Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Stefan Buschmann

OMNeT++ basierte Simulation von FlexRay Netzwerken zur Analyse von Automotive Anwendungen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Stefan Buschmann

OMNeT++ basierte Simulation von FlexRay Netzwerken zur Analyse von Automotive Anwendungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Bernd Schwarz

Eingereicht am: 23. November 2012

Stefan Buschmann

Thema der Arbeit
OMNeT++ basierte Simulation von FlexRay Netzwerken zur Analyse von Automotive Anwendungen

Stichworte
FlexRay, OMNeT++, Fahrzeug-Netzwerke, Echtzeitsysteme, diskrete ereignisbasierte Simulation, Synchronisation

Kurzzusammenfassung
In einem modernen Fahrzeug gibt es eine Vielzahl an Assistenz- und Informationssystemen. Um eine sichere und schnelle Kommunikation zwischen den Steuergeräten in einem Automobil zu gewährleisten, wurde das Kommunikationssystem FlexRay entwickelt. Es bietet eine höhere Bandbreite als Protokolle wie LIN und CAN und eignet sich außerdem auch für den Einsatz bei kritischem Datenverkehr. In dieser Arbeit wird dieses Kommunikationssystem in der diskreten ereignisbasierten Simulationsumgebung OMNeT++ implementiert. Es ist möglich FlexRay-Netzwerke individuell aufzubauen und zu konfigurieren. Jedem Teilnehmer können Zeitpunkte zugeordnet werden, zu denen sie Nachrichten an die anderen Knoten schicken. Die lokale Uhr eines jeden Knotens kann durch einen Drift von der eigentlichen Zeit abweichen. Diese Abweichung wird mittels eines Synchronisationsverfahrens kompensiert.

Title of the paper
OMNeT++ based simulation of FlexRay networks for analysis of automotive applications

Keywords
FlexRay, OMNeT++, in-vehicle networks, real time systems, discrete eventbased simulation, synchronization

Abstract
In a modern car, there are a variety of assistance and information systems. To ensure a secure and fast communication between the controllers in an automobile, the FlexRay communication system was developed. It offers a higher bandwidth than protocols like CAN and LIN, and is also suitable for use in critical traffic. In this work, this communication system is implemented in the discrete event-based simulation environment OMNeT++. It is possible to build up and configure FlexRay networks individually. To each subscriber slots can be assigned at which they send messages to the other nodes. The local clock of each node may differ by a drift of the actual time. This deviation is compensated by a synchronization process.

Inhaltsverzeichnis

1 Einleitung

Im Kfz-Bereich werden die Anforderungen an die Bussysteme immer größer. In einem modernen Fahrzeug gibt es inzwischen eine Vielzahl an Assistenz- und Informationssystemen. Aber auch die Multimediaanwendungen finden Einzug in ein Automobil. Als Beispiel kann man hier integrierte Navigationssysteme, Rückfahrkameras, Einparkassistenten oder Bildschirme in den Kopfstützen nennen. Schon im Jahr 2006 prognostizierte der Verband der Automobilindustrie (VDA), dass der Wertschöpfungsanteil der Elektronik von 20 % bis zum Jahr 2015 auf ca. 35 % ansteigen wird (vgl. VDA, 2006, S. 149). Hier ist ein klarer Trend zu erkennen. Mit dem Anstieg der Anforderungen durch diese Systeme steigt natürlich auch die Anforderung an die bisherigen Bus-Technologien. Ältere Bussysteme wie beispielsweise der Controller Area Network (CAN) Bus können bestimmte Aufgaben möglicherweise gar nicht mehr oder nicht ausreichend übernehmen.
Weitere Anforderungen an das Kommunikationssystem entstehen durch Anwendungen mit Echtzeitanforderungen. Ein solcher Fall tritt auf, wenn ein „X-By-Wire“ System realisiert werden soll. Bei dieser Technologie werden Steuerbefehle elektronisch übertragen und bisherige mechanische Systeme ersetzt. Hierbei ist es essentiell, dass z. B. die Lenkdaten bei einem „Steer-By-Wire“ System rechtzeitig beim Empfänger ankommen, da es sonst schwerwiegende Folgen haben könnte. Für einen solchen Anwendungsfall wird ein Bussystem benötigt, das die Ergebnisse in der geforderten Zeit liefert. Aus diesem Grund wurde im September 2000 das FlexRay-Konsortium gegründet, das sich mit der Entwicklung eines genau solchen Systems widmete.

> „In den letzten Jahren hat der Umfang von Elektronik im Fahrzeug wesentlich zugenommen. Aktuelle Fahrzeuge stellen ein komplexes, verteiltes elektronisches System dar. Die große Anzahl an elektronischen Steuergeräten (ECUs) und die daraus erforderlichen Datentransfers benötigen eine hohe Bandbreite und verlässliche Kommunikationssysteme. FlexRay beschreibt ein Kommunikationsprotokoll, das diese Anforderungen erfüllt.“ (Muller und Valle, 2011, S. 228)

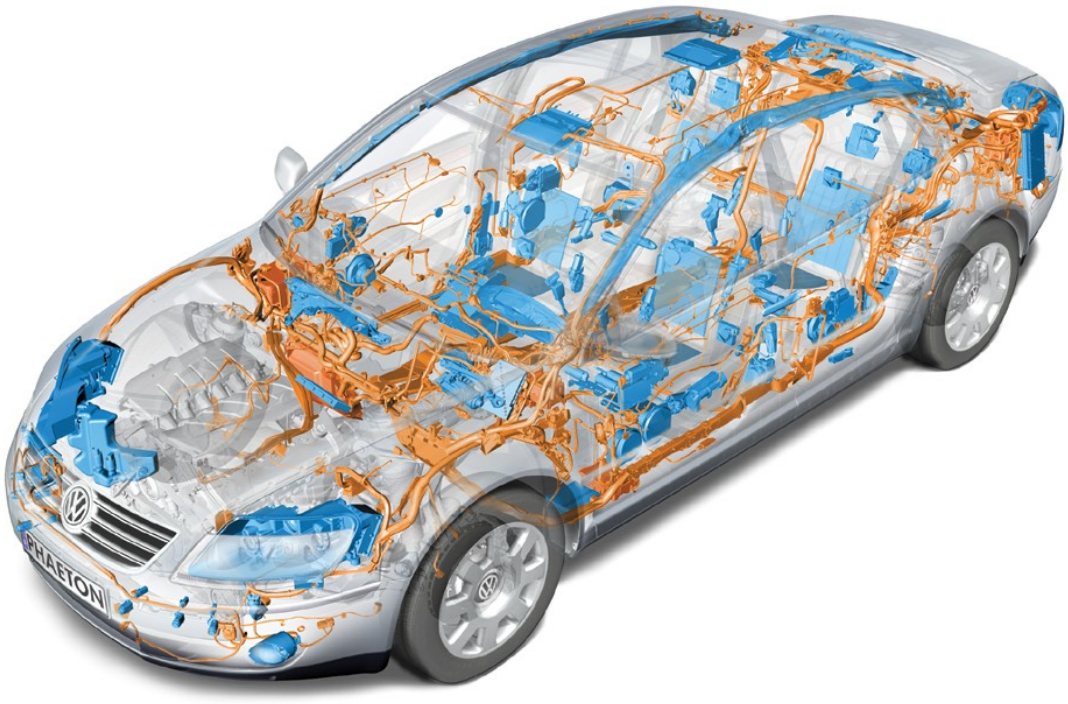

Abbildung 1.1: Bordnetz im VW Phaeton (Quelle: NYFEGA Elektro-Garage AG)

Motivation

Im CoRE-Projekt (vgl. CoRE RG, a) der Hochschule für Angewandte Wissenschaften Hamburg, in dessen Umfeld diese Bachelorarbeit entsteht, wird am Einsatz von Time-Triggered-Ethernet (TTEthernet oder TTE) im Automobil gearbeitet. Bei dieser Technologie wird das klassische Ethernet echtzeitfähig gemacht. Im Rahmen dieses Projektes wurde eine Simulation dieser Ethernet Variante erstellt. Als Simulationsplattform dient hier OMNeT++ (siehe Abschnitt 2.3) (vgl. OMNeT++ Community, a).
Da in einem Fahrzeug je nach Anwendungsbereich und Anforderungen unterschiedliche Bussysteme zum Einsatz kommen, ist es interessant die Technologien auch in der Simulation miteinander zu verbinden, um so komplexere Systeme simulieren zu können. Deswegen sollen neben dem TTEthernet auch für Bussysteme wie z. B. FlexRay oder CAN Simulationen erstellt werden. Für die busübergreifende Kommunikation sollen Gateways verwendet werden, die die Nachrichten zwischen den unterschiedlichen Bussen übersetzen und weiterleiten. Neben

der Verbindung der unterschiedlichen Techniken können sie auch für die gleiche Aufgabe konfiguriert und anschließend die Ergebnisse miteinander verglichen werden, um so die Stärken und Schwächen aufzeigen zu können.

Zielsetzung und Abgrenzung

In dieser Arbeit soll das FlexRay-Bussystem in der Simulationsumgebung OMNeT++ implementiert werden, um eine Simulation von FlexRay zu ermöglichen. Hierzu werden basierend auf der FlexRay-Spezifikation (vgl. FlexRay Consortium, 2010b) die unterschiedlichen Komponenten erstellt und angepasst. Der grundsätzliche Aufbau der Simulationskomponenten soll an die TTEthernet-Simulation angelehnt sein.
Die OMNeT++ Umgebung wurde gewählt, da diese Software im Rahmen des CoRE-Projekts der HAW-Hamburg bereits für die Simulation von TTEthernet verwendet wird. Es ist unter anderem zu einem späteren Zeitpunkt geplant diese Technologien innerhalb des Simulationsframeworks miteinander zu verbinden.

Struktur der Arbeit

Die Arbeit gliedert sich wie folgt:
Kapitel 2 befasst sich mit den Grundlagen der Arbeit. Hier wird zum einen das Kommunikationssystem FlexRay erläutert und zum anderen werden Grundlagen der Simulation erklärt. Abschließend wird die Simulationsumgebung OMNeT++ beschrieben. In Kapitel 3 werden die Anforderungen, die die Simulation erfüllen soll aufgeführt und das Konzept wie diese umgesetzt werden erläutert. Kapitel 4 enthält die Umsetzung in OMNeT++. Hier werden die einzelnen Komponenten der Simulation näher gebracht und die Konfiguration eines Netzwerks anhand eines Beispiels beschrieben. Kapitel 5 befasst sich mit den Ergebnissen der Arbeit. Letztendlich befasst sich Kapitel 6 mit einer Zusammenfassung der Ergebnisse und einem Ausblick wie die Simulation mit zukünftigen Arbeiten erweitert werden kann.

2 Grundlagen

Dieses Kapitel befasst sich mit dem FlexRay-Kommunikationssystem sowie den Grundlagen einer Simulation. Eine anschließende Beschreibung der Simulationsumgebung OMNeT lässt das Kapitel enden.

2.1 FlexRay

In diesem Abschnitt werden die für diese Arbeit notwendigen Funktionen des FlexRay-Kommuni-kationssystems beschrieben.

2.1.1 Grundlegendes und Entwicklung

Die Entwicklung von FlexRay begann im Jahr 2000 mit der Gründung des FlexRay-Konsortiums. Die Gründungsmitglieder (BMW, DaimlerChrysler, Motorola und Philips) haben sich zum Ziel gesetzt ein Kommunikationssystem zu entwickeln, das auf die speziellen Anforderungen im Automobil zugeschnitten ist. Im Laufe der Jahre traten immer mehr Unternehmen dem Konsortium bei, unter ihnen auch einige weitere namhafte Firmen aus dem Automobilbereich wie z. B. der Volkswagenkonzern, General Motors und Toyota. Es wurden am Anfang sowohl ökonomische als auch technische Ziele definiert (vgl. Rausch, 2008). Als technische Ziele wurden diese Punkte festgelegt:

- Busgeschwindigkeit von 10 MBit/s
- Redundante Kommunikationskanäle
- Bereitstellung einer globalen synchronisierten Zeitbasis
- Garantierte Nachrichtenübertragung bei geringer Zeitvarianz
- Funktion des Protokolls ohne Wissen über die Netzwerktopologie
- Unterstützung von kontinuierlicher und sporadischer Kommunikation

Die Arbeit wurde mit der Veröffentlichung der finalen FlexRay-Spezifikation Version 3.0.1 im Jahre 2010 beendet. Sie wurden an die ISO (International Organization for Standardization) übergeben, um als Standard für Automobile veröffentlicht zu werden.
Diese Spezifikationen sind auf der offiziellen Homepage des FlexRay-Konsortiums als Download in mehreren Versionen frei erhältlich (vgl. FlexRay Consortium).

2.1.2 Technische Eigenschaften

Die technischen Eigenschaften des FlexRay-Systems ergeben sich letztendlich aus den zuvor definierten Zielen. So kann jeder Knoten an zwei Kommunikationskanäle (Channel A und Channel B) angeschlossen werden. Jeder dieser Kanäle kann theoretisch eine Datenrate von bis zu 10 MBit/s erreichen. Bei der Übertragung der Daten über die Leitungen kann frei gewählt werden, ob dies redundant geschehen soll oder ob auf beiden Kanälen verschiedene Daten für eine höhere Datenrate verschickt werden sollen.
Jeder Teilnehmer verfügt über eine synchronisierte Zeitbasis (siehe Abschnitt 2.1.5). Dies und die vor dem Systemstart festgelegte Reihenfolge im Kommunikationszyklus (siehe Abschnitt 2.1.4) erlauben es FlexRay auch für sicherheitskritische Anwendungen, wie z. B. „X-By-Wire“ Systeme, eingesetzt zu werden.
Im OSI-Referenzmodell arbeitet das Protokoll hauptsächlich in den Schichten 1 und 2.

2.1.3 FlexRay Bustopologien

Ein FlexRay-Netzwerk besteht aus mehreren Kommunikationsteilnehmern (Knoten) und einem oder mehreren Kommunikationskanälen (Channels). Alle Komponenten können in zwei Gruppen eingeteilt werden: aktive (z. B. Knoten) und passive (z. B. Bus). Während Passive die Nachrichten lediglich weiterleiten, verteilen die Aktiven die Nachrichten aktiv an andere Teilnehmer. Es ist möglich die Komponenten in verschiedenen Topologien zu arrangieren. Mögliche Varianten sind hier:

- Bus-Netzwerk mit einem Kanal
- Bus-Netzwerk mit zwei Kanälen
- Stern-Netzwerk mit einem Kanal
- Stern-Netzwerk mit zwei Kanälen
- Stern-Netzwerk mit kaskadierenden Sternkopplern
- Kombinationen aus Bus- und Stern-Topologien

In den Abbildungen 2.1, 2.2 und 2.3 sieht man einige Beispiele, wie solche Topologien aussehen können.

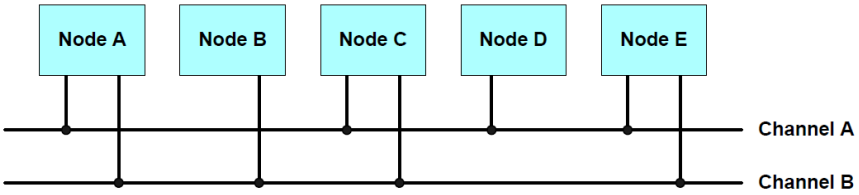

Abbildung 2.1: passiver Bus mit zwei Kanälen (Quelle: FlexRay Consortium, 2010b)

Abbildung 2.1 zeigt einen passiven Bus mit zwei Kanälen. Die Knoten A, C und E sind mit beiden Kanälen verbunden. B und D sind nur an Kanal A bzw. Kanal B angeschlossen.

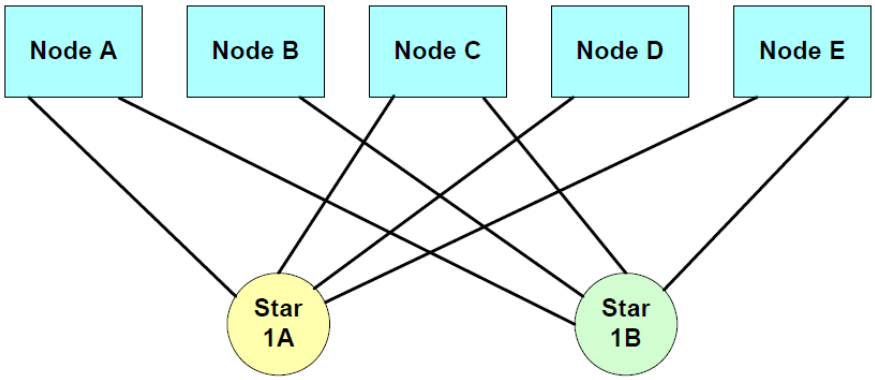

Abbildung 2.2: Sterntopologie mit zwei Kanälen (Quelle: FlexRay Consortium, 2010b)

Abbildung 2.2 zeigt eine Sterntopologie mit zwei Kanälen. Für Channel A und Channel B gibt es einen so genannten Sternkoppler, der die Weiterleitung der Nachrichten übernimmt. Wie schon bei dem passiven Bus ist es auch hier möglich die Knoten an beide oder nur an einen Koppler anzuschließen.
In der dritten Beispielkonfiguration 2.3 sieht man eine einkanalige Kombination aus Bus- und Sterntopologie. Die Knoten der Bustopologie sind an den Sternkoppler “2A“ angeschlossen. Dieser ist wiederum mit einem weiteren Sternkoppler verbunden und sie bilden somit einen kaskadierenden Stern.
Wie oben erwähnt, gibt es noch weitere Konfigurations- und Kombinationsmöglichkeiten der Topologien als die drei hier gezeigten Beispiele. Für ausführlichere Informationen und Beschreibungen kann in der Electrical Physical Layer Specification (vgl. FlexRay Consortium, 2010a) nachgeschlagen werden.

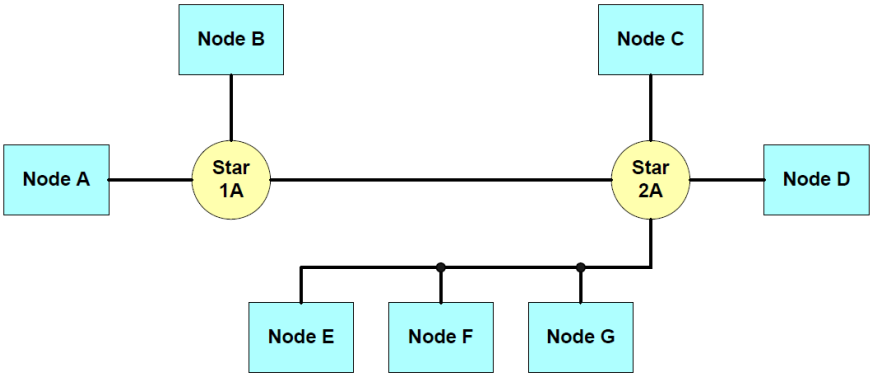

Abbildung 2.3: Kombination von Bus- und Sterntopologie auf einem Kanal(Quelle: FlexRay Consortium, 2010b)

2.1.4 Der Kommunikationszyklus

Die Kommunikation in einem FlexRay-System ist in Zyklen unterteilt, die sich immer wiederholen. Ein Zyklus kann aus bis zu vier unterschiedlichen Elementen bestehen: dem statischen Segment, dem dynamischen Segment, dem Symbol Window und der Network Idle Time (NIT). Das statische Segment und die NIT sind auf jeden Fall in einem Zyklus vorhanden. Das dynamische Segment und das Symbol Window sind optional. Das statische Segment und das dynamische Segment werden noch weiter in statische Slots bzw. Minislots unterteilt.
Die nächstkleinere Einheit in einem Zyklus ist der Macrotick. Bei dem Parameter für die Dauer des Macroticks handelt es sich um einen globalen Wert. Er ist also in jedem Knoten eines Netzwerks gleich. Die kleinste Zeiteinheit ist der Microtick. Dieser Wert wird mit Hilfe eines Taktgebers gewonnen. Die Dauer eines Microticks kann einen von drei Werten annehmen: 12,5 ns, 25 ns oder 50 ns. In unterschiedlichen Knoten können diesem Parameter unterschiedliche Werte zugeordnet werden. Die Anzahl der Microticks pro Macrotick muss auf die unterschiedlichen Werte abgestimmt sein. Dauert der Microtick in einem Knoten 12,5 ns und in einem anderen 25 ns, so muss die Anzahl der Microticks pro Macrotick im ersten Knoten doppelt so hoch sein wie im Zweiten. Der Macro- und der Microtick dienen für viele Parameter des FlexRay-Systems als Einheit.

Abbildung 2.4: Aufbau eines Kommunikationszyklus mit allen Elementen

Statisches Segment

Das statische Segment ist das erste Element in einem Kommunikationszyklus. Die Übertragung der Frames wird mit Hilfe eines Time Division Multiple Access (TDMA)-Verfahren realisiert. Das Segment ist in eine definierbare Anzahl an gleichgroßen, aufsteigend nummerierten Slots unterteilt (siehe Abbildung 2.5). Für die Kommunikation im statischen Segment wurden folgende Regeln festgelegt:

- Wenn ein Knoten einen *key slot* (Slot, in dem Synchronisationsnachrichten versendet werden) besitzt, dann muss ein Frame in diesem Slot an alle angeschlossenen Kanäle und in jedem Kommunikationszyklus übertragen werden.
- In Slots, die nicht als *key slot* definiert sind, kann entweder auf einem oder auf beiden Kanälen eine Nachricht übertragen werden.
- In einem Kommunikationszyklus darf pro Slot und Channel nur ein Knoten zur Zeit senden. In unterschiedlichen Zyklen ist es den Knoten gestattet auf dem gleichen Kanal und mit gleicher Frame-ID Frames zu verschicken.

Den Knoten werden nun vor dem Systemstart Frame-IDs zugeordnet, die den Slots entsprechen in denen sie senden sollen. Im Laufe der Zeit betritt der Knoten einen Slot nach dem anderen. Sobald die Frame-ID mit der Nummer des Slots übereinstimmt, ist der Punkt erreicht, zu dem eine Nachricht versendet werden kann. Allerdings kommt nach dem Beginn des statischen Slots noch ein Offsetwert zur Anwendung, damit die Nachrichten von den anderen Knoten immer im richtigen Slot empfangen werden können. Erst nach dieser global gültigen Dauer wird mit der Übertragung begonnen.
Wurde der aktuelle Slot als *key slot* definiert, so wird der zu versendende Frame als *sync frame* gekennzeichnet und somit als Synchronisationsnachricht verwendet.
Der Sendevorgang muss bis zum Ende des Slots abgeschlossen sein.

Dynamisches Segment

Das dynamische Segment arbeitet nach einem dynamischen Minislot-Verfahren. Hierbei wird das Segment, genau wie im statischen Segment, in mehrere gleichgroße Slots, den Minislots, unterteilt (siehe Abbildung 2.6). Diese sind allerdings kleiner als die im statischen Segment. Auch hier gibt es festgelegte Regeln, nach denen sich die Knoten richten müssen:

- Es dürfen keine Synchronisationsnachrichten versendet werden.

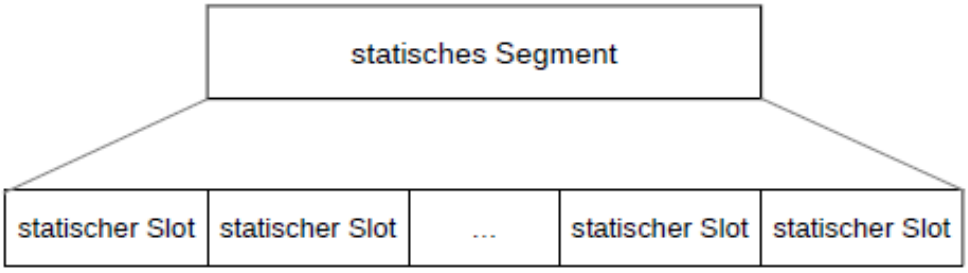

Abbildung 2.5: statisches Segment mit Unterteilung in statische Slots

- In einem dynamischen Slot ist es erlaubt sowohl auf beiden als auch auf einem Kanal den Frame zu übertragen. Auch wenn einem Knoten ein Slot zugeteilt wurde, muss die dynamische Nachricht nicht zwingend übermittelt werden. Es ist auch möglich, dass der gleiche Frame zu unterschiedlichen Zeitpunkten auf den beiden Kanälen ankommt.
- In einem Kommunikationszyklus darf pro Slot und Channel nur ein Knoten zur Zeit senden. In unterschiedlichen Zyklen ist es den Knoten gestattet auf dem gleichen Kanal und mit gleicher Frame-ID Frames zu verschicken.

Den Knoten werden hier IDs zugeordnet, mit deren Hilfe der Zeitpunkt ermittelt wird, zu dem sie senden dürfen. Es steht ihnen während des dynamischen Segments frei, ob sie zu dem Zeitpunkt senden oder nicht. Die Dauer der Übertragung ist nur durch das Ende des dynamischen Segments begrenzt.
In diesem Segment werden die Frames auf beiden Kanälen unabhängig voneinander verschickt. Es kann also vorkommen, dass auch wenn für beide Kanäle die gleiche Frame-ID für die gleiche Nachricht definiert wurde, dieser Frame auf einem Kanal früher verschickt wird als auf dem anderen. Abbildung 2.7 zeigt solch einen Fall. Der Frame mit der ID m+4 wird auf Kanal B früher verschickt, da auf Kanal A zuerst noch der Frame mit der ID m+2 übertragen wird.

Symbol Window

Das Symbol Window befindet sich im Zyklus direkt vor der NIT und kann dazu genutzt werden um Symbole zu übertragen. Diese dienen dazu Kollisionen während des Systemstarts zu vermeiden und außerdem als Wakeup-Pattern während des Systemstarts.

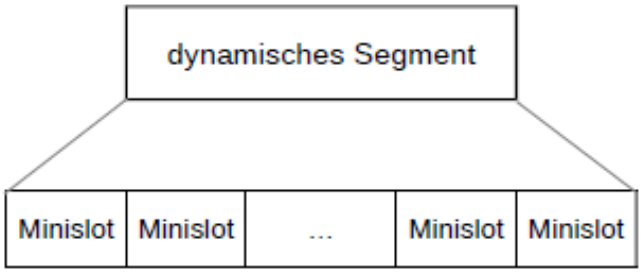

Abbildung 2.6: dynamisches Segment mit Unterteilung in Minislots

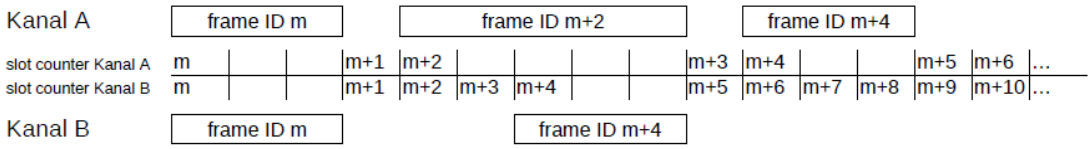

Abbildung 2.7: Übertragung auf dem dynamisches Segment

Network Idle Time

Die NIT befindet sich immer am Ende eines Kommunikationszyklus. Zu diesem Zeitpunkt findet keine Kommunikation auf dem Medium statt. In dieser Phase wird die Offsetkorrektur der Uhren vorgenommen und die Zeit kann dazu genutzt werden um Berechnungen sowie Datenverarbeitung durchzuführen.

Ablauf

Bevor ein FlexRay-System gestartet werden kann, muss für alle Knoten festgelegt werden, wer zu welchem Zeitpunkt senden darf und senden muss. Hierzu wird jedem Frame der gesendet werden soll eine eindeutige ID zugeordnet, die Frame-ID. Im statischen Segment wird in jedem Slot, egal ob dieser Slot einem Knoten zugeordnet ist oder nicht, diese Frame-ID erhöht. Mit dem Start des dynamischen Segments wird diese Nummerierung fortgesetzt. Hier wird diese ID jedes Mal erhöht, wenn in einem Minislot kein Knoten sendet. Sobald aber die Frame-ID mit der im Knoten konfigurierten ID übereinstimmt, fängt dieser an zu senden. Das Senden kann mehrere Minislots in Anspruch nehmen. Während dieser Zeit wird die Frame-ID nicht weitergezählt. Erst wenn der Sendevorgang abgeschlossen ist, wird die ID wieder inkrementiert. So kann es dazu kommen, dass ein Knoten mit einer hohen ID im dynamischen Segment in einem Zyklus nicht senden darf. Somit entspricht eine niedrige ID einer hohen Priorität und eine hohe ID einer niedrigen Priorität.

2.1.5 Uhrensynchronisation

Jeder Knoten in einem FlexRay-System hat eine eigene interne Uhr. Diese wird benötigt, damit jeder Knoten weiß, wann er beispielsweise Daten senden muss oder interne Berechnungen gestartet werden sollen, ohne auf einen externen Controller angewiesen zu sein. Da diese Uhren aber aufgrund von Abweichungen in der Fertigung und im Material oder aufgrund von äußeren Einflüssen nicht zu 100 % synchron laufen und sich die interne Zeit der einzelnen Knoten somit unterscheiden kann, müssen sich die Knoten miteinander synchronisieren. Hierzu gibt es bei der FlexRay-Technologie eine Kombination aus zwei Synchronisationsverfahren: die Offset-Correction und die Rate-Correction.
Die Offset-Correction dient dazu, den Abstand der Uhren zueinander wieder anzugleichen. Bei diesem Verfahren wird die Dauer der NIT verändert. Je nachdem, ob nach der Berechnung des Offset-Korrekturwerts ein positiver oder negativer Wert ausgerechnet wurde, werden am Ende der NIT Microticks hinzugefügt oder abgezogen. Nach diesem Verfahren wird der Abstand der Uhren zwar verringert aber an der Ursache, dass die Uhren voneinander abweichen, ändert sich nichts. Mit diesem Problem befasst sich die Rate-Correction. Bei dieser Korrektur wird die Länge des gesamten Zyklus angepasst. Dies geschieht, indem die Anzahl der Microticks pro Zyklus verändert wird. Nach der Anwendung dieser Methode laufen die Uhren nicht mehr so stark auseinander.

Zeitmessung

In einem FlexRay Netzwerk können bis zu 15 Knoten als Synchronisationsknoten definiert werden. Diesen Knoten werden *key slots* zugeteilt, in denen sie in jedem Zyklus Synchronisationsnachrichten verschicken müssen.
Während des statischen Segments ermittelt und speichert jeder Teilnehmer für sich die Abweichung von den Synchronisationsknoten. Zu einem Messergebnis gehören neben dem gemessenen Wert noch die Gültigkeit der Nachricht. Gesichert wird das Ergebnis in einem dreidimensionalen Array. Ein Beispiel kann man in Abbildung 2.8 sehen. Eine Dimension bildet die Zeilennummer, die es für jeden Synchronisationsknoten gibt. Eine weitere Dimension ist für den Kommunikationskanal (channel A oder channel B) vorgesehen, auf dem der Frame empfangen wurde. Als letztes wird noch unterschieden, ob der Frame in einem geraden oder ungeraden Zyklus empfangen wurde.

Mit Hilfe der Ankunftszeit des Frames und dem Action Point des statischen Slots wird für jede Synchronisationsnachricht der zu speichernde Wert berechnet. Dieser wird aus der

	even				odd			
	Kanal A		Kanal B		Kanal A		Kanal B	
	value	valid	value	valid	value	valid	value	valid
1	17	true	13	true	4	true	11	true
2	12	true	11	true	14	true	21	true
3	14	true	19	true	24	true	5	true
4	13	true	10	true	-11	true	-8	true
5	-7	true	-6	true	-6	true	27	true
6	-5	true	-1	true	9	true	-18	true
7	1	true	-5	true	23	true	12	true

Abbildung 2.8: dreidimensionales Array für Messwerte

Differenz zwischen der erwarteten Ankunftszeit und der tatsächlichen Ankunftszeit ermittelt. Die Zeitmessung für einen Zyklus ist mit dem Ende des statischen Segments abgeschlossen.

Fault-Tolerant Midpoint (FTM) Algorithmus

Dieses Verfahren wird sowohl von der Offset-Correction als auch von der Rate-Correction zur Berechnung der Korrekturwerte genutzt. Das Ergebnis wird mit Hilfe von Werten aus sortierten Listen, die von den Synchronisationsverfahren bereitgestellt werden, ermittelt. Zuerst wird ein Parameter, k, definiert (siehe Tabelle 2.1). Als Grundlage zur Berechnung dient die Anzahl der Werte aus der Liste.

Anzahl Werte	k
1 - 2	0
3 - 7	1
> 7	2

Tabelle 2.1: Ermittlung des Parameters k

In der Liste werden anschließend die k-größten und k-kleinsten Werte gestrichen. Nun werden der größte und der kleinste noch verbliebene Wert addiert und durch zwei geteilt. Dieses Ergebnis spiegelt den Korrekturwert des FTM-Algorithmus wider. Zu beachten ist, dass diese Zahl als Integer behandelt wird. Nachkommastellen spielen also keine Rolle. Als Beispiel wird dieses Verfahren einmal auf die Werte aus Abbildung 2.8 vom geraden Zyklus auf Kanal A angewendet:

$$
\begin{array}{ll}
\not{-7} & \\
-5 & \\
1 & \\
12 & \rightarrow (-5+14)/2 = 4 \\
13 & \\
14 & \\
\not{17} &
\end{array}
$$

Da es sich insgesamt um sieben Werte handelt, enthält der Parameter k den Wert 1. Das heißt der größte und der kleinste Wert werden gelöscht (in diesem Fall -7 und 17). Anschließend werden die Werte -5 und 14 addiert und durch 2 geteilt.

Offset-Correction

Der Wert für die Offset-Correction wird in jedem Zyklus berechnet. Angewendet wird dieser Wert allerdings nur am Ende eines ungeraden Zyklus. Für die Berechnung des Korrekturwertes ist in der Spezifikation kein definitiver Zeitpunkt festgelegt. Sie muss aber spätestens nach einem definierbaren Wert nach dem statischen Segment oder nach maximal einem Macrotick in der NIT abgeschlossen sein. Je nachdem welcher Fall früher eintritt. Um den Wert zu berechnen, werden die zuvor gemessenen Werte ermittelt und auf ihre Gültigkeit geprüft. Es werden nur Werte aus dem aktuellen Zyklus benutzt. Wenn es zu einer Frame-ID zwei Werte gibt (Channel A und Channel B), dann wird der Kleinere der beiden ausgewählt. Anschließend wird der Fault-Tolerant Midpoint Algorithmus ausgeführt. Die durch diesen Algorithmus berechnete Anzahl an Microticks wird noch mit festgelegten Grenzwerten verglichen, gegebenenfalls angepasst und kann am Ende der NIT angewendet werden. Während der Berechnung können zwei Fehlerzustände eintreten. Falls die Anzahl der Synchronisationsnachrichten zu gering ist oder der im FTM-Algorithmus berechnete Wert die Grenzwerte überschreitet, werden diese Fehlerzustände gesetzt.

Rate-Correction

Der Wert für die Rate-Correction wird nur in jedem ungeraden Zyklus berechnet. Angewendet wird dieser allerdings in jedem Zyklus. Ein Korrekturwert gilt also immer für einen geraden und einen ungeraden Zyklus. Genau wie die Offset-Correction muss die Rate-Correction zu einem bestimmten Zeitpunkt abgeschlossen sein. Auch hier gibt es einen definierbaren Wert nach dem die Berechnung fertig sein muss. Zusätzlich muss spätestens nach zwei Macroticks in der NIT ein Wert vorliegen. Für die Berechnung werden die Werte aus den

beiden vorangegangenen Zyklen verwendet. Es werden Paare ausgewählt, von denen die Differenz gebildet wird. Die Werte für ein Paar stammen von Synchronisationsnachrichten, die auf dem selben Kanal und in dem gleichen Slot empfangen wurden. Falls auf beiden Kanälen ein *sync frame* empfangen wurde, wird aus beiden Werten der Durchschnitt gebildet. Auch bei diesem Verfahren kommt der Fault-Tolerant Midpoint Algorithmus zum Einsatz. Bei der Rate-Correction wird dieser Wert allerdings zu dem vorherigen Korrekturwert addiert. Zusätzlich wird das Ergebnis noch mit einem Dämpfungswert verglichen und anschließend mit diesem weiter angepasst. Zum Schluss wird, wie schon bei der Offsetkorrektur, überprüft, ob sich der Wert innerhalb eines Grenzwertes befindet und gegebenenfalls angepasst. Sollte dies der Fall sein wird wieder ein Fehlerzustand gesetzt.

2.2 Simulation

Eine Simulation bildet ein reales System virtuell ab. Mit ihrer Hilfe kann man das Verhalten der Realität nachbilden und so beispielsweise Prognosen über die Zukunft treffen oder die Entwicklung eines Produkts erleichtern.
Man trifft heutzutage in vielen Bereichen auf Simulationen. In etwa als Wettersimulationen, Flug- oder Fahrzeugsimulatoren oder in der Forschung. Eine Simulation bietet viele Vorteile. Es kann ein schneller Funktionsnachweis erbracht werden, sie ist meist schneller als ein Experiment und kann somit Zeit und Kosten sparen, bietet eine gewisse Anschaulichkeit und ist ungefährlich (vgl. Scherf, 2010).

2.2.1 Ereignisorientierte Simulation

Es gibt eine gewisse Anzahl verschiedener Simulationsarten. Je nachdem was man abbilden möchte, bieten sich unterschiedliche Varianten an. Möchte man beispielsweise nur einen bestimmten Zeitpunkt betrachten, würde sich etwa die statische Simulation anbieten. Bei Prozessen oder Abläufen entscheidet man sich eher für die dynamische Simulation. Bei der in dieser Arbeit verwendeten Simulationsumgebung OMNeT++ wird eine dynamische Art verwendet: die diskrete ereignisorientierte Simulation.
Eine solches Simulationssystem besteht hauptsächlich aus dem derzeitigen Zustand des Systems, der aktuellen Simulationszeit und einer Menge an Ereignissen (Events).
Ein Ereignis stellt den Zeitpunkt dar, zu dem das System von einem Zustand in den nächsten wechselt. Währenddessen können neue Events generiert werden. Anschließend wird die Simulationszeit auf den Zeitpunkt des nächsten Events gestellt, sodass dieses abgearbeitet werden

kann. Es handelt sich hierbei also um eine Schleife (Eventloop), die sich wiederholt, bis das Simulationsende erreicht ist. In Abbildung 2.9 ist eine solche Schleife dargestellt.

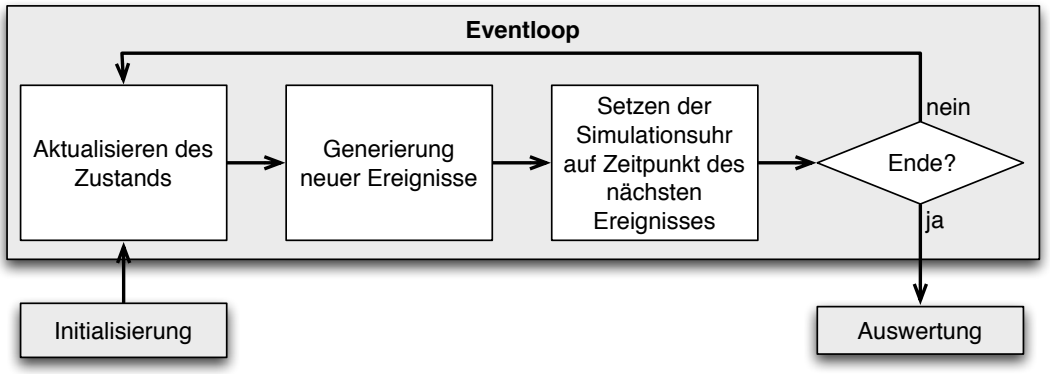

Abbildung 2.9: Ablauf einer diskreten ereignisorientierten Simulation (Quelle: Steinbach, 2011)

2.3 OMNeT++

Bei OMNeT++ (kurz für Objective Modular Network Testbed in C++) handelt es sich um eine diskrete eventbasierte Simulationsumgebung (vgl. OMNeT++ Community, a). Es wird hauptsächlich zur Simulation von Kommunikationsnetzwerken eingesetzt. Die Entwicklungsumgebung basiert auf Eclipse (vgl. Eclipse Foundation) und kann unter Linux, Windows und Mac OS X betrieben werden. Für den nicht kommerziellen Einsatz ist die Nutzung der Software kostenlos. Modelle und Komponenten werden mit Hilfe der OMNeT++-spezifischen Sprache NED (Network Description) erstellt. Die Programmierung der Logik der einzelnen Elemente geschieht mit Hilfe von C++.

2.3.1 Aufbau einer Simulation

Eine Simulation in OMNeT++ besteht aus mehreren Modulen, die miteinander kommunizieren. Die Kommunikation erfolgt über Gates. In Abbildung 2.10 sieht man zwei Module, die genau so miteinander verbunden sind. Über diese Gates können nun Nachrichten ausgetauscht werden. Der Aufbau eines solchen Netzwerks und der Aufbau der einzelnen Module wird mit der NED-Sprache beschrieben. Damit wird beispielsweise festgelegt, welche Komponenten sich in dem Netzwerk befinden (z. B. Switches, Router, Hosts) und welche Gates miteinander verbunden sind.

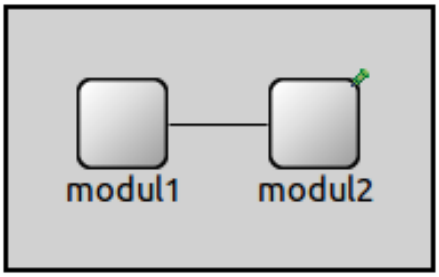

Abbildung 2.10: In diesem Netzwerk sind *modul1* und *modul2* über Gates und einen Channel miteinander verbunden

Das genaue Verhalten der Elemente wird in C++ beschrieben. So kann auf auftretende Events reagiert werden. Wenn “modul1“ eine Nachricht an “modul2“ sendet und dieses die Nachricht empfängt, dann wird bei “modul2“ die Methode “handleMessage()“ aufgerufen. Hier kann nun die weitere Vorgehensweise definiert werden.
Für jedes Projekt existiert noch eine Konfigurationsdatei *omnetpp.ini*. In dieser Datei können Parameter und andere Eigenschaften der Simulation verändert werden. Auf diese Weise kann man dann beispielsweise das gleiche Netzwerk nur mit anderen Parametern testen ohne die Module ändern zu müssen.
Wurde ein System konfiguriert und die Logik der Komponenten programmiert, kann die Simulation gestartet werden. Nachdem die einzelnen Module initialisiert sind, kann mit dem Ablauf der Eventloop begonnen werden (siehe Abschnitt 2.2.1).

2.3.2 NED Sprache

Die NED Sprache beschreibt Module und verbindet diese um beispielsweise Netzwerke zu definieren. Die Sprache bietet einige Möglichkeiten um auch große Projekte bewältigen zu können:

- Hierarchie
- Interfaces
- Vererbung
- Pakete
- Annotationen

Im Folgenden sieht man einen Ausschnitt des Codes, der das Beispiel aus Abbildung 2.10 beschreibt:

```
network SimpleNetwork
{
    types:
        channel C extends ned.DelayChannel
        {
            delay = 100ms;
        }
    submodules:
        modul1: Node;
        modul2: Node;
    connections:
        modul1.gate <--> C <--> modul2.gate;
}
```

Unter dem Punkt *submodules* wurden die beiden Module *modul1* und *modul2* definiert. Im Abschnitt *connections* werden diese beiden dann mit dem am Anfang unter *types* erstellten Channel C verbunden. Es könnten hier noch zusätzlich mit den Befehlen *gates:* und *parameters:* Gates und Parameter definiert werden.
Gibt es in einem Modul Parameter, so kann diesen hier ein Wert zugeordnet werden. Soll dieser nicht fix sein, so kann ihm per *default* ein Standardwert zugeordnet (siehe Codeausschnitt) werden und in der *omnetpp.ini* Datei nach Belieben angepasst werden.

```
parameters:
    bool sendMsgOnInit = default(false);
    int limit = default(2);
```

2.3.3 Programmierung der Logik

Wie schon erwähnt, wird die Logik der einzelnen Elemente in C++ programmiert. OMNeT bietet hier diverse Funktionen. Besonders hervorzuheben wären hier:

- *handleMessage*
- *send*
- *scheduleAt*
- *simTime*

Empfängt ein Modul eine Nachricht, wird die Funktion *handleMessage* aufgerufen und der vom Benutzer definierte Code wird ausgeführt. Über die *send*-Funktion kann ein Modul eine Nachricht an ein anderes Modul schicken. Hier muss die zu sendende Nachricht und das gewünschte Gate, von dem aus gesendet werden soll, angegeben werden. Mit *scheduleAt* kann das Modul eine Nachricht an sich selbst schicken. Dieser Funktion muss der gewünschte Zeitpunkt, wann die Nachricht empfangen werden soll, sowie die zu sendende Nachricht übergeben werden. Dieser Zeitpunkt wird meist durch *simTime* + *delta* definiert. *simTime* liefert die aktuelle Simulationszeit, wobei *delta* eine variable Zeit darstellt.

2.3.4 Ablauf einer Simulation

In diesem Abschnitt wird der Ablauf einer Simulation anhand des TicToc Tutorials für OMNeT++ (vgl. OMNeT++ Community, b) erläutert. Sobald das Netzwerk konfiguriert wurde und die Logik für die Module programmiert ist, kann die Simulation gestartet werden.
Im nächsten Schritt wird das Netzwerk mit seinen Modulen, Gates und Verbindungen erstellt und initialisiert. Zusätzlich öffnen sich zwei Fenster. Das erste Fenster (Abbildung 2.11) zeigt das Netzwerk mit den Knoten und Verbindungen. Das andere Fenster (Abbildung 2.12) listet alle Nachrichten und Events chronologisch geordnet auf.
Die Simulation an sich wird bis zu diesem Zeitpunkt noch nicht gestartet. Dies geschieht mit Hilfe einiger Schaltflächen in den Fenstern. Es besteht die Möglichkeit die Simulation in unterschiedlichen Geschwindigkeiten ablaufen zu lassen. Die erste Stufe (*run*) ist die langsamste, stellt aber sowohl die Nachrichten die verschickt werden (roter Punkt in Abbildung 2.11) als auch die Events und Nachrichten in der Auflistung dar. Die nächste Schaltfläche (*fast*) ist schon deutlich schneller, stellt den Nachrichtenverlauf im Fenster mit dem Netzwerk aber nicht mehr dar. Die letzte und schnellste Stufe (*express*) zeigt weder die Nachrichten noch die Events an.
Um die Simulation zu unterbrechen kann entweder die Schaltfläche *stop* benutzt werden oder von vornherein ein Zeitpunkt festgelegt werden (siehe Abbildung 2.13). Dieser Zeitpunkt kann entweder ein maximaler Wert der Simulationszeit (*Simulation time to stop at*), eine maximale Anzahl an Events (*Event number to stop at*) oder eine bestimmte Nachricht (*Message to stop at*) sein. Sobald keine weiteren Events anstehen, wird die Simulation automatisch gestoppt.

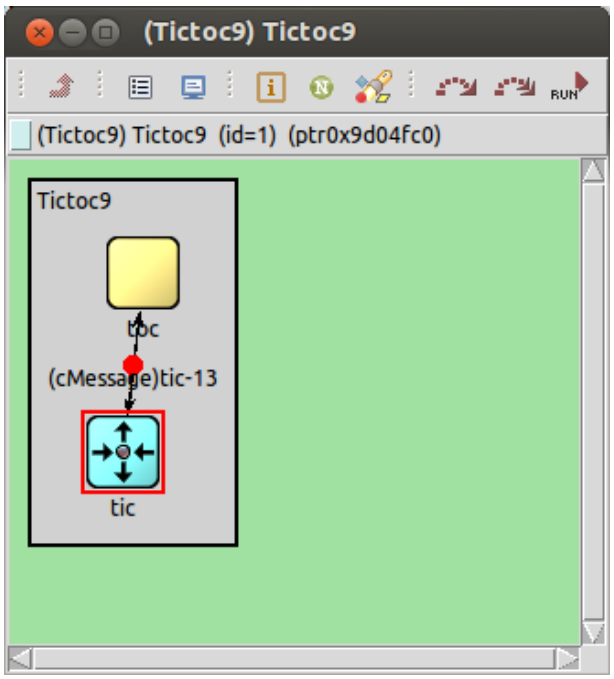

Abbildung 2.11: Darstellung des TicToc-Netzwerks während der Simulation

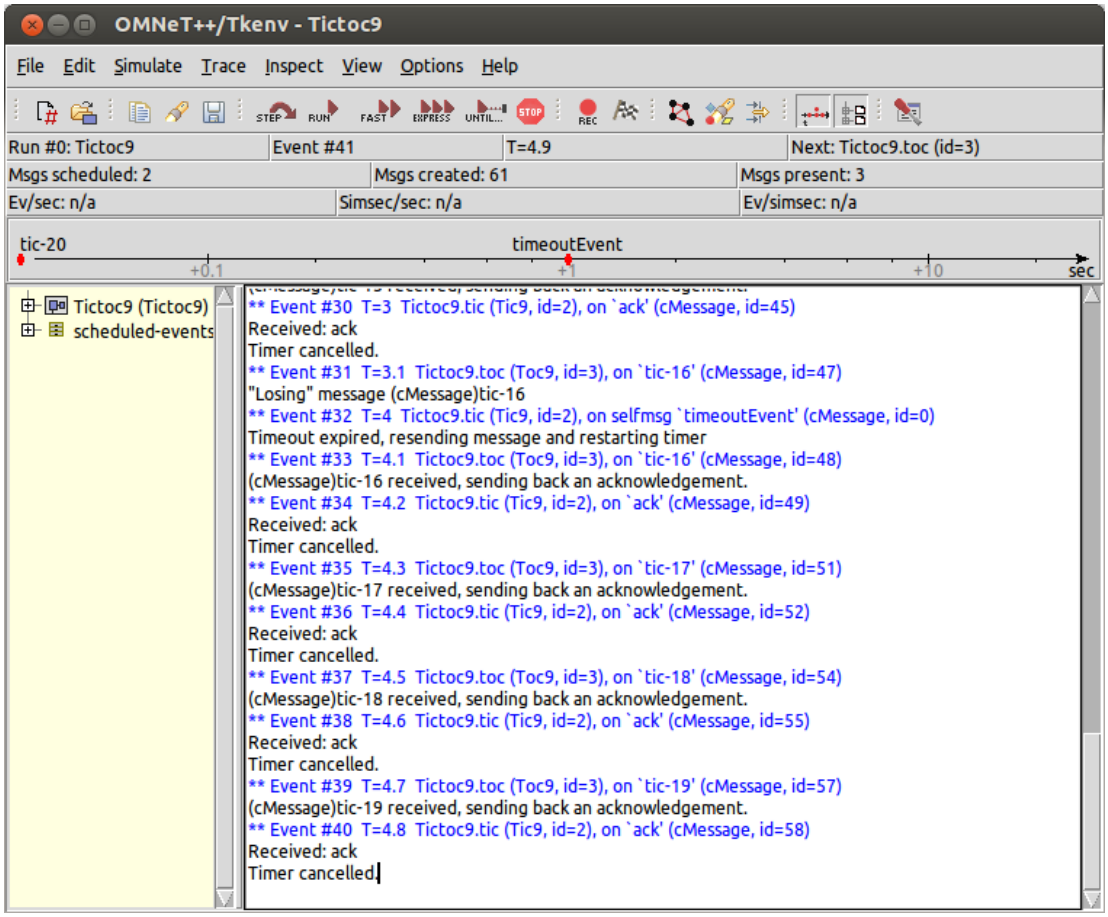

Abbildung 2.12: Auflistung der Events und Nachrichten im TicToc-Netzwerk

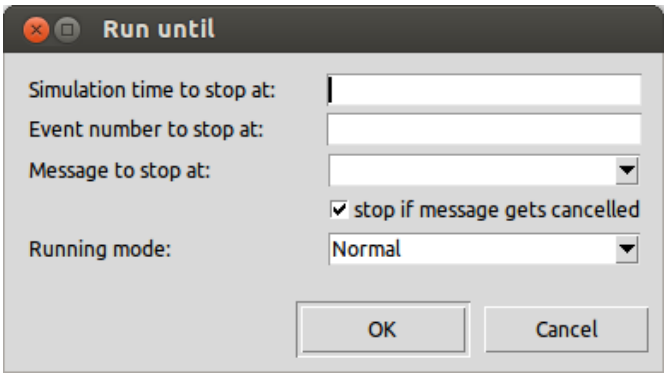

Abbildung 2.13: Auswahlfenster zum Starten der Simulation bis zu einem gewissen Zeitpunkt

3 Anforderungen und Konzept

In diesem Kapitel wird zuerst der Grad der Simulation sowie der geplante Umfang erläutert. Anschließend wird auf den geplanten Aufbau der Simulation und die Konfigurationsmöglichkeiten eingegangen.

3.1 Anforderungen an die Simulation

In der Simulation soll der grundlegende Aufbau eines FlexRay-Netzwerks nachgebildet werden. Alles was sich auf dem physical layer abspielt, soll nicht genauer betrachtet werden. Im OSI-Schichtenmodell berücksichtigt die Simulation dementsprechend den zweiten Layer.
Der Aufbau des Netzwerks sowie die Parameter der Knoten sollen frei konfigurierbar sein. Das heißt es kann z. B. frei bestimmt werden wie viele Knoten an dem Bus angeschlossen sind, auf welchen Kanälen und zu welchen Zeitpunkten diese senden. Die Konfiguration der Parameter soll möglichst analog zur FlexRay vorgenommen werden. Namen und Wertebereiche können der Spezifikation entnommen werden.
In der Simulation soll es möglich sein, gemäß dem Kommunikationszyklus, Frames im statischen und dynamischen Segment zum richtigen Zeitpunkt zu versenden und zu empfangen. Beim dynamischen Segment ist hierbei zu beachten, dass, wie in der Spezifikation definiert und im Abschnitt 2.1.4 beschrieben, auf beiden Kanäle unabhängig voneinander die Frames verschickt werden können.
Für jeden Knoten ist vorgesehen, dass er seine eigene interne Uhr besitzt. Wie auch in der Realität sollen diese Uhren voneinander abweichen, was wiederum bedeutet, dass sich die Knoten zueinander synchronisieren müssen. Hierfür wird die FlexRay-spezifische Synchronisation mittels Offset- und Rate-Correction festgelegt.
In der Simulation sollen alle vier Segmente konfiguriert werden können. Im statischen und dynamischen Segment können Nachrichten verschickt und empfangen werden. Am Anfang der NIT werden die Berechnungen für die Synchronisation gestartet. Das Symbol Window hat lediglich Auswirkungen auf die Dauer des Zyklus und keine weiteren Funktionen.
FlexRay bietet noch einige andere Funktionen, die in der Simulation zumindest vorerst nicht implementiert werden sollen. So wird die komplette Startphase nicht berücksichtigt. Es wird

angenommen, dass zum Beginn der Simulation alle Knoten bereits laufen und sich auch keine weiteren anmelden. Auch ist noch nicht geplant die versendeten Frames mit Daten zu versehen. Es ist zunächst nur wichtig, dass die Knoten ihre Nachrichten zum richtigen Zeitpunkt verschicken und keine zusätzliche Simulationszeit benötigt wird um die Daten zu generieren und zu verarbeiten. Im dynamischen Frame kann die Anzahl der benötigten Slots für einen Frame zufällig bestimmt werden, um hier eine Übertragungsdauer über mehrere Minislots zu simulieren.
Um ein Zusammenspiel zwischen der TTE-Simulation und der FlexRay-Simulation zu erleichtern, sollen Funktionen wie die Uhrenabweichung oder das Management der Events ähnlich gehandhabt werden.

3.2 Architekturkonzept

Der grundlegende Aufbau der Knoten lehnt sich an die TTE-Implementierung an. Die Einarbeitung in die neue FlexRay-Umgebung kann so vereinfacht werden, dass die einzelnen Module sich ähneln und teilweise gleiche Funktionen bieten. Dieses erleichtert zusätzlich bei dem späteren Einsatz von Gateways eine Kombination der Simulationsmodelle. Da sowohl FlexRay als auch TTEthernet ihre Echtzeitfähigkeit mit Hilfe eines TDMA-Verfahrens erreichen, sind sich diese Modelle teilweise ähnlich und es bietet sich ein ähnlicher Aufbau der Simulation an.
Jeder FlexRay Knoten wird als Modul konfiguriert. Dieses Modul wird dann weitere Module (Submodule) beinhalten, die für bestimmte Funktionen, wie z. B. die Verwaltung der Events oder die Synchronisation, zuständig sind (siehe Abbildung 3.1). Verbunden werden die Knoten mit einem weiteren Modul, welches die jeweilige Topologie darstellt. Das Topologiemodul übernimmt hauptsächlich die Aufgabe, die eingehenden Nachrichten an die anderen Busteilnehmer weiterzuleiten. Hier werden Laufzeiten und auch speziellen Eigenschaften für die Sternkoppler implementiert.
Die FlexRay Anwendung wird vom Scheduler über das Erreichen eines für den Knoten relevanten Slots informiert und leitet daraufhin das Versenden eines Frames ein.
Der Scheduler übernimmt die Verwaltung der für den Knoten wichtigen Zeitpunkte. Dies wären das Erreichen eines statischen bzw. dynamischen Slots, der NIT und des Starts eines neuen Zyklus. Sobald ein Event auftritt, können andere Module benachrichtigt werden um hier eine Bearbeitung des Events einzuleiten.
Werden im statischen Segment Synchronisationsnachrichten empfangen, ermittelt der Knoten die Abweichungen der Ankunftszeit von der eigenen internen Zeit. Auch in der Simulation

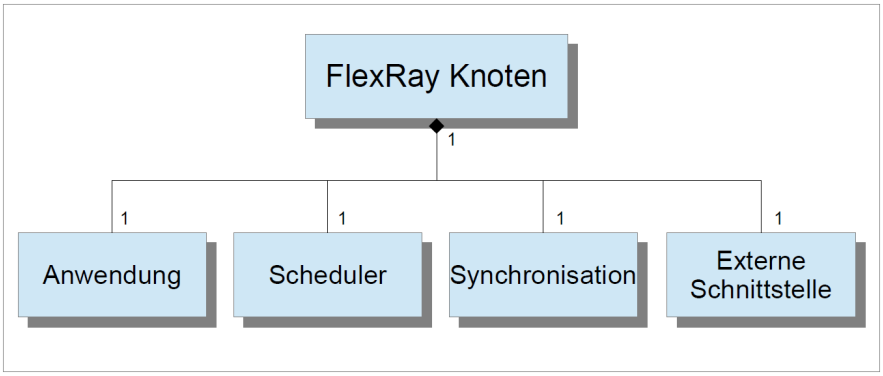

Abbildung 3.1: Konzept des FlexRay-Knotens

wird die FlexRay-spezifische Synchronisation implementiert (siehe Abschnitt 2.1.5). Das hierfür zuständige Modul ermittelt am Ende eines Zyklus die Korrekturwerte, nachdem es darüber benachrichtigt wurde.
Die externe Schnittstelle dient als Verbindung zwischen Bus und Knoten. Von hier werden Nachrichten an andere Knoten versendet und Nachrichten anderer Knoten empfangen. Die eingegangenen Frames können hier analysiert werden und anschließend an das entsprechende Modul weitergeleitet werden.

3.3 Konzept des Uhrenmodells

Die Dauer von einem Kommunikationszyklus oder der Zeitpunkt eines Events werden mit Hilfe von Macroticks definiert. Die angestrebte Dauer eines Macroticks ist fest vorgegeben. Wie lange dieser in der Simulation nun wirklich dauert, hängt mit der Zeit für einen Microtick zusammen, welcher die kleinste Einheit bei der FlexRay-Simulation darstellt.
Da von Oszillator zu Oszillator unterschiedliche Ungenauigkeiten bestehen und diese auch in der Simulation berücksichtigt werden sollen, muss ein Konzept gewählt werden um diese Abweichung zu realisieren. Für die FlexRay-Simulation wird im Prinzip das gleiche Modell umgesetzt wie bei der TTE-Simulation (vgl. Steinbach, 2011). Aufgrund der Ähnlichkeit der beiden Technologien (TDMA-Verfahren) bietet sich ein gleicher Ansatz an. Außerdem ist es für eine spätere Verbindung der Simulationen von Vorteil.
Aus Gründen der Performance kann nicht jeder Tick simuliert werden. Es müsste sonst jeder einzelne Microtick als Event registriert und bearbeitet werden. Das hätte signifikante Auswirkungen auf die Geschwindigkeit der Simulation. Deshalb wird am Anfang eines Zyklus ein Wert für die Abweichung (Drift) generiert und auf den kompletten Zyklus angewendet.

Abbildung 3.2 zeigt dieses Verhalten in vereinfachter Form.

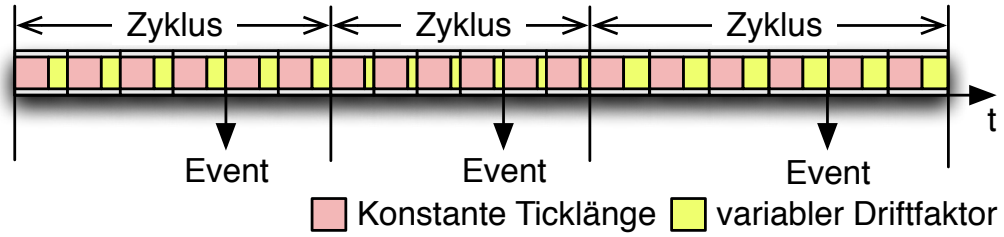

Abbildung 3.2: Zyklen mit konstanten Ticks und variablen Driftfaktoren (Quelle: Steinbach, 2011)

Mit diesem Ansatz lässt sich die Anzahl der Events auf wenige relevante reduzieren. Dies wären die Zeitpunkte, zu denen Frames verschickt werden oder andere wichtige Ereignisse wie das Erreichen der NIT oder der Start eines neuen Zyklus.

3.4 Konfigurationsmöglichkeiten

In der Simulation ist es möglich ein Netzwerk mit verschiedenen Parametern zu testen. Hierzu können die Werte dieser Parameter mit Hilfe der ini-Dateien konfiguriert werden. Bei den in diesem Abschnitt aufgeführten Parametern handelt es sich um einen kleinen Teil der für Layer 2 relevanten Daten. Einige Parameter aus der Spezifikation werden nicht berücksichtigt, da sie für Funktionen benötigt werden, die in der Simulation nicht implementiert sind (z. B. für die Startphase). Andere Parameter wie z. B. die gesamte Dauer eines Zyklus kann man aus anderen Werten errechnen und würden somit einen höheren Konfigurationsaufwand bedeuten.
Tabelle 3.1 zeigt Parameter, die für das gesamte Netzwerk gültig sind. Hier geht es hauptsächlich um die Konfiguration des Kommunikationszyklus. Die Namen und die Wertebereiche der Parameter stammen aus der FlexRay-Spezifikation.
Auch Tabelle 3.2 zeigt einige Parameter aus der Spezifikation. Hierbei handelt es sich aber um knotenspezifische Werte, die bei jedem Knoten anders konfiguriert werden können.
In Tabelle 3.3 werden Parameter aufgelistet, die nicht als Parameter in der Spezifikation aufgelistet werden. Es handelt sich hier um Werte, die speziell für die Simulation benötigt werden. Mit den Parametern *maxDrift* und *maxDriftChange*, die auch in der TTE-Simulation vorkommen, lässt sich einstellen, wie stark die Knoten von der eigentlichen Zeit abweichen können. Es gibt in der Spezifikation zwar auch derartige Werte aber diese spiegeln die Maximalwerte

für ein Netzwerk wider. Um in der Simulation aber Knoten mit unterschiedlichen Eigenschaften (z. B. sehr genaue Knoten und sehr ungenaue Knoten) zu implementieren, werden diese Parameter benötigt.

Bezeichnung	Beschreibung	Wertebereich
gNumberOfStaticSlots	Anzahl der statischen Slots in einem statischen Segment	2 — 1023
gNumberOfMinislots	Anzahl der Minislots in einem dynamischen Segment	0 — 7988
gCycleCountMax	Anzahl der unterschiedlichen Zyklen in einem Netzwerk	7 — 63
gdStaticSlot	Dauer eines dynamischen Slots in Macroticks (MT)	3 — 664 MT
gdMinislot	Dauer eines Minislots in Macroticks	2 — 63 MT
gdSymbolWindow	Dauer des Symbol Windows in Macroticks	0 — 162 MT
gdNIT	Dauer der Network Idle Time in Macroticks	2 — 15978 MT
gdMacrotick	Dauer eines Macroticks	1 — 6 µs

Tabelle 3.1: Globale Parameter nach der FlexRay-Spezifikation

Bezeichnung	Beschreibung	Wertebereich
pdMicrotick	Dauer eines Microticks (µT)	12.5, 25, 50 ns
pOffsetCorrectionOut	Maximaler Wert der Offset-Correction	15 — 16082 µT
pRateCorrectionOut	Maximaler Wert der Rate-Correction	15 — 16082 µT
pClusterDriftDamping	Ein Wert der für die Rate-Correction konfiguriert werden kann.	0 — 10 µT
pKeySlotID	ID des Slots, in dem die Synchronisationsnachricht übermittelt werden soll. Beträgt der Wert Null, wird keine solche Nachricht versandt.	0 — 1023

Tabelle 3.2: Knotenspezifische Parameter nach der FlexRay-Spezifikation

3.5 Fehlerszenarien

Vor allem bei großen Netzwerken kann es leicht zu Fehlern in der Konfiguration kommen. Um solche Fehler zu erkennen und dem Nutzer mitzuteilen, wurden einige Fälle definiert, die

Bezeichnung	Beschreibung	Wertebereich
maxDrift	Maximale Abweichung des Microticks	0 — 1500 ppm
maxDriftChange	Maximale Veränderung des Drifts pro Zyklus	*-maxDrift* — *maxDrift*
staticSlotsChA/B	In diesem Array für jeden Knoten können für jeden Knoten die Slotnummern für den statischen Slot angegeben werden, in denen dieser senden soll.	1 — *gNumberOfStaticSlots*
dynamicSlotsChA/B	In diesem Array können für jeden Knoten die Slotnummern für den dynamischen Slot angegeben werden, in denen dieser senden soll.	1 — *gNumberOfMinislots*

Tabelle 3.3: Simulationsspezifische Parameter

durch die Simulation aufgedeckt werden.

Gleichzeitiges senden: Dieses Problem tritt auf, wenn mindestens zwei Knoten im gleichen Slot auf dem selben Kanal einen Frame verschicken. Ist dies der Fall, wird die Simulation mit einer Fehlermeldung abgebrochen. Hier liegt ein Fehler in der Konfiguration vor und kann durch Anpassung der konfigurierten Slots behoben werden.

Zu viele Synchronisationsknoten: In einem FlexRay-Netzwerk darf es maximal 15 Synchronisationsknoten geben. Werden zu viele Synchronisationsnachrichten in einem Zyklus empfangen, wird ein Fehler ausgegeben, der die Simulation unterbricht und durch eine Anpassung der ini-Dateien behoben werden muss.

Frames in falschem Slot empfangen: Beim Empfang einer Nachricht wird die Frame-ID mit dem aktuellen Slot des Knotens verglichen. Stimmen diese Werte nicht überein, wurde die Nachricht im falschen Slot empfangen. Tritt dieser Fall auf, wird die Simulation zwar fortgesetzt aber jedes Mal eine Fehlermeldung ausgegeben. Dieses Problem kann mehrere Ursachen haben. Es kann sein, dass die maximalen Korrekturwerte der Synchronisationsverfahren zu niedrig konfiguriert wurden, die Drift-Werte der lokalen Uhren zu hoch eingestellt sind oder die Dauer der Slots nicht ausreichend groß ist.

4 Umsetzung

In diesem Kapitel werden die einzelnen Module und Funktionen der Simulation beschrieben. Außerdem wird der Ablauf einer Simulation anhand eines Beispiels erklärt.
Abbildung 4.1 zeigt den Aufbau einer FlexRay-Simulation. Ein Netzwerk besteht aus mehreren Knoten und einem Bus. Zwischen den Knoten und dem Bus bestehen Verbindungen, über die Nachrichten verschickt werden können. Die Knoten versenden Nachrichten an den Bus und synchronisieren sich zueinander. Der Bus ist dafür zuständig, die eingehenden Nachrichten an alle anderen Teilnehmer weiterzuleiten. Den Botschaften werden von dem sendenden Knoten Informationen mitgegeben, damit die Empfänger die eingehenden Nachrichten verarbeiten können.

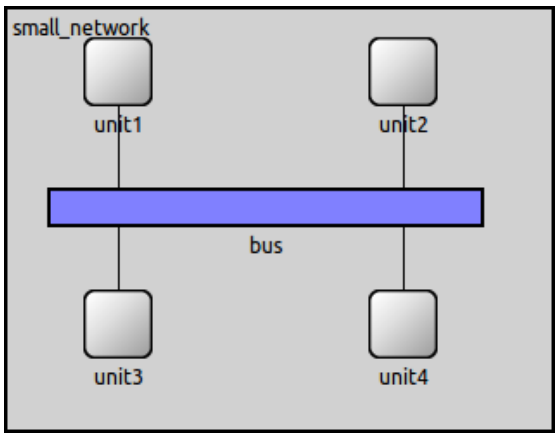

Abbildung 4.1: Beispielhafter Aufbau eines FlexRay-Netzwerks in OMNeT++

4.1 Module

Während der Entwicklung der FlexRay Simulation sind mehrere Module und eine Nachricht entstanden. Ein Knoten wird von dem Modul FRNode und vier weitere Submodulen (FRApp, FRScheduler, FRSync und FRPort) beschrieben. Für den Bus existiert das Modul FRBus und das

dazugehörige Submodul FRTopologyPort. Als Nachricht mit FlexRay-spezifischen Informationen dient die Nachrichtenklasse FRFrame.
Die Aufgaben und Funktionen dieser Elemente wird im Folgenden beschrieben.

4.1.1 FRNode

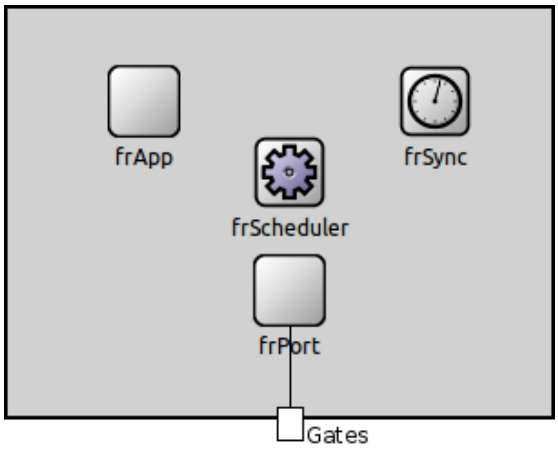

Abbildung 4.2: Layout des FRNode Moduls

Dieses Modul spiegelt den gesamten Knoten wider. Die vier Submodule sind gemäß ihrer funktionalen Blöcke unterteilt (siehe Abbildung 4.2). Außerdem lassen sich die Module bei einer solchen Unterteilung leichter austauschen oder wiederverwenden.
Das Modul FRNode fasst unter sich die Funktionen der einzelnen Submodule zusammen. So übernimmt das Submodul FRApp (siehe Abschnitt 4.1.2) die grundlegende Steuerung des Knoten. Der FRScheduler (siehe Abschnitt 4.1.3) verwaltet die Events und steuert die Synchronisation. Im FRSync-Modul (siehe Abschnitt 4.1.4) werden die Berechnungen der Synchronisation durchgeführt und FRPort (siehe Abschnitt 4.1.5) übernimmt die Kommunikation nach außen mit dem Bus.
Neben den Submodulen enthält es außerdem zwei *inout* Gates. Einen für Channel A und einen für Channel B. Diese Gates sind mit den zwei Gates von FRPort verbunden. Jeder Knoten, der in einer Simulation erstellt wird, verfügt immer über beide Kanäle. Dabei spielt es keine Rolle, ob der Knoten eventuell nur an einen Kanal angeschlossen wird oder an beide. Für einen nicht verwendeten Kanal werden keine Events generiert, was wiederum keine Auswirkungen auf die Simulationszeit hat und somit ohne Bedenken so umgesetzt werden kann. Dieses Modul verfügt über keine weitere Logik.

4.1.2 FRApp

Das FRApp-Modul wird vom Scheduler (siehe Abschnitt 4.1.3) über das Erreichen der statischen und dynamischen Slots benachrichtigt. Daraufhin werden die Frames erstellt und und das Senden eingeleitet.

Aufbau des Moduls

Dieses Modul hat keine weiteren Submodule. Es hat als einzige Komponente das *input*-Gate *schedulerIn*.

Initialisierung

Während der Initialisierungsphase wird bei dem Modul FRScheduler das Gate gesetzt, zu dem der Scheduler die Events beim Erreichen der statischen und dynamischen Slots weiterleitet.

Weitere Logik

Dieses Modul empfängt auf dem Gate Nachrichten vom Scheduler. Diese Nachrichten spiegeln die Zeitpunkte wider, zu denen statische oder dynamische Frames verschickt werden sollen. Handelt es sich um ein Event im statischen Segment, wird zuerst ermittelt, ob der Frame auf einem oder auf beiden Kanälen übertragen werden soll. Je nachdem werden entweder ein oder oder zwei FRFrame Nachrichten erstellt. Die erforderlichen Informationen über den Frame werden entweder der von Scheduler übermittelten Nachricht entnommen oder direkt beim Scheduler abgefragt. Genauere Informationen über die FRFrame-Nachricht befinden sich im Abschnitt 4.1.8. Diese Nachricht wird dann mit Hilfe des FRPort-Moduls übermittelt.
Bei einem Event für das dynamische Segment steht der Kanal auf dem der Frame zu versenden ist schon fest und wird bei der Nachricht vom Scheduler mitgeliefert. Es kann also ohne weitere Überprüfung die FRFrame-Nachricht erstellt und versendet werden.

4.1.3 FRScheduler

Die prinzipielle Funktion des Schedulers entspricht der des Schedulers aus der TTEthernet-Simulation (vgl. CoRE RG, b; Steinbach, 2011). Er wurde jedoch um einige Funktionen erweitert und die Methoden mussten angepasst werden.
Die Hauptaufgaben des Schedulers sind es, das FRApp-Modul über das Erreichen von statischen und dynamischen Slots zu informieren sowie die Steuerung der Synchronisation. Außerdem

wird hier in jedem neuen Zyklus ein neuer Wert für die Abweichung der internen Uhr generiert.

Aufbau des Moduls

Auch dieses Modul hat keine weiteren Submodule. Es werden hier aber diverse Parameter definiert, die in den ini-Dateien konfiguriert werden können. Es handelt sich hierbei um die in Abschnitt 3.4 aufgeführten Konfigurationsmöglichkeiten. Jedem dieser Parameter wird ein Standardwert zugeordnet, der im Normalfall dem Minimum entspricht.

Initialisierung

In der Initialisierungsphase werden einigen Variablen die Werte der Parameter des Moduls zugeordnet. Zusätzlich wird mit der Methode *scheduleAt* das Event NEW_CYCLE zum aktuellen Zeitpunkt eingeplant, sodass der Knoten direkt nach dem Start der Simulation eine Nachricht an sich selbst schickt und die Methode *handleMessage* aufgerufen wird.

handleMessage

Das Modul kann 4 Arten von Nachrichten empfangen: NEW_CYCLE, NIT_EVENT, STATIC_EVENT oder DYNAMIC_EVENT. Alle 4 Nachrichtenarten schickt es sich mit *scheduleAt* selber zu.
NEW_CYCLE signalisiert den Start eines neuen Zyklus. Hier wird der Zykluszähler bis zu seinem Höchstwert (*gCycleCountMax*) hochgezählt und wieder auf 0 zurückgesetzt. Nach dem Zurücksetzen werden die Events für das statische und dynamische Segment erstellt. Anschließend wird der Wert für die Uhrenabweichung in der Methode *changeDrift* geändert und die Methoden *adjustMacrotick* und *correctEvents* aufgerufen, die im späteren Verlauf noch erläutert werden. Zum Schluss werden noch die Events NEW_CYCLE für den Zeitpunkt des nächsten Zyklus und NIT_EVENT für den Zeitpunkt der NIT in dem aktuellen Zyklus eingeplant.
Die Nachricht NIT_EVENT steht für den Anfang der NIT. Während der NIT findet die Berechnung der Synchronisationswerte statt. In geraden Zyklen wird der Offsetwert nur berechnet. In ungeraden Zyklen hingegen werden sowohl der Offsetwert als auch der Rate-Correction-Wert berechnet. Anschließend wird die Dauer der NIT mit dem Wert der Offset-Correction angepasst.
Handelt es sich bei der Nachricht um STATIC_EVENT oder DYNAMIC_EVENT, so wird

die Nachricht an das FRApp-Modul (siehe Abschnitt 4.1.2) weitergeleitet und dort weiter bearbeitet.

Registrierung der Events

In jedem Knoten können die Events NEW_CYCLE, NIT_EVENT, STATIC_EVENT und DYNAMIC_EVENT erstellt werden. Die beiden Events für einen neuen Zyklus und für die NIT sind auf jeden Fall in jedem Knoten vorhanden. Statische und dynamische Events werden nur erzeugt, wenn für den Knoten Slots in den ini-Dateien konfiguriert sind.
Die maximale Anzahl an unterschiedlichen Zyklen wird von der Variable *gCycleCountMax* bestimmt. Jedes Mal, wenn der Zähler dieser Zyklen *vCycleCounter* zurückgesetzt wird, werden die Methoden *registerStaticSlots* und *registerDynamicSlots* aufgerufen. Hier werden alle statischen und dynamischen Events für die nächsten Zyklen erstellt und registriert.
Mit Hilfe der Parameter *syncFrame*, *staticSlotsChA*, *staticSlotsChB*, *dynamicSlotsChA* und *dynamicSlotsChB* aus den ini-Dateien wird für jeden Slot, in dem gesendet werden soll, ein STATIC_EVENT bzw. DYNAMIC_EVENT erstellt. Den Events wird in diesen beiden Methoden zugeordnet, zu welchen Zeitpunkten sie auftreten sollen, um welche Frame-ID es sich bei dem Slot handelt und auf welchem Kanal der Frame verschickt werden soll. Ist das Event soweit erstellt, wird es an die Methode *registerEvent* übergeben.
In *registerEvent* wird das Event zuerst an eine Liste angehängt um eine spätere Verwaltung der Events zu ermöglichen. Anschließend wird das Event mit *scheduleAt* in die Eventloop eingereiht.

Zeitverwaltung

Der Scheduler übernimmt neben der Verwaltung der Events auch die Verwaltung der internen Uhr. In jedem Zyklus wird die Abweichung und damit die Dauer eines Macroticks verändert. Die Werte der Offset- und Rate-Correction, die vom FRSync-Modul geliefert werden, wirken sich auf die Dauer des gesamten Zyklus aus und können somit die Abweichung des Knotens von den Synchronisationsknoten ausgleichen.
Um das wie in Abschnitt 3.3 dargestellte Uhrenmodell umzusetzen wird am Anfang von jedem Zyklus in der Methode *changeDrift* eine Zufallszahl generiert, die den Wert darstellt um wie viel sich der Drift ändert. Dieser Wert liegt zwischen dem positiven und negativen Wert des Parameters *maxDriftChange*. Bevor der Wert zur Anwendung kommt wird noch überprüft ob er sich innerhalb der Grenzen des Parameters *maxDrift* befindet. Abbildung 4.3 stellt den Ablauf in einem Ablaufdiagramm dar.
Um von dem Submodul FRSync die Werte für die Offset- und die Rate-Correction zu bekommen,

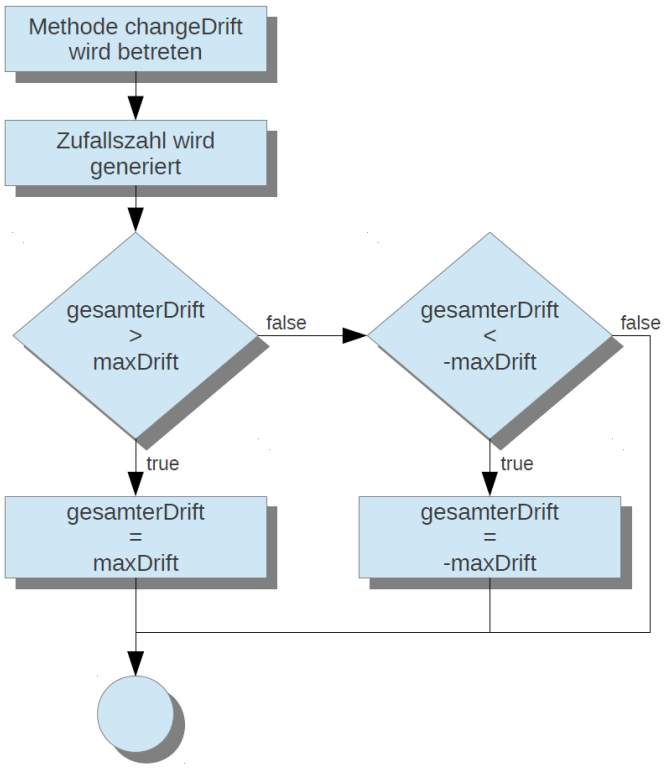

Abbildung 4.3: Ablaufdiagramm der Methode *changeDrift*

wird beim Erreichen der NIT dieses Modul benachrichtigt, vorausgesetzt es handelt sich um einen ungeraden Zyklus. Nachdem diese Werte vorliegen, wird in der Methode *correctNewCycle* der Anfang des nächsten Zyklus, mit dem Wert der Offset-Correction, korrigiert. Der Wert der Rate-Correction kommt in der Methode *adjustMacrotick* zur Anwendung. Hier wird die Dauer eines Macroticks an den aktuellen Drift und den Wert der Rate-Correction angepasst. Die Berechnung lautet:

$$DauerMT = (\frac{\mu T}{MT} + zRateCorrection) \times currentTick \tag{4.1}$$

Es wird also erst die Anzahl der Microticks pro Macrotick ermittelt, der Wert der Rate-Correction hinzuaddiert und diese Summe dann mit der aktuellen Dauer des Microticks (inklusive Drift) multipliziert.
Bei dieser Vorgehensweise unterscheidet sich die Simulation von dem realen FlexRay. Soll ein Zyklus beispielsweise einen Microtick länger dauern, so wird dieser Microtick einem Macrotick zugeordnet. Alle anderen Macroticks bleiben unverändert. Da sich diese Variante in

der Simulation nur realisieren lassen würde, wenn jeder Macrotick ein Event darstellen würde, wird dieser zusätzliche Microtick anteilig auf alle Macroticks aufgeteilt (siehe Abbildung 4.4). Da sich die in der Simulation verwendete Variante nur sehr gering von der realen Variante unterscheidet und die Dauer des Zyklus am Ende gleich ist, ist dieser Unterschied ohne Bedenken vertretbar.

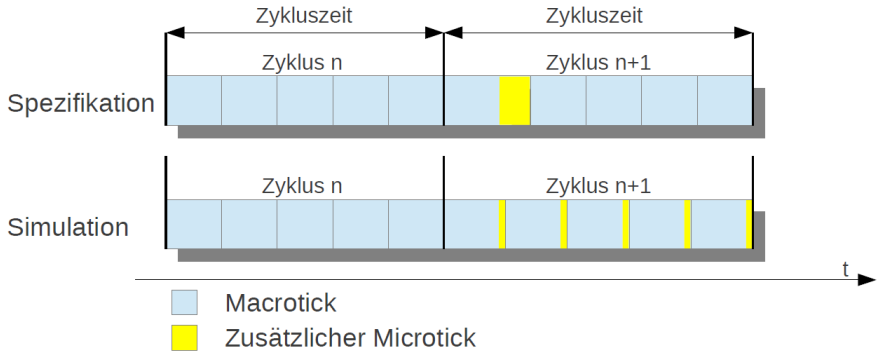

Abbildung 4.4: Verteilung eines zusätzlichen Microticks nach der Spezifikation und in der Simulation

Als letzte für die Zeitverwaltung relevante Methode ist *correctEvents* zu erwähnen. Sie wird aufgerufen, nachdem sich beim Event NEW_CYCLE die Dauer des Macroticks geändert hat. Dies ist notwendig, da die bisher registrierten Events noch nach der alten Zeit gescheduled sind. Also wird jedes registrierte Event abgebrochen und mit dem neuen Wert des Macroticks neu gescheduled.

Empfang eines dynamischen Frames

Wird von dem Knoten ein dynamischer Frame eines anderen Knotens empfangen, so wird der Scheduler darüber informiert. Da weder vorhersagbar ist, ob überhaupt eine dynamische Nachricht ankommt bzw. wie lange diese Übertragung dauert, muss auf ein solches Ereignis reagiert werden. Das Problem ist, dass sich der Sendezeitpunkt einer dynamischen Nachricht nach hinten verschiebt, das Event für diesen Zeitpunkt aber an dem ursprünglich mit *scheduleAt* registrierten Zeitpunkt auftreten würde.
Mit der Aufgabe, diese Events neu einzuordnen, befasst sich die Methode *dynamicFrameReceived*. Dauert die Übertragung länger als einen Minislot, so wird über die Liste mit den registrierten Events iteriert. Ist ein DYNAMIC_EVENT gefunden worden und ist es für den gleichen Kanal und Zyklus eingeteilt wie der empfangene Frame, dann wird dieses Event abgebrochen und muss anschließend neu zu einem späteren Zeitpunkt gescheduled werden.

Hierbei ist zu beachten, dass sich die Fälligkeit noch innerhalb des dynamischen Segments befindet.

Weitere Logik

Neben den bisher erläuterten Methoden verfügt dieses Modul noch über eine gewisse Anzahl an weiteren Methoden. Bei diesen handelt es sich hauptsächlich um kleinere Berechnungen z. B. um die genauen Zeitpunkte für ein statisches bzw. dynamisches Event zu berechnen, die Berechnung des aktuellen statischen Slots sowie die bisher verstrichenen Macroticks innerhalb des aktuellen Zyklus.
Als Letztes gibt es noch eine Methode, die während des Empfangs einer Synchronisationsnachricht zum Einsatz kommt. Sie trägt den Namen *calculateDeviationValue* und liefert den aktuellen Abstand vom ActionPoint des derzeitigen Slots.

4.1.4 FRSync

Das Modul FRSync liefert die Werte für die Offset- und die Rate-Correction. Als Grundlage dienen hier die ebenfalls in diesem Modul verwalteten Abweichungswerte aus den Synchronisationsnachrichten.

Aufbau des Moduls

Wie auch beim FRScheduler gibt es keine Submodule oder Gates. In der NED-Datei werden lediglich drei Parameter definiert und mit Standardwerten versehen. Diese Parameter werden für die Berechnungen der Korrekturwerte benötigt.

T_DevTable

Um die Abweichungswerte der Synchronisationsnachrichten zu speichern wurde in der Spezifikation ein dreidimensionales Array definiert (siehe Abschnitt 2.1.5). Ein solches Array wird in der Logik dieses Moduls ebenfalls benutzt. Die zu speichernden Datensätze können dem richtigen Kanal sowie einem geraden oder ungeraden Zyklus zugeordnet werden. Ein Datensatz enthält den Abweichungswert und einen Indikator, ob der Wert gültig ist oder nicht.

Initialisierung

In der Methode *initialize* werden den Parametern die Werte aus den ini-Dateien zugeordnet. Außerdem wird dem Array *T_DevTable* Speicher zugeteilt.

Verwaltung von *T_DevTable*

Für die Verwaltung des dreidimensionalen Arrays für die Abweichungswerte, stehen drei weitere Methoden zur Verfügung: *storeDeviationValue*, *getLineNr* und *resetTables*.
Sobald eine Synchronisationsnachricht eintrifft, wird die Methode *storeDeviationValue* aufgerufen. Sie bekommt alle nötigen Informationen um die Nachricht richtig einordnen zu können. Hierzu zählen die Frame-ID, der Kanal, der Abweichungswert, die Gültigkeit und ob der Frame in einem geraden oder ungeraden Zyklus empfangen wurde. Mit Hilfe der Frame-ID und der Methode *getLineNr* wird noch der richtige Speicherort im Array ermittelt.
Die Methode *getLineNr* verwaltet eine Liste (*position*), in der die Positionen für die Abweichungswerte innerhalb des Arrays gespeichert werden. Beim Aufruf wird die Frame-ID übergeben und die Liste wird nach schon vorhandenen Einträgen durchsucht. Ist dieser Wert bereits vorhanden, wird die entsprechende Position zurückgegeben. Ist dies nicht der Fall, wird ein neuer Eintrag in der Liste erstellt und die dazugehörige Position übertragen.
Am Ende eines ungeraden Zyklus spielen die in den beiden vorangegangenen Messbereichen ermittelten Abweichungswerte keine Rolle mehr. Deswegen werden in der Methode *resetTables* das Array und die Liste *position* zurückgesetzt. Während die Liste einfach gelöscht wird, wird die Gültigkeit der Messwerte im Array auf *false* gesetzt.

FTM Algorithmus

Eine weitere zentrale Rolle nimmt der FTM Algorithmus ein. In der dazugehörigen Methode wird das Verfahren gemäß der Spezifikation (siehe Abschnitt 2.1.5) realisiert. Wird diese Methode aufgerufen, bekommt sie eine Liste mit Werten übergeben. Anschließend wird die Liste sortiert und je nach Anzahl der Werte das jeweilige Ergebnis der Berechnung an die aufrufende Methode zurückgegeben.

Offset-Correction

Die Methode *offsetCorrectionCalculation* berechnet mit Hilfe der Abweichungswerte den Offset-Korrekturwert. Da die Simulation einige Punkte, die in der Spezifikation thematisiert werden, nicht berücksichtigt, ist der Ablauf der Offset-Correction etwas kürzer. Abbildung 4.5 zeigt das Ablaufdiagramm für die Simulation.
Im ersten Schritt müssen die Werte für die Berechnung ermittelt werden. Hierzu wird ein Wert von jedem Synchronisationsknoten in der Liste *zsMListAB* gespeichert. Liegen von einem Knoten zwei Werte vor (Kanal A und Kanal B) wird, wie in der Spezifikation gefordert, der kleinere Wert gewählt. Sofern gültige Werte vorliegen, wird die Liste an den FTM Algorithmus

übergeben. Der zurückgegebene Wert wird dahingehend überprüft, ob er sich in den definierten Grenzen des Parameters *pOffsetCorrectionOut* befindet. Wenn nicht wird dieser entsprechend angepasst und anschließend kann das Ergebnis übermittelt werden.

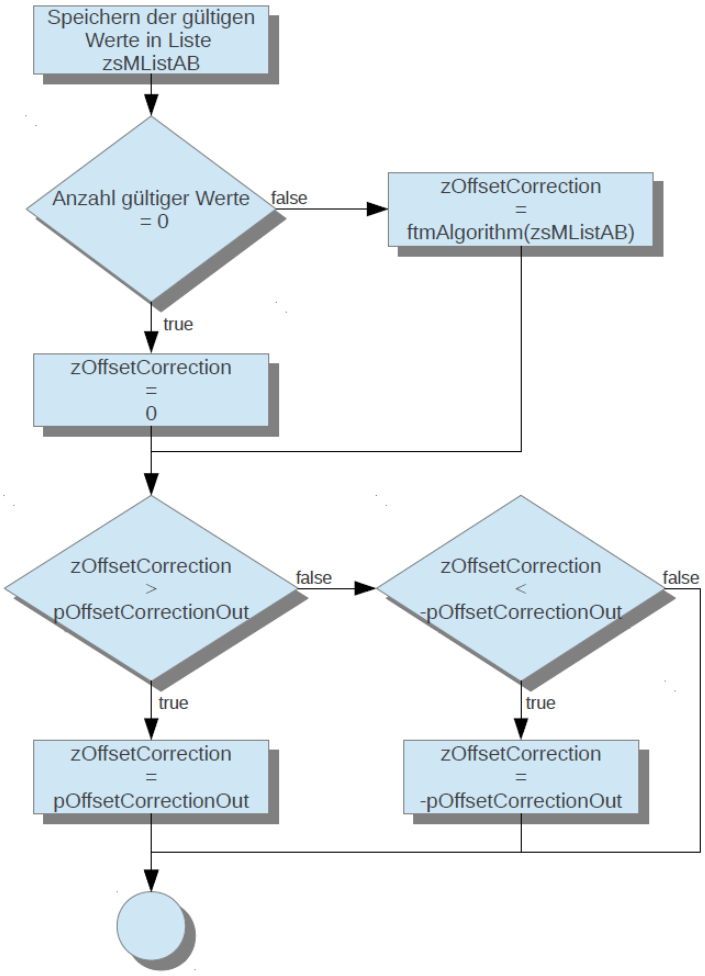

Abbildung 4.5: Ablaufdiagramm der Offset-Correction

Rate-Correction

In der Methode *rateCorrectionCalculation* wird der Rate-Korrekturwert berechnet. Wie schon bei der Offset-Correction unterscheidet sich die Vorgehensweise leicht von der in der Spezifikation. So werden die Themen startup und externe Synchronisation nicht berücksichtigt. Der Ablauf in der Simulation wird in Abbildung 4.6 dargestellt.
Zur Ermittlung der Werte, mit denen die Berechnung stattfinden soll, werden die Daten aus

dem Array *T_DevTable* herangezogen. Alle Abweichungswerte für einen Synchronisationsknoten, die in den letzten beiden Zyklen empfangen wurden (maximal 4 Werte: gerader Zyklus Kanal A und B, ungerader Zyklus Kanal A und B), werden für die Berechnung genutzt. Da in zwei Zyklen pro Kanal zwei Werte vorliegen, kann festgestellt werden, wie stark sich die Abweichung zu einem Knoten verändert hat. Es wird also pro Kanal die Differenz zwischen dem ungeraden und dem geraden Zyklus errechnet. Wurden auf beiden Kanälen *sync frames* empfangen, wird zusätzlich der Durchschnitt für beide Differenzen ermittelt. Die folgende Tabelle zeigt die Berechnungen für die jeweilige Situation:

Synchronisationsnachrichten	**Berechnung**
auf beiden Kanälen	$(ODD\_A - EVEN\_A + ODD\_B - EVEN\_B)/2$
auf Kanal A	$ODD\_A - EVEN\_A$
auf Kanal B	$ODD\_B - EVEN\_B$

Tabelle 4.1: Werteberechnung für die Rate-Correction

Für jeden Synchronisationsknoten, für den Werte vorliegen, wird das Ergebnis der Berechnung aus Tabelle 4.1 in der Liste *zsMRateAB* gesichert.
Sobald alle Einträge in dem Array abgearbeitet sind, wird die Liste an den FTM Algorithmus übergeben und das Ergebnis hiervon auf den bisherigen Korrekturwert (*zRateCorrection*) addiert. Anschließend wird in das Ergebnis noch ein gewisser Dämpfungswert (*pClusterDrift-Damping*) eingerechnet. Befindet sich der errechnete Korrekturwert zwischen dem positiven und dem negativen Wert der Dämpfung, wird er auf null gesetzt.
Bevor das Ergebnis übermittelt werden darf, wird es mit dem Grenzwert (*pRateCorrectionOut*) verglichen und, sollten die Grenzen überschritten werden, angepasst.

4.1.5 FRPort

Mittels des *FRPort*-Moduls stellt der Knoten die Verbindung zum Bus her. Hier werden Nachrichten verschickt und empfangen. Eingegangene Frames werden analysiert und die relevanten Informationen an andere Module weitergeleitet.

Aufbau des Moduls

Die beiden *inout* Gates des Moduls stellen die einzigen Komponenten innerhalb der NED-Datei dar. Sie werden direkt mit den externen Schnittstellen für Kanal A und B des Knotens verbunden und bekommen so direkt die Nachrichten vom Bus. Außerdem werden über die Gates die Frames an andere Knoten verschickt.

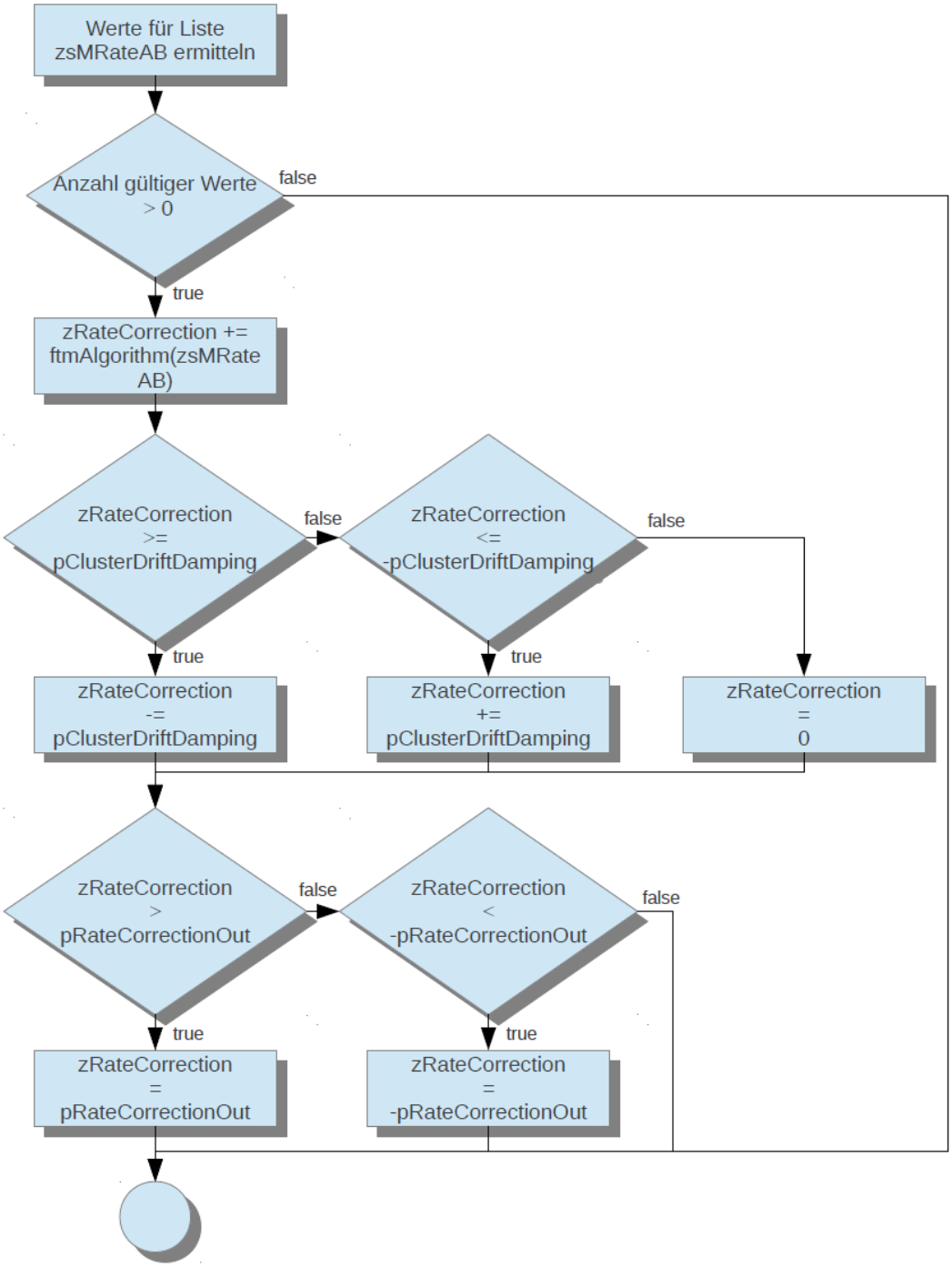

Abbildung 4.6: Ablaufdiagramm der Rate-Correction

handleMessage

Diese Methode wird aufgerufen, sobald eine Nachricht an einem der Gates eingeht. Es können zwei Arten von Frames eintreffen: statische und dynamische. Beide müssen unterschiedlich behandelt werden.
Handelt es sich um einen dynamischen Frame, wird dies dem Scheduler mitgeteilt. Die hier relevanten Informationen sind die Größe der Nachricht und der Kanal auf dem sie empfangen wurde.
Bei einem statischen Frame wird zuerst die Frame-ID der Nachricht mit der lokalen Frame-ID verglichen. Stimmen diese nicht überein, wurde der Frame im falschen Slot empfangen und eine Fehlermeldung wird ausgegeben. Stimmen sie überein, wird die Nachricht weiter verarbeitet. Es muss nun überprüft werden, ob es sich um eine Synchronisationsnachricht handelt. Ist dies der Fall wird im Modul *FRSync* die Sicherung des Abweichungswerts veranlasst (siehe Abschnitt 4.1.4).

Weitere Logik

Für dieses Modul gibt es mit *sendMsg* nur noch eine weitere Methode. Das Ausführen dieser Methode leitet das Modul *FRApp* ein (siehe Abschnitt 4.1.2). Die zu sendende Nachricht ist bereits erstellt und muss nur noch verschickt werden. Hierzu wird der Kanal, auf dem gesendet werden soll, aus der Nachricht gelesen und anschließend über das entsprechende Gate verschickt.

4.1.6 FRBus

Das *FRBus*-Modul stellt die Verbindung zwischen den einzelnen Knoten her. Jeder Knoten kann sich, je nach Bedarf, mit Kanal A und Kanal B verbinden. Der Bus empfängt dann die Nachricht eines Knotens und leitet sie an alle anderen Knoten, die an dem gleichen Kanal angeschlossen sind, weiter.

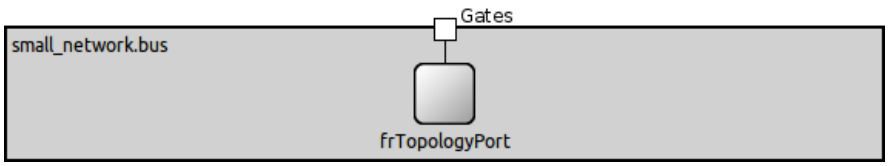

Abbildung 4.7: Aufbau des Moduls *FRBus*

Aufbau des Moduls

Das Modul besteht aus einer gewissen Anzahl an *inout*-Gates und dem Submodul *FRTopologyPort*. Das Submodul verfügt über die Logik des Busses während *FRBus* ähnlich wie *FRNode* nur den Rahmen für weitere Submodule und die Verbindung nach außen stellt. Auf dem aktuellen Stand der Simulation ist die Variante mit dem zusätzlichen Submodul nicht unbedingt nötig. Da es keine weiteren Komponenten gibt, könnte die Logik auch direkt im *FRBus*-Modul integriert werden. Da die Topologien aber noch erweitert werden sollen (z. B. Überprüfung der Nachrichten) wurde dieses Design gewählt.
Je nach Konfiguration des Netzwerks können unterschiedlich viele Knoten an den Bus angeschlossen werden. Außerdem gibt es Knoten, die nur mit einem der beiden Kanäle verbunden sind. Deswegen gibt es die Parameter *numberOfNodesChannelA* und *numberOfNodesChannelB*. Sie beinhalten die Anzahl der Knoten, die an dem jeweiligen Kanal angeschlossen werden. Beim Konfigurieren des Netzwerks müssen diese Parameter dementsprechend eingestellt werden. Die Verbindungen zwischen *FRBus* und *FRTopologyPort* werden im folgenden Codeausschnitt hergestellt:

```
1 connections:
2     for i=0..numberOfNodesChannelA-1 {
3         frTopologyPort.phyChannelA[i] <--> channelA[i];
4     }
5     for i=0..numberOfNodesChannelB-1 {
6         frTopologyPort.phyChannelB[i] <--> channelB[i];
```

4.1.7 FRTopologyPort

Dieses Modul bearbeitet die eintreffenden Nachrichten und entscheidet was mit ihnen passiert.

Aufbau des Moduls

Es gibt für jeden Kanal eine noch nicht näher definierte Anzahl an *inout*-Gates. Die genaue Anzahl wird bei der Konfiguration des Netzwerks festgelegt, wie schon im vorherigen Abschnitt 4.1.6 beschrieben wurde.

Logik

Die Logik beschränkt sich auf die Methode *handleMessage*. Trifft eine Nachricht ein, wird im ersten Schritt herausgefunden, auf welchem Kanal diese empfangen wurde. Anschließend

wird die Nachricht für jeden an dem gleichen Kanal angeschlossenen Knoten dupliziert und an diesen gesendet.

4.1.8 FRFrame

Hierbei handelt es sich um die FlexRay-spezifische Nachrichtenklasse. Damit die Knoten Informationen über den gesendeten Frame mitschicken könne, wurde diese Klasse erstellt.
In einer normalen Nachricht ist es lediglich möglich mit Hilfe der Methode *setKind* die Art der Nachricht (z. B. STATIC_EVENT oder DYNAMIC_EVENT) zu setzen. Um eine Nachricht aber zum Beispiel dem richtigen Slot zuzuordnen werden weitere Informationen benötigt. Deswegen enthält diese Nachricht zusätzlich einige Parameter:

Parameter	Funktion
frameID	spiegelt die Frame-ID wider, zu der der Frame versendet wurde
cycleNumber	spiegelt die Zyklusnummer wider, welche zum Sendezeitpunkt des Frames in dem Knoten aktuell war
size	die Größe der Nachricht, nur relevant für Nachrichten im dynamischen Segment
channel	der Kanal auf dem der Frame verschickt wurde
syncFrameIndicator	boolsche Variable, kann bei statischen Nachrichten *true* annehmen und markiert diese Nachricht somit als Synchronisationsnachricht

Tabelle 4.2: Variablen einer FRFrame-Nachricht

4.2 Beispielnetzwerk

In diesem Abschnitt wird anhand eines Beispiels beschrieben, wie ein Netzwerk aufgebaut und konfiguriert wird. Anschließend wird noch auf den Ablauf eingegangen.

4.2.1 Aufbau und Konfiguration

Bevor die einzelnen Komponenten des Netzwerks erstellt werden können, muss der Aufbau geplant werden. Es ist zu klären, wie viele Teilnehmer es geben soll, wie ein Kommunikationszyklus aufgebaut sein soll oder welchem Knoten welche Slots zugeteilt werden.

Aufbau

Ist die Planung abgeschlossen, kann mit dem Aufbau des Netzwerks gestartet werden. Dieser Aufbau wird, genau wie schon die Module, mit der NED-Sprache beschrieben. Die Knoten und das Topologiemodul werden als Submodule definiert. Unter dem Punkt *connections* werden die Knoten mit den Kanälen am Bus verbunden.
Das Beispielnetzwerk soll aus fünf Knoten bestehen, die an einen Bus angeschlossen sind (siehe Abbildung 4.8). Da als Topologiemodul bisher nur der Bus zur Verfügung steht, gibt es noch keine andere Alternative. Also wird als erstes unter dem Punkt *submodules* der Bus angelegt. Anschließend werden die Knoten erstellt. Folgender Ausschnitt aus dem Code zeigt das Erstellen des Busses und des ersten Knotens:

```
1 submodules:
2     bus: FRBus {
3     }
4
5     unit1: FRNode {
6     }
```

Im Anschluss an die Submodule müssen die Verbindungen erstellt werden. Wie aus Abbildung 4.8 ersichtlich ist, sollen die Knoten mit der Bezeichnung *Unit1, Unit3* und *Unit4* sowohl mit Kanal A als auch Kanal B verbunden werden. *Unit2* und *Unit5* nur an Kanal A bzw. B. Dementsprechend werden unter dem Punkt *connections* diese Verbindungen angelegt:

```
1 connections:
2     unit1.channelA <--> Channel <--> bus.channelA[0];
3     unit1.channelB <--> Channel <--> bus.channelB[0];
4     unit2.channelB <--> Channel <--> bus.channelB[1];
5     unit3.channelA <--> Channel <--> bus.channelA[1];
6     unit3.channelB <--> Channel <--> bus.channelB[2];
7     unit4.channelA <--> Channel <--> bus.channelA[2];
8     unit4.channelB <--> Channel <--> bus.channelB[3];
9     unit5.channelA <--> Channel <--> bus.channelA[3];
```

In diesem Beispiel wird die Verbindung mit einem *Channel* hergestellt. Hierfür lässt sich eine feste Signallaufzeit konfigurieren, die sich auf den Empfangszeitpunkt der Nachrichten auswirkt.

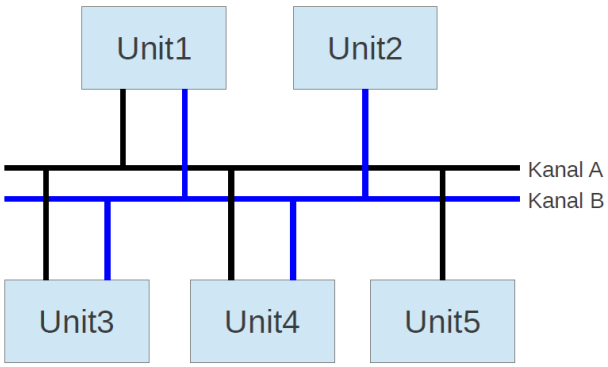

Abbildung 4.8: Aufbau des Beispielnetzwerks

Konfiguration des Netzwerks

Ist das Netzwerk soweit eingerichtet, wird es mit der Datei *omnetpp.ini* konfiguriert. Um diese möglichst übersichtlich zu halten, ist es empfehlenswert für jeden Knoten eine eigene ini-Datei anzulegen (z. B. *unit1.ini*). Diese zusätzlichen Konfigurationsdateien müssen mittels *include* in *omnetpp.ini* importiert werden.
Im Beispiel werden in *omnetpp.ini* die global gültigen Parameter deklariert. Zuerst wird die Anzahl der Knoten pro Kanal angegeben. Für die Parameter *numberOfNodesChannelA* und *numberOfNodesChannelB* wird in diesem Fall also der Wert 4 gewählt. Außerdem werden hier diverse FlexRay-spezifische Parameter bestimmt: die Dauer der einzelnen Elemente des Kommunikationszyklus, die Anzahl der unterschiedlichen Zyklen, die Dauer für einen Macrotick sowie die Anzahl der statischen Slots und der Minislots. Der folgende Ausschnitt zeigt die Datei *omnetpp.ini* mit den für das Beispiel konfigurierten Werten:

```
1  [General]
2  network = small_network
3
4  **.numberOfNodesChannelA = 4
5  **.numberOfNodesChannelB = 4
6
7  **frScheduler.gNumberOfStaticSlots = 10
8  **frScheduler.gNumberOfMinislots = 10
9  **frScheduler.gCycleCountMax = 7
10 **frScheduler.gdMacrotick = 2us
11 **frScheduler.gdStaticSlot = 50
12 **frScheduler.gdMinislot = 10
13 **frScheduler.gdSymbolWindow = 0
```

```
**frScheduler.gdNIT = 20
**frScheduler.gdActionPointOffset = 4
**frScheduler.gdMinislotActionPointOffset = 2
**frScheduler.busSpeed = 10Mbps

include unit1.ini
include unit2.ini
include unit3.ini
include unit4.ini
include unit5.ini
```

Die Anzahl der Macroticks pro Kommunikationszyklus lässt sich nun wie folgt berechnen:
Dauer statisches Segment in Macroticks:
$static = gNumberOfStaticSlots * gdStaticSlot$
Dauer dynamisches Segment in Macroticks:
$dynamic = gNumberOfMinislots * gdMinislot$
Dauer gesamter Zyklus in Macroticks:
$Zyklus = static + dynamic + gdSymbolWindow + gdNIT$
Für das Beispiel würde sich also Folgendes ergeben:
$(10 * 50MT) + (10 * 10MT) + 0MT + 20MT = 620MT$
Wird dieses Ergebnis mit dem Wert *gdMacrotick* multipliziert, erhält man die Dauer eines Kommunikationszyklus:
$620MT * 2\frac{\mu s}{MT} = 1240\mu s$

Konfiguration der Knoten

Da für das Beispiel jeder Knoten in einer eigenen Datei beschrieben wird, wird hier nur auf die Datei für den Knoten *Unit1* eingegangen. Der Aufbau der anderen ini-Dateien ist identisch, während sich die Werte unterscheiden können.
Bei der Dauer des Microticks, die ja 3 unterschiedliche Werte annehmen kann, muss beachtet werden, dass die Zeit für einen Macrotick durch diesen Wert ganzzahlig teilbar ist. Bei den anderen Werten ist im Prinzip nur darauf zu achten, dass sie sich innerhalb der Wertebereiche befinden. Wird beispielsweise ein sehr hoher Driftwert ermöglicht, aber die maximalen Korrekturwerte sehr niedrig gehalten, kann dieses dazu führen, dass sich die Knoten nicht richtig synchronisieren können.
Den aufwendigsten Part nimmt die Konfiguration der statischen und dynamischen Slots ein. Da sich der Schedule erst wiederholt, wenn der Zykluszähler zurückgesetzt wird, müssen für jeden Zyklus von 0 – *gCycleCountMax* die Slots verteilt werden. Hierzu werden die jew-

eiligen Slotnummern den Parametern *staticSlotsChA*, *staticSlotsChB*, *dynamicSlotsChA* und *dynamicSlotsChB* zugeordnet. Die Nummerierung der Slots wird fortlaufend vorgenommen. Besteht das Netzwerk, wie das Beispiel, aus 10 statischen Slots besteht der erste Zyklus aus den Slotnummern 1 – 10, der zweite Zyklus aus 11 – 20 usw.. Die Abbildung 4.9 verdeutlicht dieses Vorgehen und zeigt gleichzeitig die Konfiguration des statischen Segments der Beispielsimulation.

Die Knoten *Unit1*, *Unit3* und *Unit4* wurden als Synchronisationsknoten ausgewählt und sie senden ihre *sync frames* immer im gleichen Slot eines Zyklus. Die restlichen Slots können frei verteilt werden. Die Parameter für das statische Segment werden in *unit1.ini* folgendermaßen konfiguriert:

```
1 **.unit1.frScheduler.syncFrame = 1
2 **.unit1.frScheduler.staticSlotsChA = "1 11 21 31 41 50 51 61 71"
3 **.unit1.frScheduler.staticSlotsChB = "1 11 21 27 31 41 51 61 71"
```

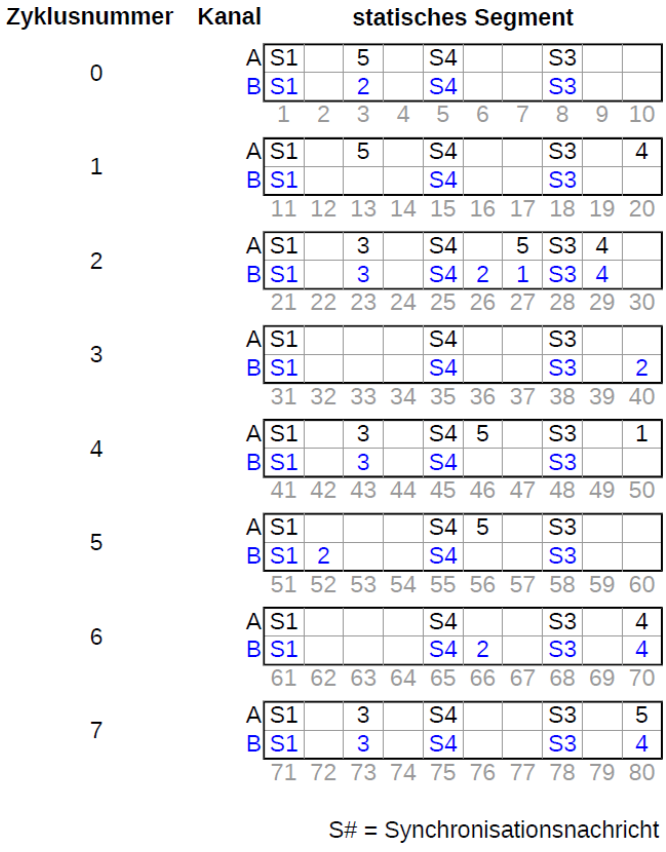

Abbildung 4.9: Verteilung der statischen Slots im Beispielnetzwerk

Bei der Konfiguration des dynamischen Slots werden die Slots der unterschiedlichen Zyklen ebenfalls fortlaufend nummeriert. Aufgrund der nicht vorhandenen Synchronisationsnachrichten können hier aber alle frei vergeben werden.

4.2.2 Ablauf

Nach der Konfiguration kann die Simulation gestartet werden. In den beiden Fenstern der OMNeT-Simulation (siehe Abschnitt 2.3) kann nun der Ablauf der Simulation verfolgt werden. Hier kann man sehen, dass die Nachrichten nach und nach von den Knoten verschickt und vom Bus verteilt werden.
Die ersten Events finden alle zum Zeitpunkt $T = 0$ statt. T steht für die Anzahl verstrichener Sekunden in der Simulation. Bei diesen Ereignissen handelt es sich um die NEW_CYCLE-Events am Anfang eines jeden Knotens. Hier wird schon das erste Mal der Drift der Uhren simuliert. Dies hat sofort Auswirkungen auf die Zeitpunkte der Nachrichten.
Da nach der Konfiguration des Beispielnetzwerks der Knoten *Unit1* direkt im ersten Slot des statischen Segments eine Synchronisationsnachricht senden soll, ist dies auch das erste STATIC_EVENT in der Simulation. Der Zeitpunkt dieses Ereignisses lautet $T = 7,999981^{-6}$. Der eigentliche Zeitpunkt für die erste Nachricht würde bei 8 μs liegen (*gdActionPointOffset* * *gdMacrotick*). Daraus ist ersichtlich, dass der Drift von *Unit1* einen relativ geringen negativen Wert angenommen hat. Auch für die anderen Knoten hat sich die Dauer eines Macroticks verändert. In dem Testlauf des Beispielnetzwerks wurden die Zeiten der einzelnen Knoten mittels Debugausgaben ausgegeben. Für *Unit2* liegt der Wert bei $1,99999^{-6}$ und für *Unit5* bei $2,00001^{-6}$. Da diese beiden Knoten im dritten statischen Slot gescheduled sind, müssten sie eigentlich zur gleichen Zeit ihre Frames verschicken. Aufgrund der Tatsache, dass *Unit2* einen etwas kürzeren Macrotick hat, tritt dieses Event bei ihm allerdings etwas früher auf. Deswegen wird die Nachricht von diesem Knoten zuerst an den Bus geleitet. In der grafischen Oberfläche kann man genau dieses Verhalten auch beobachten.

5 Ergebnis

In diesem Kapitel soll die Lauffähigkeit der implementierten Funktionen der Simulation in Hinsicht auf die FlexRay-Spezifikation und den in Kapitel 3 definierten Anforderungen verifiziert werden. Zudem werden bewusst fehlerhafte Netzwerke erzeugt, um die geforderten Fehlerszenarien zu testen.

5.1 Lauffähigkeit der Funktionen

Als erstes soll die Lauffähigkeit der in den Anforderungen definierten Funktionen festgestellt werden.

Aufbau des Netzwerks

Es ist möglich den Kommunikationszyklus nach belieben zu konfigurieren. Man kann die Dauer der einzelnen Elemente festlegen und sowohl für das statische als auch das dynamische Segment die genaue Unterteilung in Slots vornehmen.

Konfiguration des Netzwerks

Die Parameter eines Netzwerks können frei konfiguriert werden. Neben den Segmenten des Kommunikationszyklus, kann auch die Zeit für einen Macrotick für das gesamte Netzwerk definiert werden.
Die Parameter eines Knotens lassen sich unabhängig voneinander gestalten. Ihnen können die genauen Slots in den statischen und dynamischen Segmenten zugeordnet werden, in denen sie senden sollen und falls gewünscht ein *key slot* bestimmt werden.

Zeitabweichung und Synchronisation

Mit den Parametern *maxDrift* und *maxDriftChange* kann die Abweichung der lokalen Uhr eines jeden Knotens eingestellt werden. Dies hat zur Folge, dass jeder Busteilnehmer von der eigentlichen Zeit abweicht.

	even				odd			
	Kanal A		Kanal B		Kanal A		Kanal B	
	value	valid	value	valid	value	valid	value	valid
1	0	true	0	true	0	true	0	true
2	-7	true	-7	true	-6	true	-6	true
3	-12	true	-12	true	-14	true	-14	true

Abbildung 5.1: Dreidimensionales Array mit Messwerten aus dem Beispielnetzwerk

Die Knoten mit einem definierten *key slot* verschicken in den hierdurch definierten statischen Slots Synchronisationsnachrichten, die von allen anderen Knoten genutzt werden um die Werte der Offset- und Rate-Correction zu errechnen. Anhand eines zufälligen Beispiels aus dem Beispielnetzwerk soll der Synchronisationsvorgang einmal nachgerechnet werden:
Der Knoten *unit1* empfängt von den Knoten *unit3* und *unit4* Synchronisationsnachrichten. Da es sich bei ihm selbst ebenfalls um einen Synchronisationsknoten handelt, ist das Array *T_DevTable* am Ende des ungeraden Kommunikationszyklus wie in Abbildung 5.1 mit Werten gefüllt. Da sowohl auf Kanal A als auch auf Kanal B die gleichen Werte empfangen wurden, setzt sich die Berechnung im FTM-Algorithmus für die Offset-Correction wie folgt zusammen:

$$\begin{array}{l} \cancel{\emptyset} \\ -6 \\ \cancel{-14} \end{array} \quad \rightarrow (-6-6)/2 = -6$$

Der Wert für die Offset-Correction müsste also -6 betragen. Von der Simulation wird genau dieser Wert als Korrekturwert ausgegeben.
Für die Rate-Correction müssen auch drei Werte an den FTM-Algorithmus übergeben werden. Die Formel für diese Werte lautet: $(ODD\_A - EVEN\_A + ODD\_B - EVEN\_B)/2$ und es ergibt sich somit die folgende Berechnung:

$$\begin{array}{l} \cancel{2} \\ 0 \\ \cancel{4} \end{array} \quad \rightarrow (0+0)/2 = 0$$

Der Parameter *zRateCorrection* betrug im vorherigen Zyklus -41 und ändert sich vorerst nicht. Da aber noch der Dämpfungswert *pClusterDriftDamping* zu dem Ergebnis addiert wird, ergibt

sich ein Wert von -39. Auch dieser Wert wird von der Simulation als neuer Korrekturwert angezeigt.

5.2 Fehlererkennung

Die Simulation kann bestimmte Fehlerszenarien (siehe Abschnitt 3.5) erkennen und diese an den Benutzer weiterleiten. Um fehlerhaftes Verhalten zu simulieren, wurden Netzwerke angelegt, die bewusst falsche Parameter enthalten.

Gleichzeitiges senden: Es wurde ein Netzwerk erstellt, in dem zwei Knoten die gleichen Parameter für die Werte *syncFrame* und *staticSlotsChA* enthalten. Sobald das erkannt wurde, gibt die Simulation folgende Fehlermeldung aus:

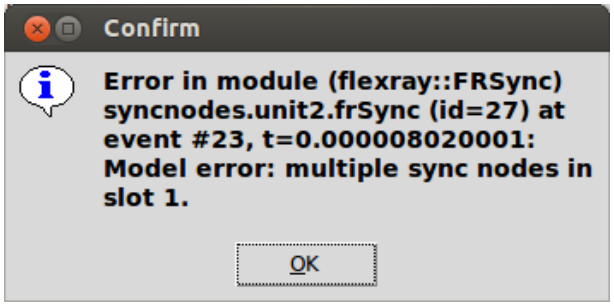

Abbildung 5.2: Fehlermeldung beim gleichzeitigen Senden zweier Synchronisationsknoten

Zu viele Synchronisationsknoten: Beim Erstellen des Netzwerks für diesen Fehler wurden 16 Knoten angelegt, von denen jeder in einem eigenen Slot eine Synchronisationsnachricht verschickt. Nachdem der letzte Knoten seinen Frame verschickt hat und empfangen wird, gibt die Simulation folgende Fehlermeldung aus:

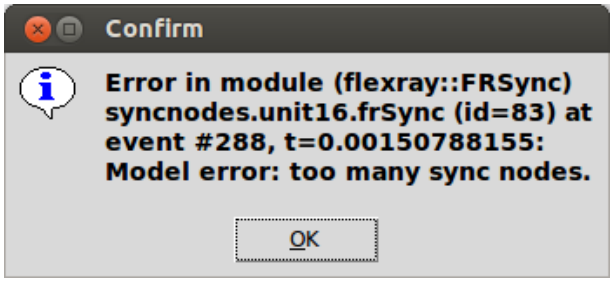

Abbildung 5.3: Fehlermeldung bei zu vielen Synchronisationsknoten

Frames in falschem Slot empfangen: Auch für den letzten Fall wurden die Parameter einiger Knoten verändert um die Fehlermeldung zu erhalten. So wurden in einem Knoten die Werte für den Drift über den erlaubten Wert erhöht, die Dauer eines statischen Slots auf ein Minimum reduziert und die maximalen Korrekturwerte der Synchronisationen sehr niedrig eingestellt.
Werden die Driftwerte bei einem Knoten sehr hoch eingestellt, fängt dieser Knoten schnell an in falschen Slots zu senden. Die Information über den Empfang in einem falschen Slot wird von den Knoten wie gewollt angezeigt. Bis auf die Tatsache, dass zu falschen Zeitpunkten Frames über den Bus versendet werden, hat dies aber keine weiteren Einflüsse auf das Netzwerk, da, selbst wenn es sich um einen Synchronisationsknoten handelt, die hohen Abweichungswerte durch den FTM-Algorithmus nicht berücksichtigt werden.
Bei der Verringerung der maximalen Korrekturwerte *pOffsetCorrectionOut* und *pRateCorrectionOut* verhält es sich genau wie bei den Driftwerten. Solange nur ein Knoten betroffen ist, ist die Synchronisation nicht gefährdet. Bei mehreren Knoten können sich diese Einstellungen auf das gesamte Netzwerk auswirken.
Sind die Slots sehr kurz eingestellt und betragen die Driftwerte gleichzeitig keine geringeren Werte, hat diese Einstellung Auswirkungen auf das gesamte Netzwerk und es ist möglich, dass die komplette Synchronisation in allen Knoten nicht mehr funktioniert.

6 Zusammenfassung und Ausblick

In dieser Arbeit soll das FlexRay-Kommunikationssystem in der ereignisorientierten Simulationsumgebung OMNeT++ implementiert werden. In diesem letzten Kapitel werden die Ergebnisse der Arbeit zusammengefasst (siehe Abschnitt 6.1) und ein Ausblick auf weiterführende Arbeiten gegeben (siehe Abschnitt 6.2).

6.1 Zusammenfassung der Ergebnisse

Die in Kapitel 3 gestellten Anforderungen können als erfüllt betrachtet werden. In der Simulation ist es möglich ein Netzwerk nach den eigenen Wünschen aufzubauen und zu konfigurieren. Während des Betriebes laufen die Uhren der einzelnen Knoten um einen zufälligen Wert auseinander und somit hat jeder einen anderen Blick auf die aktuelle Zeit. Mit Hilfe der Synchronisationsverfahren können die Knoten ihre Abweichung von den anderen Teilnehmern kompensieren um so ihre Nachrichten zum richtigen Zeitpunkt zu versenden.
Die Grundlage für eine Verbindung der TTE-Simulation mit der FlexRay-Simulation ist also gelegt. Dies kann dann in zukünftigen Arbeiten realisiert werden.

6.2 Ausblick

Im Rahmen der Arbeit wurde die grundlegende Funktionsweise von FlexRay in OMNeT++ implementiert. Das Kommunikationssystem bietet aber noch einige weitere Funktionen und auch im Zusammenhang mit der TTE-Simulation stehen noch Arbeiten an.

FlexRay Funktionen

Ein FlexRay-Netzwerk kann neben der implementierten Bustopologie auch als Sterntopologie oder als eine Kombination aus diesen beiden Topologien aufgebaut werden. Sind die Sternkoppler implementiert, bietet sich zusätzlich die Möglichkeit zur Integration von so genannten Busguardians. Diese kontrollieren die Kommunikation im Netzwerk und nehmen gegebenenfalls Einfluss darauf, ob ein Knoten senden darf oder nicht.

Zur Zeit wird angenommen, dass bereits alle Knoten im Netzwerk arbeiten und zueinander synchron sind. Die komplette Startphase wird also übersprungen. Würde das startup-Verfahren von FlexRay implementiert werden, kann auch diese Phase simuliert werden und es wäre möglich, dass Knoten zu beliebigen Zeitpunkten sich in das Netzwerk integrieren und an der Kommunikation teilnehmen. In diesem Zusammenhang könnte man den Knoten auch ermöglichen das Senden von Frames einzustellen (z. B. wenn der Knoten nicht mehr synchron ist) und sich im weiteren Verlauf der Simulation wieder in das Netzwerk zu integrieren.
Um dem Verhalten eines realen Systems näher zu kommen, können noch Störsignale auf dem Bus und der Verlust von Nachrichten simuliert werden.
Um ein konfiguriertes Netzwerk besser bewerten zu können, wäre es von Vorteil, wenn man den Verlauf einiger Variablen (z. B. die Höhe der Korrekturwerte) im Nachhinein analysieren könnte. Hierzu müssen im Programmcode Signale erstellt werden, die die einzelnen Werte regelmäßig speichern.

Weitere Funktionen

Das schon in der Einleitung erwähnte Gateway zwischen TTE- und FlexRay-Simulation, um eine Kommunikation der beiden Systeme miteinander zu ermöglichen, ist noch eine weitere zu realisierende Komponente. Um hier auch einen Austausch von Daten zu ermöglichen, könnte das Versenden von Payload innerhalb der Frames ermöglicht werden.

Literaturverzeichnis

[CoRE RG a] CoRE RG: *Communication over Real-time Ethernet*. – URL http://core.informatik.haw-hamburg.de

[CoRE RG b] CoRE RG: *TTE for INET*. – URL http://tte4inet.realmv6.org

[Eclipse Foundation] Eclipse Foundation: *Eclipse IDE*. – URL http://www.eclipse.org/. – Zugriffsdatum: 2011-01-29

[FlexRay Consortium] FlexRay Consortium: *FlexRay specifications*. – URL http://flexray.com/index.php?sid=8d1041c86459c6fe9ea6bf6f721349c8&pid=94&lang=de. – Zugriffsdatum: 2012-10-08

[FlexRay Consortium 2010a] FlexRay Consortium: FlexRay Communications System Electrical Physical Layer Specification / FlexRay Consortium. Stuttgart, Oktober 2010 (3.0.1). – Specification

[FlexRay Consortium 2010b] FlexRay Consortium: Protocol Specification / FlexRay Consortium. Stuttgart, Oktober 2010 (3.0.1). – Specification

[Muller und Valle 2011] Muller, C. ; Valle, M.: Design and simulation of automotive communication networks: the challenges. In: *e & i Elektrotechnik und Informationstechnik* 128 (2011), S. 228–233. – URL http://dx.doi.org/10.1007/s00502-011-0008-6. – ISSN 0932-383X

[NYFEGA Elektro-Garage AG] NYFEGA Elektro-Garage AG: *Phaeton Bordnetz*. – URL http://www.nyfega.ch/bilder/phaeton_bordnetz.jpg. – Zugriffsdatum: 2011-02-23

[OMNeT++ Community a] OMNeT++ Community: *OMNeT++ 4.2*. – URL http://www.omnetpp.org

[OMNeT++ Community b] OMNeT++ Community: *TicToc Tutorial for OMNeT++*. – URL http://www.omnetpp.org/doc/omnetpp/tictoc-tutorial/index.html

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

[Scherf 2010] Scherf, Helmut: *Modellbildung und Simulation dynamischer Systeme*. 4. München : Oldenbourg, 2010. – ISBN 978-3-486-59655-7

[Steinbach 2011] Steinbach, Till: *Echtzeit-Ethernet für Anwendungen im Automobil: Metriken und deren simulationsbasierte Evaluierung am Beispiel von TTEthernet*. Hamburg, Hochschule für Angewandte Wissenschaften Hamburg, Masterthesis, Februar 2011

[VDA 2006] VDA: Auto Jahresbericht 2006 / Verband der Automobilindustrie e. V. Frankfurt am Main, 2006. – Jahresbericht. http://www.vda.de/de/downloads/476/?PHPSESSID=41mq8ff4ipph19li0a420u0ua3. – ISSN 0171-4317

Abbildungsverzeichnis

Tabellenverzeichnis

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

Hamburg, 23. November 2012 Stefan Buschmann