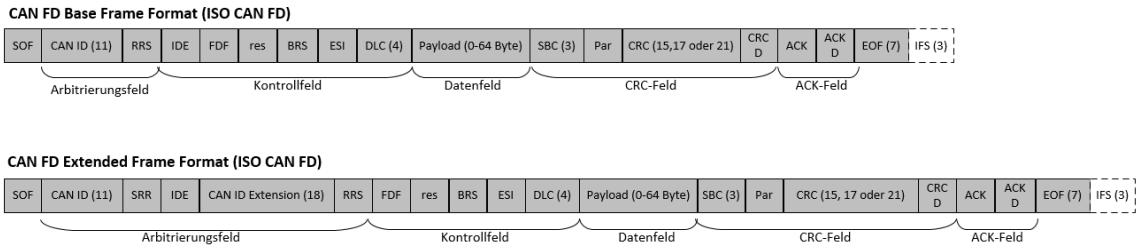

Abbildung 2.9. CAN FD Daten-Frame (ISO CAN FD) im Base Frame und Extended Frame Format

Folgende Veränderungen gegenüber der ersten CAN FD Spezifikation sind durchgeführt worden

- **Umbenennung des Felds r1 in das Feld RRS** (Remote Request Substitution)

- **Umbenennung des Felds EDL in FDF**
  Das Feld FDF steht für Flexible Data Frame und hat dieselbe Bedeutung, wie das Feld EDL selbst.

- **Einführung des Felds SBC im CRC Feld**
  SBC steht für den Stuffbit Zähler. Die Bitsequenz, die der Stuffbit-Regel unterliegt, ist im neuen ISO CAN FD Nachrichtenformat bis zum Ende des Nutzdatenfelds verkleinert worden, während bei allen anderen CAN Daten- und Remote-Frames oder bei der CAN FD Spezifikation 1.0 das CRC-Feld bis zum CRC Trenner mit in die Stuffbit-Berechnung einbezogen wird. An ersten Stelle des CRC Felds kommt nun der Stuffbit Zähler, der 3 Bit lang ist. Der Stuffbit Zähler enthält die Anzahl von dynamisch eingefügten Stuffbits Modulo 8 (einen Rest, der sich nicht durch 8 teilen lässt) des Senders. Die Empfänger können dadurch noch einmal sichergehen, wie viele Stuffbits der Sender in die gesendete Nachricht eingefügt hat.

- **Einführung des Felds Paritätsbit**
  Das Paritätsbit[2] sichert zusätzlich den vorangehenden Stuffbit Zähler gegen Störeinwirkungen ab.

- **Einführung von festen Stuffbits in der CRC-Bitsequenz**
  Nach dem Paritätsbit folgt anschließend das erste feste Stuffbit der CRC-Bitsequenz, das eine inverse Polarität zum Paritätsbit besitzt. Weiterhin werden jeweils zusätzliche feste Stuffbits in 4er Schritten in die CRC-Bitsequenz eingefügt, die jeweils eine andere Polarität zum letzten Bit der 4er Bitsequenz, nach der sie eingefügt werden, haben.

[2] Ein Paritätsbit kennzeichnet ein Kontrollbit und gibt bei einer Bitsequenz an, ob die Anzahl der vorhandenen 1er Bitzustände der Bitsequenz gerade oder ungerade ist. Ist die Anzahl von 1en gerade, wird von einer geraden, sonst ungeraden Parität gesprochen.

2.2.3.4 CAN Error-Frame

Stellt ein Empfänger (der zugleich auch der Sender sein kann) einen, der möglichen Fehler fest, signalisiert er ihn mit einem Error-Frame. Dabei kann ein Error-Frame an beliebiger Position innerhalb eines Daten- oder Remote-Frames gesendet werden. Ein Error-Frame wird aus folgenden Bitfeldern aufgebaut

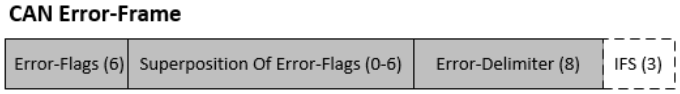

Abbildung 2.10. CAN Error-Frame (vgl. (4))

Error-Flags

Die Error-Flags bestehen aus 6 aufeinander folgenden Bits gleicher Polarität. Die Entscheidung, welche Polarität in diesen Bitstellen eigentragen wird, hängt von dem Busteilnehmer und seinem Fehlererkennungszustand ab. In dem aktiven Fehlererkennungszustand werden an der Stelle von Error-Flags 6 dominante Bits eingetragen. Sendet ein CAN oder CAN FD Busteilnehmer einen Error-Frame im aktiven Fehlererkennungszustand, so zerstören die 6 dominant gesendeten Bits jedes auf dem Bus gesendete Daten- oder Remote-Frame aufgrund einer Nichtbeachtung der Bitstuffing-Regel. Alle anderen Busteilnehmer, die daraufhin auch den Fehler bemerken, senden ihrerseits auch einen Error-Frame. In einem passiven Fehlererkennungszustand werden, an den Stellen von Error-Flags, 6 rezessive Bits eingetragen, die keinen zerstörerischen Einfluss auf die Übertragung eines Daten- oder Remote-Frames haben.

Superposition von Error-Flags

Da die Signale eine Ausbreitungszeit auf dem Bus aufweisen und alle CAN Busteilnehmer, die einen Error-Frame erkennen, selbst mit einem Error-Frame antworten, muss für den Versand von Error-Flags zusätzlich Zeit einkalkuliert werden, die maximal doppelt so hoch sein kann, wie die Zeit zum Versenden der Error-Flags selbst. Dieses Verhalten wird als Superposition bezeichnet und kann eine zusätzliche Zeit zum Versenden von 0 bis 6 Bits betragen.

Error-Delimiter

Der Error-Delimiter definiert den Trenner, der aus 8 rezessiv gesendeten Bits besteht, und kennzeichnet das Ende des gesendeten Error-Frames.

2.2.3.5 CAN Overload-Frame

Wenn ein Busteilnehmer nicht mit der Verarbeitung von Nachrichten hinterherkommt, kann er diesen Zustand mit einem Overload-Frame den anderen Busteilnehmern und damit eine Pause für sich reservieren.

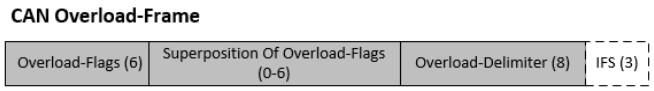

Abbildung 2.11. CAN Overload-Frame (vgl. (7))

Jeder Busteilnehmer, der einen Overload-Frame empfängt, antwortet auch mit einem Overload-Frame. Overload-Frames werden nach dem Empfang eines CAN Daten- oder Remote-Frames oder CAN FD Daten-Frames innerhalb des Interframe Space-Fensters gesendet. Maximal können zwei Overload-Frames nacheinander versandt werden.

CAN Error-Frame und CAN Overload-Frame sehen in ihrem Aufbau identisch aus. Auch das Overload-Frame beginnt mit 6 Overload-Flags, die 6 Bit lang sind und alle dominant gesendet werden. Im Gegensatz zu einem Error-Frame, dürfen diese 6 Bits nur dominant übertragen werden. Weil alle Empfänger beim Erkennen eines Überlastzustands auch mit einem Overload-Frame antworten, entspricht die Erläuterung zur Superposition, der des Error-Frames aus dem vorangehenden Abschnitt.

Genau, wie das Error-Frame, wird ein Overload-Frame mit einem Overload-Trenner (Overload-Delimiter), der 8 Bit lang ist und bei dem alle Bits rezessiv gesendet werden, abgeschlossen.

2.3 Simulation

2.3.1 Der Simulationsbegriff

Eine Simulation bildet mit Hilfe eines Computers möglichst realitätsnah Geschehen der Wirklichkeit ab. Sie ist aus Sicherheits- und Kostengründen fast für jede Problemstellung notwendig, um diese von der Realität zu lösen und sie abstrakt zu behandeln. Simulationen liefern Erkenntnisse, die auf die Wirklichkeit übertragen werden können.

2.3.2 System-Level Simulation

System-Level Simulationen adressieren die Architektur und funktionale Performanz komplexer Systeme. Eine System-Level Simulation ist ein Satz von Techniken die mit Hilfe des Computers Aktionen eines komplexen Systems oder Prozesse in der Simulation nachbilden. Computer werden dazu verwendet, um Modelle zum Zweck der Beschreibung oder Visualisierung einer komplexen Interaktion aufzubauen. Die Komplexität des Systems entsteht aufgrund stochastischer Natur von Ereignissen, Regeln der Interaktionen von Elementen und der Wahrnehmung des Systemverhaltens als Ganzes nach der Zeit.

2.3.3 Ereignisgesteuerte diskrete Simulation

Eine diskrete Simulation verwendet die Zeit, um nach messbaren oder zufälligen Zeitintervallen bestimmte Ereignisse hervorzurufen, die zu Änderung des Systemzustands führen. Ein Ereignis ist die Bitte an das Betriebssystem ein stattgefundenes Geschehen zu behandeln. Die hauptsächliche

Anwendung von diskreten Simulationen zur Lösung von praxisrelevanten Problemen findet meistens in Produktions- und Logistikumfeld statt. Die Modelle dieser Art von Simulation lassen sich gut durch die standardisierte Elemente, wie Zufallszahlen, Warteschlangen und Wahrscheinlichkeiten entwickeln. Nach genügend häufiger Wiederholung einer simulierten Problemstellung können anschließend Aussagen über die Wahrscheinlichkeit des zu erwartenden Systemzustands getroffen werden.

2.4 OMNeT++

Zu der Kategorie der ereignisgesteuerten diskreten Simulationssoftware gehört das OMNeT++ Framework. Es ist ein Framework, weil es neben der OMNeT++-Entwicklungsumgebung, die auf der open-source Entwicklungsumgebung Eclipse basiert, eine Klassenbibliothek zur Erzeugung von Simulationsmodellen und nützliche Werkzeuge, wie das Result Analysis Tool, zur anschließenden Analyse von Ergebnissen einer simulierten Problemstellung, anbietet. OMNeT++ ist besonders für die Simulation von Netzwerkprotokollen und -architekturen geeignet.

2.4.1 Modellentwurf

Modelle unter OMNeT++ werden in einer Network Description Language (NED) entwickelt. Die NED unterstützt Sprachelemente zum hierarchischen Aufbau eines komplexen Systems. Dabei kann die Architektur des Systems in eine Reihe von Subsystemen und elementaren Komponenten zerlegt werden, die in separaten `.ned` Dateien entwickelt werden können. Die Hierarchietiefe der Architektur ist dem Anwender selbst überlassen.

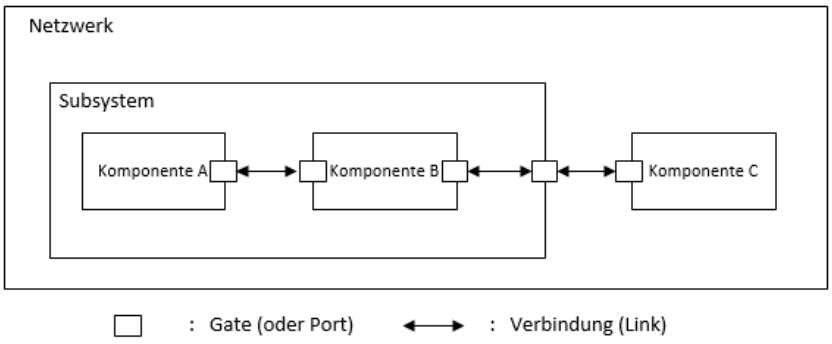

Abbildung 2.12. Modellentwurf unter OMNeT++ (vgl. (10))

Die Subsysteme werden unter OMNeT++ als zusammengesetzte Module (`compound modules`) und elementaren Komponenten als einfache Module (`simple modules`) bezeichnet und mit NED-Sprachelementen `module`, für ein zusammengesetztes Modul und `simple`, für ein einfaches Modul, erzeugt. Die Komponente in der obersten Hierarchieebene definiert das Netzwerk selbst und ist genau genommen auch ein zusammengesetztes Modul, das aber mit dem NED-Sprachelement `network`

ausgezeichnet wird. Der Unterschied zwischen den zusammengesetzten und einfachen Modulen liegt darin, dass einfache Module eine Logik implementieren können, die dann in C++-ähnlichen Klassen, bestehend aus .h- und .cc-Dateien, realisiert wird. Die zusammengesetzte Module bilden ein Subsystem ab und definieren eine logische Einheit, die entweder aus weiteren zusammengesetzten Modulen oder einfachen Modulen bestehen kann.

Zusammengesetze und einfache Module können Eigenschaften definieren. Die Eigenschaften von zusammengesetzten und einfachen Modulen können in `.ned`-Dateien mit Default-Argumenten belegt werden.

Die NED-Sprache unterstützt Vererbungsmechanismen, die aus den objekt-orientierten Programmiersprachen bekannt sind und mit denen einfache Modulerweiterungen erzeugt werden können.

2.4.2 Kommunikationsform

Einfache Module und zusammengesetzte Module können Gates (oder Ports) angeben, um mit anderen zusammengesetzten oder einfachen Modulen über diese Gates kommunizieren zu können. Die Gates von Subsystemen definieren die Schnittstellen mit der Außenwelt, die ausgehende Nachrichten von internen einfachen Modulen entweder nach außen oder eingehende Nachrichten zu internen einfachen Modulen weiterleiten können. Gates können Namen haben. Den Gates kann in der Gate-Definition eine Kommunikationsrichtung als Eingang, Ausgang oder Beides zugewiesen werden.

Die Gates von einfachen und zusammengesetzten Modulen können über Verbindungspfade (Channels) miteinander verbunden werden. Die Verbindungspfade können als ideal oder verknüpft mit einer Verzögerungszeit modelliert werden. Eine Form der Gate-Verbindung ohne Channels unter Angabe von Ausgangs- und Eingangsgates von Modulen ist erlaubt. Aus den C++-Klassen können Module über direkte Methodenaufrufe innerhalb des Netzwerks die Verhaltensänderung anderer Module anstoßen.

2.4.3 Nachrichtendefinition

In OMNeT++ können Module, neben direkten Funktionsaufrufen, einander Nachrichten in Form von Ereignissen senden. Der Entwurf eigener Nachrichten findet in Nachrichten-Dateien mit der Endung `.msg` statt. Eigene Nachrichtenentwürfe leiten von den OMNeT++-Klassen `cMessage` oder `cPacket` ab, wenn sie in den `.msg` Dateien mit Schlüsselwörtern `message` oder `packet` ausgezeichnet werden.

In eigenen Nachrichten können Parameter definiert werden. Der Nachrichten-Compiler übersetzt während der Kompilierzeit die `.msg`-Dateien in die zugehörigen C++-Dateien. Die C++-Dateien erhalten automatisch sowohl die nützliche Funktionen der abgeleiteten Klassen als auch Funktionen zum Setzen und Abfragen von eigenen definierten Parametern, die als Getter und Setter bezeichnet werden.

Nachrichten, die von der cPacket-Klasse ableiten, stehen sowohl Funktionen zum Setzen der Länge einer Nachricht in Bits oder Bytes als auch zum Einbetten und Extrahieren weiterer cPacket- oder seiner Derivat-Klassen bereit. Das Setzen von Bit oder Byte Längen einer cPacket-Nachricht erfolgt mit dem Aufruf der Funktionen setBitLength oder setByteLength, die die Länge eines Packets in Bits oder Bytes erwarten. Für die Setter-Funktionen gibt es in der cPacket-Klasse der OMNeT++-Klassenbibliothek korrespondierende Getter-Funktionen getBitLength und getByteLength, die eine entsprechende Anzahl von Bits oder Bytes zurückliefern. Für die Einbettung eines Packets in ein anderes Packet steht die Funktion encapsulate zur Verfügung, die eine einzubettende cPacket-Nachricht erwartet. Die Extrahierung einer cPacket-Nachricht erfolgt mit der Funktion decapsulate. Wird eine cPacket-Nachricht, die eine Bitlänge aufweist, in eine andere cPacket-Nachricht eingekapselt, wächst die Bitlänge der äußeren cPacket-Nachricht um die Bitlänge der inneren cPacket-Nachricht automatisch mit. Beim Extrahieren einer inneren cPacket-Nachricht aus einer äußeren cPacket-Nachricht lässt die Bitlänge der äußeren cPacket-Nachricht um die Bitlänge der inneren cPacket-Nachricht schrumpfen.

2.4.4 Logikimplementierung

Die Implementierung der Logik von einfachen Modulen erfolgt in C++-ähnlichen Klassen, die genauso heißen, wie das Modul in .ned Datei selbst. Eigene Klassendefinitionen erben von der OMNeT++-Klasse cSimpleModule, die ein reaktives Verhalten für eigene Klassendefinitionen anbietet und Funktionen zum Senden von Nachrichten unterstützt. Die Funktion initialize wird zur Laufzeit für alle einfachen Module aufgerufen, in der der Initialisierungscode hinzugefügt wird. Für diese Methode existiert eine weitere Variante – initialize(stage: int), die Initialisierung von Parametern im mehreren Stufen erlaubt. Dafür ist die Anzahl von Stufen mit dem Überschreiben der Funktion numInitStages festzulegen.

Die Nachrichten, die über Eingangsgates eines einfachen Moduls eintreffen, werden von der Laufzeitumgebung, in dem Funktionsaufruf handleMessage, an das Modul übergeben.

Ein Modul kann Nachrichten in Form von Timeouts an sich selbst senden, um Ausführungszeit oder Zeitverzögerung zu simulieren. Sollen Nachrichten an sich selbst gesendet werden, werden sie mit dem Funktionsaufruf scheduleAt, unter Angabe der zukünftigen Ausführungszeit und der Nachricht, der Laufzeitumgebung übergeben. Beim Ablauf der Zeit liefert die Laufzeitumgebung die Nachricht an den jeweiligen Erzeuger zurück. In diesem Fall wird wieder die Funktion handleMessage des Moduls aufgerufen. Über die Funktion isSelfMessage, die in der cMessage-Klasse definiert ist, kann ein Modul unterscheiden, ob eine neue Nachricht über ein Eingangsgate des Moduls eingeliefert oder die eigene Timeout-Nachricht ist.

Mit einer von mehreren überlagerten Funktionen send wird eine Nachricht unter Angabe des Ausgangsgates an andere Module oder Subsysteme übertragen. Die Laufzeitumgebung sorgt dafür, dass die Nachricht über den Verbindungspfad des Ausgangsgates mit dem Eingangsgate eines anderen

Subsystems oder einfachen Moduls übertragen wird, bis am Ende des Pfads in der C++-Klasse des anderen einfachen Modul wieder die Nachricht behandelt wird.

2.4.5 Future Event Set (FES)

Das Future Event Set (FES) beschreibt eine Liste von zukünftig auszuführenden Ereignissen der Laufzeitumgebung, die als Nachrichten definiert werden. Solange FES mit zukünftigen Ereignissen gefüllt wird, läuft die Simulation ununterbrochen weiter. Wird die Liste vollständig abgearbeitet, und es liegen keine weiteren Ereignisse der FES vor, wird die Simulation abgebrochen.

2.4.6 Netzwerkkonfiguration

Die Netzwerkkonfiguration erfolgt in einer omnetpp.ini-Datei, die den Einstiegspunkt der Simulation bildet.

2.4.7 Führen von Statistiken

OMNeT++ erlaubt das Sammeln von Simulationsergebnissen, die als Statistiken bezeichnet werden. Um Simulationsergebnisse sammeln zu können, werden im Programmcode Signale vom Typ `simsignal_t` angelegt und mit einem Signalnamen registriert. Die Signalnamen von Signalen werden von der Laufzeitumgebung auf spezielle Identifier abgebildet, die im gesamten System global sind. Unterschiedliche Module können sich für ein Signal über den Namen registrieren und über den Signal-Identifier bei Bedarf für dieses Signal Ereignisse erzeugen. Den Ereignissen können Simulationsergebnisse zugewiesen werden, die von der Laufzeitumgebung erfasst und nach dem Simulationsende in spezielle Dateien abgelegt werden. Unter Anwendung von unterschiedlichen Analyse-Werkzeugen können Simulationsergebnisse anschließend analysieren werden. Das OMNeT++-Framework integriert in die Entwicklungsumgebung das Result Analysis Werkzeug, mit dem verschiedene Diagramme aus den Simulationsergebnissen erzeugt werden können.

2.4.8 Starten einer OMNeT++-Simulation

Eine Simulation unter OMNeT++ wird in der Entwicklungsumgebung mit einem Beispielprojekt wie folgt gestartet. Rechtsklick auf eine omnetpp.ini Netzwerk-Konfigurationsdatei und in dem Popupfenster „Run as…“ als „OMNeT++-Simulation“ auswählen. Es wird eine grafische Laufzeitumgebung gestartet und alle an der Simulation beteiligten Module initialisiert. Der Klick im oberen Menü-Band auf den Run-Knopf startet die Simulation. Zur grafischen Laufzeitumgebung gibt es unter OMNeT++ zur Ausführung von Simulationen eine kommandozeilenbasierte Alternative.

3 CAN Bus-Modell

In diesem Kapitel wird das CAN Bus-Modell unter OMNeT++ erläutert. Beginnend mit einem Überblick über das CanBus-Modell und seine Modellentitäten, werden anschließend die Nachrichtenformate, die bei einer CAN Bus-Modellsimulation ausgetauscht werden, beschrieben. Die Erläuterung zu internem Aufbau von Entitäten und die Beschreibung von Aktionen in den Modulen wird dieses Kapitel abschließen.

3.1 Überblick

Jede Problemstellung, das unter OMNeT++ unter Anwendung des CAN Bus-Modells simuliert wird, wird aus folgenden CAN Bus-Modellentitäten aufgebaut.

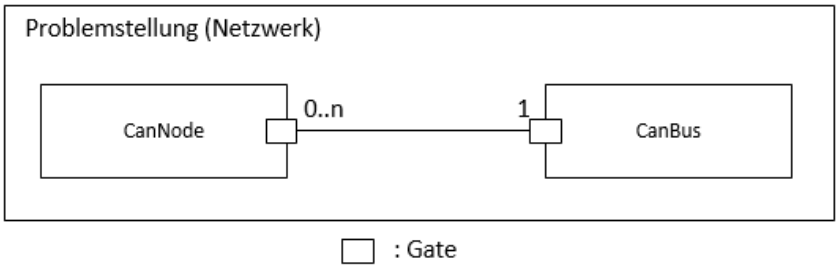

Abbildung 3.1. Blockschaltbild des CAN Bus-Simulationsmodells

CanBus

Die CanBus Entität modelliert den physikalischen CAN Bus, der nicht als eine einfache Übertragungsleitung, sondern abstrahiert, als ein System mit einer ausführbaren Logik dahinter, aufgefasst wird. Eine CanBus-Entität kann keine oder mehrere CanNode-Entitäten verwalten.

CanNode

Ein CAN Busteilnehmer wird durch die CanNode Entität in einem CAN Bus Simulationsmodell modelliert. CanNode-Entitäten können mit einer CanBus Entität über Gate-Verbindungen verknüpft werden, um Nachrichten austauschen zu können. Nachrichten, die über diese Verbindungen laufen, werden in der Simulationsumgebung visualisiert.

Die Nachrichtenübertragung zwischen `CanNode`- und `CanBus`-Entitäten wird im CAN-Simulationsmodell als ideal simuliert. Das Vergehen der Zeit wird in den Empfänger-Modulen bei Bedarf selbst simuliert.

3.2 Nachrichtenformate

Der CanBus definiert zwei Nachrichtenformate für das Versenden von CAN Daten-, Remote- und Error-Frames

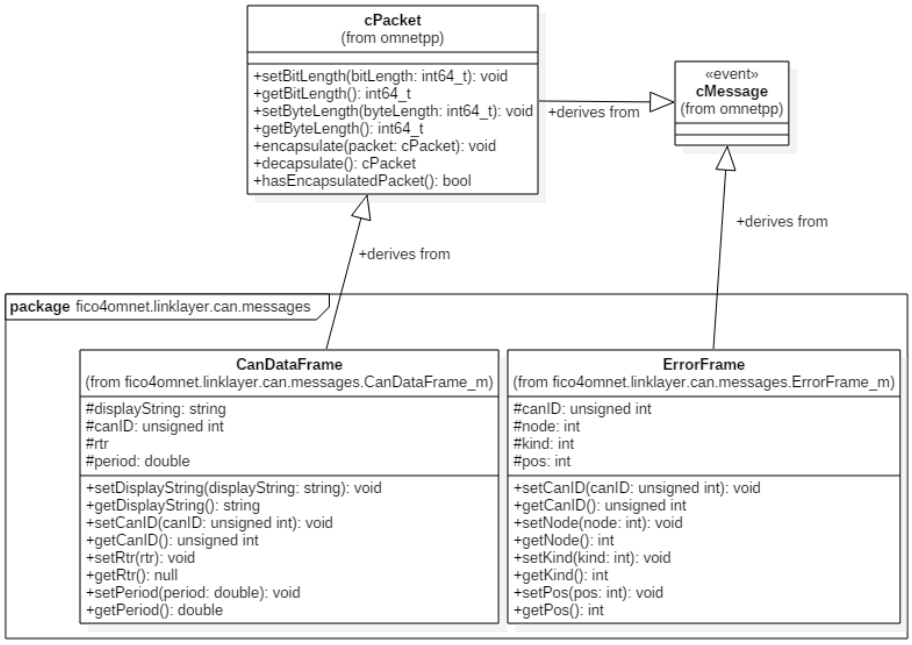

Abbildung 3.2. Nachrichtenformate im CAN-Modell

Ein `CanDataFrame`-Datentyp wird zur Modellierung von CAN Daten- oder Remote-Frames verwendet. Dieser Typ leitet von der Klasse `cPacket` ab, die wiederum von der cMessage-Klasse ableitet. Der `CanDataFrame`-Nachrichtentyp ist keine Nachbildung eines realen CAN Daten- oder Remote-Frames. Vielmehr beschreibt dieser Typ eine logische Übertragungsnachricht, die Parameter zum Zweck der Nachverfolgung enthält. Der `CanDataFrame`-Nachrichtentyp definiert folgende Parameter

Parameter	Datentyp	Beschreibung
displayString	string	Beschreibung der Nachricht
canID	unsigned int	Kennzeichnet den CAN Identifier.
rtr	bool	Kennzeichnet, ob es sich bei dieser Nachricht um ein CAN Daten- oder Remote-Frame handelt.
period	double	Kennzeichnet die Periode, die den Zyklus des Nachrichtenversands für diese Nachricht festlegt.

Tabelle 3.1: CanDataFrame-Parameter

Der ErrorFrame-Nachrichtentyp leitet von der `cMessage`-Klasse ab und wird zur Modellierung von CAN Error-Frames verwendet. Im CAN Bus-Modell werden Fehler in Sende- und Empfangsfehler eingeteilt. Zu Sendefehlern gehören:

(a) **Bit-Fehler** und
(b) **Form-Fehler**

Zu den Empfangsfehlern gehören

(c) **CRC-Fehler**
(d) **Bitstuffing-Fehler**

Genau, wie das `CanDataFrame`, definiert der `ErrorFrame`-Nachrichtentyp nur eine logische Nachricht und ist keine originalgetreue Abbildung eines CAN Error-Frames.

Der ErrorFrame-Nachrichtentyp definiert folgende Parameter

Parameter	Datentyp	Beschreibung
canID	unsigned int	Hier wird die CAN ID des CanDataFrames festgehalten, das entweder als CAN Daten- oder Remote-Frame versendet und während seiner Übertragung ein Fehler erzeugt wird
node	int	Beschreibt die ID des CAN Knotens, der den Fehler ausgelöst hat
kind	int	Beschreibt einen Fehlertyp
pos	int	Beschreibt die Position des Fehlers in einem CanDataFrame, das als CAN Daten- oder Remote-Frame gesendet wird

Tabelle 3.2. Parameterbeschreibung des ErrorFrame-Datentyps

3.3 CAN-Bus Modellentitäten

Jedes einfache Modul und Subsysteme besitzen eine `Modulname.ned`-Datei mit Parameter- und Gate-Definitionen, die für die Module festgelegt sind.

3.3.1 CanBus

Das CanBus-Modul besteht aus zwei einfachen Modulen, der `CanBusLogic`- und dem `BusPort`-Modul.

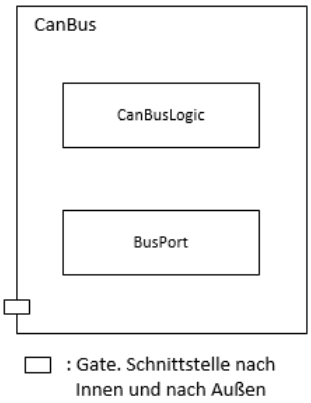

Abbildung 3.3. Blockschaltbild des CanBus-Moduls

Für die CanBus-Modulentität sind folgende Parameter in der `CanBus.ned` Datei festgelegt.

Parametername	Datentyp	Einheit/Typ und Wertebereich	Standardwert	Beschreibung
bandwidth	double	MBit/s	1	Legt die Datenrate des CAN Busses fest
version	string	Aufzählung {2.0A, 2.0B}	2.0A	Unterscheidet zwischen anzuwendender CAN Spezifikation Teil A oder Teil B
bitStuffingPercentage	double	%	0	Legt den Wert für die Berechnung von Stuffbits im CanNode fest.

Tabelle 3.3: Parameterbeschreibung des CanBus-Moduls in der CanBus.ned Datei

Die Parameter der CanBus-Entität können in der Konfigurationsdatei omnetpp.ini während der Simulationskonfiguration angepasst werden, um ihre Standardwerte zu überschreiben. Sie werden von allen CanNode Entitäten benötigt, um Information über die festgelegte Datenrate, den Prozentwert zur Berechnung von Stuffbits oder nach welcher Protokollspezifikation die Bitlängen von CanDataFrames berechnet werden sollen, zu erfahren.

3.3.1.1 CanBusLogic

Im CanBusLogic-Modul ist die Logik der CanBus Entität integriert. Hier sind Mechanismen festgelegt, um den CanBus-Simulationsablauf zu regeln.

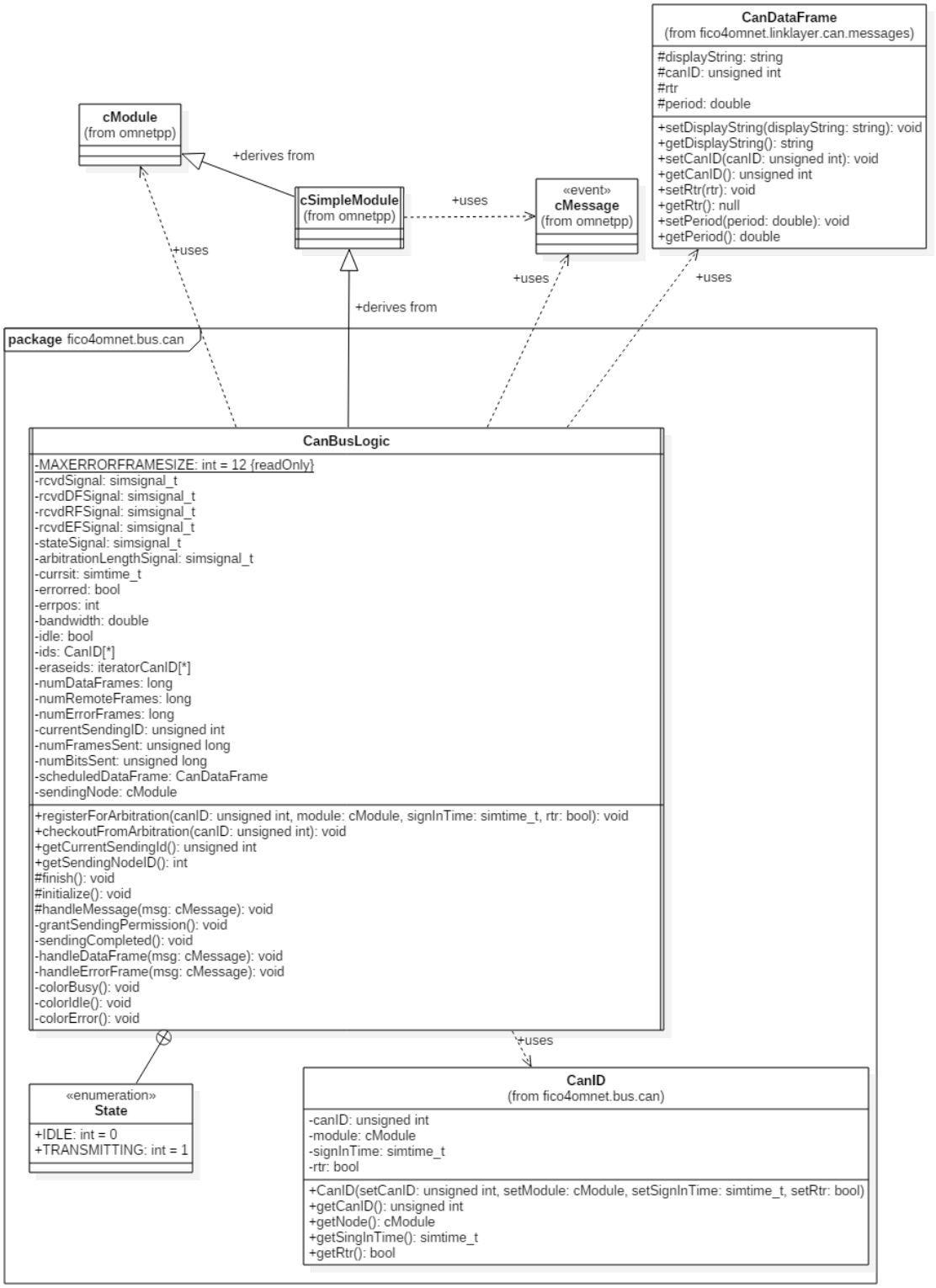

Abbildung 3.4. CanBusLogic Klassendiagramm

Die Funktion `registerForArbitration` in dem CanBusLogic-Modul wird von dem CanOutputBuffer-Modul (s. Abschnitt 3.3.2.3) eines CanNode-Busteilnehmers, wenn das CanOutputBuffer-Modul über sein Eingangsgate ein neues CanDataFrame von den Source Apps (s. Abschnitt 3.3.2.2) erhält, aufgerufen. Das CanOutputBuffer-Modul übergibt ihr die CAN ID des CanDataFrames, einen Verweis auf sich selbst, die Übergabezeit und den `rtr` Parameter zur Unterscheidung, ob es sich beim CanDataFrame um ein Daten- oder Remote-Frame handelt. Aus den übergebenen Argumenten erzeugt das CanBusLogic-Modul ein Zustandsobjekt vom Typ CanID und merkt sich dieses in der Liste `ids`. Zur Laufzeit, wenn beteiligte CanOutputBuffer-Module die ersten CAN IDs von CanDataFrames, die sie von den Source Apps empfangen haben, beim CanBusLogic-Modul für die Übertragung registrieren, erzeugt der erste Aufruf der Funktion `registerForArbitration` eine Timeout-Nachricht als einfache cMessage-Nachricht für das CanBusLogic-Modul. Die zukünftige Ausführungszeit der Timeout-Nachricht wird mit

$$\text{aktuelle Simulationszeit} + \text{ die Zeit von } \frac{1\text{Bit}}{\text{Datenrate}}$$

berechnet. Die aktuelle Simulationszeit kann mit der Funktion `simTime` erfragt werden. Die Datenrate, die mit dem Parameter `bandwidth` festgelegt und während der Initialisierungsphase der Simulation aus der Konfigurationsdatei `omnetpp.ini` über eine gleichnamige Variable gelesen und zugewiesen wird, wird als eine Fließkommazahl in MBit/s angegeben. Unter Verwendung der Funktion `scheduleAt` (diese Funktion steht jedem einfachen Modul zur Verfügung, das von der Klasse `cSimpleModule` der OMNeT++-Klassenbibliothek erbt) wird die erzeugte Timeout-Nachricht zusammen mit der berechneten zukünftigen Ausführungszeit dem FES der Laufzeitumgebung übergeben. Anschließend wird noch die Variable `idle`, die den anfänglichen Ruhezustand des CanBusLogic-Moduls kennzeichnet, geändert. Das CanBusLogic-Modul ist nun bereit die Funktion eines Schedulers für die Buszuteilung und Nachrichtenabfertigung zu übernehmen. Während vor dem ersten Timeout weitere CAN IDs von den CanOutputBuffer-Modulen registriert werden können, findet nach dem Timeout in dem Aufruf der Funktion `handleMessage`, die den Verweis auf die Timeout-Nachricht erhält, die Buszuteilung als Funktion einer Busarbitrierung statt. Die Buszuteilung wird nach jeder erfolgreicher oder fehlgeschlagener Übertragung von CanDataFrames, die als CAN Daten- oder Remote-Frames versendet werden, erneut durchgeführt. Anhand folgendem Aktivitätsdiagramm werden in der Funktion `handleMessage` der CanBusLogic folgende Aktionen durchgeführt

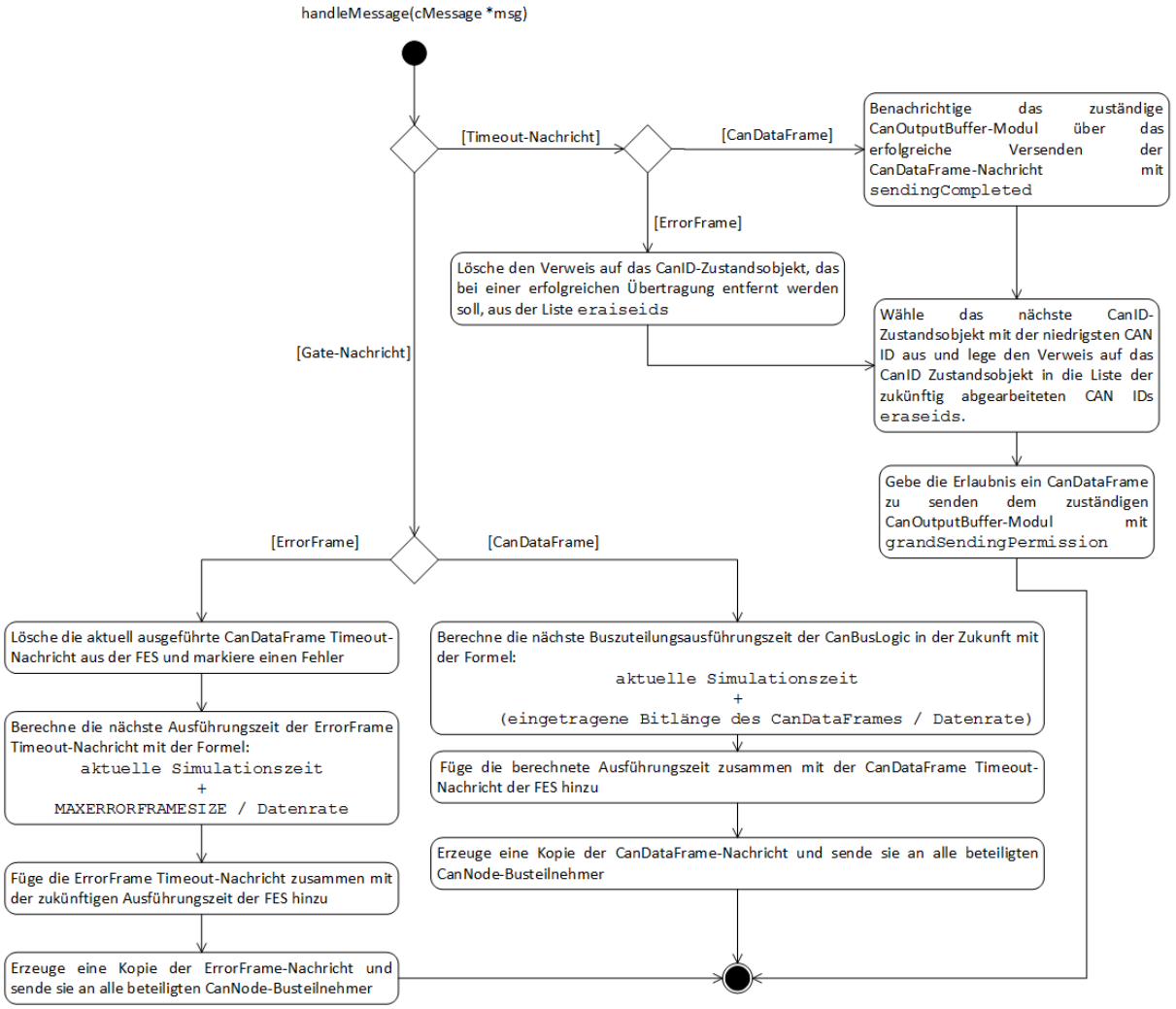

Abbildung 3.5. CanBusLogic. Busarbitrierung und Weiterleitung von CanDataFrames und ErrorFrames

3.3.1.2 BusPort

Das BusPort-Modul definiert zwei bidirektionale Gates. Über ein Gate werden Nachrichten von den CanBus-Teilnehmern angenommen und über das Ausgangsgate an das CanBusLogic-Modul weitergeleitet. Die Nachrichten, die über das Eingangsgate von dem CanBusLogic-Modul eintreffen, werden mit der Anzahl der Verbindungspfade zu den CanNode-Modulen vervielfältigt und an diese gesendet.

3.3.2 CanNode

Der CanNode simuliert einen CAN Busteilnehmer und wird aus folgenden Modulen und Subsystemen aufgebaut

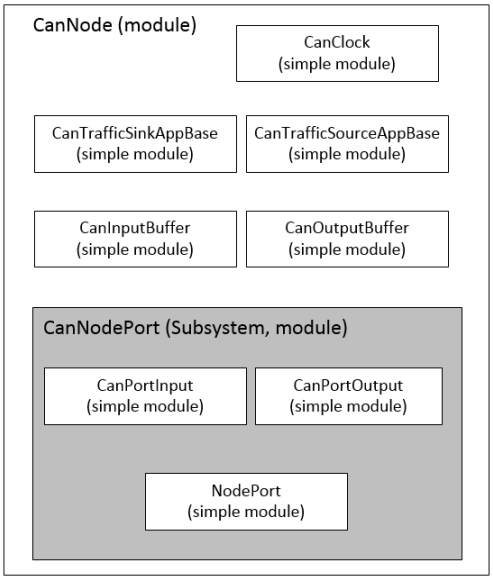

Abbildung 3.6. Blockschaltbild eines CanNode-Busteilnehmers

Für die Simulation von Sende- und Empfangsfehlern definiert das CanNode-Modul den Parameter `errorperc`, der mit einem Standardwert von `0` in der `CanNode.ned` Datei angelegt ist. Zur Fehlersimulation wird dieser Parameter von `CanPortInput`- und `CanPortOutput`-Modulen gelesen und verwendet.

3.3.2.1 CanClock

Die CanClock modelliert eine lokale Uhr eines CAN Busteilnehmers (CanNode). Damit wird das Verhalten einer Uhrenungenauigkeit simuliert, deren aktuelle Abweichung (auch als Drift bezeichnet) in dem Aufruf der Funktion `getCurrentDrift` als Zahlenwert in Sekunden zur zukünftigen Ausführungszeit einer Nachricht hinzuaddiert wird.

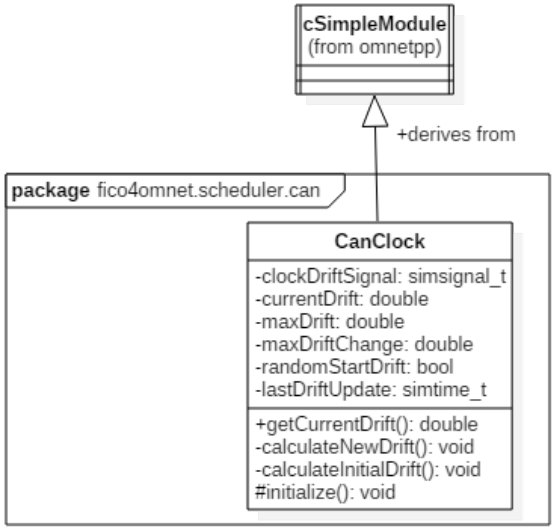

Abbildung 3.7. CanClock Klassendiagramm

3.3.2.2 CanTrafficSourceAppBase

Source Apps modellieren Anwendungen, die zyklisch CAN Nachrichten mit Daten generieren und über Ausgangsgates an das CanOutputBuffer-Modul senden.

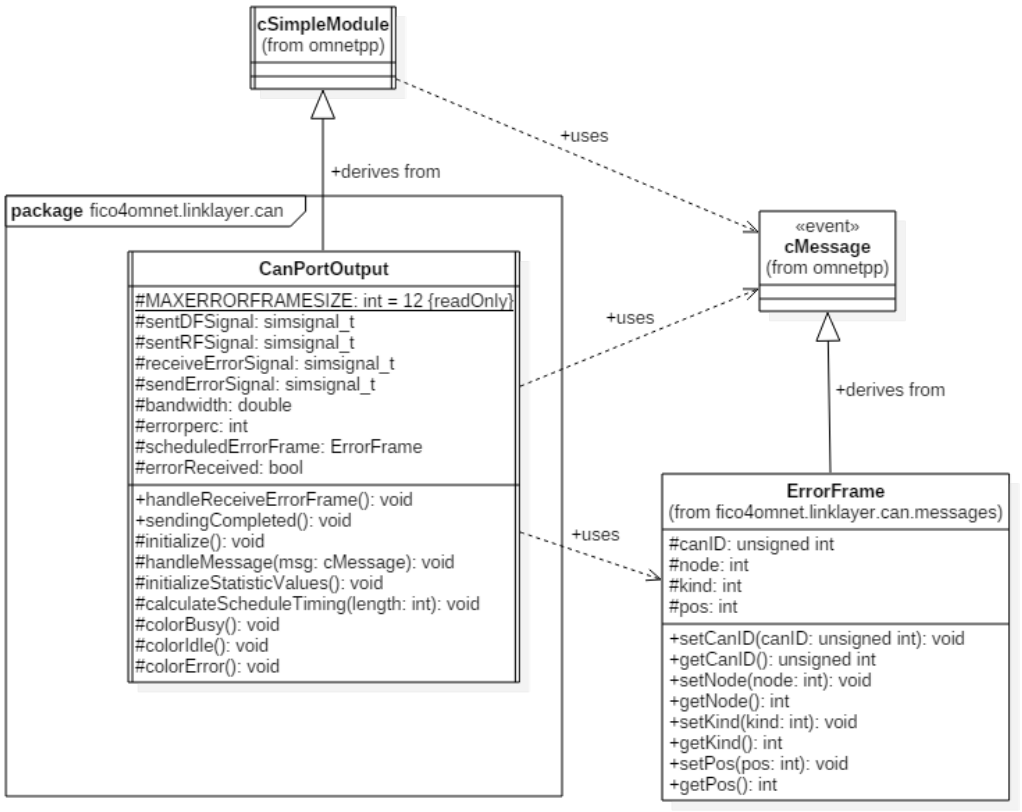

Abbildung 3.8. CanTrafficSourceAppBase Klassendiagramm

Das `CanTrafficSourceApp`-Modul leitet von der `cSimpleModule`-Klasse der OMNeT++-Klassenbibliothek ab.

3.3.2.2.1 Nachrichteninitialisierung

Source Apps erzeugen in der Initialisierungsphase, während des Aufrufs der Funktion `initialize(stage: int)` CAN Daten- und Remote-Frames als CanDataFrames (s. Abschnitt 3.2.1).

Für die Source Apps werden in der `CanTrafficSourceAppBase.ned` Datei folgende Parameter festgelegt.

Parameter	Datentyp	Beschreibung
idDataFrames	string	Feld zur Festlegung von CAN IDs für Daten-Frames
dataLengthDataFrames		Feld zur Festlegung von Nutzdatenfeldgrößen in Bytes
periodicityDataFrames		Feld zur Festlegung von Perioden in Sekunden
initialDataFrameOffset		Feld zur Festlegung von Offsets in Sekunden für den ersten Versand eines Daten-Frames
idRemoteFrames	string	Feld zur Festlegung von CAN IDs für Remote-Frames
dataLengthRemoteFrames		Feld zur Festlegung von Nutzdatenfeldgrößen in Bytes
periodicityRemoteFrames		Feld zur Festlegung der Periode in Sekunden
initialRemoteFrameOffset		Feld zur Festlegung von Offsets. Offsets definieren die Zeit für den ersten Versand eines Remote-Frames.
periodInaccurracy	string	Definiert eine Periodenungenauigkeit, die bei der Berechnung der Ausführungszeit eines Daten- oder Remote-Frames verwendet wird

Tabelle 3.4. Parameterbeschreibung von Source Apps aus CanTrafficSourceAppBase.ned Datei

In der Funktion `initialize(stage: int)` wird für die Erzeugung von CanDataFrame als CAN Daten-Frames `initialDataFrameCreation` aufgerufen. Der Aufruf dieser Funktion führt Aktionen durch in Abbildung 3.9 dargestellt sind

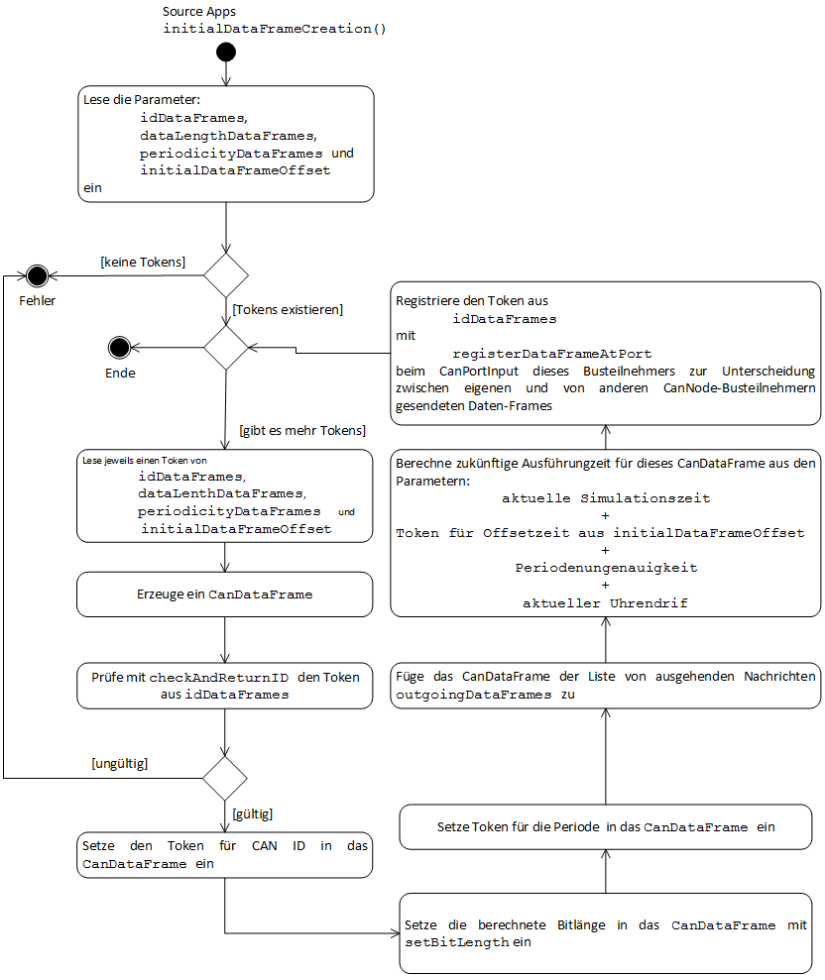

Abbildung 3.9. Initialisierungsphase von Source Apps. Anlegen von CanDataFrames, die als CAN Daten-Frames versendet werden.

Jedes generierte CanDataFrame als CAN Daten-Frame wird mit einer cPacket-Nachricht, die die Länge des Nutzdatenfelds in Bytes erhält, erweitert.

Die Erzeugung und Initialisierung von CanDataFrames als CAN Remote-Frames erfolgt in der gleichen Art und Weise. In der `initialize(stage: int)`-Funktion wird für die Erzeugung von CanDataFrames als CAN Remote-Frames die Funktion `initialRemoteFrameCreation` aufgerufen. In dieser Funktion werden unter die Parameter für Remote-Frames aus CanTrafficSourceAppBase.ned Datei: `idRemoteFrames`, `idDataLengthRemoteFrames`, `periodicityRemoteFrames` und `initialRemoteFrameOffset` eingelesen und die Werte zur Initialisierung von CanDataFrames als Remote-Frames verwendet. Wenn bei den Daten-Frames der Parameter rtr den Wert `false` erhielt, wird dieser Parameter bei Remote-Frames auf `true` gesetzt. Weiterhin wird für die Remote-Frames die Funktion `registerRemoteFrameAtPort(canID: unsigned int)` beim `CanPortInput` (s. Abschnitt 3.3.2.4.3) zur Registrierung ihrer CAN IDs aufgerufen.

3.3.2.2.2 Berechnung von Bitlängen für CanDataFrames

Der CanBus-Parameter `canVersion` legt für eine CanBus-Modellsimulation die CAN Spezifikation Teil A oder Teil B fest (vgl. CAN Specification, 1991, S. 2ff., CAN Specification, 1991, S. 32ff.) (s. Hinweis im Abschnitt 3.5). In der nachfolgender Abbildung wird die Wertbelegung von statischen Konstanten aus der Abbildung 3.8 zur Bestimmung von Bitlängen und Stuffbits gezeigt

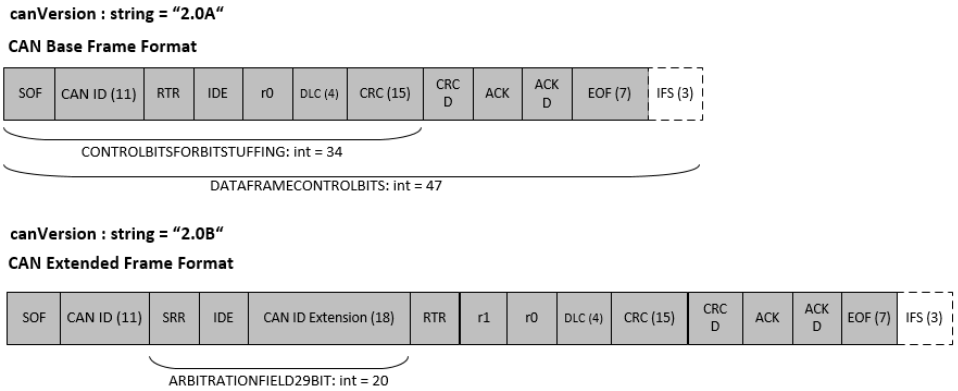

Abbildung 3.10. Abbildung von statischen Konstanten aus Abbildung 3.8 auf die CAN Daten- und Remote-Frame Formate ohne das Nutzdatenfeld

Die Bits IDE und r1 sind im Extended Frame Format zum Zweck der Vorführung, wie die 20 Bits zustande kommen, getauscht worden.

Die Parameter `canVersion`, `bitStuffingPercentage` und `bandwidth` der CanBus-Modulentität werden von jeder Source App abgefragt. Über den Parameter `canVersion` wird die Länge für gesendete Daten- und Remote-Frames ohne das Nutzdatenfeld bestimmt. Für CAN Daten- oder Remote-Frames im Base Frame Format beträgt die Bit Länge ohne Stuffbits – 47 und für CAN Daten- oder Remote-Frames im Extended Frame Format ohne Stuffbits – 67 Bits (berechnet mit `DATAFRAMECONTROLBITS + ARBITRATIONFIELD29BIT`). Die vollständige Bit Länge ohne Stuffbits mit dem Nutzdatenfeld ergibt sich entweder aus 47 oder 67 Bits + Nutzdatenfeld in Bits.

Für die Berechnung von Stuffbits unterliegen nach der CAN Spezifikation nur die Bitfelder beginnend mit SOF und endend mit der CRC-Bitsequenz von 15 Bit inklusive dem Nutzdatenfeld. Unter diesen Voraussetzungen wird für die Berechnung von Stuffbits die Konstante `CONTROLBITSFORBITSTUFFING` definiert, die die ersten 34 Bits ohne das Nutzdatenfeld definiert. Für CAN Daten- oder Remote-Frames im Extended Frame Format wird dieser Wert um `ARBITRATIONFIELD29BIT` erweitert und entspricht 54 Bits. In der nachfolgenden Formeln zur Berechnung von Stuffbits werden diese Bits als Standardbits, die von dem Parameter `canVersion` abhängen, bezeichnet

$$\frac{\text{Standardbits} + \text{Nutzdatenfeldgröße in Bits} - 1}{4}$$

Die Subtraktion von 1 bedeutet an dieser Stelle, die Anpassung der gesendeten Datenbits, um die richtige Berechnung von Stuffbits zu erreichen. Die nachfolgende Abbildung verdeutlicht diesen Sachverhalt

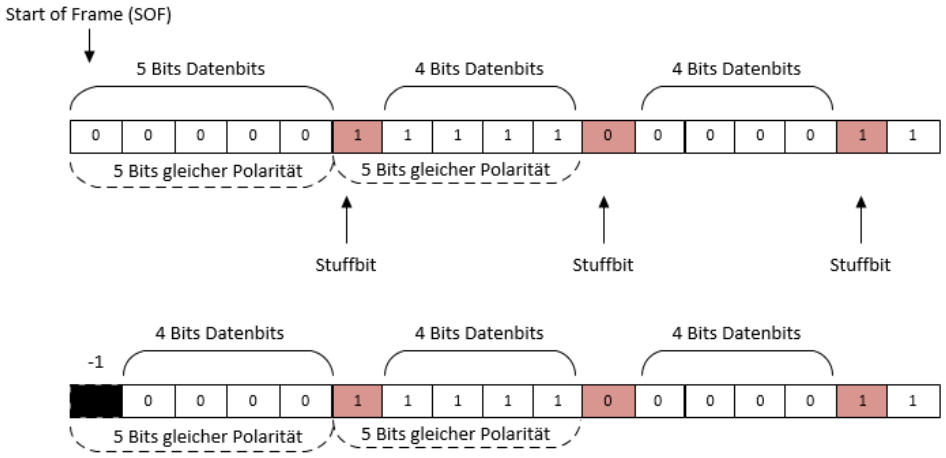

Abbildung 3.11. Erläuterung zur Berechnung der Formel 3.2

Nach dem Senden des Bits SOF werden noch 4 Datenbits gesendet, bevor das erste Stuffbit im ungünstigsten Fall an der 6-ten Bitposition eingefügt wird. Alle anderen Stuffbits werden jeweils nach 4 Datenbits unter Berücksichtigung, dass vor den 4 Bits auch ein Stuffbit derselben Polarität kommt, eingefügt. Für die richtige Berechnung von Stuffbits muss also die Datenbitlänge um 1 verringert werden, um beim Berechnen von Stuffbits gleichen Intervallschritte zu berücksichtigen.

3.3.2.2.3 Nachrichtenversand und externe Remote-Frame Behandlung

Wenn der Timeout einer eigenen CanDataFrame-Nachricht stattfindet, wird sie von der Laufzeitumgebung wieder der Source App übergeben, indem seine `handleMessage`-Funktion aufgerufen und ihr der Verweis auf das CanDataFrame übergeben wird. Die Source App erzeugt eine Kopie dieser Nachricht, berechnet für die nächste Ausführung dieses CanDataFrames den neuen Timeout in der Zukunft und fügt wieder die ursprüngliche CanDataFrame-Nachricht unter Angabe der zukünftigen Ausführungszeit mit der Funktion `scheduleAt` der FES hinzu, während die Kopie der Nachricht über das Ausgangsgate der Source App zum CanOutputBuffer-Modul gesendet wird. Diesen Vorgang beschreibt die Abbildung 3.12.

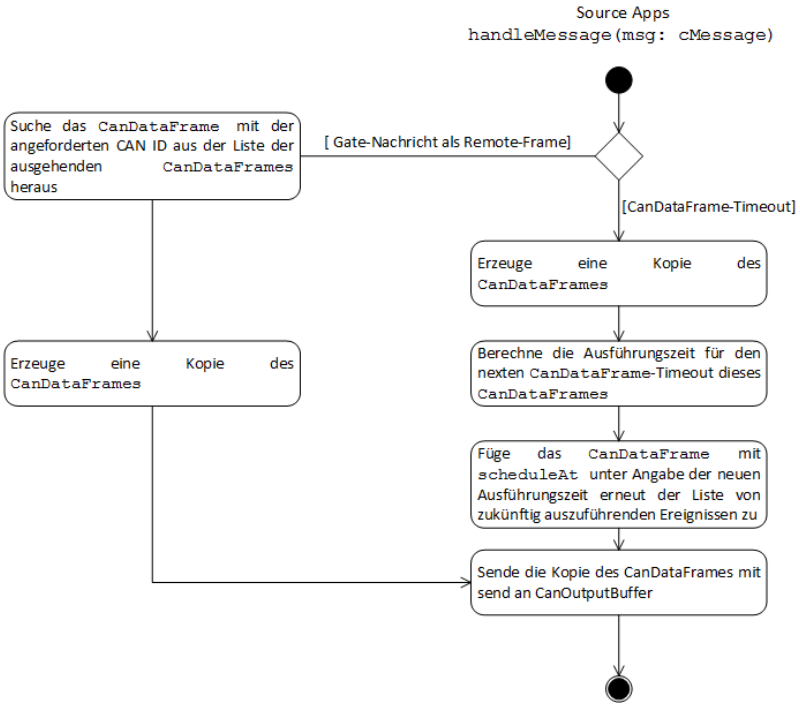

Abbildung 3.12. Nachrichtenübertragung von Source Apps.

Aus der Abbildung 3.10 ist auch erkennbar, dass ein CanDataFrame, der als CAN Remote-Frame von einem anderen Busteilnehmer versendet wird, auch den Source Apps zugespielt wird, wenn das Remote-Frame für sie bestimmt ist und sie darauf antworten können (Überprüfung der CAN ID im canID-Feld des Remote-Frames). In diesem Fall wird das zugehörige CanDataFrame, mit dem geantwortet werden soll, aus der Liste der ausgehenden CanDataFrames ausgewählt und über das Ausgangsgate der Source Apps für die Übertragung zum CanOutputBuffer gesendet.

3.3.2.3 CanOutputBuffer

Das CanOutputBuffer-Modul modelliert einen Nachrichtenspeicher für ausgehende Nachrichten.

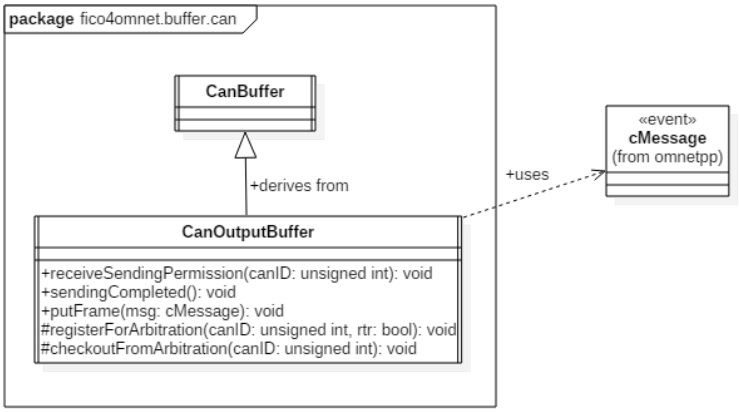

Abbildung 3.13. CanOutputBuffer Klassendiagramm

Empfängt das CanOutputBuffer-Modul eine CanDataFrame-Nachricht von einer Source App, wird das CanDataFrame mit dem Aufruf der Funktion `putFrame` zu der Liste der ausgehenden Nachrichten hinzugefügt und über den Funktionsaufruf `registerForArbitration`, die CAN ID des empfangenen CanDataFrames für die Busarbitrierung beim CanBusLogic-Modul (s. Abschnitt 3.3.1.1) registriert. Der Parameter rtr dient zur Unterscheidung zwischen Daten- und Remote-Frames.

Wenn von dem CanBusLogic-Modul der Erlaubnis zur Übertragung einer bestimmten CAN ID erfolgt (die Busarbitrierung ist zugunsten dieses CanNode-Busteilnehmers entschieden worden), wählt das CanOutputBuffer-Modul das entsprechende CanDataFrame aus der Liste der ausgehenden Nachrichten aus und sendet es über das Ausgangsgate an das `CanNodePort`-Subsystem des CanNode-Busteilnehmers. Das `CanNodePort`-Submodul leitet anschließend die Nachricht an das Eingangsgate vom subsysteminternen `CanPortOutput`-Moduls weiterleitet.

Ein CanOutputBuffer-Modul kann mehrere Source Apps bedienen. Über einen speziellen Parameter MOB (Message Object Buffer), wird entschieden, ob die von den Source Apps ankommenden neuen CAN Nachrichten, die in der Liste noch vorhandenen CAN Nachrichten mit derselben CAN IDs überschrieben sollen oder nicht. Das kann z. B. auftreten, wenn die Source Apps des CanNode-Busteilnehmers neue Nachrichten schneller erzeugen, als die Freigabe für die alten Nachrichten mit gleicher CAN ID im CanOutputBuffer durch das CanBusLogic-Modul erfolgt. Ist das Überschreiben erwünscht, wird die alte CAN ID von der Busarbitrierung des CanBusLogic-Moduls mit dem Aufruf der Funktion `checkoutFromArbitration` unter Angabe der CAN ID ausgeschlossen, die Nachricht aus der Liste von ausgehenden Nachrichten überschrieben und die neue CAN ID für die Busarbitrierung erneut bei dem CanBusLogic-Modul registriert.

3.3.2.4 CanNodePort

Das CanNodePort-Modul definiert ein Subsystem eines CanNode-Busteilnehmers, das die Transceiver-Charakteristiken implementiert und aus den einfachen Modulen CanPortOutput, NodePort und CanPortInput besteht.

Es beschreibt mehrere Eingangs- und Ausgangsgates, über die die Nachrichten entweder von den anderen internen CanNode-Modulen eintreffen oder an sie ausgeliefert werden und ein bidirektionales Gate, das mit dem Ein- und Ausgangsgate (auch ein bidirektionales Gate) des CanNode-Moduls verknüpft wird, um Nachrichten an das CanBus-Modul zu senden oder Nachrichten von dem CanBus-Modul zu empfangen.

3.3.2.4.1 CanPortOutput

Das CanPortOutput-Modul führt zwei Aktionen durch. Die erste Aufgabe besteht in der Weiterleitung der CanDataFrames, die das Modul vom CanBufferOutput-Modul erhält, wenn das CanBufferOutput-Modul die Erlaubnis zum Senden eines CanDataFrames von der CanBusLogic mit `grantSendingPermission` erhält. Die zweite Aufgabe des CanPortOutput-Moduls besteht in der Simulation von eigenen Sendefehlern.

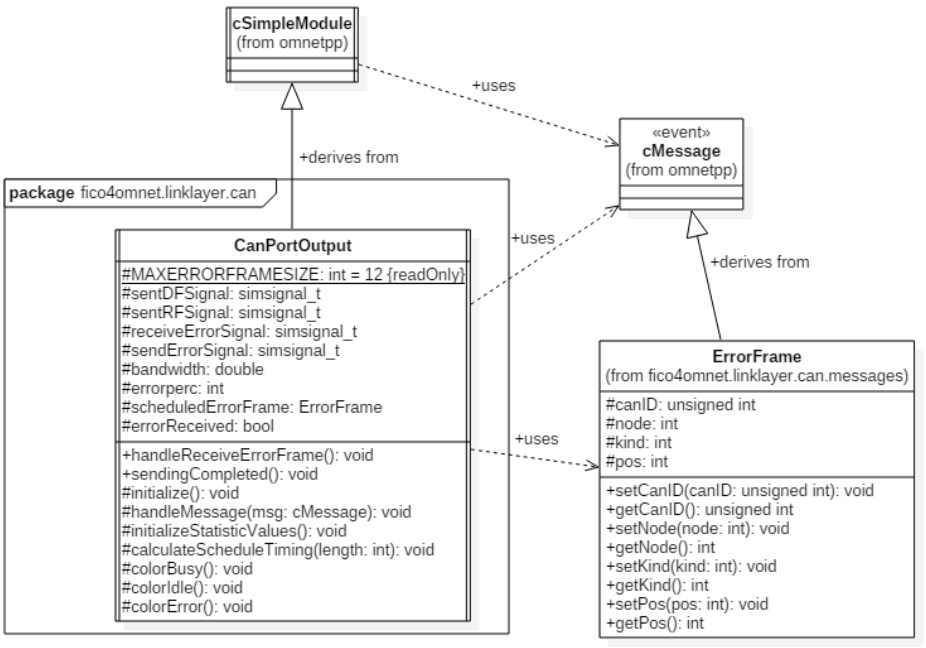

Abbildung 3.14. CanPortOutput Klassendiagramm

Dazu wird der Parameter `errorperc`, das in der CanNode.ned definiert ist, herangezogen. Der Parameter `errorperc` gibt den Prozentwert für den Versand von Fehlern an. In der handleMessage Funktion wird für das Versenden von Sendefehlern folgender Algorithmus, der nachfolgend in einer Pseudo-Sprache beschrieben wird, durchgeführt

(1) Setze die Variable `errorReceived` auf false.
(2) Lege eine Variable `senderr` an und speichere hier einen Zufallswert zwischen 0 und 99.
(3) Ist der Wert im `senderr` kleiner als `errorperc`, wenn ja - weiter mit (4), wenn nein – weiter mit (12).

(4) Erzeuge eine `ErrorFrame`-Nachricht und bezeichne sie als „senderror“.
(5) Trage die CAN ID der CanDataFrame-Nachricht ein, für die der Sendefehler erzeugt wird.
(6) Lege eine Variable `position` an und speichere hier einen Zufallswert zwischen 0 und Größe des CanDataFrames in Bits minus `MAXERRORFRAMESIZE`
(7) Lege eine Variable `kind` an und speichere hier ein Zufallswert zwischen 0 und 1, die den Bit- oder Form-Fehler kennzeichnen.
(8) Setze alle restlichen Parameter in die erzeugte `ErrorFrame`-Nachricht ein und berechne ihre Ausführungszeit mit der Formel:

$$\text{aktuelle Simulationszeit} + \text{die Zeit von } \frac{\text{Bitposition}}{\text{Datenrate}}$$

(9) Füge die berechnete Ausführungszeit und die `ErrorFrame`-Nachricht mit `scheduleAt` dem FES hinzu.
(10) Beim Ablauf des Timeouts überprüfe die Variable `errorReceived`, ob nicht ein Empfangsfehler vom CanPortInput-Modul gemeldet worden ist. Wenn ja – lösche eigene ErrorFrame-Nachricht und gehe zu (11), falls nicht – weiter mit (12).
(11) Sende die `ErrorFrame`-Nachricht als Sendefehler an das Ausgangsgate.
(12) Ende.

Das Setzen der Variable `errorReceived` wird von dem CanPortInput-Modul im Aufruf der Funktion `handleReceivedErrorFrame` des CanPortOutput-Moduls durchgeführt.

Simulationen von Sendefehlern können visuell infolge der Übertragung zur CanNode Entität, die dann die ErrorFrame-Nachricht vervielfältig und an alle Busteilnehmer sendet, in der Simulationsumgebung beobachtet werden.

3.3.2.4.2 NodePort

Über das NodePort-Modul werden Nachrichten von dem CanPortOutput-Modul empfangen und über das bidirektionale Gate des NodePort-Moduls an das bidirektionale Gate des CanNodePort-Moduls gesendet. CanNodePort überträgt die Nachricht seinerseits an das bidirektionale Gate des CanNode-Moduls, über den die Nachrichten schließlich an das CanBus-Modul übertragen werden.

Treffen Nachrichten in entgegengesetzter Richtung beim NodePort ein, werden sie an das CanPortInput-Modul weitergeleitet.

3.3.2.4.3 CanPortInput

Das CanPortInput-Modul erhält eingehende Nachrichten von dem NodePort-Modul.

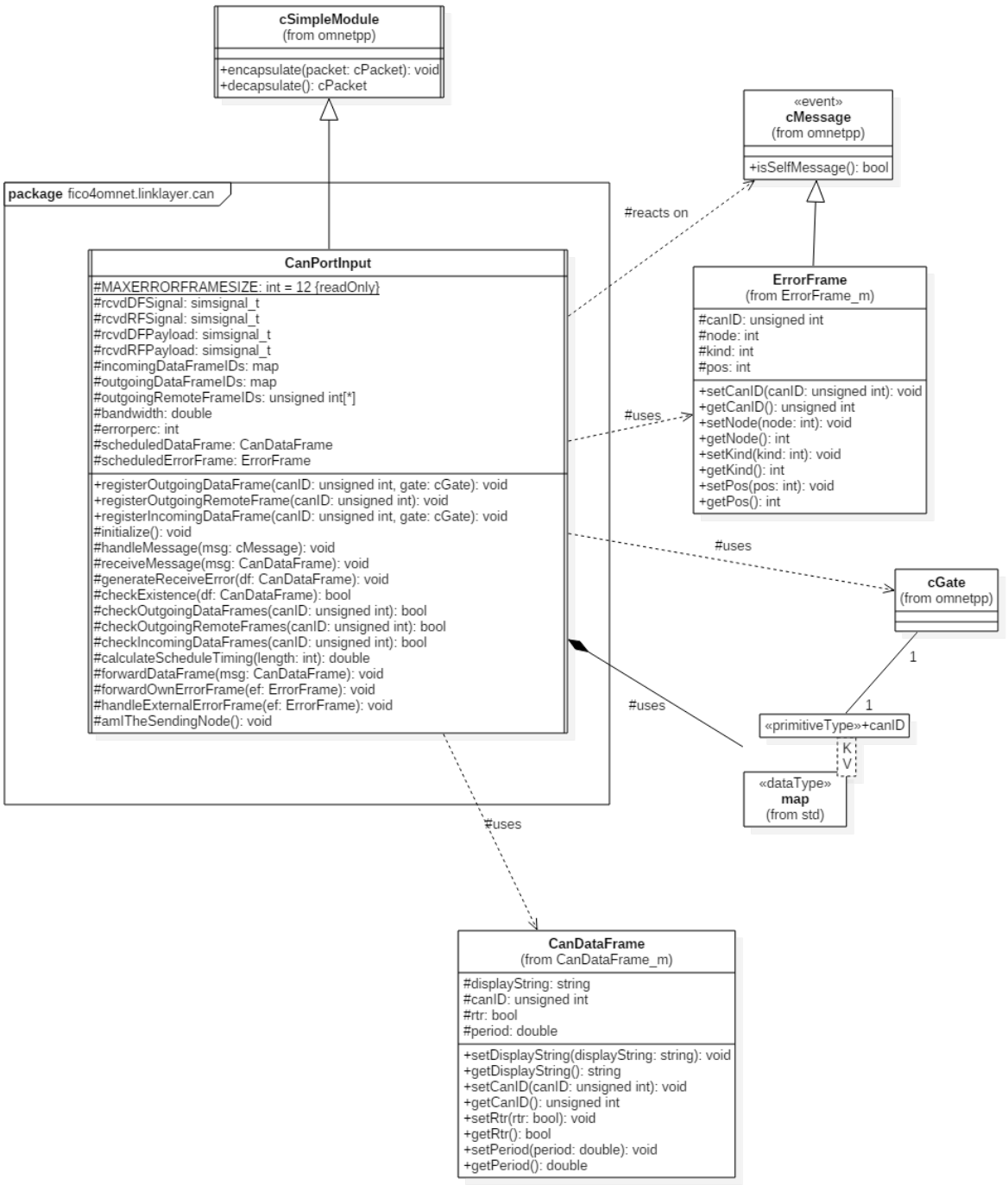

Abbildung 3.15. CanPortInput Klassendiagramm

Im CanPortInput-Modul werden drei mögliche Aktionen durchgeführt. Die erste Aktion besteht in der Simulation des Empfangs einer CanDataFrame-Nachricht. Wenn eine CanDataFrame-Nachricht vom NodePort-Modul empfangen wird, wird zuerst überprüft, ob die Nachricht für diesen Busteilnehmer relevant ist und ob nicht dieser Busteilnehmer der Sender dieser Nachricht gewesen ist. Ist die

Überprüfung erfolgt und es wird festgestellt, dass dieser Busteilnehmer nicht der Sender dieser Nachricht ist und das die Nachricht für diesen Busteilnehmer relevant ist, wird die empfangene CanDataFrame-Nachricht als eine Timeout-Nachricht behandelt, und unter Berechnung der zukünftigen Empfangszeit für diese Nachricht, die sich aus der Formel

$$\text{aktuelle Simulationszeit} + \text{ die Zeit von } \frac{\text{Bitlänge des CanDataFrames}}{\text{Standard Datenrate}}$$

ergibt, zusammen mit der Timeout-Nachricht über `scheduleAt` der FES der Laufzeitumgebung übergeben.

Die zweite Aufgabe des CanPortInput-Moduls besteht in der Simulation der Empfangsfehler. Dafür wird die Funktion `generateReceiveError` aufgerufen und ihr das empfangene CanDataFrame übergeben. Analog zu den Sendefehlern des CanPortOutput-Moduls wird derselbe Algorithmus mit wenigen Ausnahmen durchlaufen. Aus „senderror“ wird „receiveError“ eingetragen und die Fehlerart über eine Zufallszahl zwischen CRC- und Bitstuffing-Error als `kind` eingetragen. Der erzeugte ErrorFrame wird zusammen mit der zukünftigen Ausführungszeit, die wieder über eine zufällige Position der Bit Länge CanDataFrame minus MAXERRORFRAMESIZE als Zufallszahl bestimmt wird, in dem FES der Laufzeitumgebung eingelagert. Wird in dieser Zeit eine externe ErrorFrame-Nachricht empfangen, so werden sowohl das zuvor empfangene CanDataFrame als auch der erzeugte Empfangsfehler, deren Timeout noch nicht stattgefunden ist, in dem Funktionsaufruf `handleExternalErrorFrame` aus dem FES entfernt. Falls der Sender dieser externen Fehlernachricht dieser Busteilnehmer gewesen ist, das über die eingetragene CAN ID festgestellt werden kann, und es sich um einen Sendefehler handelt, wird über den Funktionsaufruf `handleReceivedErrorFrame` des CanPortOutput-Moduls der CanPortOutput-Modul dieses Busteilnehmers über den Fehlerempfang informiert, damit er auch entsprechend reagieren kann. Wenn der Timeout der eigenen Empfangsfehlernachricht stattgefunden ist, und bis dahin keine weiteren Fehler über das Eingangsgate für eingehende Nachrichten empfangen wird, wird der Empfangsfehler über ein spezielles Gate im Funktionsaufruf `forwardOwnErrorFrame` an das CanPortOutput-Modul dieses Busteilnehmers gesendet, wo diese Nachricht einfach gelöscht wird.

Wird der Timeout für die CanDataFrame-Nachricht erzeugt, das eine erfolgreich empfangene CanDataFrame-Nachricht kennzeichnet, wird die Nachricht an das entsprechende CanInputBuffer-Modul in dem Funktionsaufruf `forwardDataFrame` weitergeleitet.

3.3.2.5 CanInputBuffer

CanInputBuffer modelliert einen Nachrichtenspeicher für eingehende Nachrichten.

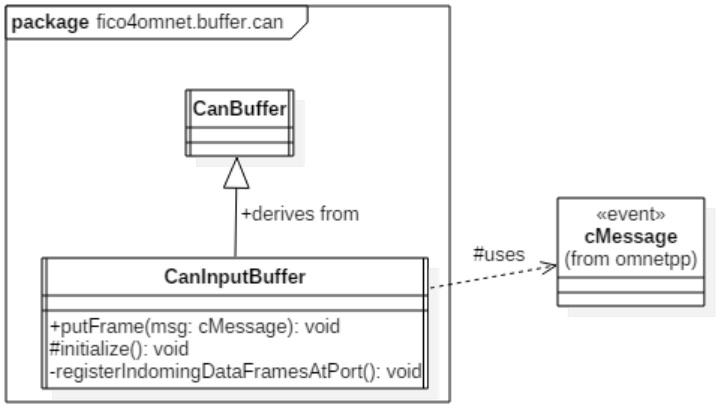

Abbildung 3.16. CanInputBuffer Klassendiagramm

Über das Eingangsgate für Nachrichten werden die zuständigen Sink Apps informiert, die anschließend, über eine Anfrage zur Nachrichtenübermittlung, die für sie bestimmte Nachricht zugesandt bekommen. An die Sink Apps ausgelieferte Nachrichten werden aus dem Nachrichtenempfangsspeicher gelöscht.

Das CanInputBuffer-Modul unterstützt genauso, wie das CanOutputBuffer-Modul, die MOB (Message Object Buffer) Variable, über die entschieden wird, ob die Nachrichten mit gleicher CAN ID, die noch nicht an die Sink Apps ausgeliefert sind, überschrieben werden sollen oder nicht.

3.3.2.6 CanTrafficSinkAppBase

Sink Apps modellieren Komponenten, die auf Nachrichten von anderen CAN Busteilnehmern, um diese weiter zu verarbeiten, warten (Stellglieder).

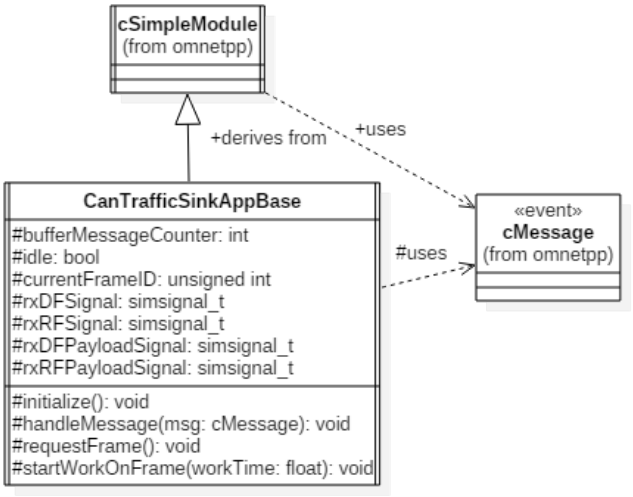

Abbildung 3.17. CanTrafficSinkAppBase Klassendiagramm

Die Sink Apps simulieren die Verarbeitung von Nutzdaten, die sie aus den empfangenen CanDataFrame-Nachrichten extrahieren. In definierten Zeitabständen können sie eine Anfrage an das CanInputBuffer-Modul des Busteilnehmers mit dem Aufruf der Funktion `requestFrame` stellen, um ihm dabei ihren Ruhezustand zu vermitteln. Falls für die Sink Apps in dem CanInputBuffer-Modul CanDataFrame-Nachrichten existieren, wird eine Nachricht ausgewählt und diese über ein spezielles Eingangsgate der Sink Apps an die Sink Apps weitergeleitet. Mit dem Aufruf der Funktion `startWorkOnFrame`, die eine spezifische Zeit für die Datenverarbeitung erhält, kennzeichnen die Sink Apps den Zustand der Datenverarbeitung.

4 Anforderungen

Für die Integration von CAN FD in das bestehende CAN Bus-Simulationsmodell resultieren aus Kapitel 3 Anforderungen, die bei der Entwicklung von CAN FD beachtet werden müssen.

4.1 Funktionale Anforderungen

4.1.1 Entwicklung eines CAN FD Nachrichtenformats

Für die Datenübertragung mit CAN FD soll ein geeignetes Nachrichtenformat für CAN FD Daten-Frames unter OMNeT++ entwickelt werden.

4.1.2 Anwendung der Bitstuffing-Regel bei CAN FD Daten-Frames

Bei einem CAN FD Daten-Frame soll die Möglichkeit zur Anwendung der Bitstuffing-Regel als Prozentwert auf die Länge des CAN FD Daten-Frames gegeben werden.

4.1.3 Implementierung von Sende- und Empfangsfehlern

Die Sende- und Empfangsfehler sollen bei der CAN FD Simulation unterstützt werden. Der Ablauf zum Erzeugen von unterschiedlichen Fehlern soll der CanBus-Modellsimulation entnommen werden.

4.2 Nichtfunktionale Anforderungen

4.2.1 Performanz bei der Nachrichtenübertragung

Die bit-orientierte Nachrichtenübertragung, wie es in realen CAN oder CAN FD Netzwerken der Fall ist (s. Abschnitt 2.1), ist im CAN Modell unter OMNeT++ nicht implementiert. Der Hintergrund dafür liegt in der Ereignisflut, die dabei entstehen kann, wenn die Performanz der Datenübertragung des CAN Modells im Mischbetrieb mit anderen Busarchitekturen, die im FiCo4OMNeT-Teilprojekt der CoRE-Arbeitsgruppe verwaltet werden, betrieben wird. Die Ereignisse werden nicht nur während der

Datenübertragung erzeugt, sondern in den beteiligten Modellierungsentitäten innerhalb von zugehörigen Busarchitekturen selbst. Die dadurch entstehende Gesamtereignismenge kann den Simulationskernel stark verlangsamen. Um die Anzahl von Ereignissen einzudämmen, soll sowohl zur Übertragung von CAN FD Nachrichten zwischen CAN FD Busteilnehmern, als auch Datenaustausch innerhalb eines CAN FD Knotens zwischen implementierten Modulstrukturen – ein Ereignis pro Nachricht – gewählt werden.

5 Konzept

In diesem Kapitel werden Konzepte zur Umsetzung von CAN FD im CAN Bus-Simulationsmodell beschrieben.

5.1 Modulentwicklung zur Unterstützung von CAN FD

CAN FD definiert die Obermenge von CAN, deshalb muss ein CAN FD Busteilnehmer entworfen werden, der alles können muss, was ein CAN Busteilnehmer kann. Diese Aussage schließt das Verstehen und Anwenden von allen CAN Nachrichtenformaten ein. Wenn CAN FD Busteilnehmer im Mischbetrieb mit CAN Busteilnehmern simuliert werden sollen, dürfen CAN FD Busteilnehmer unter dem Gesichtspunkt, dass die CAN Busteilnehmer nicht das neue CAN FD Daten-Frame Nachrichtenformat verstehen, nur die älteren CAN Nachrichten austauschen. Damit wirkt der CAN FD Busteilnehmer nach Außen, wie ein gewöhnlicher CAN Busteilnehmer. Dieser Vorgang führt zur Anlegung eines Parameters, der bei einem CAN FD Busteilnehmer den Betriebsmodus bestimmt, um mit der Spezifikation konform zu bleiben (vgl. (4)).

Zur Übertragung von CAN FD Nachrichten und ihrem Scheduling müssen folgende Module entwickelt werden:

1. Source Apps, die zyklisch CAN FD Daten-Frames generieren und senden können. Die Source Apps unterstützen einen Parameter zur Nachrichtengenerierung. Damit können die Source Apps so konfiguriert werden, dass sie entweder nur CAN Daten- oder Remote-Frames, nur CAN FD Daten-Frames oder die Mischung aus beiden Varianten unterstützen (siehe dazu Abschnitt 5.4).
2. Sink Apps, die aus empfangenen CAN FD Daten-Frames Nutzdaten extrahieren und verarbeiten können.

Sowohl Source Apps als auch die Sink Apps müssen entsprechend dem Auswahlmoduls des CAN FD Busteilnehmers arbeiten. Die Entscheidung im welchen Modus der CAN FD Busteilnehmer arbeitet, liegt bei dem Simulationsanwender selbst.

3. Entwicklung des PortOutput-Moduls für die Weiterleitung von CAN FD Daten-Frames und Generierung von Sendefehlern.
4. Entwicklung des PortInput-Moduls für den Empfang von CAN FD Daten-Frames und Generierung von Empfangsfehlern.
5. Erweiterung der CanBusLogic um das Scheduling eines CAN FD Daten-Frame Nachrichtenformats.

5.2 CAN FD Nachrichtenformat

Beim CAN FD Daten-Frame kann das Frame in zwei Übertragungsphasen unterteilt werden.

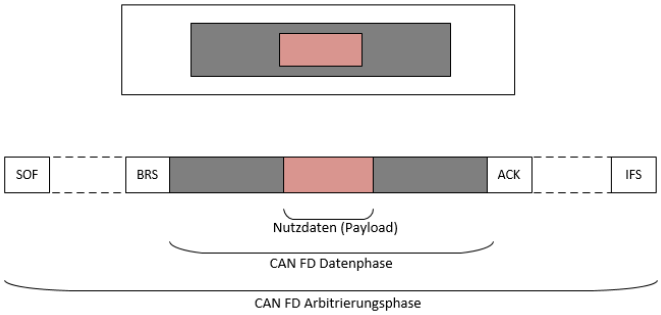

Abbildung 5.1. Konzept des CAN FD Daten-Frames

Das äußere Packet definiert die CAN FD Arbitrierungsphase, das nächst innere Packet - die CAN FD Datenphase. Das Datenphasenpacket kapselt schließlich die Nutzdaten ein. Diese Form des Nachrichtenversands unterstützt die Anforderung eines Ereignisses pro Nachricht.

5.3 Anwendung der Bitstuffing-Regel

Zur Umsetzung der Berechnung von Stuffbits für ein CAN FD Daten-Frame gibt es zwei Alternativen. Die erste Alternative besteht darin, die Anzahl von Stuffbits beginnend vom SOF Bit eines CAN FD Daten-Frames und endend mit dem letzten Bit der Bitsequenz, das noch der Stuffbit-Regel unterliegt, mit einer Prozentwertangabe aus dem ungünstigsten Fall von möglichen Stuffbits für diesen Abschnitt zu berechnen. Die berechneten Stuffbits werden in diesem Fall einfach von links nach rechts in die CAN FD Bitsequenz, die der Stuffbit-Regel unterliegt, eingefügt. Je nachdem, an welcher Bitposition sich die Stuffbits befinden, gehören sie entweder der CAN FD Arbitrierungsphase oder der CAN FD Datenphase an.

Die zweite Alternative besteht in der Berechnung von Stuffbits mit zwei Prozentwertangaben, die über entsprechende Parameter definiert werden. Der eine Prozentwert wird zur Berechnung von Stuffbits in der CAN FD Arbitrierungsphase, beginnend mit dem SOF und endend mit dem BRS Bit, verwendet. Der andere Prozentwert wird zur Berechnung von Stuffbits in der CAN FD Datenphase, beginnend mit dem ersten Bit nach dem BRS Bit und endend mit der Bitsequenz, die noch der Stuffbit-Regel entsprechend der gewählten CAN FD Nachrichtenstandard (ISO oder non-ISO CAN FD) unterliegt, verwendet.

5.4 CAN FD Fehlersimulation

Die Simulation von Sende- und Empfangsfehlern erfolgt äquivalent der Fehlersimulation, die in den CanPortInput- und CanPortOutput-Modulen durchgeführt wird. Dazu wird die Position aus dem Bitlängenabschnitt eines CAN FD Daten-Frames, beginnend mit dem SOF und endend mit dem CRC Trenner Bit, über eine Zufallszahl bestimmt. Aus der berechneten Bitposition resultiert die Anzahl von Datenraten, die für die Berechnung der zukünftigen Timeout-Zeit für das Versenden einer Fehlernachricht herangezogen werden können.

5.5 Unterstützung mehrerer Simulationsszenarien

Mit einer entsprechenden Parameterbelegung der CAN FD Source Apps und der Netzwerkkonfiguration kann der Anwender selbst entscheiden welche Nachrichten die CAN FD Source Apps senden sollen. Dies betrifft folgende Simulationsszenarien:

1. Ein CAN FD Busteilnehmer soll nur ältere CAN Nachrichten senden, weil er in einem Mischbetrieb mit älteren CAN Busteilnehmern, die von CAN FD keine Ahnung haben, Nachrichten austauschen soll. In diesem Fall wird der CanBus-Modulparameter `CAN_CANFD_mixedmode` gesetzt. Damit werden CAN FD Source Apps keine CAN FD Nachrichten generieren und senden.

2. Ein CAN FD Busteilnehmer soll nur CAN FD Daten-Frames senden, weil er sich in einem homogenen Netzwerk, bestehend nur aus CAN FD Busteilnehmern, befindet. Das ist der Standardfall und erlaubt dem Anwender selbst zu entscheiden, ob ein CAN FD Busteilnehmer nur CAN Daten- oder Remote-Frames oder nur CAN FD Daten-Frames oder aber eine Mischung aus beiden Nachrichtenvarianten generiert.

3. Ein CAN FD Busteilnehmer soll die älteren CAN Daten- oder Remote-Frames aber auch die neuen CAN FD Daten-Frames senden, weil das Netzwerk auch aus zukünftigen CAN Busteilnehmern besteht, die zwar nur CAN sprechen aber auch entsprechende Behandlungsmethoden für den Empfang von CAN FD Daten-Frames anbieten.

6 Umsetzung

In diesem Kapitel wird die Integration von CAN FD in das bestehende CAN Bus-Simulationsmodell beschreiben. Hier wird die Umsetzung von Konzepten unter Einhaltung von definierten Anforderungen dargelegt.

6.1 Überblick

Die Umsetzung von CAN FD betrifft die Definition eines CAN FD Daten-Frame Nachrichtenformats, Änderung in der CanBusLogic zum Scheduling von CAN FD Daten-Frames und die Definition eines CAN FD Busteilnehmers, der die Obermenge von CAN bildet und neben CAN Nachrichtenformaten auch CAN FD Daten-Frames senden und empfangen kann.

6.2 CAN FD Nachrichtenformat

Das CAN FD Daten-Frame Nachrichtenformat besteht aus einem äußeren `CanFDArbitrationPhaseFrame` und einem inneren `CanFDDataPhaseFrame`-Nachrichtentyp. Beide Nachrichtentypen entsprechen den CAN FD Übertragungsphasen. Das `CanFDDataPhaseFrame` kann Nutzdaten einkapseln.

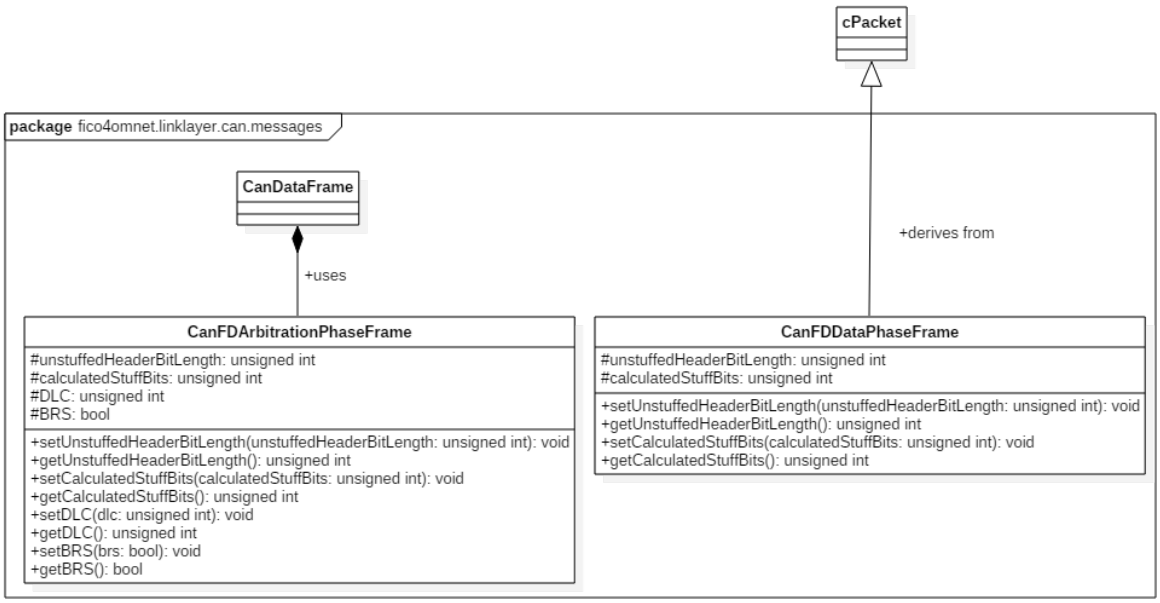

Abbildung 6.1. CanFDArbitrationPhaseFrame und CanFDDataPhaseFrame. Klassendiagramm

`CanFDArbitrationPhaseFrame` leitet von dem `CanDataFrame` ab und kennzeichnet die CAN FD Bitsequenz, die mit einer CAN-üblichen Datenrate übertragen wird. Im Parameter `BRS` kann entschieden werden, ob das innere `CanFDDataPhaseFrame` mit einer alternativen höheren Datenrate oder mit der CAN-üblichen Datenrate versendet werden soll. Das `DLC` Feld kennzeichnet den Data Length Code, der die Bitlänge des Nutzdatenfelds als eine vorzeichenlose ganze Zahl adressiert.

Sowohl das `CanFDArbitrationPhaseFrame` als auch das `CanFDDataPhaseFrame` beinhalten die Parameter `unstuffedFrameHeaderBitLength` und `calculatedStuffBits`. In `calculatedStuffBits` können die berechneten Stuffbits in dem jeweiligen Nachrichtentyp gespeichert werden. Der Parameter `unstuffedFrameHeaderBitLength` wird als Eintrag für Bitlängen verwendet, die im jeweiligen Nachrichtentyp zur Berechnung von Stuffbits benötigt werden.

6.3 Die Hilfsklasse CanUtils

Die Klasse CanUtils beinhaltet nützliche statische Validierungs-, Berechnungs- und Konvertierungsfunktionen, deren Anwendung die Umsetzung von CAN oder CAN FD Eigenschaften stark vereinfachen und die Codezeilen während der Entwicklung reduzieren. Folgende Funktionen und Attribute werden von dieser Klasse unterstützt

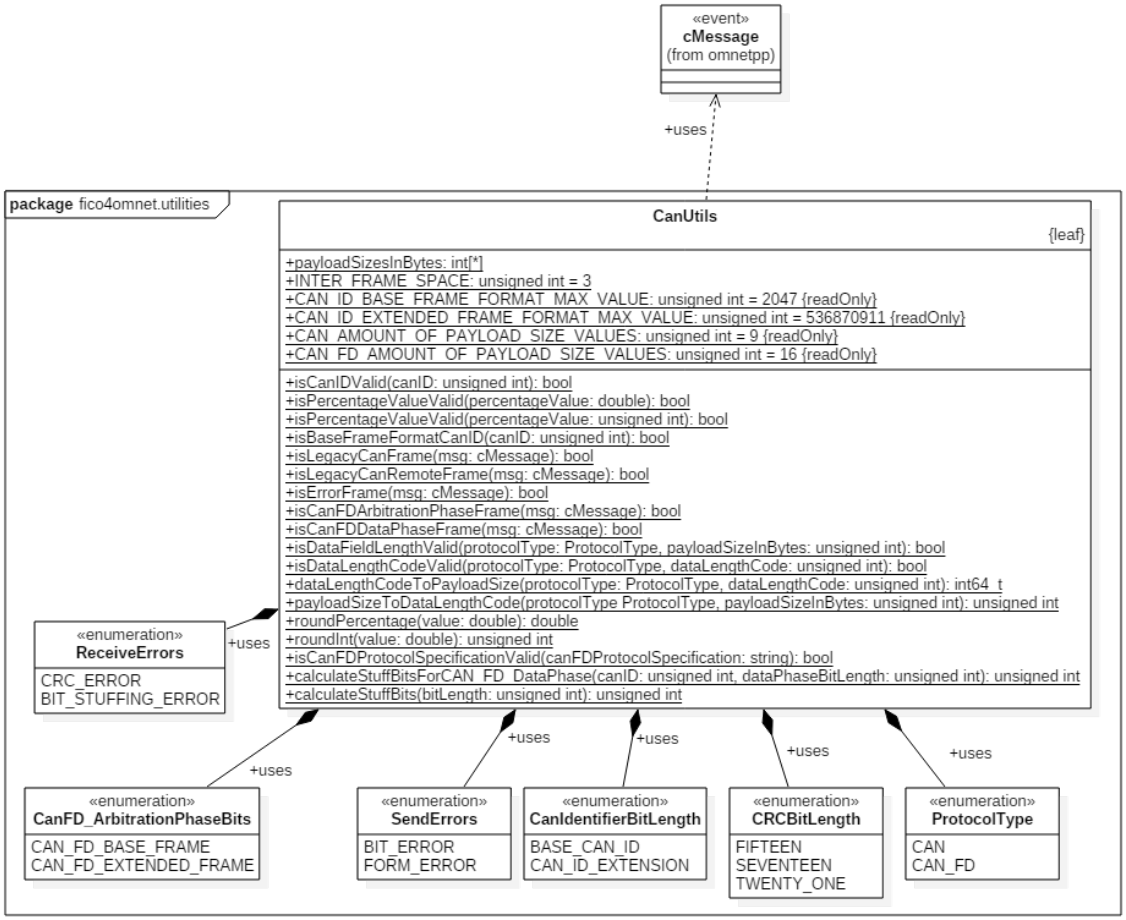

Abbildung 6.2. CanUtils Klassendiagramm

6.4 CanBus

6.4.1 Parameterdefinitionen

Im CanBus-Modul sind neue Parameter definiert worden. Diese sind in der nachfolgenden Tabelle 6.1 gelistet

Parameter	Datentyp	Einheit/Wertauswahl	Standardwert
CANFD_DataphaseBandwidth	double	bps	$2*10^6$
CANFD_BitStuffingPercentageDataPhase	double	%	0.0
CANFD_ProtocolType	enum	{non-ISO-CAN-FD, ISO-CAN-FD}	non-ISO-CAN-FD
CAN_CANFD_mixedmode	bool	{true, false}	false

Tabelle 6.1. Neue Parameter in der canBus.ned Datei

Die Parameter werden, wie folgt, definiert:

CANFD_DataphaseBandwidth

Kennzeichnet die Datenrate, die für eine CAN FD Datenübertragung in der Datenphase verwendet wird.

CANFD_BitStuffingPercentageDataPhase

Kennzeichnet den alternativen Prozentwert zur Berechnung von Stuffbits für die Bitsequenz in der CAN FD Datenphase.

CANFD_ProtocolType

Wählt zwischen zwei möglichen CAN FD Protokollvarianten aus. Hier kann entweder der Wert `non-ISO-CAN-FD` für die CAN FD Spezifikation 1.0 oder der Wert `ISO-CAN-FD` für die CAN FD im ISO Standard ausgewählt werden. Auf der Grundlage von diesen CAN FD Daten-Frame Varianten werden zur Laufzeit der Simulation CAN FD Daten-Frames aufgebaut.

CAN_CANFD_mixedmode

Die Migration von CAN FD in die CAN Netzwerke kann über das Setzen dieses Parameters simuliert werden. Wird dieser Parameter gesetzt, so werden CAN FD Busteilnehmer sich wie ältere CAN Busteilnehmer verhalten und ausschließlich ältere CAN Nachrichten senden und empfangen. In diesem Parameter wird also der Modus des Netzwerks festgelegt, der einen direkten Einfluss darauf hat, welche Nachrichten während der Simulation von CAN und CAN FD Busteilnehmern ausgetauscht werden

twoBitStuffingPercValues

Kennzeichnet den Auswahlmodus zur Bestimmung von Stuffbits für einen CAN FD Daten-Frame. Die Stuffbits bei einem CAN FD Daten-Frame können mit einem oder zwei Prozentwertangaben bestimmt werden.

6.4.2 Erweiterung des CanBusLogic-Moduls

Das CanBusLogic-Modul ist mit der Funktion zum Scheduling von CAN FD Daten-Frames erweitert worden. Die Funktion `handleCanFDFrame` empfängt ein `CanFDArbitrationPhaseFrame` und untersucht den BRS Parameter dieser Nachricht. Falls dieser Parameter gesetzt ist, wird das `CanFDDataPhaseFrame` aus dem `CanFDArbitrationPhaseFrame` extrahiert. Infolge dessen reduziert sich die Bit Länge des `CanFDArbitrationPhaseFrames`. Anschließend wird der zukünftige Idle-Zustand der `CanBusLogic` in Form eines Timeouts in Abhängigkeit von zwei Datenraten bestimmt und das `CanFDDataPhaseFrame` in das `CanFDArbitrationPhaseFrame` erneut verpackt. Mit `scheduleAt` Funktion wird der berechnete Timeout und die `CanFDArbitrationPhaseFrame`-Nachricht, die die aktuell übertragende Nachricht kennzeichnet, in das Future Event Set der Laufzeitumgebung eingelagert, sowie Kopie der Nachricht an das Ausgangsgate des Moduls gesendet. Im BusPort-Modul wird diese Nachricht vervielfältigt und an alle Busteilnehmer verteilt (siehe Abschnitt 3.3.1).

6.5 CanFDNode

Der CanFDNode modelliert einen CAN FD Busteilnehmer. Ein CanFDNode realisiert alles, was ein CAN Busteilnehmer kann und ermöglicht zusätzlich den Versand und Empfang von CAN FD Daten-Frames.

Das CAN Bus-Simulationsmodell ist um das Modell eines CAN FD Busteilnehmers `CanFDNode` erweitert worden, der die Modellarchitektur eines CAN Busteilnehmers CanNode beibehält und für die Umsetzung von Konzepten und Anforderungen neue Module definiert. Die neuen Module ersetzen die Module des CanNode-Busteilnehmers, indem sie von ihnen ableiten und ihr Verhalten um CAN FD Eigenschaften und Verhalten erweitern. Zur Erkennung tragen die neuen Module denselben Namen des Moduls, von dem sie ableiten, in den der FD-Kürzel eingetragen ist. Die Abbildung 6.1 zeigt einen CAN FD Busteilnehmer `CanFDNode` mit neuen Modulentwürfen, die farblich unterlegt sind

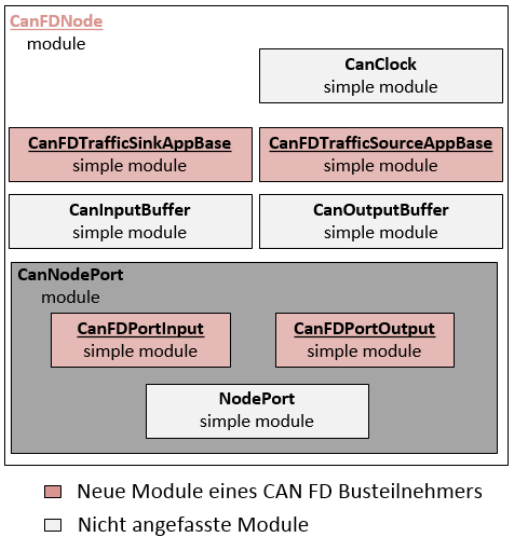

Abbildung 6.3. Umsetzung eines CAN FD Busteilnehmers

6.5.1 CanFDTrafficSourceAppBase

CAN FD Source Anwendungen modellieren Anwendungen, die in Abhängigkeit von der Konfiguration der CanBus-Modulparameter, entweder ältere CAN Daten- oder Remote-Frames oder auch CAN FD Daten-Frames generieren können. Abhängig von der Parameterbelegung der CAN FD Source Apps und der globalen Netzwerkkonfiguration kann der Anwender selbst entscheiden, welche Nachrichten die CAN FD Source Apps senden sollen. Für die Simulation von drei beschriebenen Szenarien (siehe Abschnitt 5.4), werden in der `CanFDTrafficSourceAppBase.ned` Datei zusätzliche Parameter definiert, die abhängig vom Parameter `CAN_CANFD_mixedmode`, der in der CanBus.ned Datei definiert ist, die Erzeugung von CAN FD Daten-Frames ermöglichen. Die nachfolgende Tabelle listet sie tabellarisch auf

Parameter	Datentyp
idCanFDDataFrames	string
periodicityCanFDDataFrames	
dataLengthCanFDDataFrames	
initialCanFDDataFramesOffset	
bitRateSwitch	
dataLengthCode	int

Tabelle 6.2. Parameterbeschreibung von CanFDTrafficSourceAppBase.ned

Die CAN FD Source Apps unterstützen Parameter und Funktionen, die im nachfolgenden Klassendiagramm abgebildet sind

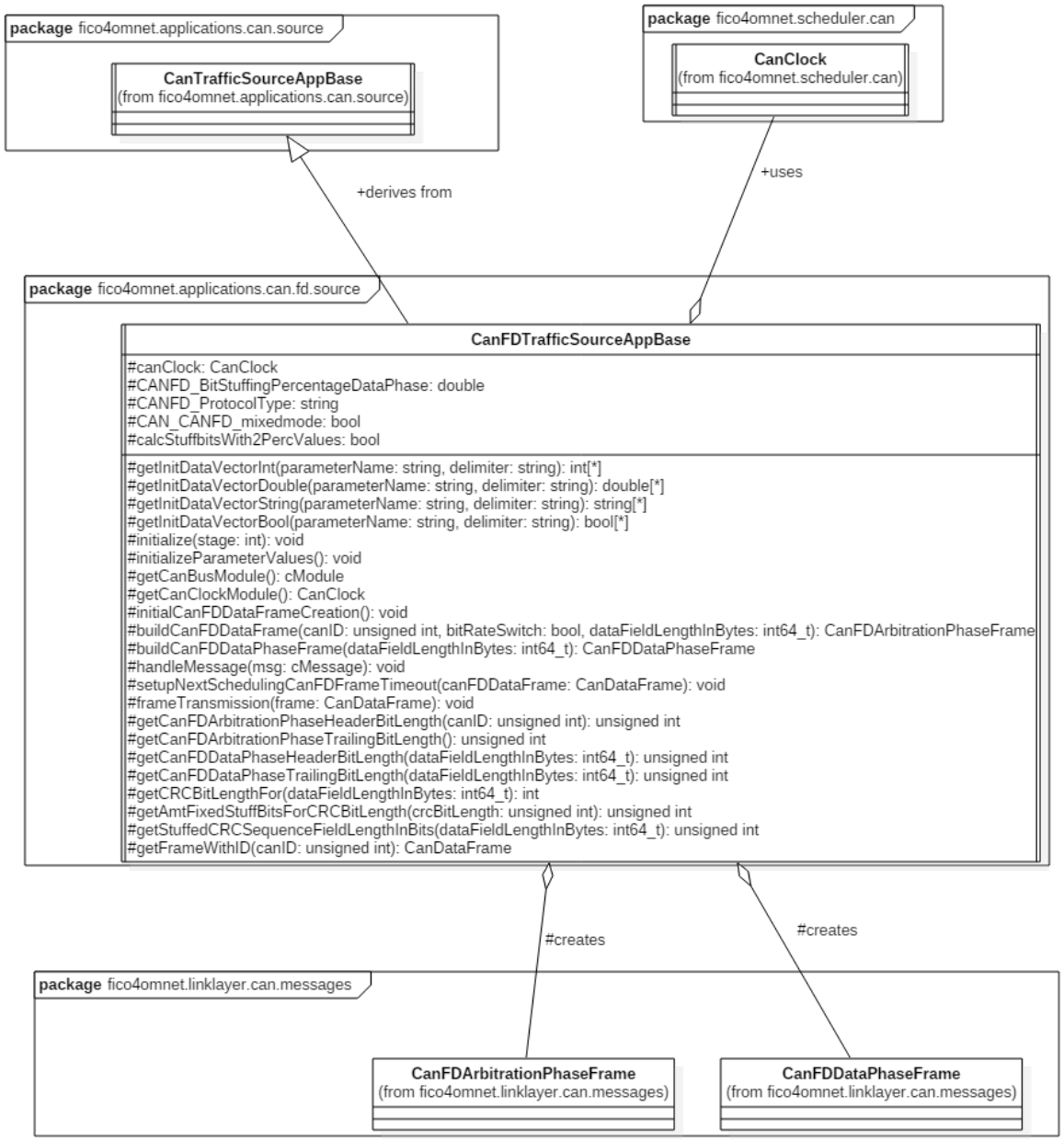

Abbildung 6.4. CanFDTrafficSourceAppBase Klassendiagramm

Soll ein CAN FD Busteilnehmer CAN FD Daten-Frames generieren, werden während der Initialisierungsphase der Simulation in dem Aufruf der Funktion `initialize(stage: int)`, die Funktionen `initializeParameterValues` und `initialCanFDDataFrameCreation` aufgerufen. Die Funktion `initializeParameterValues` liest die notwendigen Konfigurationsparameter des Netzwerks ein, während die Funktion `initialCanFDDataFrameCreation` die CAN FD Daten-Frames erzeugt, in der Liste `outgoingDataFrames` speichert, die CAN IDs beim `CanFDPortInput`-Modul registriert und

erzeugt und die ersten Scheduling-Zeiten für die Nachrichten berechnet und als Timeout-Nachrichten mit den kalkulierten Ausführungszeiten der FES übergibt.

6.5.1.1 Generierung von CAN FD Daten-Frames

Die Erzeugung von CAN FD Daten-Frames, sowie die Berechnung von Stuffbits erfolgt in dem Funktionsaufruf `buildCanFDDataFrame`, die die Argumente `canID`, `bitRateSwitch` und die Länge des Nutzdatenfelds in Bytes erhält. CAN FD Daten-Frames werden auf die Nachrichtentypen `CanFDArbitrationPhaseFrame` und `CanFDDataPhaseFrame` abgebildet. Der Grund für diese Aufteilung besteht darin, dass ein CAN FD Daten-Frame mit zwei Datenraten übertragen werden kann (siehe Abbildung 2.8). Die Bits, die mit der CAN-üblichen Standarddatenrate versendet werden, werden in die `CanFDArbitrationPhaseFrame`-Nachricht berechnet und eingetragen. Die restlichen Bits, die der Datenphase angehören und mit einer höheren Datenrate übertragen werden können, werden in eine `CanFDDataPhaseFrame`-Nachricht genauso eingefügt. Ein `CanFDArbitrationPhaseFrame`- und ein `CanFDDataPhaseFrame`-Typ ergeben zusammen ein vollständiges CAN FD Daten-Frame, das hinten auch die Bit-Länge des Interframe Space erhält und damit die Implementierung eines `CanDataFrame`-Typs für CAN Daten- und Remote-Frames nachbildet.

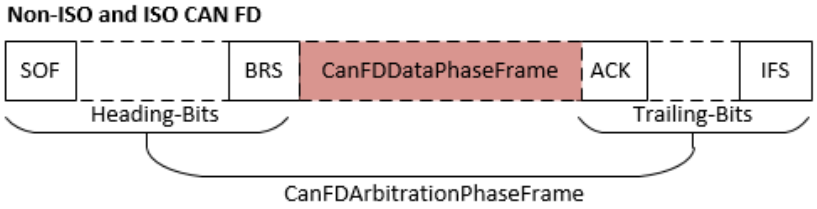

Abbildung 6.5. Unterteilung eines CanFDArbitrationPhaseFrames in Heading- und Trailing-Bits

Die Abbildung 6.4 veranschaulicht noch einmal das Bild, welche Bits in einem CanFDArbitrationPhaseFrame als eine Ganzzahl eingetragen werden. Weil die vorderen Heading-Bits eines CanFDArbitrationPhaseFrames der Stuffbit-Regel unterliegen und die hinteren Bits nicht, wird die Anzahl von Bits während der Berechnung für ein `CanFDArbitrationPhaseFrame`-Typ in Heading- und Trailing-Bits eingeteilt. Für die Berechnung von Heading- und Trailing-Bits definiert die Klasse CanFDTrafficSourceAppBase die Funktionen

(a) `getCanFDArbitrationPhaseHeaderBitLength` und
(b) `getCanFDArbitrationPhaseTrailingBitLength`,

die die Anzahl von Heading- und Trailing Bits für ein CAN FD Daten-Frame berechnen und zurückgeben. Dabei wird die Anzahl von Heading-Bits aus der gegebenen CAN ID erschlossen. Übersteigt der Wert der CAN ID von 2047, werden die Heading-Bits nach dem CAN FD Extended Frame Format kalkuliert und zurückgegeben. Andernfalls wird die Anzahl von Heading-Bits nach dem CAN FD Base Frame Format berechnet. Für die Anzahl von Heading-Bits eines CAN FD Daten-Frames im Base Frame Format

sind die vorderen 17 Bits und für die Anzahl von Heading-Bits eines CAN FD Daten-Frames im Extended Frame Format sind die vorderen 36 Bits relevant (siehe Abbildung 2.7 und 2.9).

Die vorderen Heading-Bits eines CAN FD Daten-Frames werden in dem `CanFDArbitrationPhaseFrame`-Parameter `unstuffedFrameHeaderBitLength` gespeichert für eine anschließende Stuffbit-Berechnung gespeichert.

Für die Bestimmung der Bit-Länge für ein `CanFDDataPhaseFrame`-Typ wird das Gleiche Verfahren angewendet. Ein `CanFDDataPhaseFrame` wird auch logisch in Heading- und Trailing-Bits unterteilt (siehe Abbildung 6.5).

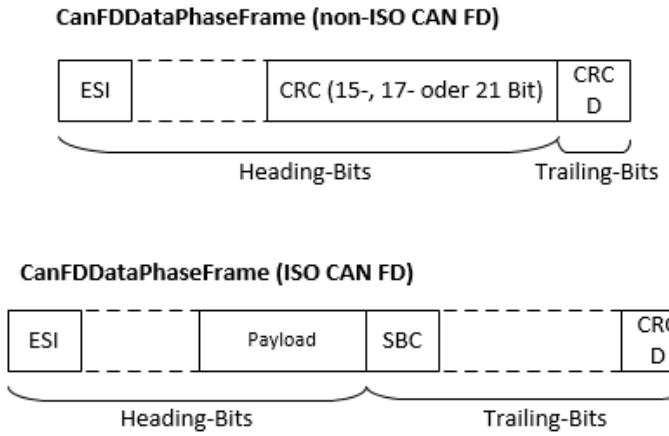

Abbildung 6.6. Unterteilung eines CanFDDataPhaseFrames in Heading- und Trailing-Bits abhängig vom Spezifikationstyp

Während bei der non-ISO CAN FD Variante die Bitfolge vom Bit ESI bis zum Ende der CRC-Bitsequenz der Stuffbit-Regel unterliegt und deshalb in Heading-Bits einfließt, ist diese Bitsequenz bei der ISO CAN FD Variante kürzer und endet nach dem Nutzdatenfeld (Payload). Die Entscheidung nach welchem CAN FD Daten-Frame Nachrichtentyp ein `CanFDDataPhaseFrame` aufgebaut wird, kann in der CanBus.ned Datei über den Parameter `CANFD_ProtocolType` getroffen werden. Möglich sind beide CAN FD Daten-Frame Varianten.

Die Bestimmung von Bitlängen für ein CanFDDataPhaseFrame in Abhängigkeit vom Spezifikationstyp erfolgt in den Funktionen der CanFDTrafficSourceAppBase-Klasse

(a) `getCanFDDataPhaseHeaderBitLength`,
(b) `getCanFDDataPhaseTrailingBitLength`,
(c) `getCRCBitLengthFor`
(d) `getAmtFixedStuffBitsForCRCBitLength`
(e) `getStuffedCRCSequenceFieldLengthInBits`

Die beiden unteren Funktionen `getAmtFixedStuffBitsForCRCBitLength` und `getStuffedCRCSequenceFieldLengthInBits` dienen nur der CRC Feld-Bitlängenbestimmung für CAN FD Daten-Frames im ISO Standard. Die oberen drei Funktionen werden für beide Protokollvarianten verwendet. Das `CanFDDataPhaseFrame` definiert, wie das

CanFDArbitrationPhaseFrame, den Parameter `unstuffedFrameHeaderBitLength`, der anschließend für die Berechnung von Stuffbits in der Datenphase verwendet wird.

6.5.1.2 Berechnung von Stuffbits

Die Berechnung von Stuffbits kann nach zwei möglichen Varianten erfolgen. Die Varianten zur Berechnung von Stuffbits können über den Parameter `calcStuffBitsWith2PercValues`, der in der CanBus.ned Datei definiert ist, angegeben werden. Wird dieser Parameter gesetzt, so wird die Stuffbit-Berechnung unter Anwendung von zwei Prozentwertangaben durchgeführt. Zur Anwendung von zwei Prozentwertangaben müssen die Parameter `bitStuffingPercentage` und `CanFD_BitStuffingPercentageDataPhase` bestimmt werden. Mit `bitStuffingPercentage` wird die Anzahl von Stuffbits für die vorderen Heading-Bits des CanFDArbitrationPhaseFrames bestimmt. Dafür wird der ungünstigste Fall von Stuffbits aus der eingetragenen Bit-Länge im Parameter `unstuffedFrameHeaderBitLength` berechnet und die nachfolgende Formel eingesetzt

$$\frac{\text{unstuffedHeaderBitLength} - 1\ \text{Bit}}{4}$$

Die Subtraktion mit 1 bedeutet auch hier eine Korrektur von Bitschritten, durch die anschließend geteilt wird, um auf die korrekte Anzahl von Stuffbits im ungünstigsten Fall für die vorderen Heading-Bits eines CAN FD Daten-Frames zu kommen (siehe dazu die Abbildung 3.11). Abhängig vom CAN FD Frame Format können hier entweder 4, berechnet mit

$$(17 - 1)/4$$

oder 8, berechnet mit

$$(36 - 1)/4$$

Stuffbits im ungünstigsten Fall für die Heading-Bits eines `CanFDArbitrationPhaseFrames` berechnet werden, die dann mit der Zufallszahl im Bereich von 0 bis zum angegebenen Prozentwert im Parameter `bitStuffingPercentage` multipliziert werden, um die Anzahl von Stuffbits zu erhalten, die der Bit-Länge des `CanFDArbitrationPhaseFrames` hinzugefügt wird.

Der gleiche Ablauf wird für die Berechnung von Stuffbits für die CAN FD Datenphase initiiert.

$$\frac{\text{n} + \text{unstuffedHeaderBitLength}}{4}$$

Für die korrekte anschließende Berechnung von Stuffbits der CAN FD Datenphase wird um n Bits (mit $\text{n} \in \mathbb{N}$) bis zum vorhergehenden möglichen Stuffbit zurückgegangen und die Anzahl von Heading-Bits im Parameter `unstuffedFrameHeaderBitLength` des `CanFDDataPhaseFrames` vergrößert. Das hat den Hintergrund, dass bei der Berechnung von Stuffbits in der CAN FD

Arbitrierungsphase ein möglicher Rest übrig bleibt, der in die Stuffbit-Berechnung im ungünstigsten Fall in der Arbitrierungsphase nicht eingeflossen ist, aber für die korrekte Stuffbit-Berechnung in der CAN FD Datenphase eine Rolle spielt. Da CAN FD entweder im Base Frame oder Extended Frame Formaten existiert, resultieren für n unterschiedliche Zahlenwerte, um die der `unstuffedFrameHeaderBitLength` Parameter logisch für die korrekte Stuffbit-Berechnung vergrößert wird. Die so gebildete Bitanzahl bedarf keiner weiteren Korrektur und wird durch die 4 Bit Schritte geteilt, um auf die Anzahl von Stuffbits im ungünstigsten Fall in der CAN FD Datenphase zu kommen. Das Ergebnis wird wieder mit der Zufallszahl aus dem Bereich von 0 bis zum angegebenen Prozentwert, das im Parameter `CANFD_BitStuffingPercentageDataPhase` festgehalten wird, multipliziert. Dadurch resultiert die Anzahl von Stuffbits, die in der Bit-Länge für CAN FD Datenphase berücksichtigt wird.

Das zweite mögliche Verfahren für die Berechnung von Stuffbits kann mit nur einem angegebenen Prozentwert durchgeführt werden. Falls der Parameter `calcStuffBitsWith2PercValues` nicht gesetzt wird, werden die beiden Bit-Längen aus den Parametern `unstuffedFrameHeaderBitLength` eines CanFDArbitrationPhaseFrames und CanFDDataPhaseFrames zusammen addiert und zur Bestimmung von Stuffbits im ungünstigsten Fall angewendet. Dazu wird die Formel

$$\frac{\text{Anzahl Bits} - 1}{4}$$

angewendet. Das Ergebnis wird mit der Zufallszahl, die wieder von 0 bis zum angegebenen Prozentwert in `bitStuffingPercentage`, generiert wird, multipliziert und ergibt die Anzahl von Stuffbits, die von links nach rechts, beginnend mit dem Bit SOF und endend mit der Bitsequenz, die noch der Stuffbit-Regel unterliegt, verteilt. Übersteigt die Anzahl von berechneten Stuffbits die maximale Anzahl von Stuffbits im ungünstigsten Fall, die in der CAN FD Arbitrierungsphase eingetragen werden können, so werden sie in die CAN FD Datenphase eingetragen.

6.5.1.3 Bestimmung von Bitlängen

Die Bestimmung von Heading- und Trailing-Bits für beide CAN FD Daten-Frame Nachrichtentypen `CanFDArbitrationPhaseFrame` und `CanFDDataPhaseFrame`, sowie die Berechnung von zugehörigen Stuffbits für jeden Nachrichtentyp, erzeugt die Gesamtbitlänge für den jeweiligen Typ durch das Aufsummieren dieser drei Werte: Heading-Bits, berechnete Stuffbits und Trailing-Bits.

Für die Berechnung von Stuffbits für das `CanFDDataPhaseFrame` werden die Nutzdaten in Bits während der Berechnung von Heading-Bits auch hinzugefügt. Deshalb werden diese Bits beim Setzen der Bitlänge in das `CanFDDataPhaseFrame` von der Gesamtbitlänge subtrahiert. Diese Korrektur ist notwendig, weil die Nutzdaten als ein separates Packet mit einer vorgegebenen Bitlänge in ein `CanFDDataPhaseFrame` eingekapselt werden, das folglich zur automatischen Vergrößerung der Bitlänge des `CanFDDataPhaseFrames` und damit zur korrekten Gesamtbitlänge eines `CanFDDataPhaseFrames` führt.

Die resultierten Gesamtbitlängen werden mit der Funktion setBitLength, die in der `cPacket`-Klasse definiert ist, in die Nachrichtentypen gesetzt. Die Einkapselung von Nachrichtentypen erzeugt ein gültiges CAN FD Daten-Frame Nachrichtentyp.

6.5.2 CanFDPortOutput

Das CanFDPortOutput-Modul realisiert neben der Weiterleitung von CAN Nachrichtenformaten auch die Weiterleitung von CAN FD Daten-Frames und simuliert für CAN FD Daten-Frames den Versand von Sendefehlern.

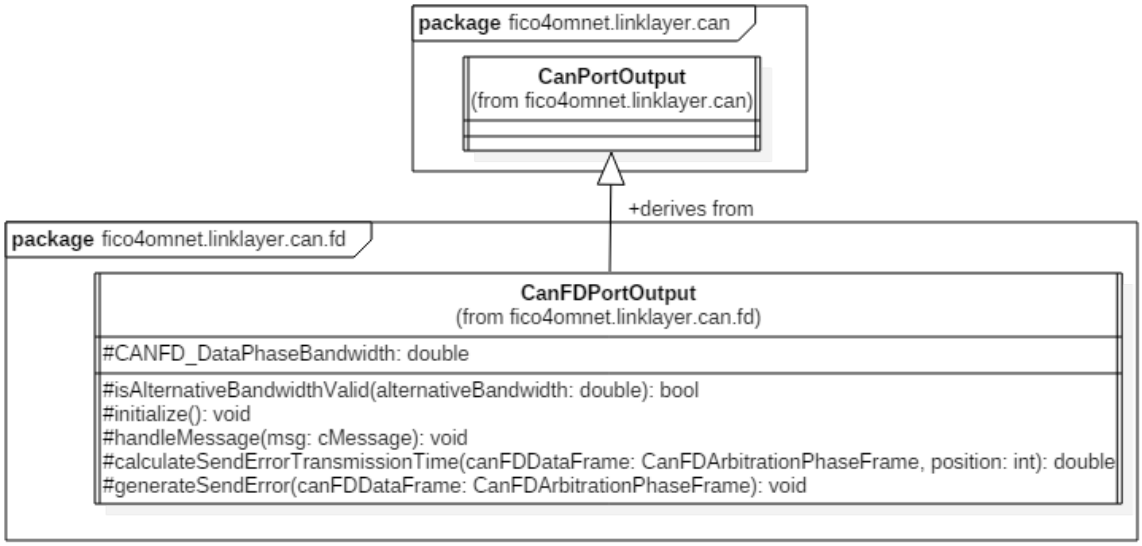

Abbildung 6.7. CanFDPortOutput Klassendiagramm

Ein CanFDPortOutput-Modul leitet von dem CanPortOutput-Modul ab und verwendet alle seine Variablen, um die Senderfehler für weitergeleitete `CanFDArbitrationPhaseFrames` zu generieren.

Empfängt das `CanFDPortOutput`-Modul ein `CanFDArbitrationPhaseFrame` vom `CanOutputBuffer`-Modul, wird für diese Nachricht im Aufruf der Funktion `generateSendError`, die ein CAN FD Daten-Frame erhält, der Timeout für die Ausführung des Sendefehlers generiert. Dabei folgt der Algorithmus dem Algorithmus des CanPortOutput-Moduls aus dem Abschnitt 3.3.2.4.1. Der Unterschied liegt nur in der Bestimmung der Ausführungszeit für die erzeugte Fehlernachricht, die bei einem CAN FD Daten-Frame von zwei Datenraten abhängt. Die Bestimmung der Position für das Zeitfenster im CanFDArbitrationPhaseFrame, an dem der Fehler gesendet werden soll, wird als eine Zufallszahl bestimmt. Die Zufallszahl wird im Bereich von 0 und der Gesamtbitlänge des `CanFDArbitrationPhaseFrames` subtrahiert mit `MAXERRORFRAMESIZE` generiert. Ist die Position des Fehlers bestimmt, wird dafür die zukünftige Ausführungszeit berechnet. Dafür werden aus dem `CanFDArbitrationPhaseFrame` die Parameter

`unstuffedFrameHeaderBitLength` und `calculatedStuffBits` gelesen und zusammenaddiert. Ist die bestimmte Position größer, als das Ergebnis der Addition, wird die zukünftige Ausführungszeit der Fehlernachricht folgend berechnet

anzArbitrationPhaseHeaderFrameBits = unstuffedFrameHeaderBitLength + calculatedStuffBits

wenn position > anzArbitrationPhaseHeaderFrameBits **dann**

$$(\text{position} - \text{anzArbitrationPhaseHeaderFrameBits})/(\text{höhere Datenrate}) + \text{anzArbitrationPhaseHeaderFrameBits}/(\text{Standard Datenrate})$$

sonst

$$\text{position}/(\text{Standard Datenrate})$$

6.5.3 CanFDPortInput

Der CanFDPortInput-Modul simuliert den Empfang sowohl von CAN Daten- und Remote-Frames als auch von CAN FD Daten-Frames. Weiterhin ist hier die Simulation von Empfangsfehlern für CAN FD Daten-Frames integriert.

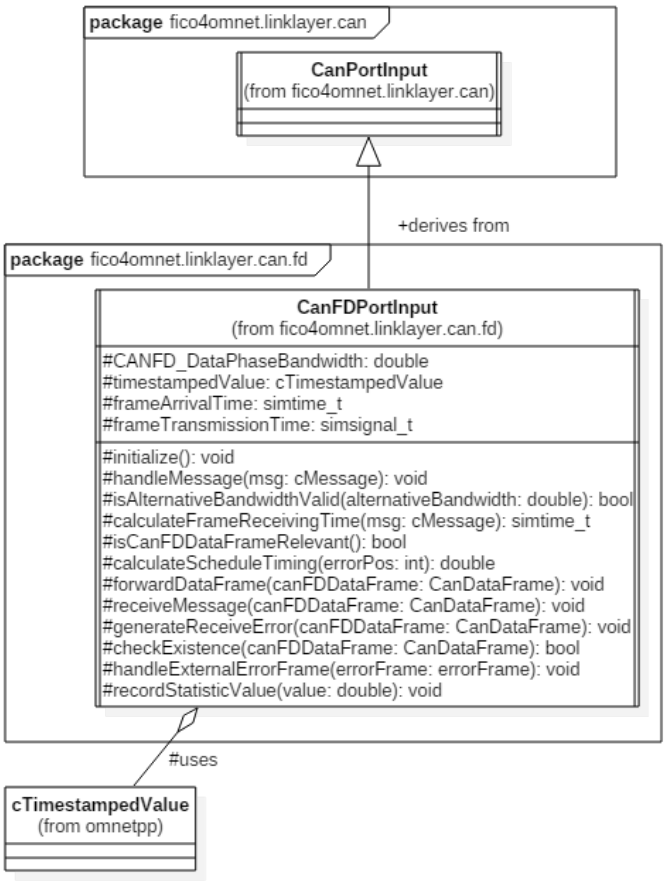

Abbildung 6.8. CanFDPortInput Klassendiagramm

Die Umsetzung von Empfangsfehlern erfolgt äquivalent der Umsetzung von Sendefehlern des CanFDPortOutput-Moduls aus dem Abschnitt 6.5.3. Statt der Funktion `generateSendError` wird für die Erzeugung von Empfangsfehlern die Funktion `generateReceiveError`, die das empfangene CAN FD Daten-Frame erhält, aufgerufen. In der Funktion wird derselbe Algorithmus, wie es in der Abbildung 6.8 beschrieben ist, zur Erzeugung eines Empfangsfehlers während des Nachrichtenempfangs durchlaufen.

6.5.4 CanFDTrafficSinkAppBase

Die CAN FD Sink Apps erweitern die `handleMessage` Funktion der CanTrafficSinkAppBase-Klasse

7 Qualitätssicherung

Qualitätssicherung beschreibt Verfahren zur Messung von Softwarequalität auf unterschiedlichen Stufen der Softwareentwicklung. Dieses Kapitel befasst sich mit messbaren Größen der Softwareentwicklung, die als Metriken bezeichnet werden.

7.1 Methoden und Maßnahmen

Zur Messung von Softwarequalität auf der Komponentenebene werden sogenannte Black- und White-Box-Tests eingesetzt. Bei Black-Box-Tests werden die äußeren Schnittstellen der Komponente durch zufällige Eingabeparameter unter Einhaltung von Vorbedingungen getestet und Ergebnisse durch den Vergleich mit Nachbedingungen beobachtet. Beim einem White-Box-Test liegt das Wissen über jedes Verhalten der Komponente vor. Der geschriebene Code liegt offen: Anhand des Ausführungsgraphen können Testfälle zur Ermittlung der Codequalität definiert werden. Dabei wird der Code nach sogenannten Überdeckungskriterien untersucht. Der White-Box-Test unterscheidet mehre Überdeckungskriterien:

(a) Codeüberdeckung
(b) Anweisungsüberdeckung
(c) Pfadüberdeckung, u.a.

7.2 Beispiel

Anhand eines Beispiels soll eine Anweisungsüberdeckung gezeigt werden. Dafür wird eine `handleMessage` Funktion aus dem `CanFDPortOutput`-Modul herangezogen. Aus der Funktion entsteht folgender Kontrollflussgraph

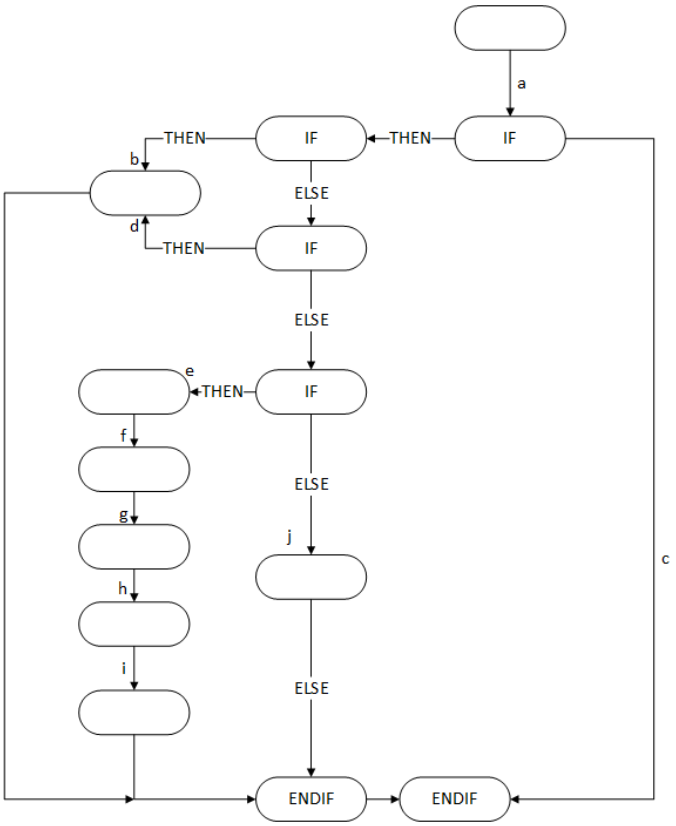

Abbildung 7.1. Kontrollflussgraph der Funktion handleMessage des CanFDPortOutput-Moduls

Für eine vollständige Anweisungsüberdeckung müssen demnach folgende Testfälle generiert werden:

(a) a, c
(b) a, b
(c) a, d
(d) a, e, f, g, h, i
(e) a, j

8 CAN FD-Modellevaluierung

In diesem Kapitel wird die Integration von CAN FD in das bestehende CanBus-Simulationsmodell infolge von Versuchsdurchführungen auf die Korrektheit der Umsetzung überprüft. Anhand von Erwartungswerten und simulierten Ergebnissen kann ermittelt werden, ob das CAN FD Simulationsmodell die CAN FD Spezifikation erfüllt.

8.1 Versuchsdurchführung 1

Die CanFDArbitrationPhaseFrame-Nachricht unterstützt den Parameter BRS mit dem das Versenden eines CAN FD Daten-Frames mit zwei oder einer Datenrate simuliert werden kann. Die Übertragung eines CAN FD Daten-Frames mit einer Datenrate entspricht nahezu der Übertragung einer CAN Nachricht unter Voraussetzung, dass gleiche Nutzdatenlängen übertragen werden. Die Bit-Länge eines CAN Daten-Frames im Base Frame Format ohne Nutzdaten inklusive dem IFS Feld beträgt 47 Bits. Beim CAN FD Daten-Frame im gleichen Frame Format ohne Nutzdaten inklusive dem IFS Feld beträgt die Bit-Länge 51 Bits. Bei dieser Versuchsdurchführung wird das BRS Feld für die Übertragung von CAN FD Daten-Frames nicht gesetzt. Dadurch kann angenommen werden, dass die Latenzzeiten bei der Übertragung von CAN Daten- oder Remote-Frames und von CAN FD Daten-Frames nahezu identisch aussehen.

Die Simulation des beschriebenen Szenarios liefert die Zeitmessung beim Nachrichtenempfang eines CAN FD Busteilnehmers mit dem Namen `CAN_FD_Node1`. Dafür wird die Zeit beim Eintreffen der Nachricht am Eingangsgate des Moduls und die anschließende simulierte Empfangszeit der Nachricht in Form eines Timeouts im Modul festgehalten und daraus die Differenz gebildet. Die Differenz wird anschließend in einer Liste von Statistikwerten gesammelt, die schließlich im Result Analysis Tool von OMNeT++ als nachfolgendes Liniendiagramm beobachtet werden kann. Für die Datenübertragungsrate ist der Standardwert 1MBit/s gewählt worden.

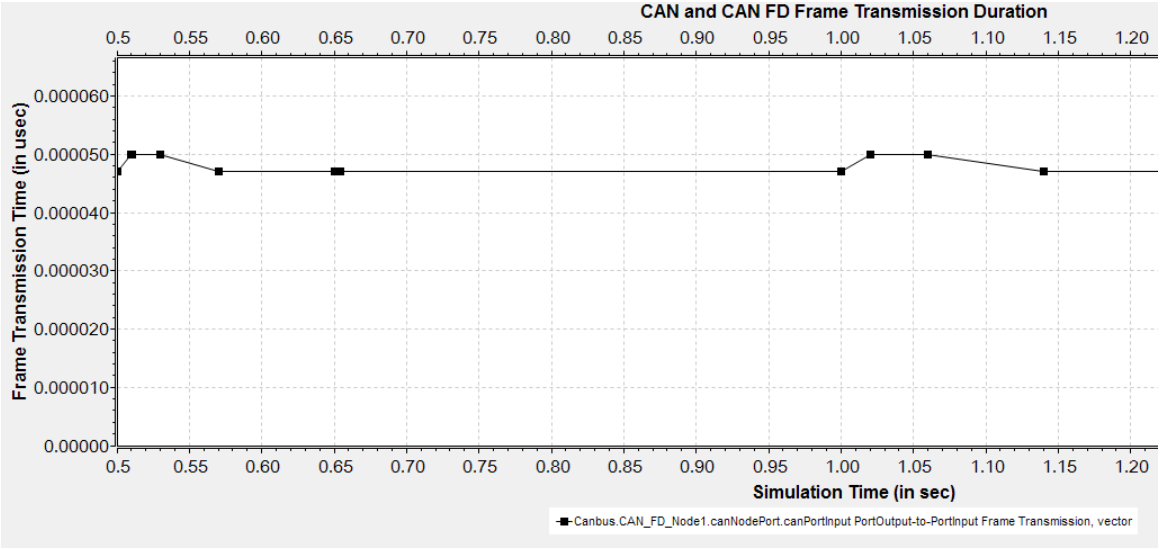

Abbildung 8.1: Simulierte Empfangszeiten bei der Übertragung von CAN und CAN FD Daten-Frames.

An diesem Diagramm sind deutlich die Nachrichtengrößen von CAN und CAN FD Daten-Frames aus der Übertragungszeit auf der Y-Achse erkennbar und die Nachrichten in CAN und CAN FD Nachrichten leicht unterschieden werden können. Die Empfangszeit von CAN FD Daten-Frames mit 50 Bit liegt etwas höher als die der CAN Daten-Frames mit 47 Bit, und resultiert aus der Bit-Länge geteilt durch die Übertragungsrate, die links auf der X-Achse beobachtet werden kann.

8.2 Versuchsdurchführung 2

In dieser Versuchsdurchführung werden die CAN FD Eigenschaften gezeigt. Dafür werden die übertragenen CAN FD Daten-Frames in zwei Gruppen aufgeteilt. Die eine Gruppe überträgt die CAN FD Daten-Frames unter Anwendung einer zweiten Datenrate und die andere Gruppe überträgt die CAN FD Daten-Frames unter Anwendung der CAN Standard Datenrate. Beide Gruppen übertragen ein CAN FD Daten-Frame im Base Frame Format ohne die Nutzdaten. Die Länge des CAN FD Daten-Frames ohne Nutzdaten beträgt 50 Bits. Für die Standard Datenrate wird die Datenübertragungsrate 1MBit/s und für die zweite höhere Datenrate die Datenübertragungsrate 2MBit/s gewählt. Das nachfolgende Liniendiagramm führt zu diesem Ergebnis

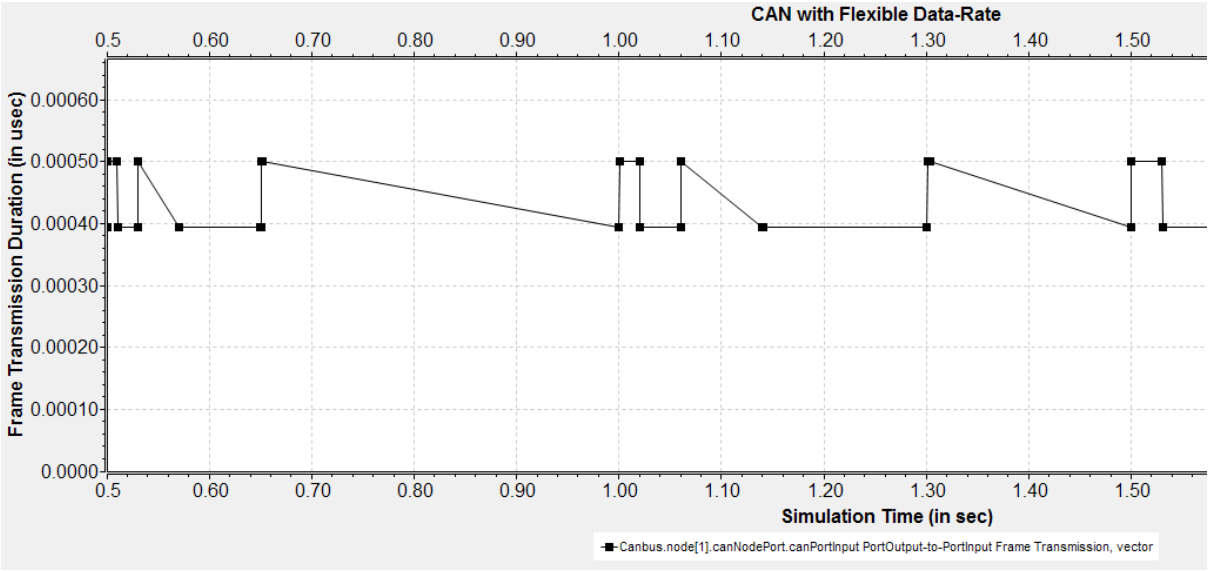

Abbildung 8.2: CAN with Flexible Data-Rate

Für die Übertragung eines CAN FD Daten-Frames ohne das Nutzdatenfeld von 50 Bits mit der Standard Datenrate von 1MBit/s sind 50 Mikrosekunden notwendig, die auch in der Versuchsdurchführung 1 beobachtet werden können. Für die Übertragung eines CAN FD Daten-Frames ohne das Nutzdatenfeld mit zwei Datenraten, werden 29 Bit mit der Standard Datenrate von 1MBit/s und 21 Bit mit der höheren Datenrate von 2MBit/s übertragen. Das führt zu einem Ergebnis von 39,5 Mikrosekunden und einer Leistungssteigerung von ca. 27 Prozent.

9 Zusammenfassung und Ausblick

9.1 Zusammenfassung

In dieser Arbeit sollte zur Simulation von CAN FD die bestehende CanBus-Simulationsumgebung um CAN FD erweitert werden, damit CAN FD auch in gemischten Netzwerken zusammen mit CAN simuliert werden kann. CAN FD ist die Zukunft von CAN Netzwerken und bietet eine Leistungsverbesserung dank zwei wesentlichen Eigenschaften:

(a) Während einer Nachrichtenübertragung die Anwendung einer zweiten höheren Datenrate und
(b) Im Vergleich zu CAN lassen sich mit CAN FD mehr Nutzdaten in einer Nachricht versenden, als es bisher möglich war.

Die Verbesserung der CAN Datenübertragung mit CAN FD für CAN Netzwerke bringt viele Vorteile und lässt den Einsatz von CAN in der Automobilbranche aufrecht erhalten, ohne auf die neuen revolutionären Bussystemlösungen, wie FlexRay oder Ethernet in dem CAN Segment zugreifen zu müssen. Die direkte Migration von CAN FD in CAN Netzwerke kann nur unter bestimmten Voraussetzungen erfolgen.

Simulationen bieten eine gute Gelegenheit für neuen Entwicklungen, wie für CAN FD, bevor sie erst eingesetzt werden, über Modelle und Analysen Erfahrungen zu sammeln, die auf ein reales Umfeld projiziert werden können.

9.2 Ausblick

Bei der Integration von CAN FD in das vorhandene CAN Bus-Simulationsmodell sind viele Szenarien umgesetzt worden, die für CAN Netzwerke den Bedarf an Weiterentwicklung des CAN FD Simulationsmodells einschränken.

Als eine mögliche Erweiterung des CAN-Simulationsmodell könnte ein CRC Wert für Daten- und Remote-Frames simuliert werden, durch den eine Nachrichten verifiziert werden kann.

Interessant wären Arbeiten, die sich mit protokollübergreifender CAN FD Simulation befassen oder vielleicht neues Scheduling-Konzept zur Übertragung von Nachrichten in CAN Netzwerken einsetzten.

10 Literaturverzeichnis

1. **Decker, Peter.** Wege des klassischen CAN zum verbesserten CAN FD. [Hrsg.] elektroniknet.de. *Elektronik automotive.* April 2013, 4, S. 38-41.

2. **Zimmermann, Werner und Schmidgall, Ralf.** Bussysteme in der Fahrzeugtechnik. Wiesbaden : Springer Vieweg, 2014, S. 96-118.

3. —. Bussysteme in der Fahrzeugtechnik. Wiesbaden : Springer Vieweg, 2014, S. 138-148.

4. **Robert Bosch Gmbh.** CAN with Felxible Data-Rate, Specification, Version 1.0. *www.can-newsletter.org.* [Online] 17. April 2012. [Zitat vom: 18. Dezember 2016.] https://can-newsletter.org/assets/files/ttmedia/raw/e5740b7b5781b8960f55efcc2b93edf8.pdf.

5. **Can in Automation.** CAN FD - The basic idea. *www.can-cia.org.* [Online] [Zitat vom: 16. März 2017.] https://www.can-cia.org/can-knowledge/can/can-fd/.

6. **Wiesinger, Johannes.** Der CAN-Bus - Grundlagen von Automobil Bussystemen. *www.kfztech.de.* [Online] 17. März 2017. [Zitat vom: 2. Mai 2017.] http://www.kfztech.de/kfztechnik/elo/can/can_grundlagen_1.htm.

7. **Robert Bosch GmbH.** CAN Specification, Version 2.0. *www.bosch-semiconductors.de.* [Online] September 1991. [Zitat vom: 16. Dezember 2016.] http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf.

8. *CAN FD verbessert.* **Zeltwanger, Holger.** [Hrsg.] elektroniknet.de. 23. März 2015, Elektronik automotive.

9. *ISO CAN FD or non-ISO CAN FD.* **CAN in Automation.** [Hrsg.] Can in Automation. 12. September 2014, CAN Newsletter Online.

10. **Supke, Jorg und Zimmermann, Werner.** Diagnosesysteme im Automobil, Bussysteme im Fahrzeug. *www.emotive.de.* [Online] [Zitat vom: 12. März 2017.] https://www.emotive.de/documents/Seminare/DiagnosesystemeKomplettseminar.pdf.

11. **Zimmermann, Werner und Schmidgall, Ralf.** Bussysteme in der Fahrzeugtechnik. Wiesbaden : Springer Vieweg, 2014, S. 57-79.

12. **Monroe, Scott, Stout, David und John, Griffith.** Solutions of CAN and CAN FD in a mixed network topology. *www.can-cia.org.* [Online] 2013. [Zitat vom: 10. Dezember 2016.] https://www.can-cia.org/fileadmin/resources/documents/proceedings/2013_monroe.pdf.

13. **omnetpp.org.** OMNeT++ Simulation Manual. *omnetpp.org.* [Online] [Zitat vom: 2. Mai 2017.] https://omnetpp.org/doc/omnetpp/manual/.

11 Abbildungsverzeichnis

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

*Hamburg, den*_____ _____

Andrej Wittmann