UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

Dirk Försterling*

Interaktive Visualisierung der Medienzugriffskontrolle beim FlexRay-Protokoll

Gedruckt am 25. September 2006

*flexray@domain-mx.de

INTERNE BERICHTE
INTERNAL REPORTS

Lehrstuhl XII (Technische Informatik und Eingebettete Systeme und Didaktik der Informatik)
Fachbereich Informatik
Universität Dortmund
Gutachter:
Birgit Sirocic
Prof. Dr. Peter Marwedel

GERMANY · D-44221 DORTMUND

Danksagung

An dieser Stelle möchte ich mich zunächst bei Birgit Sirocic bedanken, die mir bei der Ertellung meiner Diplomarbeit einen großzügigen kreativen Freiraum gelassen hat, meine vielen Fragen nie als Belästigung empfand und stets gemeinsam mit mir geeignete Antworten darauf fand.

Ich möchte auch meinen Dank an Marianne und André Bialojahn, Andre Dierker, Conny und Jürgen Glitsch und Ralf Kellermann aussprechen, die mir ihre Zeit geschenkt haben, um mir bei der Korrektur des Textes wertvolle Hinweise und Anregungen zu geben.

Inhaltsverzeichnis

Bilderverzeichnis **vii**

Tabellenverzeichnis **x**

1 Einführung **1**
1.1 Motivation . . . 1
1.2 Ziel . . . 2
1.3 Aufbau der Arbeit . . . 4

2 Grundlagen **5**
2.1 Feldbusse und Feldbus-Protokolle . . . 5
2.1.1 Controller Area Network (CAN) . . . 5
2.1.2 Media Oriented Systems Transport (MOST) . . . 6
2.1.3 Time Triggered Protocol (TTP) . . . 7
2.1.4 byteflight . . . 8
2.1.5 FlexRay . . . 8
2.1.6 Medienzugriffskontrolle beim FlexRay-Protokoll . . . 10
2.1.7 Buswächter . . . 14
2.2 Komponenten einer interaktiven Visualisierung . . . 14
2.2.1 Was sind Lernprogramme? . . . 14
2.2.2 Behaviorismus, Kognitivismus und Konstruktivismus . . . 15
2.2.3 Interaktivität in Lernprogrammen . . . 22
2.2.4 Simulation . . . 24
2.2.5 Visualisierung . . . 26
2.2.6 Die interaktive Visualisierung . . . 28
2.3 Zusammenfassung . . . 28

3 Existierende Lehrmittel und Programme **31**
3.1 Aus der Lehre . . . 31
3.2 Aus der Industrie . . . 33
3.2.1 Programme von Vector Informatik . . . 34
3.2.2 dSPACE Simulator und Simulink . . . 34
3.2.3 Mirabilis Design VisualSim FlexRay-Modell . . . 36
3.3 Zusammenfassung . . . 38

4 Konzeption **41**
4.1 Zielgruppe und Voraussetzungen . . . 41
4.2 Präzisierung der Ziele . . . 41
4.3 Vereinfachungen . . . 42
4.4 Visualisierung . . . 47
4.5 Simulation . . . 50
4.5.1 Modell . . . 50
4.5.2 Arbeitsweise . . . 53
4.5.3 Konfiguration . . . 53
4.6 Interaktion . . . 54
4.6.1 Tutorial . . . 54
4.6.2 Steuerung der Simulation . . . 55
4.6.3 Feedback für den Lernenden . . . 55
4.7 Zusammenfassung . . . 55

5 Umsetzung **57**
5.1 Anforderungen . . . 57
5.2 Allgemeines zur Programmierung . . . 58
5.3 Ablauf der Simulation . . . 60
5.4 Visualisierung und Steuerung . . . 61
5.4.1 Feingestaltungen und Auflockerungen . . . 65
5.5 Tutorial und Hilfe . . . 69
5.6 Sonstiges . . . 70
5.6.1 Speicherung der Modelldaten . . . 70
5.6.2 Mehrere Modelle . . . 70
5.7 Zusammenfassung . . . 70

6 Fazit **71**
6.1 Bewertung des Lernprogramms . . . 71
6.2 Einordnung im didaktischen Kontext . . . 72

7 Ausblick **73**
7.1 Simulation . . . 73
7.2 Visualisierung . . . 74
7.3 Benutzerführung und Interaktion . . . 74

Anhang **75**

A Programmbedienung **77**
A.1 Programmfenster . . . 77
A.2 Funktionen entlang der Menüstruktur . . . 78
A.2.1 FlexRay-Cluster . . . 78

A.2.2 Bearbeiten 78
A.2.3 Simulation 79
A.2.4 Ansicht 80
A.2.5 Einstellungen 80
A.2.6 Hilfe 81
A.3 Erstellung und Konfiguration von Modellen 81
A.4 Simulation 85

B FlexRay-Protokollparameter 87

C Besonderheiten der Implementierung 89
C.1 Komponentenbauweise der Visualisierung 89
C.2 SpringUtilities 90
C.3 Transparenz 90
C.4 Drehknopf 90
C.5 Verschieben von Komponenten mit der Maus 91
C.6 Speicherung der XML-Daten 91
C.7 Fehler in den Java-Bibliotheken 92
C.7.1 6358355: Monotonie bei *float*-Literalen 92
C.7.2 6296446: XML wird nicht eingerückt 92
C.7.3 6357387: Statt Kreissegmenten werden Vollkreise gezeichnet . . 93
C.7.4 6292739: Spracheinstellung wirkt sich nicht auf Tastenkürzeltexte aus 93
C.7.5 4626632: *null*-Zeiger von Actions werden dereferenziert 93

Literaturverzeichnis 95

Bilderverzeichnis

2.1 Zeitzählerstruktur bei FlexRay 9
2.2 Zeitvorgaben im FlexRay-Kommunikationszyklus 11
2.3 NAND-Übung: „Falsch“ 17
2.4 Vokabeltrainer-Feedback: „Falsch“ 18
2.5 Vokabeltrainer-Feedback: Die Lösung 19
2.6 HYDRO.EXE 21
2.7 Java Tutorial Einstieg 23
2.8 Java Tutorial Getting Started 24
2.9 Java Tutorial Vergleichsbild 27

3.1 FlexRay-Beispiel aus der Lehre 31
3.2 TDMA-Beispiel aus der Lehre 32
3.3 Simulationskonfiguration bei CANoe 35
3.4 Darstellung der Daten bei CANoe 36
3.5 FlexRay-Modell für dSPACE-Simulator in Simulink 37
3.6 FlexRay-Modell für dSPACE-Simulator: Konfiguration 38
3.7 Mirabilis Design VisualSim FlexRay-Modell 39

4.1 Zeitverhalten im statischen Segment. Quelle: [Fle05c]. 44
4.2 Einzelbild-Darstellung der Medienzugriffskontrolle 47
4.3 Konzeptgrafik der interaktiven Visualisierung 48
4.4 Slot-Zuordnung FlexConfig 49
4.5 Modellstruktur 51

5.1 Klassenstruktur, einfach 59
5.2 Programm im Editmodus 62
5.3 Visualisierung der Fragmentierung 63
5.4 Visualisierung der Nachrichtenanzahl 63
5.5 Kommunikationszyklus, statisches Segment 64
5.6 Kommunikationszyklus, dynamisches Segment 64
5.7 Programm während der Simulation 65
5.8 Drehknopf für die Simulationsgeschwindigkeit 66
5.9 Markierungen für Knoten 66
5.10 Abschaltbare Kantenglättung 67
5.11 Transparenz beim Verschieben von Knoten 67

5.12 Größenänderung der Darstellung des Kommunikationszyklusses 68
5.13 Automatische Verkürzung von Texten 68
5.14 Verschieben der Kommunikationszyklusdarstellung 69
5.15 Navigationspfad im Tutorial 69

A.1 Programmfenster 77
A.2 Menü: FlexRay-Cluster 78
A.3 Menü: Bearbeiten 79
A.4 Menü: Simulation 79
A.5 Menü: Ansicht 80
A.6 Menü: Einstellungen 80
A.7 Menü: Hilfe 81
A.8 Der erste Kommunikationszyklus 82
A.9 Konfiguration des Kommunikationszyklusses 82
A.10 Erstes Modell ohne Verbindungen 83
A.11 Erster Knotendialog 84
A.12 Erweiterter Knotendialog 84
A.13 Vier Knoten verbunden mit Kanal A 85
A.14 Knotenauswahl für Slot 0, Kanal A 86
A.15 Kommunikationszyklus nach Knotenzuordnung 86

C.1 Fehlerhafte Menüdarstellung 93

Tabellenverzeichnis

B.1 FlexRay-Parameter-Präfixe 87
B.2 Verwendete Einheiten 87
B.3 FlexRay-Parameter 88

1 Einführung

1.1 Motivation

Rechner werden heutzutage in vielen alltäglichen Geräten als universelles Instrument benutzt. Sie finden Verwendung nicht nur zur Anzeige von Daten, sondern auch zur halbautomatischen oder automatischen Steuerung. Diese Systeme sind meistens für ihre Benutzer unsichtbar in die Geräte integriert[1].

Je mehr Arbeiten diese Systeme übernehmen und je kritischer ihre Anwendungsgebiete sind, um so umfangreicher und komplizierter werden sie. Für die Benutzer dieser Systeme und für Interessierte, wie zum Beispiel Studierende, wird es immer schwieriger, die Funktionsweise dieser Systeme oder Teile von ihnen zu verstehen. Für die Lehre wird es gleichzeitig schwieriger, diese Systeme auf eine geeignete Weise zu präsentieren. Texte und Bilder genügen oft nicht, um deren Funktionsweise anschaulich zu vermitteln. Ein solches System im Betrieb vorzuführen ist selten praktikabel.

Als ein sehr plastisches Beispiel könnte man sich vorstellen, dass in einer Informatik-Vorlesung ein Automobil an eine Wand gefahren wird, um die Kommunikation zwischen Aufprallsensoren und Airbag-Zündung vorzuführen. Die Präsentation ist sicher sehr eindrucksvoll und wird den Studierenden auch im Gedächtnis bleiben. Neben dem mangelnden Platz in einem Hörsaal und der durch den Aufprall entstehenden Lärmbelästigung dürften auch die Reparaturkosten pro Semester kaum tragbar sein. Der Lehreffekt wird dabei verhältnismäßig gering sein. Der eigentliche Anschauungsgegenstand – die Kommunikation, die während des Aufpralls in der Bordelektronik stattfindet – kann bei diesem Versuch nicht direkt beobachtet werden. Die Messung der Signale und deren Sichtbarmachung während des Versuchs ist zwar möglich, doch die Konzepte, die dahinter stehen, werden dadurch nicht erkennbar. Zudem läuft der Versuch unveränderbar schnell ab und ist ohne aufwändige Vorbereitung nicht wiederholbar. Wichtige Details können so sehr leicht übersehen werden.

Der zu betrachtende Vorgang kann trotz seiner umfangreichen und komplizierten Spezifikation einfach genug sein, um mit Bildern und Texten beschrieben zu werden. Eine einfache Beschreibung genügt jedoch oft nicht, um den Inhalt dem Lernenden eingängig zu machen oder ihn zu motivieren, sich selbstständig damit auseinander zu setzen. Die Folge ist, dass die Vermittlung der Inhalte trotz des sorgfältig vorbereiteten Lehrstoffs noch vergleichsweise zeitaufwändig ist. Durch die geringe aktive Einbindung der Studierenden in den Lehrprozess leidet auch die Nachhaltigkeit der vermittelten

[1] Solche Systeme nennt man *Eingebettete Systeme*.

Inhalte. Es existieren oft keine geeigneten Hilfsmittel für aktuelle technische Entwicklungen, um sie – oder für bestimmte Fachgebiete interessante Teilaspekte von ihnen – zu lehren. Ein Lehrmittel, das den Inhalt anschaulich vermitteln hilft, die Studierenden motiviert und ihnen Übungsmöglichkeiten eröffnet, erscheint in solchen Fällen sinnvoll. FlexRay ist eines der Systeme, für die ein Bedarf an anschaulichen Lehrmitteln besteht.

Bei FlexRay handelt es sich um einen so genannten Feldbus (siehe hierzu Abschnitt 2.1) zum Einsatz speziell in Automobilen. FlexRay soll als Ersatz für die verschiedenen Bussysteme dienen, die in Automobilen für unterschiedliche, spezialisierte Aufgaben eingesetzt werden. Gleichzeitig soll FlexRay die zukünftigen Anforderungen an Bussysteme in Automobilen erfüllen können. Dies gilt insbesondere für kritische Aufgaben wie die Fahrzeugkontrolle (X-By-Wire[2]).

Eine zentrale Komponente von FlexRay ist dabei die im FlexRay-Protokoll spezifizierte Medienzugriffskontrolle, also die Vorgehensweise für den geteilten Zugriff auf den Kommunikationsbus. Es würde sicherlich den Rahmen einer Vorlesung sprengen, zur Erläuterung des Buszugriffs jede Einzelheit des Protokolls zu erläutern oder gar ein Automobil mit einem FlexRay-Prototypen vorzuführen. Dennoch ist es im Rahmen einer Vorlesung über Eingebettete Systeme oder Kommunikationssysteme interessant, FlexRay vorzustellen und seine spezielle Medienzugriffskontrolle zu verdeutlichen. Der Lehrende wünscht sich, ein Mittel zur Verfügung zu haben, mit dem ihm eine anschaulichere Präsentation möglich ist, als sie es nur mit Bildern und Texten wäre. Zugleich soll den Studierenden ein Mittel an die Hand gegeben werden, mit dem sie die erläuterten Vorgänge frei im Selbststudium nacharbeiten oder anhand von vorgegebenen Szenarien Übungsaufgaben bearbeiten können.

1.2 Ziel

Ziel dieser Arbeit ist die Konzeption eines Lernprogramms zur Medienzugriffskontrolle beim FlexRay-Protokoll in Form einer so genannten interaktiven Visualisierung (Abschnitt 2.2) sowie eine einfache Realisierung dieses Programms. Der Schwerpunkt liegt hierbei auf der Konzeption und der Gestaltung.

Die Idee ist, die dynamischen Vorgänge bei der Medienzuteilung zu simulieren und darzustellen. Das Programm muss es ermöglichen, dass ein Modell eines FlexRay-Clusters[3] erstellt werden kann. Eine animierte Simulation dieses Modells soll dann die Funktionsweise der Medienzuteilung verdeutlichen. Der Lernende soll mit Hilfe des Programms in der Lage sein, mit nur geringen Vorkenntnissen die Medienzugriffskontrolle beim FlexRay-Protokoll zu untersuchen und sein Verständnis davon zu verbessern.

[2] *X-By-Wire* (engl. X-per-Leitung) bezeichnet elektronische Steuerungen im Gegensatz zu rein mechanischen Steuerungen, insbesondere von Systemen wie Lenkung und Bremsen.

[3] Ein Netz aus FlexRay-Geräten.

Lernziele hierbei sind:

- Fördern des Verständnisses der Funktionsweise der Medienzugriffskontrolle beim FlexRay-Protokoll.
- Gewinn von Erkenntnissen über die Auswirkung von Änderungen bei der Gestaltung von FlexRay-Clustern auf der Ebene der Medienzuteilung.
- Sensibilisieren für die Wichtigkeit von Details bei der Konfiguration eines FlexRay-Clusters und deren Auswirkung auf den Kommunikationsverlauf[4].

Um Dozenten in einer Vorlesung die Möglichkeit zu geben, vorbereitete Modelle zur Veranschaulichung vorzuführen, ist ein Präsentationsmodus vorgesehen.

Es dürfen keine Abhängigkeiten zwischen dem Programm und FlexRay-Hardware bestehen, um den Nutzungsaufwand und die Anschaffungskosten, insbesondere für die Studierenden, gering zu halten. Abhängigkeiten von speziellen Software-Simulationssystemen sollen auch vermieden werden. Auch wenn solche Software-Systeme eventuell für die Lehre erschwinglich sein könnten, sind sie durch ihre Universalität zu umfangreich. Dadurch können sie zu schwierig zu beherrschen sein und lenken durch ihre Funktionsvielfalt oft auch vom eigentlichen Thema ab.

Von technischen Details, die für die Medienzugriffskontrolle nebensächlich sind, soll bei der Konzeption des Programms abstrahiert werden, um dem Lernenden einen leichten Einstieg zu ermöglichen und ihn nicht mit unnötigen Details abzulenken. Ein interaktives Tutorial soll dem Lernenden den Einstieg in die Benutzung des Programms vereinfachen.

Um das Ziel zu erreichen, ist es erforderlich, dass ich folgende Teilaufgaben bearbeite:

- Inhaltlich-fachliche Seite:
  - Beurteilen, welche Einzelheiten aus der FlexRay-Protokollspezifikation relevant sind, um die Funktionsweise der Medienzugriffskontrolle darstellen und simulieren zu können.
  - Feststellen derjenigen Aspekte, die weggelassen werden dürfen, ohne die Richtigkeit der Darstellung zu gefährden.
  - Identifizieren von Möglichkeiten für die Vereinfachung der darzustellenden Zusammenhänge und Erarbeiten einer geeigneten Umsetzung dieser Vereinfachungen.
- Didaktische Seite:
  - Finden geeigneter Methoden für die Gestaltung des Programms. Dies umfasst das Aussehen und das Verhalten des Programms.

[4] Es geht hierbei nur um diejenigen Konfigurationsdetails auf Protokoll- und Geräteebene, die eine Auswirkung auf die Medienzugriffskontrolle haben.

- Feststellen der Bedingungen, die ein solches Programm erfüllen muss, damit es dem Lernenden dabei helfen kann, die angestrebten Lernziele zu erreichen.
- Berücksichtigen der primären Zielgruppe des Programms: Studierende, die eine Vorlesung über Eingebettete Systeme hören.

1.3 Aufbau der Arbeit

In Kapitel 2 gehe ich zunächst auf Grundlagen zum technischen Hintergrund von FlexRay ein und stelle Feldbusse vor, die bereits Einsatz in Automobilen finden. Danach stelle ich pädagogische Konzepte und Erkenntnisse aus dem Bereich des rechnergestützten Lernens vor, die auch in diesem Programm Berücksichtigung finden müssen.

Danach zeige ich in Kapitel 3 Materialien aus der Lehre und der Industrie und diskutiere ihre Eigenschaften.

In Kapitel 4 dokumentiere ich die Konzeption des Programms und beschreibe und begründet die hierzu getroffenen didaktischen und technischen Entscheidungen.

Einen Überblick über die eigentliche Realisierung sowie hierbei aufgetretene Besonderheiten bei der Programmentwicklung gibt Kapitel 5.

Abschließend stelle ich in Kapitel 6 das Ergebnis der Arbeit vor und biete in Kapitel 7 einige Ansätze zur möglichen Weiterführung an.

2 Grundlagen

2.1 Feldbusse und Feldbus-Protokolle

Feldbusse oder Feldbussysteme sind digitale Netze, die in den 1980er Jahren entwickelt wurden, um die in der Industrie häufig gebrauchte analoge parallele Verkabelung von zentralen Steuerungen mit Sensoren[1] und Aktoren[2] – sogenannte *Feldgeräte* – zu ersetzen. Dadurch, dass Feldbusse den Anschluss mehrerer Komponenten an eine Kommunikationsleitung erlauben, reduziert sich der Verkabelungsaufwand gegenüber den früher genutzten Zweipunktverbindungen, bei denen jeder Sensor und Aktor eine eigene Verbindung zum zentralen Steuerungsrechner hatte.

Es wurden Feldbussysteme für verschiedene Einsatzzwecke entwickelt und hergestellt; zum Beispiel für die Fertigungs- und Gebäudeautomatisierung oder für die Fahrzeugtechnik. Die verschiedenen Anwendungsgebiete führten zu einer Vielzahl von Feldbussystemen mit unterschiedlichen elektrischen Eigenschaften und Protokollen.

Um einen Eindruck von Feldbussen in Automobilen zu bekommen und um die besonderen Eigenschaften von FlexRay einordnen zu können, stelle ich zunächst Eigenschaften einiger Feldbussysteme vor und komme danach zu FlexRay selbst.

2.1.1 Controller Area Network (CAN)

Dieses Anfang der 1980er Jahre von Bosch und Intel vorgestellte Feldbussystem war ursprünglich für Kraftfahrzeuge vorgesehen und sollte dort die bestehenden Verkabelungen reduzieren, um Kosten und Gewicht zu sparen und Übersicht zu schaffen. CAN ist auch in der industriellen Regel- und Steuerungstechnik populär geworden. Dies wird auf die einfache Bedienbarkeit, die vergleichsweise geringen Kosten und die frühzeitig erfolgte Standardisierung zurückgeführt [Rei98, Bus96].

Die Datenübertragung ist ereignisgesteuert und verwendet für den geteilten Buszugriff ein CSMA/CA[3]-Verfahren. Hierbei ist ein Bit auf dem Bus dominant. Angenommen, das Bit mit dem binären Wert 0 sei dominant. Wird von einem Knoten[4] eine

[1] Schalter oder Messfühler, die manuelle Steuerungsaktionen oder physikalische Größen in elektrische Signale umwandeln.

[2] Aktoren sind vereinfacht gesagt das Gegenteil von Sensoren und setzen elektrische Steuersignale in Aktionen um. Zum Beispiel Ventile, Motoren, Hydrauliken.

[3] Carrier Sense Multiple Access / Collision Avoidance.

[4] Die an einen Feldbus angeschlossenen Geräte werden oft auch als Steuer- und Feldgerät oder Teilnehmer bezeichnet. Ich wähle die Bezeichnung *Knoten*, die auch als allgemeine Bezeichnung

binäre 0 und von einem anderen Knoten eine binäre 1 auf den Bus gegeben, dann resultiert dies in einem Signal für eine binäre 0 auf dem Bus. Um das auszunutzen, senden die Knoten einen eindeutigen Bezeichner vor der eigentlichen Nachricht und horchen gleichzeitig am Bus. Sobald ein Knoten feststellt, das ein anderes Bit auf dem Bus angekommen ist als das selbst gesendete, stoppt er die Übertragung. Dadurch kann nur ein Knoten seinen Bezeichner komplett übertragen. Dieser Knoten kann danach mit dem Versenden seiner Nachricht fortfahren. Die anderen Knoten, die die Übertragung abgebrochen haben, warten auf die Freigabe des Busses, um danach erneut die Übertragung zu versuchen.

Aufgrund der durch die ereignisgesteuerte Übertragung möglichen Kollisionen und der hierzu notwendigen Kollisionsauflösung ist CAN für harte Echtzeitsysteme[5] nicht geeignet.

Es gibt ein auf CAN aufsetzendes Protokoll TTCAN (Time Triggered CAN – zeitgesteuertes CAN [Rob06b]), das den harten Echtzeitbetrieb durch eine Zeitsteuerung ermöglicht. Hierfür wird ein Referenzknoten als Zeitquelle verwendet. Dieser Knoten sendet regelmäßig eine spezielle Nachricht, die alle anderen Knoten dazu verwenden, um ihre lokale Zeit zu korrigieren. Die Kommunikation selbst findet dann in von dieser Zeit abhängigen, vordefinierten Zeitfenstern statt. In einer Erweiterung von TTCAN, dem so genannten „extension level 2“, können mehrere Referenzknoten als Zeitquelle verwendet werden. Dies vermeidet, dass das gesamte Netz zum Stillstand kommt, falls die einzige Zeitquelle ausfällt.

Die Bitrate bei CAN ist auf 1 MBit/s beschränkt. Dadurch eignet sich CAN nicht besonders gut für datenintensive Anwendungen, wie beispielsweise die Übertragung von Videos.

2.1.2 Media Oriented Systems Transport (MOST)

Das MOST Feldbussystem wurde ab etwa 1997 von der MOST-Cooperation [MOS06b] speziell für die Multimedia-Datenübetragung in Kraftfahrzeugen entwickelt. MOST benutzt Lichtwellenleiter in einer Ring-Topologie. Die Struktur der Kanäle auf einem Medium ist dreigeteilt: Es sind gesonderte Kanäle für synchrone und asynchrone Datenübertragung, sowie für Kontrolldaten (zur Ansteuerung der synchronen und asynchronen Kanäle) vorgesehen.

Ein Knoten übernimmt die Rolle eines Timing-Masters und liefert die globale Zeit für alle Knoten. Diese Abhängigkeit stellt einen Schwachpunkt dar, denn die Funktion des gesamten Netzes kommt mit dem Ausfall des Timing-Masters zum Erliegen. Obwohl pro MOST-Ring genau ein Timing-Master vorgesehen ist [Vie03] und bei

für ein an einem Netzwerk angeschlossenes Gerät dient und im Gegensatz zum Wort *Teilnehmer* nicht mit dem Bediener des Gerätes verwechselt werden kann.

[5] In *harten Echtzeitsystemen* sind zeitliche Vorgaben zwingend einzuhalten, da ansonsten erheblicher Schaden entstehen kann. Beispiel: Es ist nicht akzeptabel, wenn ein Bremsimpuls erst Sekunden später die Bremsen betätigt.

den Herstellern meist von „dem Timing-Master“ die Rede ist (beispielsweise Vector Informatik [Vec06b]), wird auf der MOST-Webseite angedeutet, dass mehrere Timing-Master unterstützt werden. Diese Unterstützung besteht darin, ein MOST-Netz aus mehreren MOST-Ringen zusammenzusetzen, von denen jeder einen Timing-Master hat [Whe06].

Die synchronen Kanäle sind für kontinuierliche Datenübertragung vorgesehen, wie dies beispielsweise bei Video- oder Audiodatenströmen benötigt wird. Um einen kontinuierlichen Datenfluss zu gewährleisten, werden die synchronen Kanäle vermittelt, ähnlich der Funktionsweise des öffentlichen leitungsvermittelten Telefonnetzes. Dadurch entfällt nach dem Aufbau der Verbindung die Übertragung von Kontrolldaten. Die asynchronen Kanäle sind für die schnelle Übertragung von nicht kontinuierlichen Daten gedacht, wie Bilder, Navigationsdaten oder Internet-Datenverkehr. Der Zugriff auf die asynchronen Kanäle findet paketweise statt.

Details über die Buszuteilung bei MOST-Bussen werden trotz vorhandener öffentlicher Spezifikation nicht bekannt gemacht. Die Spezifikation enthält nur Hinweise, dass die Buszuteilung nach einem doppelten Verfahren automatisch vom MOST Netzwerkcontroller auf eine faire, echtzeitfähige Weise durchgeführt wird ([MOS05], Seiten 116, 118, 173).

2.1.3 Time Triggered Protocol (TTP)

TTP ist ein von der TTTech Computertechnik AG [TTT06b] in Zusammenarbeit mit der TU Wien und weiteren nicht öffentlich genannten Partnern entwickeltes Protokoll für harte Echtzeitsysteme, das nicht speziell auf den Einsatz mit Feldbussen festgelegt ist[6]. Es gibt verschiedene TTP-Systeme auf dem Markt, die eine Netto-Datenrate von bis zu 25 MBit/s bieten. In Laborexperimenten wurden erfolgreich Versuche mit 1 GBit/s durchgeführt [TTT04].

Die Steuerung des Medienzugriffs bei TTP ist ein Zeitmultiplexverfahren (*Time Division Multiple Access*, TDMA). Jeder Knoten bekommt vor Inbetriebnahme des Netzes Zeitschlitze zugewiesen, die er in jedem Kommunikationszyklus benutzen darf. Um die für diese Art der Zugriffskontrolle benötigte globale Zeit kümmert sich ein verteilter, fehlertoleranter Algorithmus, der ständig die Uhren der Knoten synchronisiert.

Ein unabhängiger Buswächter garantiert bei TTP, dass auch tatsächlich immer nur ein Knoten gleichzeitig sendet.

Die TTP-Spezifikation ist zwar kostenlos verfügbar, doch ist ihr Besitz an mehrere Auflagen gebunden, wie zum Beispiel die ausschließliche Nutzung des Dokuments für interne Informationszwecke innerhalb des Unternehmensteils, zu dem der Empfänger gehört oder, dass ohne die schriftliche Genehmigung der Firma TTTech keine Hard-

[6] Der Beginn der Entwicklung war nicht genau in Erfahrung zu bringen. Auf der Webseite des Herstellers befand sich im April 2006 die Aussage, dass das Protokoll seit über 25 Jahren entwickelt wird.

ware gebaut wird, die die Spezifikation erfüllt [TTA06a, TTT06a]. Derartige Restriktionen sollten bei der Entwicklung von FlexRay (Siehe Abschnitt 2.1.5) von Anfang an ausgeschlossen werden.

2.1.4 byteflight

byteflight [BMW06a] wurde von BMW, Motorola, ELMOS, Infineon, Siemens EC, Steinbeis Transferzentrum für Prozessautomatisierung und IXXAT ab 1996 entwickelt. Es bestand nach Aussage von BMW Bedarf an einem Bussystem mit einem deterministischen Protokoll für sicherheitsrelevante Anwendungen, das hohe Datenraten erlaubt und fehlertolerant ist [BMW06b]. Die zu diesem Zeitpunkt verfügbaren CAN- und TTP-Systeme erfüllten diese Anforderungen noch nicht. Um elektromagnetische Störungen auf dem Bus zu vermeiden, wurde ein optisches Übertragungsmedium für byteflight gewählt.

Bei byteflight wird der Medienzugriff zeitgesteuert kontrolliert. Anstelle eines einfachen Zeitmultiplex wird hier ein Verfahren benutzt, bei dem die tatsächlichen Sende-Zeitschlitze aus kleineren Zeitschlitzen zusammengesetzt werden (*flexible time division multiple access*, FTDMA). Dieses Verfahren ist auch ein Teil der Medienzugriffskontrolle von FlexRay. Ich werde es dort, auf Seite 12, genauer erläutern. Grundlage für die Zeitsteuerung bei byteflight ist ein beliebiger als SYNC-Master konfigurierter Knoten im byteflight-Netz. Er erzeugt den für alle Knoten gültigen Takt, indem er entsprechende Synchronisierungspulse generiert. Der Abstand zwischen zwei Synchronisierungspulsen ist ein Kommunikationszyklus. Um Gesamtausfälle des Systems durch Ausfall des SYNC-Masters zu vermeiden, wird eine Übernahmestrategie vorgeschlagen [GBP00]. Ein bestimmter Knoten kann bei Fehlen des SYNC-Pulses die Funktion des SYNC-Masters übernehmen. Wie vorgegangen wird, falls nur zufällig diesem Knoten der SYNC-Puls gefehlt hat, oder was passiert, wenn der Master nur einen zeitweiligen Ausfall hatte, wird nicht beschrieben. Als Alternative wird die vollständige Redundanz vorgeschlagen, also der parallele Betrieb zweier identischer byteflight-Netze.

2.1.5 FlexRay

FlexRay wird vom FlexRay-Konsortium entwickelt [Fle06b]. Dem FlexRay-Konsortium gehören Hersteller aus verschiedenen Bereichen der Automobilbau- und -zulieferindustrie an. Kernmitglieder sind BMW; Bosch; DaimlerChrysler; Freescale Semiconductor, Inc.; General Motors; Philips und die Volkswagen AG. Das FlexRay-Konsortium hat sich zum Ziel gesetzt, die in FlexRay verwendeten Techniken für jedermann verfügbar zu halten und FlexRay in der Praxis als herstellerübergreifenden Standard für die Automobilvernetzung durchzusetzen.

Das Ziel von FlexRay ist es, als Ersatz für die bestehenden Feldbus-Systeme in Automobilen zu dienen und gleichzeitig für zukünftige Anwendungen geeignet zu sein.

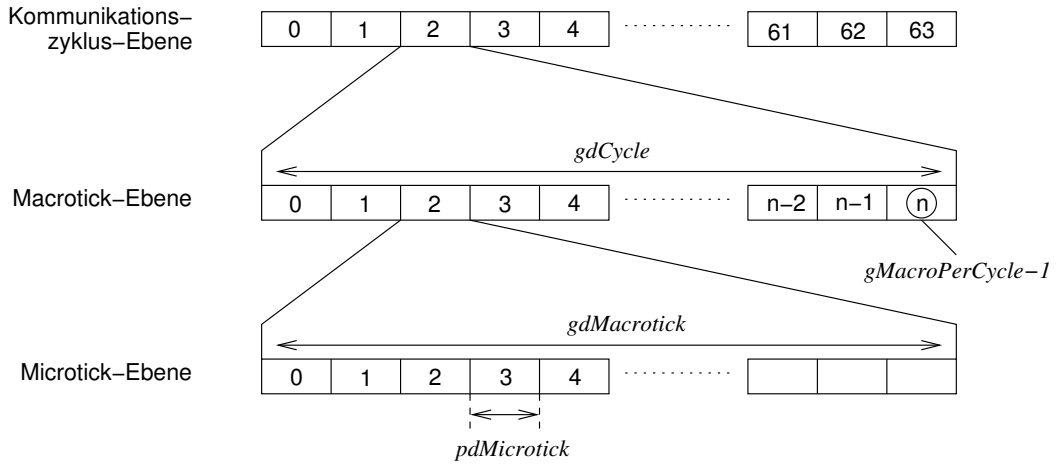


Bild 2.1: Zeitzählerstruktur bei FlexRay

FlexRay unterstützt hohe Datenraten, deterministische Kommunikation zur Erfüllung harter Echtzeitbedingungen, Redundanz und Fehlertoleranz. Ein FlexRay-Cluster kann einkanalig (Kanal A) oder zweikanalig (Kanäle A und B) gebaut und betrieben werden. Ein Kanal bezeichnet hierbei einen Hardware-Kanal, also ein eigenes Medium oder zu einem Netz zusammengeschaltete Medien. Diese Medien müssen den Anforderungen der FlexRay Electrical Physical Layer Specification [Fle05b] genügen. Die FlexRay-Protokollspezifikation schreibt nicht vor, wie der optionale zweite Kanal verwendet werden soll. Beispielhaft gibt sie jedoch an, dass der zweite Kanal verwendet werden kann, um durch Redundanz die Ausfallsicherheit zu erhöhen. Eine Anwendung zur besseren Ausnutzung der Kanalkapazität ist jedoch auch möglich: Ein Knoten, der nur an Kanal A angeschlossen ist und ein Knoten, der nur an Kanal B angeschlossen ist, dürfen in den selben Zeitschlitzen (*Slots*) senden. Auf diese Weise können beide Knoten gleichzeitig statt hintereinander Daten übertragen.

Die Medienzugriffskontrolle bei FlexRay ist zeitgesteuert und vereint in sich Eigenschaften der TTP- und byteflight-Protokolle. Eine Beschreibung der FlexRay-Medienzugriffskontrolle gebe ich im nächsten Kapitel.

Grundlage für die Zeitsteuerung ist das Vorhandensein einer globalen Zeit. Um diese zu ermitteln und von den instabilen Oszillatoren der Knoten zu entkoppeln, wurde für FlexRay eine dreistufige Zeitzähler-Hierarchie entworfen (Bild 2.1). Auf unterster Ebene sind die *Microticks* (μT), eine Zeiteinheit, die direkt aus den Oszillatoren der Knoten ermittelt wird und daher für jeden Knoten unterschiedlich sein kann. Aus den Microticks berechnet jeder Knoten die *Macroticks* (MT), so dass die Macrotics clusterübergreifend innerhalb vorgegebener Toleranzen gleich lang sind (*gdMacrotick*[7] im Bild). Die knotenspezifische Anzahl der Microticks pro Macrotick hängt von der Dau-

[7]Die hier genannten FlexRay-Parameter habe ich in Anhang B zusammengefasst.

er der Microticks (*pdMicrotick* im Bild) und damit von der Stabilität des Oszillators des Knotens ab. Jeder Macrotick eines Knotens kann daher aus einer unterschiedlichen Anzahl an Microticks zusammengesetzt sein. Aus einer konfigurierbaren Anzahl an Macroticks (*gMacroPerCycle*) wird dann ein Kommunikationszyklus aufgebaut. Die Macroticks werden während eines Zyklusses von 0 bis *gMacroPerCycle-1* durchgezählt. Die Knoten zählen die Kommunikationszyklen von 0 bis 63 (*gCycleCounterMax*) und beginnen danach wieder mit 0.

Um die zeitgesteuerte Kommunikation durchführen zu können, benötigen die Knoten[8] eine gemeinsame Zeit (oder *einen gemeinsamen Takt* – die Begriffe werden teilweise synonym verwendet). Anders als bei MOST, TTCAN oder byteflight ist bei FlexRay von vorneherein eine redundante Ermittlung einer globalen Zeit eingeplant worden. Diese Zeit wird bei FlexRay mit Hilfe eines fehlertoleranten, verteilten Algorithmus nach Welch und Lynch [WL88] ermittelt. Jeder Knoten synchronisiert sich selbst mit dem Cluster. Dazu verarbeiten sie die auf den Bus gesendeten Synchronisierungssnachrichten der anderen Knoten, aktualisieren damit ihre Zeitwerte und senden selbst eine Synchronisierungsnachricht. Jeder Knoten hat dadurch eine eigene Sicht auf die globale Zeit. Der Algorithmus stellt dabei sicher, dass die Macroticks über alle Knoten innerhalb festgelegter Grenzen synchron und gleichmäßig sind. Durch diese verteilte aktive Ermittlung der globalen Zeit besteht keine Gefahr, dass das gesamte System fehlerhaft arbeitet oder zum Stillstand kommt, falls Taktgeber-Knoten ausfallen.

FlexRay im Einsatz

Der erste Einsatz von FlexRay in einem Fahrzeug ist für den neuen BMW X5 vorgesehen [Bay06][9]. In [Völ06] wird beschrieben, wie beim optional erhältlichen Fahrwerkregelsystem „Adaptive Drive“ FlexRay verwendet wird, um die Steuergeräte der regelbaren Stoßdämpfer und die zentrale Steuereinheit zu vernetzen. Laut BMW sind leider keine veröffentlichungsfähigen Informationen über Details der FlexRay-Protokollnutzung beim Adaptive Drive verfügbar.

2.1.6 Medienzugriffskontrolle beim FlexRay-Protokoll

Die Medienzugriffskontrolle bei FlexRay besteht aus zwei Teilen. Ein Teil sind die Vorgaben darüber, wie ein Knoten die Medienzugriffskontrolle handhaben soll und wird in Kapitel 5 der FlexRay-Protokollspezifikation beschrieben. Der zweite Teil der

[8] Nach der FlexRay-Protokollspezifikation ist hier der Kommunikationscontroller gemeint, der im Knoten eingebaut ist. Für die bessere Lesbarkeit verwende ich hier und im Folgenden den Begriff *Knoten* auch für Komponenten des Knotens, weil die Unterscheidung im Rahmen dieser Arbeit nicht relevant ist.

[9] Eine direke Angabe der Webseite mit den Informationen zum neuen BMW X5 ist aufgrund der Verwendung von JavaScript auf [Bay06] leider nicht möglich.

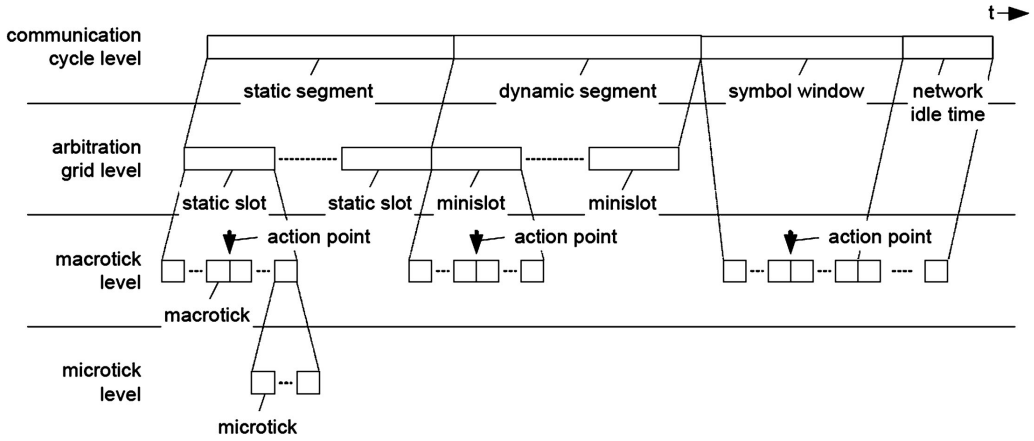


Bild 2.2: Zeitvorgaben im FlexRay-Kommunikationszyklus (Quelle: [Fle05c])

Medienzugriffskontrolle ist ihre Konfiguration, die die Entwickler beim Entwurf eines konkreten Clusters festlegen. Ohne eine sorgfältige Konfiguration können die im Rahmen der Protokollspezifikation beschriebenen Techniken nicht richtig funktionieren. Details zur Konfiguration erläutere ich ab Seite 13.

Knotenverhalten und Kommunikationszyklusstruktur

Grundlage für die Medienzugriffskontrolle bildet der auf der Zeitzählerstruktur aus Bild 2.1 aufbauende, wiederkehrende Kommunikationszyklus. Bild 2.2 zeigt, wie die zusätzlichen Kontrollelemente zur Buszuteilung in diese Hierarchie eingebettet sind.

Dazu ist auf oberster Ebene der Kommunikationszyklus und seine Aufteilung in statisches Segment (*static segment*), dynamisches Segment (*dynamic segment*), Symbolfenster (*symbol window*) und Ruhephase (*network idle time*, NIT) dargestellt. Darunter ist in der Ebene der Buszuteilung (*arbitration grid level*) der Aufbau dieser Segmente detaillierter dargestellt: Das statische Segment ist seinerseits in statische Slots (*static slots*) unterteilt, und das dynamische Segment besteht aus *Minislots*. Alle Slots werden von Beginn bis Ende des Zyklusses unterbrechungsfrei durchgezählt. Das bedeutet, dass der erste Minislot im dynamischen Segment eine Slotnummer hat, die um eins höher ist, als die Nummer des letzten statischen Slots. Wie bei den Macroticks beginnt mit jedem Zyklus auch die Nummerierung der Slots wieder mit Null. Das Symbolfenster und die NIT sind nicht weiter unterteilt. Die beiden Ebenen darunter sind dieselben wie bei der Zeitzählerstruktur. Zur Erinnerung: Die Macroticks repräsentieren die globale Zeit in einem FlexRay-Cluster, während die Microticks knotenlokale Zeiteinheiten darstellen.

Die Dauer des Kommunikationszyklusses ist die Summe aus den Dauern seiner Seg-

mente. Die Segmentdauern werden vom Entwickler des Clusters festgelegt und bleiben während der Laufzeit des Clusters auf Macrotick-Ebene gleich.

Das statische Segment dient zur Übertragung zeitkritischer Daten. Um die Anforderungen für Echtzeitsysteme zu erfüllen, wird dort ein TDMA-Verfahren wie bei TTP (Abschnitt 2.1.3) zur Medienzugriffskontrolle verwendet. Die Dauer der Nachrichtenübertragung und der Zeitpunkt, zu dem eine bestimmte Nachricht versendet wird, sind durch die vorherbestimmte Konfiguration festgelegt. Die statischen Slots dürfen eine Länge zwischen 4 und 661 Macroticks haben.

Die in Bild 2.2 außerdem erkennbaren *action points* sind die Zeitpunkte, zu denen ein FlexRay-Knoten seinen Sender anweisen soll, die Übermittlung eines Rahmens zu beginnen. Die action points der statischen Slots dienen zusätzlich als Referenzzeitpunkte für die Uhrensynchronisierung.

Zur Realisierung der Slot-Zuordnung bei diesem TDMA-Verfahren legt die Spezifikation fest, dass in einem Slot nur ein Rahmen übertragen werden darf, dessen Rahmenbezeichner (*Frame ID*) mit der Nummer des Slots übereinstimmt. Die Knoten sorgen dafür, dass die Rahmen die passenden Bezeichner enthalten. Damit dies gelingt, zählt ein Knoten die Slots mit, indem er einen internen Zähler immer dann um eins erhöht, wenn er das Ende eines Slots auf dem Bus festgestellt hat. Endet der Kommunikationszyklus, setzt der Knoten seinen internen Slot-Zähler wieder auf Null zurück.

Das dynamische Segment ist für die Übertragung von Daten für Komfortfunktionen vorgesehen. Solche Daten sind nicht zeitkritisch (zum Beispiel Steuerdaten für den Fensterheber) und können auch variable Anforderungen an die Kanalkapazität haben (beispielsweise Bilddaten für ein Navigationssystem). Im dynamischen Segment findet die Medienzugriffskontrolle wie in byteflight (Abschnitt 2.1.4) nach dem FTDMA-Schema statt. Die hierfür verwendeten Minislots können eine Länge von 2 bis 63 Macroticks haben. Aus diesen Minislots setzen die Knoten während der Kommunikation dynamische Slots (*dynamic slots*) zusammen, das sind die in diesem Segment zur Kommunikation verwendeten Zeitschlitze variabler Länge. Solange kein Knoten sendet, werden im dynamischen Segment die Minislots als Slots gezählt. Wenn ein Knoten an der Reihe ist, dann sendet er einen Rahmen, dessen Länge zwischen einem Minislot und der noch verfügbaren Anzahl an Minislots liegt. Die Behandlung der daraus eventuell resultierenden Fragmentierung der Daten wird im FlexRay-Protokoll nicht spezifiziert und muss auf höherer Ebene realisiert werden. Während ein Knoten sendet, steigt aus Sicht der anderen Knoten die Länge des dynamischen Slots schrittweise um die Länge eines Minislots an. Dadurch verschiebt sich das Ende des Slots und die Knoten zählen entsprechend später ihren Slot-Zähler hoch. Auf diese Weise kann es durchaus passieren, dass ein Knoten nicht in einem Kommunikationszyklus senden kann, obwohl Daten zur Übertragung vorhanden sind. Andere Knoten, die vor ihm im selben Zyklus senden, können die gesamte Sendedauer verbrauchen. Diese Eigenschaft erlaubt es dem Entwickler, die Zugriffe auf das dynamische Segment zu priorisieren: Eine niedrigere Slot-Nummer bedeutet eine höhere Priorität.

Nach dem dynamischen Segment folgt ein optionales, bis zu 142 Macroticks langes, Symbolfenster (*symbol window*), in dem ein nicht näher spezifiziertes Symbol übertragen werden kann. Eine Medienzugriffskontrolle durch das FlexRay-Protokoll ist für das Symbolfenster nicht vorgesehen und muss bei Bedarf auf einer höheren Ebene implementiert werden.

Die NIT bildet den letzten Teil im Kommunikationszyklus, mit einer Dauer zwischen 2 und 805 Macroticks. In der NIT findet keine Kommunikation statt. Die Knoten nutzen diese Zeit, um interne Berechnungen durchzuführen, zum Beispiel, um Korrekturwerte für die Synchronisierung zu ermitteln. Die NIT spielt bei der Synchronisierung auch noch eine weitere Rolle: Innerhalb der NIT wird der zeitliche Versatz zwischen den einzelnen Knoten korrigiert. Dazu ist in der NIT ein Bereich definiert (*offset correction segment*), der von jedem Knoten entsprechend der errechneten Korrekturwerte um einige Microticks verlängert oder verkürzt wird.

Konfiguration eines FlexRay-Clusters

Die Konfiguration des FlexRay-Clusters ist für die Medienzugriffskontrolle wichtig, da sich einige der Parameter direkt auf sie auswirken. Durch die Konfiguration legt der Entwickler vor Inbetriebnahme des Clusters fest, wie dessen Kommunikationsverhalten ist. Unter den Parametern befinden sich clusterglobale und knotenlokale Parameter, die das Verhalten des Clusters auf der Ebene des Protokolls und des physischen Medienzugriffes bestimmen. Dabei gelten für die Parameter bestimmte Einschränkungen und Abhängigkeiten, die in [Fle05c, Anhänge A und B] beschrieben werden. Beispielsweise ist die minimale Anzahl an Microticks pro Macrotick mit 20 fest vorgegeben [Fle05c, Tabelle A-1], oder die maximale Anzahl der Minislots im dynamischen Segment hängt von der Übertragungsgeschwindigkeit des verwendeten Mediums ab: Bei 2,5 MBit/s darf es maximal 3977 Minislots geben, bei 5 MBit/s bereits 7977 und bei 10 MBit/s sind 7986 erlaubt [Fle05c, Tabelle B-19].

Für die Medienzugriffskontrolle sind nur einige der dort angegebenen Parameter von Bedeutung. Neben der Definition der Struktur des Kommunikationszyklusses besteht die eigentliche Konfiguration der Medienzugriffskontrolle darin, die Zuordnung von Knoten, Slots und Kanälen konfliktfrei zu gestalten. Die Zuordnung zwischen Kanal und Knoten ergibt sich daraus, an welchen Kanälen die Knoten angeschlossen werden. Bei der Slot-Zuordnung muss der Entwickler dann darauf achten, dass er keine Doppelbelegungen innerhalb eines Kanals vornimmt. Eine Doppelbelegung führt immer zu einer nicht auflösbaren Kollision auf dem Bus.

Weiterhin wird mit der Konfiguration die Priorisierung im dynamischen Segment festgelegt. Der Entwickler ist verantwortlich dafür, dass keine Zuordnungen vorgenommen werden, bei denen Knoten niemals senden können. In einem konkreten Cluster kann es zudem wichtig sein, dass er die Prioritäten für im dynamischen Segment sendende Knoten sehr genau plant, damit sie genügend Übertragungszeit für ihre ordnungsgemäße Funktion eingeräumt bekommen. Dies kann insbesondere dann schwierig

werden, wenn die Anzahl der Knoten steigt, die im dynamischen Segment Nachrichten variabler Länge senden werden.

2.1.7 Buswächter

Für FlexRay sind – ähnlich wie in TTP (siehe Seite 7) – Buswächter (*bus guardians*) vorgesehen. Sie sollen sicherstellen, dass keine Knoten außerhalb ihres Zugriffsplans auf den Bus senden können. Die Buswächter waren ursprünglich in der FlexRay-Protokollspezifikation verankert, werden in der momentan aktuellen Version [Fle05c] jedoch nicht mehr berücksichtigt, weil ihre Spezifikation [Fle05a] noch nicht abgeschlossen ist.

2.2 Komponenten einer interaktiven Visualisierung

In diesem Abschnitt gebe ich einen Überblick über didaktische und gestalterische Grundlagen, die bei der Entwicklung eines rechnergestützten Lehrmittels wichtig sind. Danach vertiefe ich den Begriff der Interaktivität und nenne Beispiele zu ihrer Umsetzung. Als Mittel zur Gestaltung von Interaktivität führe ich danach kurz in die rechnergestützte Simulation ein. Mit diesen Grundlagen erläutere ich abschließend, was eine interaktive Visualisierung ist.

2.2.1 Was sind Lernprogramme?

Für Lehrmittel, die irgendwie mit der Hilfe von Rechnern benutzt werden, kommen in der Literatur verschiedene Begriffe vor, die oft den Zusatz „Multimedia“ enthalten. Obwohl dies hauptsächlich bedeutet, dass solche Lehrmittel aus mehreren Medien wie Film, Bild, Ton, Animation und Text bestehen, ist meistens auch gemeint, dass diese Lehrmittel „digital“ sind, da der Begriff „Multimedia“ gerade im Zusammenhang mit Rechnern populär wurde. Kerres [Ker01] wählte für rechnergestützte Lehrmittel die Bezeichnung „Multimediale und telemediale Lernumgebungen“[10] für digitale Medien, die in der Lehre eingesetzt werden sollen. In [RM05] werden Überlegungen zu diesen Lehrmitteln unter dem Titel „Didaktik und Neue Medien“ zusammengefasst und in [IP02] ist schlicht nur von „Multimedia“ die Rede, weil hierin auch die Bedeutung von Multimedia als die Kombination mehrerer, nicht notwendigerweise computerbasierter, Medien verstanden wird. Die Programmierung, die der Begriff Lernprogramm ausdrückt, wird dabei nicht immer als die Implementierung von Algorithmen in einer Programmiersprache verstanden, sondern als strukturierte Aneinanderreihung von Inhalten und deren Anreicherung mit Interaktionsmöglichkeiten, zum Beispiel unter Verwendung von Multimedia-Autorensystemen [IP02, Seite 222-240], [Ker01, Teil D].

[10] Eine *Lernumgebung* umfasst die äußeren Bedingungen des Lernens, wie zum Beispiel die verwendeten Lehrmittel, den Ort an dem das Lernen stattfindet oder die Voraussetzungen des Lernenden.

Ein Grund für die verschwommene Begriffssituation ist die universelle Nutzbarkeit des Rechners. Er ist gut als Werkzeug zur Erzeugung und Präsentation von multimedialen Inhalten geeignet, wodurch auch die Grenze zwischen digitalen und klassischen Medien verschwimmt. Die digitalen Äquivalente zu den klassischen Medien haben dieselben Namen: Film, Bild, Ton, Text. Eine weitere Schwierigkeit bei der Begriffsbestimmung ist die mit der Herstellung von rechnergestützten Lehrmitteln verbundene Interdisziplinarität. So schreibt Niegemann in [NHHM+04]:

> *E-Learning ist ein weites Feld, es lässt sich keiner klassischen akademischen Disziplin alleine zuordnen: Informatik, pädagogische Psychologie, Didaktik und Grafikdesign liefern wichtige Beiträge.*

Er verwendet dabei *E-Learning* als Oberbegriff für die rechnergestützten Lehrmittel. Kritzenberger schreibt in [Kri05], dass dieser Begriff eigentlich ein Modebegriff sei, und sich seit etwa dem Jahr 2000 international etabliert hat. E-Learning umfasst dabei unter Anderem telemediale Lernangebote (Lernen über das Internet), Hypertextsysteme, Anschauungsobjekte aus digitalen Medien (zum Beispiel Präsentationen oder Animationen) und auch Lernprogramme mit dem heute gebräuchlichen Oberbegriff „Lernsoftware“.

Bei Lernprogrammen im klassischen Sinn geht es darum, dem Lernenden Inhalte zu vermitteln und diese gelernten Inhalte durch Wiederholung und Übung zu festigen, so wie dies in einer typischen Lehrer-Schüler-Unterrichtssituation der Fall ist. Ein Hauptziel von Lernprogrammen ist es, den Lehrer zu ersetzen, insbesondere bei sich wiederholenden Lehrtätigkeiten [Ker01, RM05, NHHM+04, GWGK05]. Das Wort Lernprogramm wird heute aber auch schon für Programme gebraucht, die den Lehrer bei der Arbeit unterstützen und nur unter seiner Mitwirkung eingesetzt werden können. Ein Beispiel hierfür sind Programme, die zwar Aufgaben präsentieren und Antworten abfragen, es aber dem Lehrer überlassen, die Richtigkeit der Ergebnisse zu überprüfen.

Bei der Diskussion darüber, was ein solches Programm noch leisten muss, werden eine oder mehrere lernpsychologische Richtungen berücksichtigt. Kritzenberger geht in [Kri05] auf Behaviorismus, Kognitivismus und Konstruktivismus ein (Abschnitt 2.2.2), und betont, dass die verschiedenen Sichten sich nicht ausschließen müssen. Robert M. Gagné vertrat eine ähnliche Ansicht [GWGK05]. Für ihn gab es nicht die Frage nach der einen richtigen Lernmethode. Sein Konzept des Instruktionsdesigns (*instructional design*, ID) baut darauf auf, für jede Lernsituation die Umgebung und geeigneten Methoden zu wählen und zu kombinieren.

2.2.2 Behaviorismus, Kognitivismus und Konstruktivismus

Im Folgenden erläutere ich die drei lernpsychologischen Richtungen *Behaviorismus*, *Kognitivismus* und *Kontruktivismus* und beschreibe anhand von Beispielen, wie Er-

kenntnisse aus diesen Richtungen in die Entwicklung von Lehrmitteln, insbesondere rechnergestützten Lehrmitteln, eingehen können.

Behaviorismus

Der Behaviorismus ist eine Theorie der Verhaltensforschung (amerikanisch-englisch: *behavior* = Verhalten). Grundlage des Behaviorismus ist beobachtbares Verhalten und die Beeinflussbarkeit dieses Verhaltens durch Konditionierung. Mit der klassischen Konditionierung, wie sie von Iwan Pawlow beschrieben wurde, hat das nur noch bedingt zu tun. Pawlow's Experiment mit seinem Hund ist wohl eines der bekanntesten Beispiele für die klassische Konditionierung. Dem Hund wurde ein neutraler Reiz (ein Klingelton) präsentiert und in zeitlichen Zusammenhang mit einem biologisch signifikanten, unkonditionierten Reiz (die Fütterung) gestellt. Die Reaktion des Hundes auf die Fütterung (Speichelfluss) wurde durch wiederholte Präsentation des neutralen Reizes auf diesen übertragen, so dass der Speichelfluss bereits bei der Präsentation des Klingeltons einsetzte. Der Hund hat „gelernt", dass der Klingelton etwas mit Futter zu tun hat. Skinner hat den Begriff der operanten Konditionierung eingeführt und darauf aufbauend den programmierten Unterricht beschrieben [Kri05, Kapitel 1], [NHHM+04, Kapitel 14]. Die operante Konditionierung findet nicht mehr durch die Paarung eines neutralen und eines biologisch signifikanten unkonditionierten Reizes statt, sondern mit Verstärkung durch Feedback[11]. Das bedeutet beispielsweise, dass der Lernende bei einer richtigen Antwort auf eine Frage eine Belohnung erhält. Dadurch erhöht sich die Wahrscheinlichkeit, dass unter ähnlichen Umständen dieselbe richtige Antwort gegeben wird. Der programmierte Unterricht nutzt dies aus, indem er die Umstände konstant hält. Der Lernende bekommt bei jeder Wiederholung denselben Lernstoff in unveränderter Reihenfolge präsentiert. Die Aufgabe des Lehrers besteht dabei darin, die für die Lernsituation optimale Reihenfolge der zu vermittelnden Inhalte festzulegen[12] und geeignete Feedbacks zu entwickeln.

Behavioristisch orientierte Lehrmittel sind daher auf die gleichförmige Wiederholung von Inhalten und deren Verstärkung durch geeignetes Feedback ausgelegt. Der programmierte Unterricht kann direkt mit der Hilfe von Rechnern umgesetzt werden, ohne auf besondere Eigenschaften digitaler Medien Rücksicht nehmen zu müssen. Visuelle Aspekte der Gestaltung treten gegenüber der Gestaltung der Funktionaliät in den Hintergrund. Da es kein absolutes Maß für die Güte von Feedback gibt, liegen die ergriffenen Maßnahmen im Ermessen des Lehrers und hängen im Detail von der

[11] Rückmeldung an den Lernenden. Ich verwende jedoch weiterhin das Wort *Feedback*, da es im pädagogischen Kontext gebräuchlich ist.

[12] Obwohl ich keine konkrete Aussage in der Literatur darüber finden konnte, gehe ich davon aus, dass auch die zufällige Reihenfolge optimal sein kann. Beispiele hierfür sind Vokabeltrainer, die als klassisches Modell der behavioristischen Lernprogramme stehen. Sie geben die abzufragenden Phrasen und Wörter oft in zufälliger Reihenfolge vor. Die eigentliche Umgebung bleibt dabei jedoch konstant.

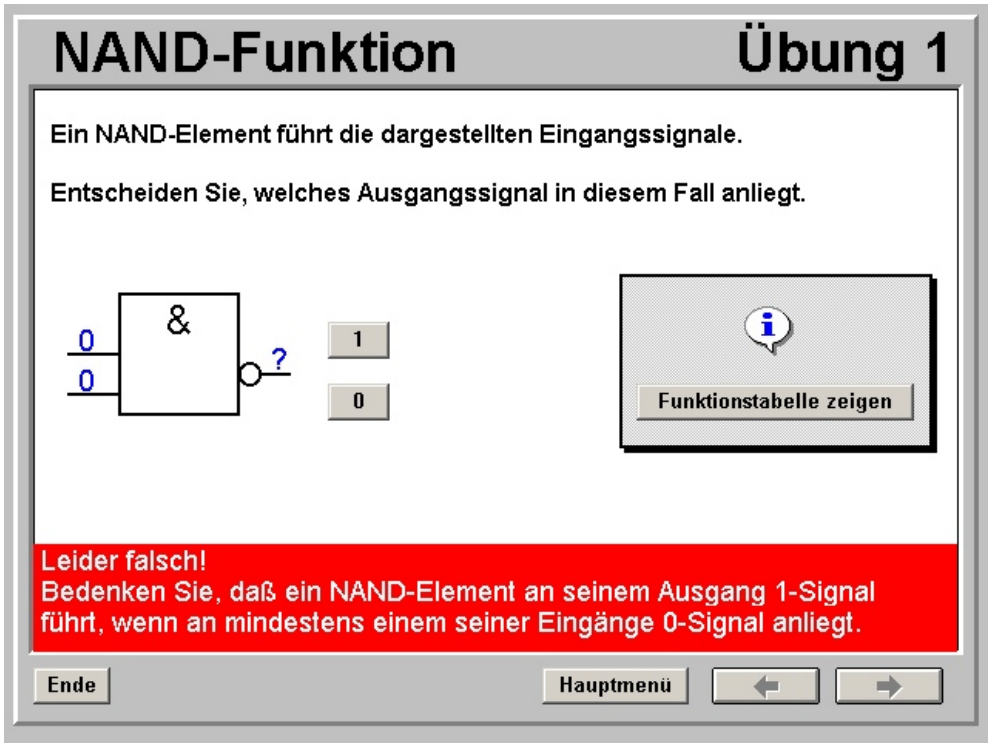


Bild 2.3: NAND-Übung: Feedback „Falsch“ (Quelle: [Lüt04])

individuellen Lehrsituation ab. Euler [Eul92] und Niegemann [Nie01] haben einige grundsätzliche Eigenschaften für Feedback formuliert:

- Keine zu einfachen Rückmeldungen geben (Nur „richtig“ oder „falsch“ ist nicht ausreichend)
- Bei Fehlern ist mindestens die korrekte Antwort mit anzugeben, besser noch eine Erklärung (Bild 2.3)
- Wünschenswert wäre ein Feedback auf Grundlage einer Fehleranalyse
- Feedback sollte sachlich sein und frei von Beschimpfung und Herabsetzung
- Vermittlung einer realistischen Erfolgszuversicht
- Lob für Lernfortschritte anhand des Verlaufes des bisherigen Lernvorganges, jedoch kein überschwängliches Lob für simple Lösungen

Die Erfüllung dieser teilweise selbstverständlich erscheinenden Grundsätze kann mitunter schwierig sein. Beispielsweise kann eine Fehleranalyse für gegebene Antworten sehr komplex werden, wenn ein Lernprogramm frei formulierbare Antworten zulässt.

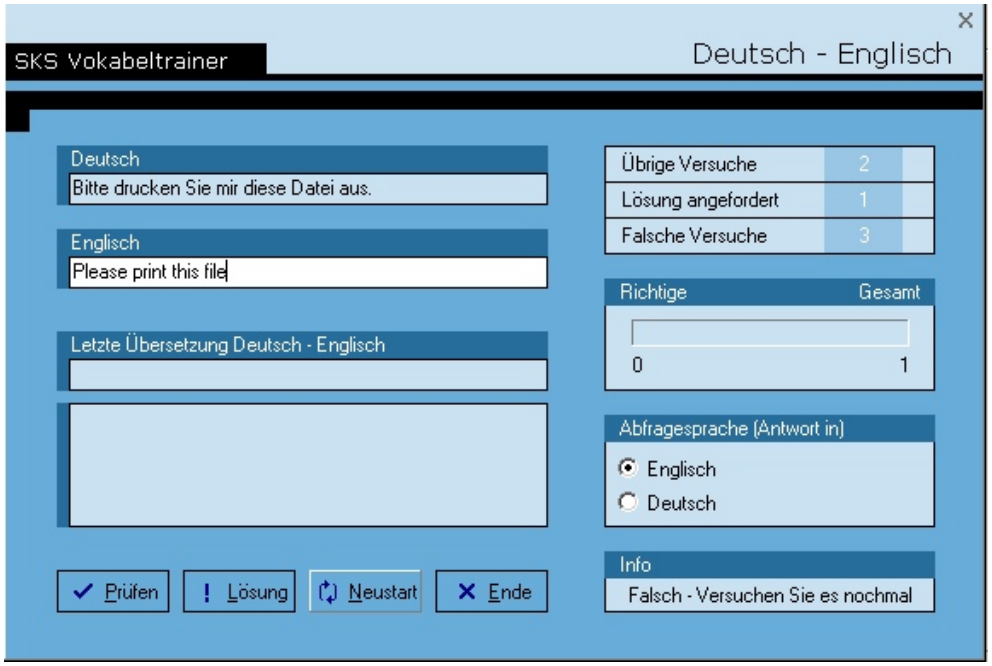


Bild 2.4: Vokabeltrainer: Feedback „Falsch“ (Quelle: [Kes04])

Bilder 2.4 und 2.5 zeigen dies. Obwohl die gegebene Antwort sachlich richtig ist, hat das Programm eine leicht andere Antwort erwartet und reagiert nur mit einer Variation des simplen „falsch“.

Kognitivismus

Die Wahrnehmung (*Kognition*) und ihre Verarbeitung im Gehirn ist zentrales Element des Kognitivismus. Kerres beschreibt im Zusammenhang mit kognitivistischen Ansätzen Lernen als besondere Form der Wahrnehmung und Speicherung von Informationen [Ker01, Kapitel 2]. Bei der Gestaltung von Lehrmitteln nach kognitivistischer Sicht geht es darum, die kognitiven Prozesse im Gehirn des Lernenden auszunutzen und so eine dauerhafte Speicherung und Abrufbarkeit von Wissen zu erzielen. Die Wahrnehmung der Inhalte und die Repräsentation von Wissen sind dabei wichtige Aspekte, wie sie Anderson in [And01, Kapitel 2 bis 5] beschreibt. In Kapitel 3 stellt er dar, wie die Automatisiertheit von Aufgaben deren Lösung erleichtert. Als Beispiel nennt er den *Stroop-Effekt* und das zugehörige Experiment [And01, Seite 101], in dem Probanden eine Reihe von Wörtern vorgelegt bekommen:

> Die Aufgabe verlangt von den Probanden, die Druckfarbe zu nennen, mit der die Wörter geschrieben sind. Das Wort, dessen Druckfarbe zu benennen ist, kann ein Farbwort wie *rot* oder ein neutrales Wort wie *Lob* sein. Wenn

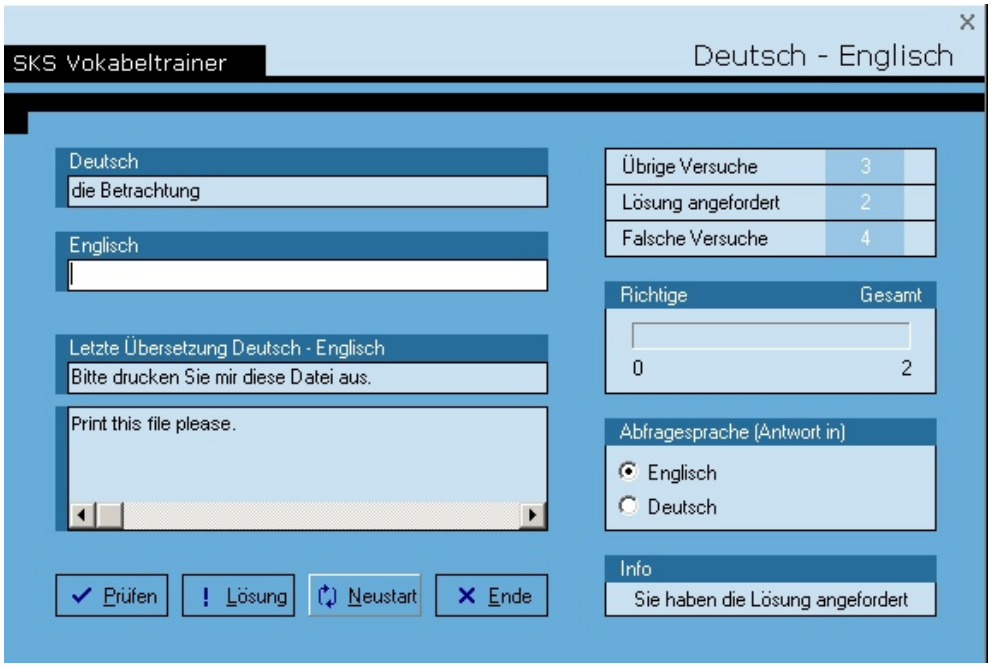


Bild 2.5: Vokabeltrainer: Ausgabe der korrekten Lösung (Quelle: [Kes04])

> es sich um ein Farbwort handelt, kann es in der jeweils bezeichneten Farbe oder in einer anderen Farbe gedruckt sein.

Das Experiment hat ergeben, dass es für die Probanden wesentlich leichter ist, die Farbe zu nennen, wenn sie mit dem Wort übereinstimmt, als wenn ein Konflikt vorliegt, zum Beispiel, wenn das Wort *rot* in grün gedruckt wurde. Das hängt damit zusammen, dass das Lesen eine stark automatisierte Tätigkeit ist, das Benennen von Farben jedoch nicht. Neben Erkenntnissen zur Wahrnehmung spielen beim Kognitivismus auch Erkenntnisse über die Funktion des Gedächtnisses eine Rolle, wie sie zum Beispiel in [And01, Kapitel 6 und 7] dargestellt werden. Anderson beschreibt dort, wodurch das Gedächtnis beeinflusst werden kann. Er beschreibt dort zum Beispiel, dass die *elaborative Verarbeitung* zu besserem Behalten führt. Unter elaborativer Verarbeitung versteht er die Anreicherung des zu lernenden Materials um zusätzliche Informationen. Zur Verdeutlichung beschreibt er das folgende Experiment [And01, Seite 192]:

> ... Probanden [wurden angewiesen,] sich einfache Sätze wie *Der Arzt haßte den Rechtsanwalt* zu merken. Unter einer Versuchsbedingung lernten die Probanden den Satz lediglich, unter der anderen wurden sie aufgefordert, eine Elaboration zu generieren, beispielsweise weil er ihn wegen eines Kunstfehlers anzeigte. Später sahen die Probanden Subjekt und Verb des Originalsatzes (zum Beispiel Der Arzt haßte) und sollten das Objekt wiedergeben (zum Beispiel den Rechtsanwalt).

Ohne die Elaboration konnten sich 57 Prozent der Probanden das Objekt abrufen, mit Elaboration war es 72 Prozent der Probanden möglich. Die zusätzlich erzeugten Inhalte der Elaboration sorgen für eine erweiterte Wahrnehmung. Dadurch wird eine tiefergehende Verknüpfung der Information mit den bereits im Gedächtnis gespeicherten Inhalten hergestellt, wodurch ein späterer Abruf einfacher wird.

Bei der Entwicklung von kognitivistischen Lehrmitteln ist die Gestaltung der Lehrmittel auf ihre Darbietung konzentriert. Ähnlich wie bei Gagné wird hier nicht *die* richtige Lehrmethode kodiert, sondern die geeignete Methode gewählt und deren Umsetzung dann nach Erkenntnissen der kognitiven Psychologie vorgenommen. Diese Aufgabe kann sehr aufwändig sein, da es eine große Menge an Ergebnissen zur kognitiven Psychologie gibt. Einige allgemeine Beispiele für die Berücksichtigung dieser Ergebnisse können dem alltäglichen Leben entnommen werden. So ist es beispielsweise ungünstig, in einem Lernprogramm einen „Weiter"-Knopf rot und einen „Anhalten"-Knopf grün zu färben. Dies erschwert die Wahrnehmung der Bedeutung des Knopfes (ähnlich dem Stroop-Effekt). Wie stark dieser Effekt ausfällt, hängt mitunter vom Lernenden ab – in Kulturen, in denen Lichtzeichenanlagen zur Weiterfahrt rot anzeigen, wird ein roter „Weiter"-Knopf keine Schwierigkeiten bereiten.

Ein weiteres Beispiel ist die Überlastung der Lernenden. Während des Multimedia-Booms Anfang bis Mitte der 1990er Jahre bedeutete „Multimedia", dass alles möglichst bunt und laut sein musste. Anderson gibt Experimente an, aus denen ersichtlich wird, dass die Aufmerksamkeit nur auf wenige Inhalte gleichzeitig gelenkt werden kann [And01, Kapitel 3]. Bei zu großzügig angereicherten Lehrmitteln besteht so die Gefahr, dass genau diejenigen Inhalte, die eigentlich vermittelt werden sollen, nicht mehr wahrgenommen werden. Bild 2.6 zeigt beispielhaft eine Seite aus einem Lernprogramm mit verbesserungswürdiger Gestaltung, bei dem die Lernenden durch Überlastung demotiviert werden können.

Kritzenberger nennt in [Kri05, Kapitel 2.2] drei Arten der kognitiven Belastung: Neben der bereits genannten Belastung durch die kognitive Verarbeitung nennt sie noch die intrinsische und die extrinsische Belastung. Die intrinsische Belastung bezeichnet die Belastung bei der Verarbeitung des Inhalts. Sie steigt mit dem Schwierigkeitsgrad des Inhalts. Die extrinsische Belastung wird durch den Lernraum, also auch direkt durch die Gestaltung der Lehrmittel, beeinflusst. Das Lehrmittel muss so gestaltet sein, dass der Aufwand für den Lernenden, um an die zu vermittelnde Information zu gelangen, möglichst gering ist. Kritzenberger verweist hier auf ihre Multimediaprinzipien im selben Buch [Kri05, Kapitel 4.2], auf die ich hier nicht näher eingehe, da das in dieser Arbeit entstandene Programm keine Inhalte vermittelt, wie dies dort angenommen wird, sondern Zusammenhänge anhand einer simulierten Darstellung verdeutlicht.

Konstruktivismus

Beim konstruktivistischen Ansatz konzentriert man sich mehr auf den Lernenden und seine Umgebung. Der Lernprozess wird von der Konstruktion von Wissen durch den

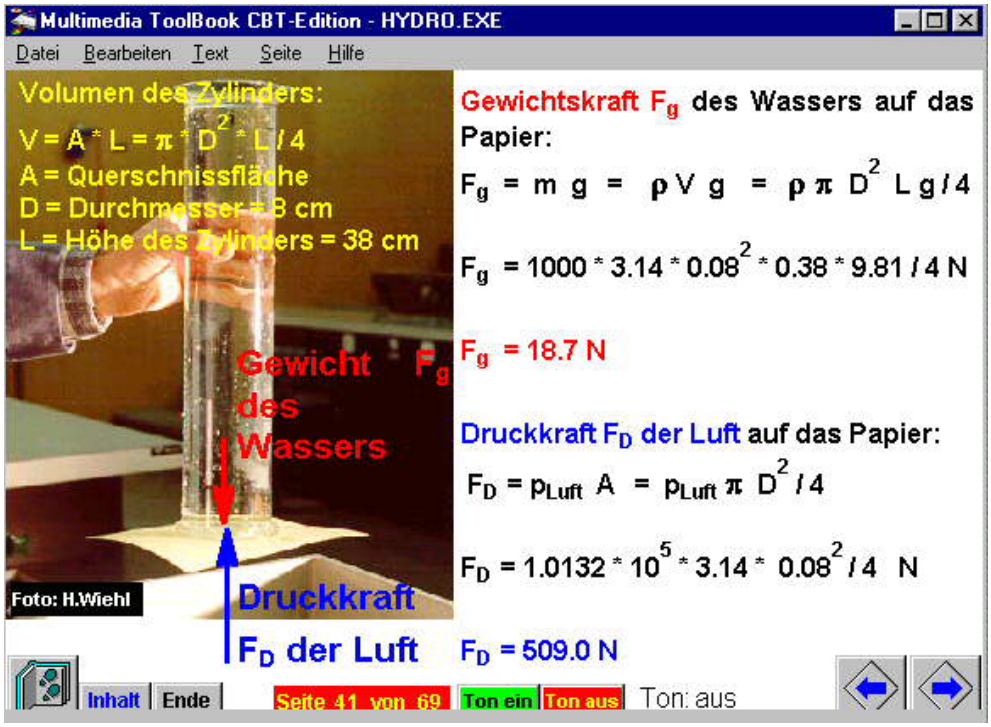


Bild 2.6: Bildschirmfoto von HYDRO.EXE (Quelle: [WIZ06])

Lernenden beherrscht, der dieses neue Wissen dann seinem bisherigen hinzufügt [Kri05, Kapitel 1.3.3]. Diese Konstruktion kann nicht durch bloße Weitergabe von Wissen geschehen, sondern ist immer in einen aktiven Kontext eingebettet [IP02, Seite 166-178]. Aufgrund dieser Verbindung zum Kontext wird das Lernen im Konstruktivismus als *situiertes Lernen* bezeichnet. Die Grundlage hierfür beschreibt Gagné in [GWGK05, Seite 6]:

> Learned capabilities are acquired in a particular context, and the perceived utility of that context has implications for later retrieval and use. (...) The guiding principle (...) is that learning that occurs in authentic contexts where it can be meaningfully applied is more likely to be remembered and recalled when needed.

Es gibt unterschiedliche Ansichten dazu, welche Auswirkungen die Erkenntnisse zum situierten Lernen auf die Lehre haben können. Während sich beispielsweise Kerres in [Ker01, Kapitel 3.2] mit der Frage beschäftigt, ob situiertes Lernen das Ende der Lehre bedeutet, weil es die bisherigen didaktischen Konzepte in Frage stellt, ist die Aussage in [IP02, Seite 166-178] positiv und verspricht durch den Konstruktivismus bessere Lernergebnisse.

Bei der Gestaltung von konstruktivistischen Lehrmitteln liegt der Schwerpunkt wieder mehr auf der Zusammenstellung der Struktur und weniger auf der konkreten Gestaltung. Es gibt mehrere Ansätze, konstruktivistische Lehrmittel herzustellen. Einer davon ist die Gestaltung von Szenarien als *Anker* für die Aufmerksamkeit des Lernenden. Deswegen nennt man diesen Ansatz *geankertes Lehren* (siehe [Ker01, Kapitel 3.3], [Kri05, Kapitel 1.4]). Die Szenarien sollen abstrakte Darstellungen und Modelle ablösen, die den Lernenden demotivieren. Als Beisiel nennt Kerres eine Bildplatte über *Die Abenteuer des Jasper Woodbury*, in der die Schüler *Jasper* auf einer Schiffstour begleiten, wobei Informationen über Schifffahrt, Kraftstoffverbrauch, Geschwindigkeiten und Distanzen eingearbeitet wurden [Ker01, Seite 79].

Ein weiterer Ansatz ist der des *Goal Based Scenario* (GBS), in dem der Handelnde einen Auftrag erhält, den er erfüllen soll. Kritzenberger gibt ein Beispiel in [Kri05, Seite 19]:

> Der Lerner übernimmt den Auftrag eine Spedition zu leiten und diese zur Marktführerposition auszubauen. Er muss dazu Kontrolle übernehmen, Abläufe steuern, Zusammenhänge entdecken und Handlungsstrategien entwickeln.

Weitere Ansätze, die genannt werden (fallbasiertes Lernen, Projektmethode), stellen im Grunde nur Varianten des geankerten Lehrens oder des GBS dar. Von einer detaillierteren Betrachtung sehe ich daher ab.

2.2.3 Interaktivität in Lernprogrammen

Interaktion bedeutet eine „Wechselbeziehung zwischen Handlungspartnern“ [Dud03], in diesem Fall zwischen Lernendem und Lernprogramm. Ein interaktives Lernprogramm muss also auf den Lernenden mit Handlungen reagieren. Technisch gesehen könnte jede Art von Aktion und Reaktion eines Programmes als Teil einer Interaktion gewertet werden: Das Abspielen eines Tons bei Betätigen eines Knopfes, die Aufforderung des Lernenden durch das Programm, etwas zu tun, falls der Lernende längere Zeit untätig ist und Ähnliches. Im Rahmen von Lernumgebungen muss Interaktivität bestimmten Anforderungen genügen. Niegemann hat in [Nie01] und [NHHM+04] Interaktivität von Lehrmitteln beschrieben. Er zählt Grundfunktionen des Lehrens auf und schreibt, dass Interaktivität, die keine dieser Funktionen unterstützt, wahrscheinlich überflüssig, wenn nicht kontraproduktiv ist. Die Funktionen sind:

- Motivieren
- Informieren
- Verstehen fördern
- Behalten fördern

- Anwenden bzw. Transfer fördern
- Lernprozess organisieren und regulieren

Bei der Gestaltung eines interaktiven Lehrmittels sollte also nicht nur die konkrete Interaktion, sondern insbesondere ihre Wirkung im Auge behalten werden. Typische Arten von Interaktion sind Feedbacks, wie ich sie im Rahmen der Eigenschaften behavioristisch orientierter Lehrmittel auf Seite 16 wiedergegeben habe. Konkrete Ausprägungen können von der einfachen Falsch/Richtig-Aussage, wie sie in Bild 2.3 zu sehen ist, bis hin zu einer Belohnung, zum Beispiel in Form eines Films, gehen.

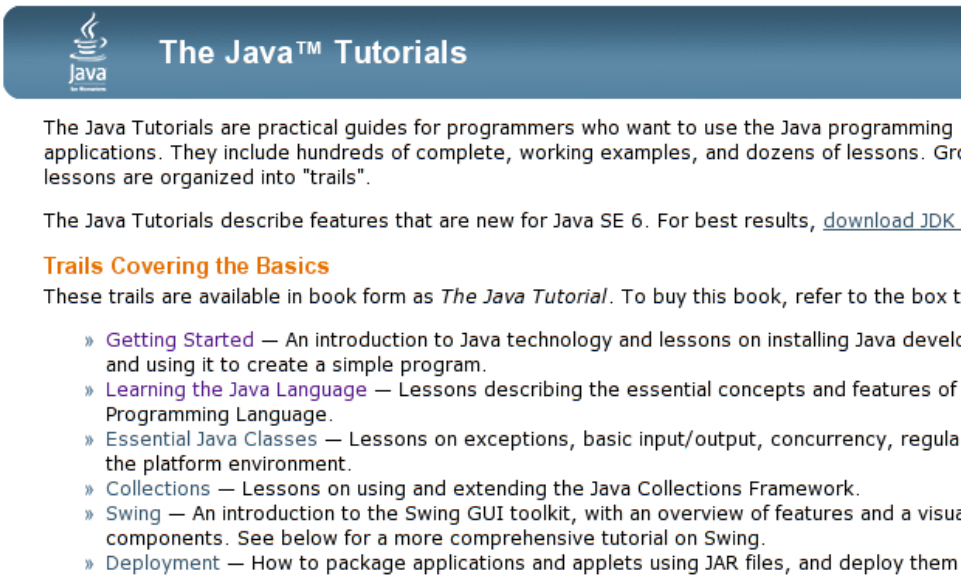


Bild 2.7: Bildschirmfoto von *The Java Tutorials*: Einstieg

Eine andere Art der Interaktion ist die direkte Manipulation des Mediums. Beispiele hierfür sind die Steuerung des Mediums, im einfachsten Fall Vorwärtsblättern, oder ein Eingriff in die Bedingungen der Lehreinheit, wie etwa die Auswahl eines Schwierigkeitsgrades. Je nach Auslegung des Mediums – zum Beispiel als Simulation – sind auch komplexere Manipulationen denkbar. Die Steuerung als Interaktion kann auch in den Lernprozess integriert sein, indem sie den Zugriff auf das Lehrmittel vereinfacht und durch ihre eigene Struktur beratend wirkt. Ein Beispiel hierfür ist die Online-Version der Java Tutorials. Bild 2.7 zeigt die Übersicht. Der Lehrstoff ist in verschiedene Bereiche, sogenannte *trails* (Pfade) unterteilt, die prinzipiell in der vorgegebenen Reihenfolge abgearbeitet werden sollen. Durch die Aufteilung kann der Lernende sich jedoch selbst aussuchen, welche Einheit er lernen möchte, falls er zum Beispiel über einige bereits Wissen besitzt. Bild 2.8 zeigt einen Teil des trails „Getting Started“. Dort existieren oben und links angepasste Orientierungs- und Navigationshilfen. Man

Bild 2.8: Bildschirmfoto von *The Java Tutorials*: Getting Started

kann erkennen, in welchem trail man sich befindet (oben) und wo innerhalb des trails man sich befindet und welche Themen das aktuelle Thema umgeben (links). Dadurch kann man sich gut im Lernstoff orientieren.

2.2.4 Simulation

Um ein interaktives Modell für einen verborgenen und in einer komplexen Umgebung eingebetteten Sachverhalt wie die FlexRay-Medienzugriffskontrolle herzustellen, benötigt man eine Animation oder eine Simulation. Für einfache Abläufe, die nahezu jedes Mal gleich sind, wie zum Beispiel die Funktion einer Schiffsschleuse, genügt eine einfache Animation. Für kompliziertere, veränderbare Vorgänge benötigt man eine Simulation.

Köcher beschreibt in [KMOS72] Simulation allgemein als „das Nachbilden eines realen Prozesses mit künstlichen Mitteln. Sie besteht darin, dass ein Stück Wirklichkeit zunächst so genau wie möglich in einem Modell dargestellt wird und dass dieses dann experimentell auf seine Eigenschaften hin untersucht wird.“ Als Ziele einer Simulation gibt Köcher Demonstration, Training, Forschung, Optimierung und Logik[13] an. Bei den verwendeten Modellen für eine Simulation unterscheidet er zwischen physikalischen Modellen, Spielmodellen und mathematischen Modellen. Eine rechnergestützte Simulation erwähnt Köcher noch nicht, nennt unter den physikalischen Modellen

[13] Er bezieht sich hiermit auf ein tiefergehendes Verständnis des simulierten Gegenstandes, das auch schon während der Konstruktion des Modells gewonnen werden kann. Er nennt dies eine „logische Durchdringung“.

aber eine elektronische Nachbildung als Möglichkeit. Physikalische Modelle würden wie Spielmodelle experimentell erforscht, während mathematische Modelle deduktiv erforscht werden und nichts mit einem Experiment gemein hätten.

Die rechnergestützte Simulation entspricht dem Experimentieren mit physikalischen Modellen. In der Realität wird ein Modell während einer kontinuierlichen Zeitspanne benutzt und beobachtet. Da ein Computer keine kontinuierlichen Größen verarbeiten kann, muss man eine Diskretisierung durchführen. Es gibt zwei Ansätze zur diskreten, rechnergestützten Simulation. Bei der zeitdiskreten Simulation, auch „schrittweise Simulation“ genannt, wird die zu simulierende Zeitspanne in endlich viele Schritte zerlegt. Für jeden Schritt wird berechnet, was im Modell passiert und wie es darauf reagiert. Bei der ereignisdiskreten Simulation, auch „ereignisorientierte Simulation“ genannt, gibt es einen Ablaufplan, in dem festgelegt ist, welche Ereignisse wann auftreten. Statt nun Zeit-Schritte durchzuführen, werden die Ereignisse der Reihe nach abgearbeitet und die entsprechenden Reaktionen berechnet.

Ein Vorteil der ereignisdiskreten Simulation ist, dass man bei längeren ereignislosen Zeitabschnitten nicht unnötig auf die Simulation warten muss. Der hierfür notwendige Programmier- und Verwaltungsaufwand kann jedoch höher sein als bei der zeitdiskreten Simulation, falls sich Ereignisse überlappen. In der speziell für Simulationen entwickelten Programmiersprache SIMULA existiert für solche Fälle das Konzept der *Koroutine*. Koroutinen laufen partiell ab. Sie können an einer beliebigen Stelle unterbrochen und später wieder fortgesetzt werden. Damit ist es möglich, die für die überlappenden Ereignisse nötige Parallelität nachzubilden.

Eine zeit- oder ereignisdiskrete Simulation kann die Grundlage für eine Interaktion im didaktischen Sinn sein. Die Simulation stellt dabei die Steuerung für eine variable Präsentation von Sachverhalten dar; sie ist sozusagen der Motor für die zumeist grafische Darstellung. Präzision und Umfang treten dabei in den Hintergrund, während Abstraktion und Steuerbarkeit wichtig werden. Dies soll nicht bedeuten, dass die Simulation grundsätzlich unpräzise sein soll. Es soll nur bedeuten, dass je nach Ziel der Simulation einige Inhalte abstrahiert werden, um dem Benutzer eine eingänglichere und kompaktere Simulation zu bieten. Zum Beispiel braucht eine Simulation, die als Grundlage für ein Lernprogramm über die Funktionsweise einer Sanduhr dient, nicht für jedes Sandkorn in der Uhr alle physikalischen Parameter präzise zu berechnen und zur Darstellung anzubieten. Es genügt völlig, den Rieselvorgang grob zu simulieren. Ebenso ist es unnötig, die Rieselgeschwindigkeit für alle möglichen räumlichen Lagen der Sanduhr berechnen zu können. Diese könnten den Beobachter eventuell sogar verwirren. Die Simulation einer aufrecht stehenden Sanduhr genügt, um die Zeitmessung zu demonstrieren.

Damit die Interaktion nicht am Benutzer vorbeigeht, muss sie durch ihn kontrollierbar bleiben. Daher ist die einfache Steuerbarkeit bei einer solchen Simulation wichtig. So ist bei Erstbetrachtungen eines Vorganges eine schrittweise Steuerung vorteilhaft, und bei wiederholtem Experimentieren ist es erforderlich, dass derselbe Ablauf schnell und mit gleichem Ergebnis wiederholbar ist. Beim Entwurf der Simulation muss dies

berücksichtigt werden. Die Manipulation der Parameter und das Eingreifen in das laufende Geschehen gehört auch zur Steuerbarkeit – ohne diese Mittel würde auch eine einfache Animation ohne Simulation genügen. Bei dem Sanduhr-Beispiel könnte man sich vorstellen, dass der Benutzer die Möglichkeit hat, den Simulationsparameter „Sand-Füllmenge“ zu ändern und direkt in das Geschehen einzugreifen, indem die Simulation auf eine Eingabe „Sanduhr umdrehen“ reagiert und daraufhin die Richtung des Rieselns ändert.

2.2.5 Visualisierung

Die Visualisierung ist meistens ein Aspekt bei der Erstellung von Lernprogrammen, auch wenn nicht immer explizit der Begriff „Visualisierung“ verwendet oder sogar definiert wird, wie in [NHHM+04, Seite 136].

In der klassischen Auslegung des Begriffs der *rechnergestützten Visualisierung* wird darunter die strukturierte Darstellung von Information verstanden. Als Beispiele für solche Informationen werden oft Daten angeführt, die Ergebnisse aus Umfragen, Messungen oder Simulationen sind [SM00]. Wie in der Definition von Niegemann erwähnt, ist Visualisierung aber auch die Sichtbarmachung von *Prozessen*, und nicht nur von Ergebnissen solcher Prozesse.

Der Charakter der Visualisierung bei Lernprogrammen ist dabei von anderen Motiven geprägt als bei der Sichtbarmachung von Daten in technischen Dokumenten. Während bei technischen Dokumenten der Schwerpunkt hauptsächlich auf einer übersichtlichen Strukturierung liegt [Rec05], ist bei Lernprogrammen oftmals eine ansprechende Gestaltung wichtig, um den Benutzer zu motivieren und zum Weitermachen zu animieren [NHHM+04, Ker01]. Dabei wurde, wie Kerres [Ker01, Kapitel 2] anmerkt, in der Vergangenheit die professionelle Produktion von Medien vernachlässigt. Niegemann und Euler geben in [Nie01, Eul92] Beispiele vom „Multimedia“-Trend und seinen Auswirkungen. Anstatt sich auf die Inhalte zu konzentrieren, wurde die Gestaltung zum Selbstzweck. Lernprogrammen wurde das Zappeln und Klingeln beigebracht, was mehr Ablenkung als Unterstützung brachte.

Es bietet sich daher die Beachtung von Regeln aus der Mediengestaltung an, wie sie zum Beispiel in [BBSZ01] zu finden sind und in der Ausbildung zum *Mediengestalter für Digital- und Printmedien* vermittelt werden. Eine umfassende Einführung in die Mediengestaltung ist mir im Rahmen dieser Arbeit leider nicht möglich. Themen der Mediengestaltung sind beispielsweise Layout, Typographie, Screendesign, Druckverfahren, Optik, Farbtheorie, Akustik, Digitalmedien, Bildverarbeitung, Videoverarbeitung und Präsentation.

Ich werde beispielhaft einige relevante Eigenschaften kurz vorstellen.

Bei der Gestaltung eines Mediums, das Informationen anbietet, ist zu berücksichtigen, dass diese Informationen auch effektiv transportiert werden. Ein einfaches Beispiel für eine nicht besonders effektive Umsetzung habe ich bereits in Abschnitt 2.2.2 erwähnt: Bild 2.4 zeigt das Feedback eines Vokabeltrainers bei Eingabe einer falschen

Antwort. Die Information ist nur schwer auszumachen – sie befindet sich rechts unten im Bild, nicht weiter kenntlich gemacht. Wie an Bild 2.5 zu erkennen ist, wird derselbe Anzeigebereich auch unverändert für andere Informationen mit weit geringerer Wichtigkeit verwendet. Die Umsetzung der Gestaltung des Feedbacks ist bei 2.3 schon besser gelungen: Es wird dem Benutzer sofort klar, wo und wie die Information aufgetreten ist. An der Stelle, an der die Information präsentiert wird, wird während der sonstigen Programmbedienung nichts angezeigt.

Die Darstellung an sich erfordert eine gewisse Ordnung und Klarheit. Hierfür genügt meist etwas Einheitlichkeit. Dies ist der Vermeidung der Überlastung der Kognition (siehe Seite 20 und Bild 2.6) nicht unähnlich.

Die Auswahl der Medien und deren Nutzung kann eine besondere Schwierigkeit darstellen. Aufgrund der Vielseitigkeit der für Endverbraucher verfügbaren Rechnersysteme kann es sehr aufwändig sein, ein Lernprogramm zu gestalten, das auf all diesen Systemen ähnlich funktioniert. Als Beispiel sind webbasierte Trainingslösungen zu nennen, bei denen nicht nur die Beschaffenheit des Zielsystems, sondern auch die Eigenschaften des verwendeten Webbrowsers die Gestaltung beeinflussen. Als Beispiel kann hier das Java-Tutorial genannt werden. Bild 2.8 entstand unter Linux mit Mozilla Firefox. Bild 2.9 zeigt denselben Ausschnitt mit Microsoft Internet Explorer unter Microsoft Windows XP Professional. Die Unterschiede in der Gestaltung sind durch die verwendeten Systeme bedingt. Eine Angleichung für alle Systeme ist entweder technisch nicht möglich oder unverhältnismäßig aufwändig.

Bild 2.9: Vergleichsbild *The Java Tutorials*: Getting Started

Die Formatierung und Speicherung der Inhalte ist ebenfalls ein Thema der Me-

diengestaltung. So ist es erforderlich, ein Verständnis von Datenformaten für digitale Medien zu entwickeln (siehe [BBSZ01], Kapitel 3.1 und Kapitel 6), um zum Beispiel zu vermeiden, dass für ein digitales Medium eine unangemessen große Datenmenge verbraucht wird oder bestimmte Kompressionsmethoden seinen Inhalt unleserlich machen.

2.2.6 Die interaktive Visualisierung

Das in dieser Arbeit entwickelte Programm ist kein Lernprogramm im klassischen Sinn. Es ist vielmehr ein Hilfsmittel, das das Lernen unterstützt. Durch die besondere Veranschaulichung eines schwierig mit Worten zu vermittelnden Sachverhaltes (Visualisierung) und – im Rahmen des Themas weitreichenden – Eingriffsmöglichkeiten (Interaktivität) hilft es dem Lernenden, den dargestellten Sachverhalt zu erfahren (hauptsächlich situiertes Lernen) und dadurch ein besseres Verständnis dafür zu entwickeln. Der wichtigste Unterschied zu einem herkömmlichen Lernprogramm oder -system – wie zum Beispiel auch DoLL [Pro93] und seine Anwendung [Spr95] – ist, dass die interaktive Visualisierung keine starren Inhalte präsentieren und abfragen soll, sondern dass sie durch eine einfache Simulation Zusammenhänge verdeutlicht.

Die Interaktionen ergeben sich hauptsächlich aus den für den Sachverhalt erforderlichen Manipulationsmöglichkeiten, wie ich sie beispielhaft schon für die Sanduhr angedeutet habe: Die Sanduhr kann herumgedreht, abgelesen oder mit verschiedenen Sandmengen befüllt werden. Besondere Möglichkeiten bei der rechnergestützten Simulation können ebenfalls ausgenutzt werden: Die Sanduhr kann angehalten werden, um den Füllstand nach einer bestimmten Zeit genauer zu untersuchen.

Die Visualisierung hat in diesem Zusammenhang hauptsächlich die Aufgabe, die während der Simulation entstehenden Daten zu präsentieren. Sie wird durch die Simulation sozusagen angetrieben. Darüber hinaus stellt die Visualisierung die Schnittstelle zur Steuerung der Simulation dar – inklusive der Manipulation von Parametern. Dieser Aspekt erlaubt es je nach Situation, diese beiden Funktionen in die Darstellung einzubetten. Bei dem Sanduhrbeispiel könnte eine grafische Darstellung einer Sanduhr direkt mit dem Mauszeiger „angefaßt“ werden, um sie durch Bewegen der Maus in die entsprechende Richtung herumzudrehen.

2.3 Zusammenfassung

Es gibt schon viele Feldbusse, doch steigende Anforderungen führen zu immer komplexeren Konzepten und daraus entstehenden Entwicklungen, die ohne geeignete Hilfsmittel nur schwierig zu verstehen sind. FlexRay ist eine solche Entwicklung, die auch den Anspruch hat, zukünftige Anforderungen erfüllen zu können.

Um für einen Teilaspekt eines komplexen technischen Systems ein geeignetes interaktives Lernprogramm zu erstellen, ist es erforderlich, dem Programm verschiedene

Eigenschaften aus Didaktik und Mediengestaltung, wie ich sie in Kapitel 2.2.2, 2.2.3 und 2.2.5 vorgestellt habe, beim funktionalen und visuellen Entwurf zu berücksichtigen. Zu den Eigenschaften gehören beispielsweise: Lernen förderndes Feedback, geringe kognitive Belastung des Lernenden oder Steigerung seiner Motivierung durch eine ansprechende Gestaltung. Damit soll sichergestellt werden, dass der Lernende einen einfachen Zugang zum Programm und zum Thema hat. Welche konkreten Mittel im Detail geeignet sind, ist dabei abhängig von der Zielgruppe und den beabsichtigten Wirkungen des Lernprogramms.

3 Existierende Lehrmittel und Programme

In diesem Kapitel beschreibe ich zuerst, wie an Universitäten eingesetzte Lehrmittel zu FlexRay, insbesondere der Medienzugriffskontrolle, aussehen. Danach beschreibe ich die Lehrsituation in der Industrie und untersuche in diesem Rahmen einige Programme daraufhin, ob sie als Lehrmittel geeignet sind.

3.1 Medien aus der Lehre

Es gibt einige Vorlesungen an verschiedenen Fachbereichen von Hochschulen, die Inhalte zu FlexRay vermitteln. Beispiele sind: Automobilelektronik, Kommunikation und Netzwerkmanagement oder Eingebettete Systeme. Die Veranstaltungen an den auf Elektrotechnik ausgelegten Fachbereichen beschäftigen sich hauptsächlich mit der Signalverarbeitung auf dem Bus und weniger mit den Protokolleigenschaften. Vorlesungen in der Informatik vermitteln dagegen mehr protokollrelevante Details, wie zum Beispiel zur Zeitsynchronisierung oder der Medienzugriffskontrolle.

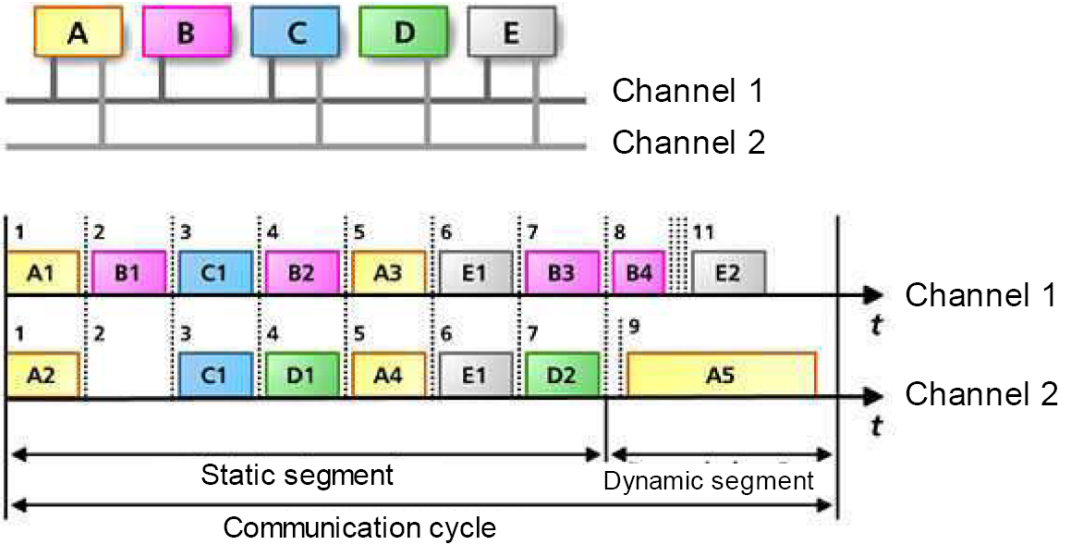


Bild 3.1: Beispiel aus der Lehre: FlexRay Kommunikationszyklus. Quelle: [Mar06]

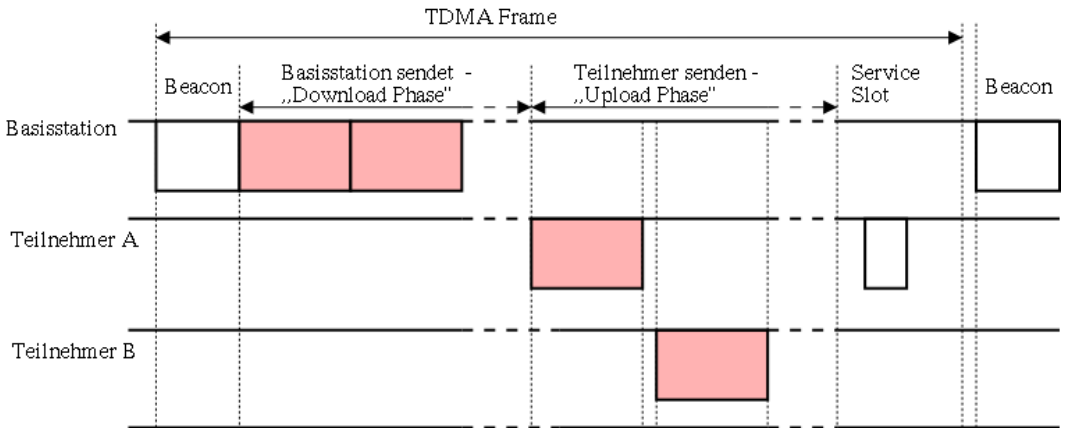


Bild 3.2: Beispiel aus der Lehre: TDMA. Quelle: [Man06].

Die Vorlesungsmaterialien enthalten entweder Bilder direkt aus der FlexRay-Spezifikation, wie zum Beispiel das Material zur Vorlesung „Kommunikation und Netzwerkmanagement“ an der FH München [Som06], oder vereinfachte, anschaulichere Darstellungen des FlexRay-Kommunikationszyklusses, wie die Materialien zu den Vorlesungen „Eingebettete Systeme“ an der Universität Dortmund [Mar06] oder an der ETH Zürich [Thi06]. Diese einfacheren Darstellungen unterscheiden sich nur minimal voneinander. Beispielhaft zeigt Bild 3.1 die Darstellung aus den Folien zur Vorlesung „Eingebettete Systeme“, die im Fachbereich Informatik an der Universität Dortmund gehalten wird. Ähnliche Darstellungen werden auch in Vorlesungen zu Rechnernetzen verwendet, zum Beispiel bei Krumm [Kru02, Seite 51]. Bild 3.2 zeigt eine weitere Möglichkeit, wie TDMA-Verfahren in der Lehre dargestellt werden. Es stammt aus einem Seminar für drahtlose Netze an der Universität Oldenburg. Hier wurde anstelle verschiedener Farben oder Bezeichner für die einzelnen Knoten eine abgesetzte Anordnung gewählt. Diese Art der Darstellung kann schnell groß und unübersichtlich werden und ist daher für ein Lernprogramm, das eine weitgehend freie Konfiguration erlaubt, nicht empfehlenswert.

Materialien wie diese eignen sich gut, um die Struktur der Medienzugriffskontrolle beim FlexRay-Protokoll zu vermitteln. Es besteht jedoch der Wunsch, den Studierenden mehr zu zeigen: Wie verändert sich die Kommunikation im Laufe der Zeit? Welche Konsequenzen haben Änderungen bei der Konfiguration des Zugriffs? Derartige Aspekte können mit einfachen Bildern nicht oder nur aufwändig erläutert werden. Der Lerneffekt von Übungsaufgaben, die sich nur auf solche Text- und Bildinformationen stützen können, ist bei Themen wie der Medienzugriffskontrolle beim FlexRay-Protokoll zudem begrenzt: Um die Aufgabe zu lösen, liest der Studierende die Methodik nach und baut sich das Ergebnis entlang der Texte zusammen. Durch die Konzentration auf das Auffinden der gerade wichtigen Information kann der Zusammenhang zwischen den

einzelnen Teilaspekten dabei leicht übersehen werden. Da das Ergebnis der Übung oft erst relativ spät in der nächsten Übungsveranstaltung kontrolliert wird, besteht zudem die Gefahr, dass aus gemachten Fehlern nicht genug gelernt wird oder das Hinzulernen neuer Informationen und Zusammenhänge während des Wartens auf die Korrektur erschwert wird. Ein Medium, das den Studierenden die Möglichkeit zum Experiment bietet, kann in solchen Fällen von Nutzen sein.

3.2 Medien aus der Industrie

Die Industrie bietet keine Medien speziell zum Erlernen von FlexRay-Eigenschaften an. Zu Beginn dieser Arbeit existierten nur Design- und Testwerkzeuge zum Thema FlexRay. Für Schulungen gab es bei verschiedenen Herstellern aus dem FlexRay-Umfeld nur Ankündigungen. Schulungen von Anbietern, die keine FlexRay-Komponenten herstellen, waren nicht zu finden.

Im Verlauf des Jahres 2006 sind den Ankündigungen konkrete Termine gefolgt. So bieten inzwischen zum Beispiel DECOMSYS [DEC06b] und Vector Informatik FlexRay-Schulungen [DEC06b, Vec06a] zum Preis von 1350 und 800 Euro an. Aus den Beschreibungen der Schulungsinhalte geht hervor, dass dort reale FlexRay-Cluster als Anschauungsobjekte dienen. Für einen Hersteller von FlexRay-Systemen bedeutet dies, im Gegensatz zur universitären Lehre, schließlich keinen besonderen Aufwand. Die in diesen Schulungen verwendeten Geräte kann der Kursteilnehmer jedoch nicht mitnehmen, um zu Hause damit zu üben. Es ist ohnehin zu erwarten, dass die meisten Teilnehmer nach der Schulung an echten FlexRay-Clustern arbeiten werden.

Es sind auch weitere Anbieter auf dem Markt erschienen, die FlexRay-Schulungen anbieten, zum Beispiel die HILF! GmbH [MC06], die sich auf ähnliche Materialien stützt, wie sie in der Lehre zurzeit gebraucht werden. Ein anderer Anbieter, Albotronic [Alb06], bietet seinen Teilnehmern am zweiten Schulungstag den Aufbau eines FlexRay-Clusters an. Hierzu werden spezielle FlexRay-Knoten verwendet, die Albotronic auch für 500 Euro pro Stück verkauft (für ein funktionierendes Cluster sind mindestens zwei Stück erforderlich). Ein umfangreiches Übungsnetz kann so für den Lernenden schnell teuer werden.

Im Folgenden beschreibe ich nun beispielhaft einige verfügbare Programme, deren Eignung als Medium für die Lehre ich untersucht habe. Über diese Beispiele hinaus existieren noch ähnliche Programme von anderen Herstellern, zum Beispiel den Multibus Analyzer für FlexRay [IXX06c] von der IXXAT Automation GmbH, die DECOMSYS::SIMTOOLS [DEC06a, WP06] von der DECOMSYS GmbH oder TTXPlan von der TTAutomotive Software GmbH. Diese Programme haben ähnliche Ziele und Voraussetzungen wie die hier vorgestellten, weshalb ich diese Programme nicht näher beschreibe.

3.2.1 Programme von Vector Informatik

Die Vector Informatik GmbH bietet mehrere Produkte zu FlexRay an: *DaVinci Network Designer FlexRay*, *CANalyzer.FlexRay*, *CANoe.FlexRay* und *CANape*.

Der *DaVinci Network Designer FlexRay* ist ein Programm zum Entwurf von FlexRay-Systemen. Er ermöglicht den grafischen Entwurf der Hardwaretopologie innerhalb eines Fahrzeugs und die Konfiguration sämtlicher Kommunikationsparameter mit einer tabellarischen Darstellung. Der Designer führt während der Eingabe Konsistenzprüfungen durch und schlägt Korrekturen vor. Das Programm unterstützt den Datenim- und -export im Field Bus Exchange Format [DKR+05], das auch von den anderen Produkten von Vector Informatik verwendet werden kann. Die Simulation eines mit dem Designer entworfenen Netzes ist nicht möglich, wodurch keine Einsicht in die Funktionsweise der Medienzugriffskontrolle genommen werden kann. Der Entwickler muss bereits vor der Benutzung des Programms mit den Eigenschaften von FlexRay vertraut sein, um die Kommunikationsparameter sinnvoll einzustellen oder die durch die Konsistenzprüfung hervorgebrachten Vorschläge beurteilen zu können.

CANalyzer.FlexRay ist ein Analyse- und Testwerkzeug für FlexRay-Busse, mit dem die am Bus angeschlossenen Steuergeräte und ihre Kommunikation untersucht werden können. Hierzu kann der Busverkehr beobachtet und die auf dem Bus gemessenen Signale können visualisiert werden. Um bestimmtes Verhalten hervorzurufen, ermöglicht es der CANalyzer, Nachrichten zu definieren und zu versenden, den Verkehr aufzuzeichnen und wiederzugeben sowie Busstörungen zu erzeugen. Die Verwendung des CANalyzer ohne FlexRay-Netz ist nicht möglich.

Das Entwicklungs-, Test- und Simulationswerkzeug *CANoe.FlexRay* ermöglicht eine Beobachtung des Verhaltens der an den Bus angeschlossenen Knoten auf Anwendungsebene. Eine grafische Darstellung des Netzes dient zur Konfiguration der Simulation (Bild 3.3). Während der Simulation werden Daten aus den Knoten tabellarisch dargestellt, wie zum Beispiel in Bild 3.4. Eine Darstellung aller Nachrichten ist auch möglich, so dass nachvollzogen werden kann, welcher Knoten welche Nachricht wann gesendet hat und ob ein anderer Knoten darauf reagiert hat. Tiefere Einblicke sind mit CANoe allerdings nicht möglich; die Funktionsweise der Medienzugriffskontrolle von FlexRay wird nicht dargestellt.

3.2.2 dSPACE Simulator und Simulink

Die Firma dSPACE bietet ein Hardware-in-the-Loop (HIL) Simulationssystem für Simulation und Test von elektronischen Steuergeräten (Electronic Control Units, ECUs) an. Bei der HIL-Simulation wird das zu testende Steuergerät[1] an ein System angeschlossen, das die Umgebung simuliert, in der das Steuergerät später arbeiten soll[2].

[1] Die Hardware.

[2] Die Schleife: Der Simulator versorgt das Gerät mit Sensordaten. Das Steuergerät verarbeitet diese, und der Simulator nimmt die Reaktionen wieder entgegen.

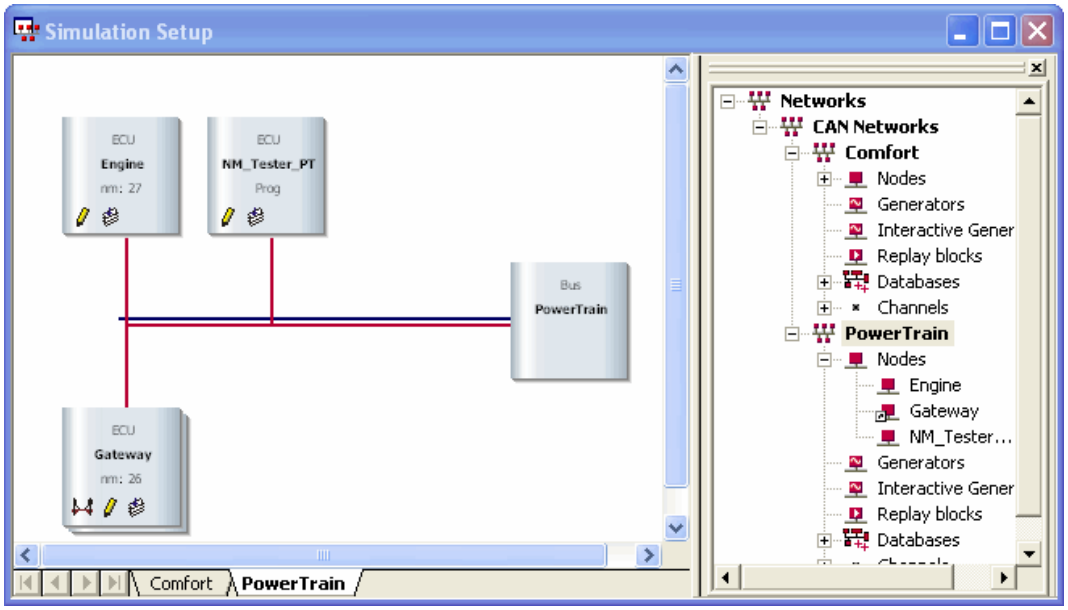


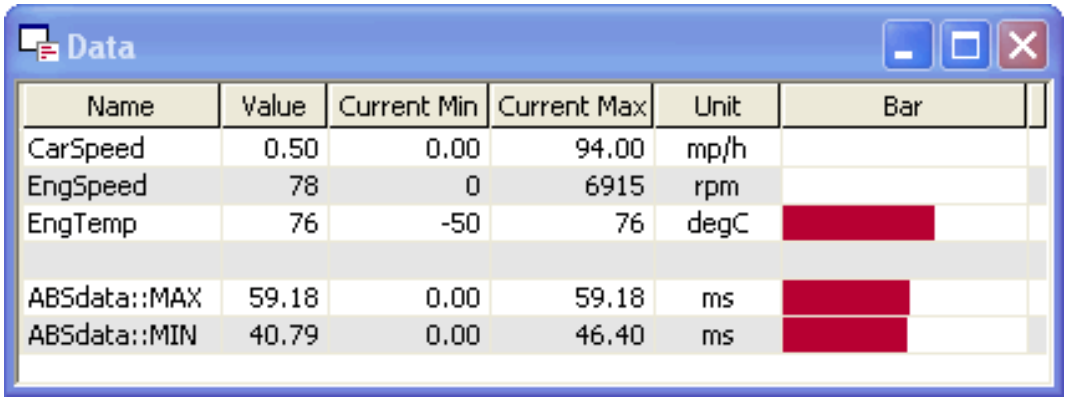
Bild 3.3: Simulationskonfiguration bei CANoe

Entwickler von Steuergeräten können diese dadurch ohne ein funktionsfähiges Gesamtsystem testen. Das kann ein finanzieller oder organisatorischer Vorteil sein: Ein in Entwicklung befindliches Gesamtsystem könnte noch gar nicht verfügbar sein. Der dSPACE-Simulator kann durch Module und Softwarekomponenten für verschiedene Systeme angepasst werden. Eines dieser Systeme ist FlexRay. Bild 3.5 zeigt ein Standbild aus der Simulation eines FlexRay-Modells und Bild 3.6 zeigt die Konfigurationsseite. Diese Abbildungen stammen von der Webseite von dSPACE [dSP06][3].

Die Modelle für den dSPACE-Simulator werden mit Hilfe der Software Simulink der Firma The MathWorks [The06] erstellt und gepflegt. Spezielle, eigenständige FlexRay-Modelle für Simulink habe ich während der Erstellung dieser Arbeit nicht finden können. Aufgrund der Arbeitsweise und Zielsetzung von Simulink – ein präzises mathematisches Simulationsinstrument zu sein – schätze ich die FlexRay-Blocksets von dSPACE als für diese Anwendung repräsentativ ein.

Das Design der Benutzeroberfläche lässt erkennen, dass die Zielgruppe des Systems vor seiner Anwendung bereits weitreichende Kenntnisse über die Funktionsweise von FlexRay haben muss. Das Systemdiagramm ist mit beschrifteten Baugruppensymbolen gestaltet und die Visualisierung der Kommunikation findet über ein simuliertes Oszilloskop statt. Eine anschauliche Darstellung der Kommunikation, die einfache Rück-

[3]Die Abbildungen sowie die Referenzen auf das gezeigte Produkt (RTI FlexRay Blockset) sind nicht mehr auf der Webseite vorhanden. Die dSPACE-Produktbroschüre zählt FlexRay aber weiterhin bei den unterstützten Systemen auf.



Data

Name	Value	Current Min	Current Max	Unit	Bar
CarSpeed	0.50	0.00	94.00	mp/h	
EngSpeed	78	0	6915	rpm	
EngTemp	76	-50	76	degC	
ABSdata::MAX	59.18	0.00	59.18	ms	
ABSdata::MIN	40.79	0.00	46.40	ms	

Bild 3.4: Darstellung der Daten bei CANoe

schlüsse auf die Arbeitsweise der Medienzugriffskontrolle erlauben würde, ist damit nicht möglich.

3.2.3 Mirabilis Design VisualSim FlexRay-Modell

Für das grafische Modellierungs- und Simulationsprogramm *VisualSim* der Firma Mirabilis Design wird ein FlexRay-Modell angeboten. VisualSim ist ein allgemeines Simulationswerkzeug, das beim Entwurf von integrierten Schaltungen, Prozessoren, Echtzeitsoftware und Netzwerksystemen helfen soll. Eine Demonstration des FlexRay-Modells ist auf der Herstellerwebseite als Java-Applet ausführbar [Mir06]. Die echte Anwendung unterscheidet sich laut Mirabilis Design hauptsächlich dadurch, dass auch andere FlexRay-Cluster damit gestaltet und simuliert werden können. Bild 3.7 zeigt ein Bildschirmfoto der Demonstration.

Da die Simulation ohne Hardware auskommt und auch Modelle als Java-Applets verfügbar gemacht werden können, die auf fast jedem Rechner ausgeführt werden können, erschien dieses Programm zunächst am geeignetsten für den Einsatz in der Lehre.

Der grafische Aufbau der FlexRay-Netze verspricht eine einfache Bedienung. Ein Blick auf die Demonstration lässt erkennen, dass dies jedoch nur einen kleinen Teil der Bedienung ausmacht. Die Parametrisierung findet nicht mehr grafisch statt und ist sehr umfangreich. Dies ist bedingt durch das Ziel, eine exakte, vollständige Simulation anzubieten, mit der Situationsanalysen möglich sind, bevor das echte Netz in den Betrieb geht.

Die Simulation ist daher auch nur wenig animiert (lediglich die Aufzeichnung der Daten wird fortschreitend dargestellt) und nicht in ihrer Geschwindigkeit regelbar. Es wird ein definierter Zeitrahmen für die Simulation vorgegeben, und die Simulation dann in der auf dem Rechner des Benutzers maximal möglichen Geschwindikgeit

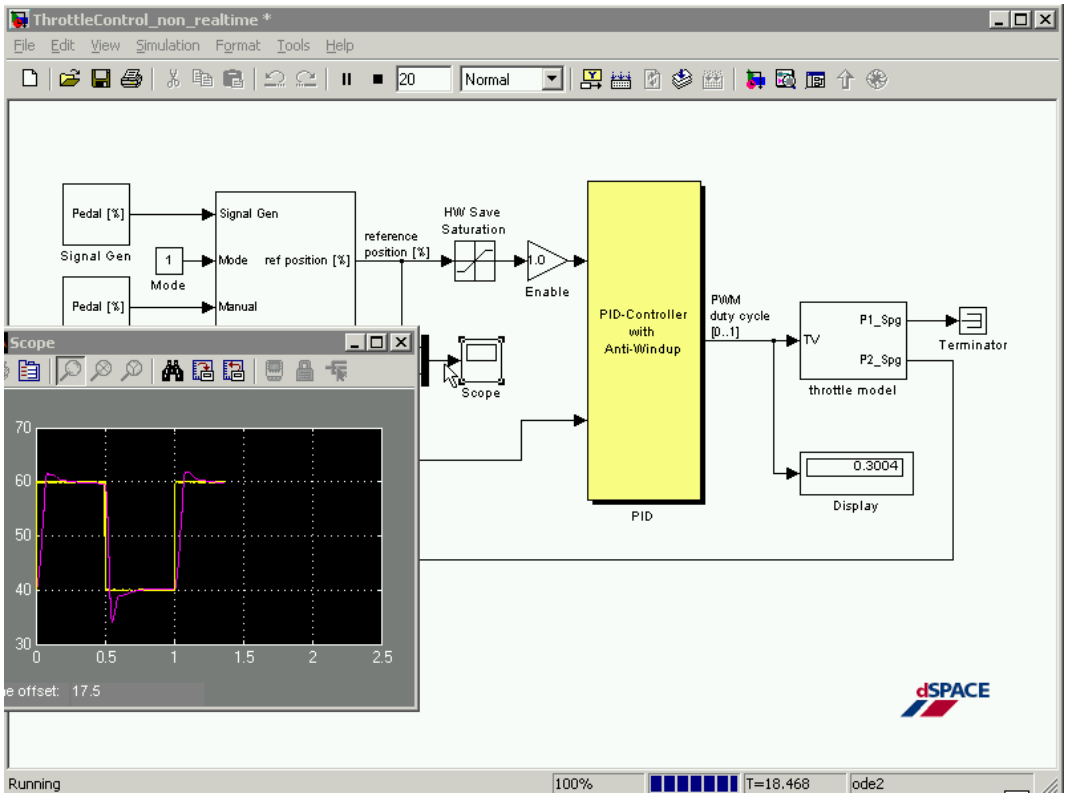


Bild 3.5: FlexRay-Modell für dSPACE-Simulator in Simulink

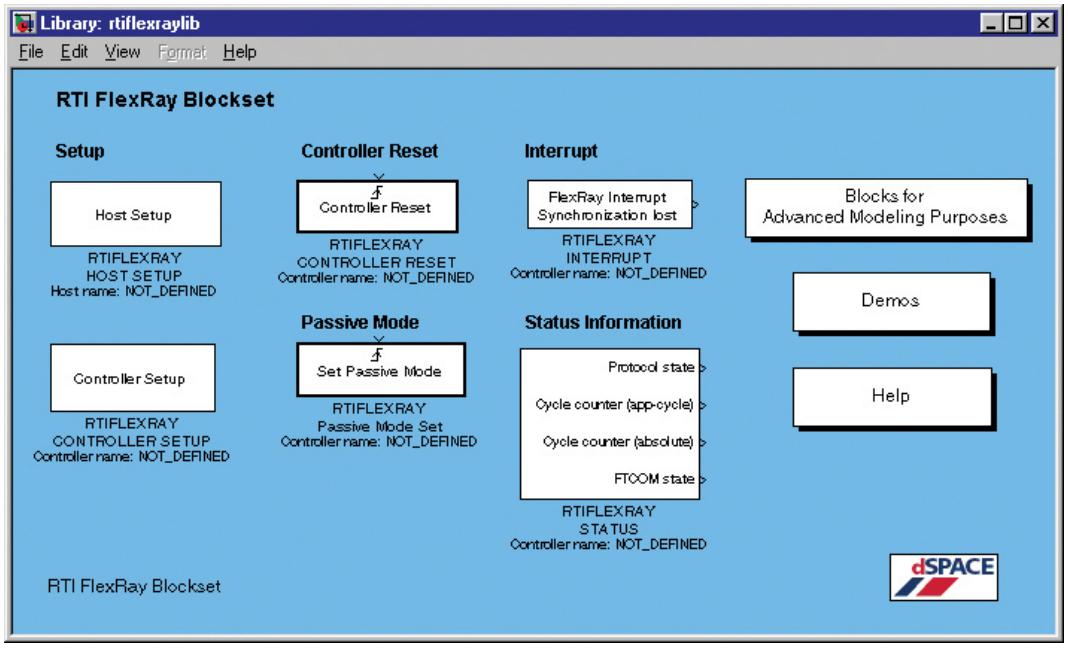


Bild 3.6: FlexRay-Modell für dSPACE-Simulator: Konfiguration

ausgeführt. Nach der Simulation erhält man Einblick in die Daten, die während der Simulation aufgezeichnet wurden. Die einzige grafische Darstellung ist die Anzeige der Kanalaktivität (in der Mitte von Bild 3.7).

Die Funktion der Medienzugriffskontrolle wird nicht gesondert dargestellt. Der Anwender muss in der Lage sein, die Ausgabedaten der Simulation richtig zu interpretieren, um zu verstehen, was passiert ist. Zumindest das Demonstrations-Applet erkennt keine Fehlkonfigurationen und bietet auch keinerlei Rückmeldung, falls der Anwender fehlerhafte Daten eingegeben hat.

3.3 Zusammenfassung

Die in der Lehre eingesetzten Materialien beschränken sich auf Texte und erläuternde Bilder. Um die Studierenden besser in den Lehrprozess einbeziehen zu können, sind darüber hinausgehende Medien wünschenswert.

Spezielle Programme, die das Verständnis für die Funktionsweise von FlexRay oder Teilen daraus zum Ziel haben, existieren nicht. Die Entwicklung von grafischen Anwendungen, die sich als Veranschaulichung verwenden ließen, beschränkt sich auf Werkzeuge zur Konfiguration und Analyse von konkreten FlexRay-Clustern. Entsprechend kompliziert und unübersichtlich stellen sich diese Werkzeuge dem Laien dar.

Durch das spezielle Ziel fehlen diesen Werkzeugen auch wichtige Merkmale, die das

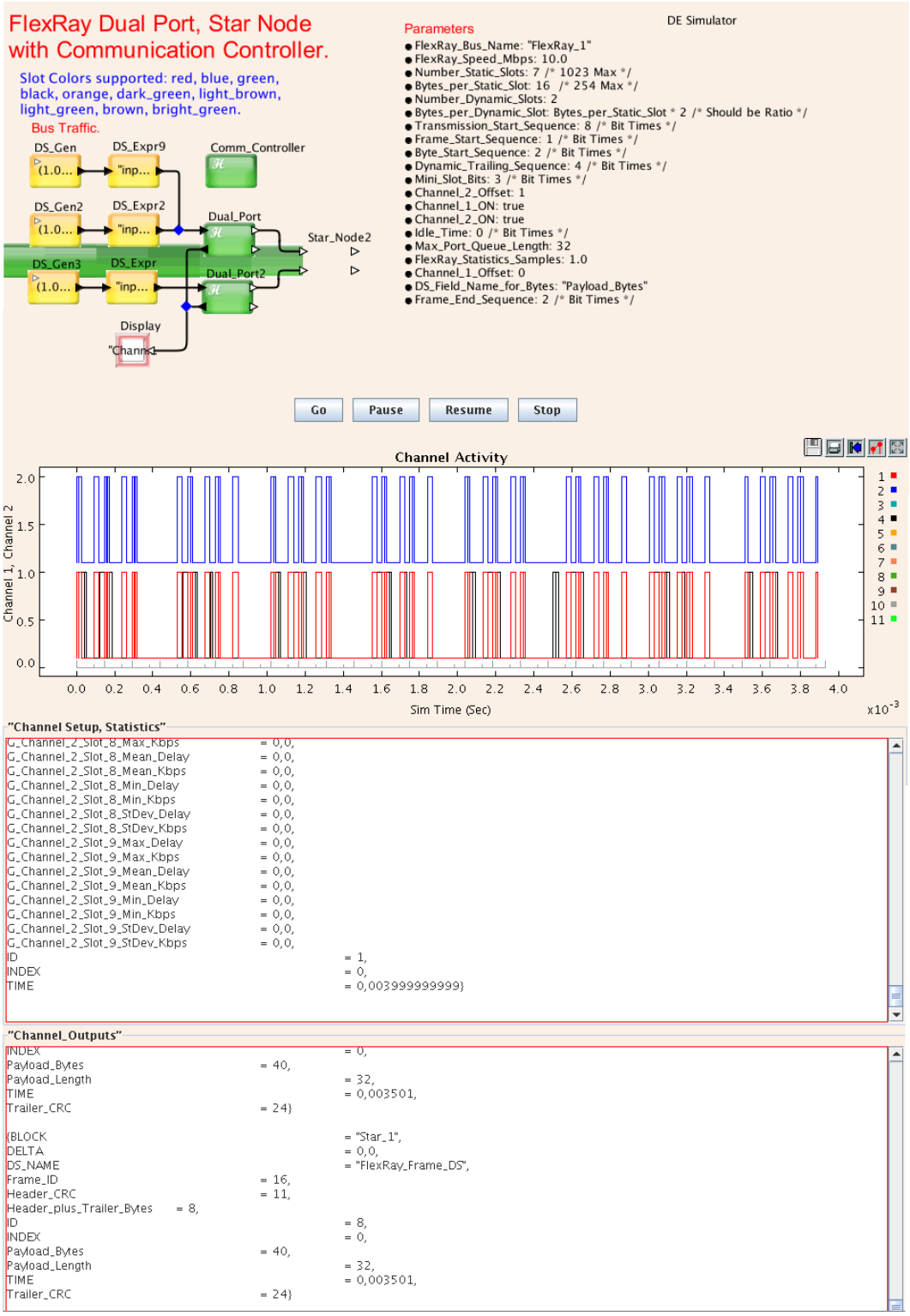


Bild 3.7: Mirabilis Design VisualSim FlexRay-Modell

Verständnis fördern können – es ist sogar so, dass die Bedienung der Werkzeuge bereits Grundlagen voraussetzt, für die diese Werkzeuge ein Anschauungsobjekt in der Lehre sein sollten.

4 Konzeption

In diesem Kapitel beschreibe ich, wie ich die interaktive Visualisierung der Medienzugriffskontrolle beim FlexRay-Protokoll konzipiert habe. Nach einer Präzisierung der Ziele beschreibe ich zunächst die von mir durchgeführte didaktische Reduktion. Das ist die Vereinfachung des Themas auf inhaltlicher und visueller Ebene. Danach beschreibe ich, wie ich die Simulation realisiert habe, und schließe das Kapitel mit meinen Überlegungen zu den eingearbeiteten Interaktionen ab.

4.1 Zielgruppe und Voraussetzungen

Die primäre Zielgruppe für die interaktive Visualisierung der Medienzugriffskontrolle beim FlexRay-Protokoll sind Studierende der Informatik, die vom Grundstudium in das Hauptstudium übergehen. Ein geplanter Einsatz für das Programm ist eine Vorlesung zu Eingebetteten Systemen an der Universität Dortmund. Interessierte aus anderen Fachbereichen oder von außerhalb des universitären Umfeldes sollten das Programm ebenso verwenden können, falls sie die notwendigen Voraussetzungen erfüllen.

Um das Programm nutzbringend verwenden zu können, sind zunächst Grundkenntnisse zu Rechnernetzen erforderlich. Der Benutzer sollte verstehen, was ein Rechnernetz ist, und wie die Kommunikation darin funktioniert. Die Lektüre eines einfachen Buches über Rechnernetze genügt, zum Beispiel [Com04]. Zusätzlich ist es aufgrund der Natur des Programms als begleitendes Lehrmittel und Übungswerkzeug notwendig, zumindest eine einfache Einführung in FlexRay zu bekommen oder einen Begleittext zu haben. Vorgesehen ist der vorlesungsbegleitende Einsatz, bei dem der Dozent bereits mit Hilfe des Programms die notwendigen Grundlagen zur weiterführenden Übung vermitteln kann. Interessenten, die nicht an einer Informatik-Vorlesung teilnehmen, könnten mit einer einfachen Anleitung, wie zum Beispiel der FlexRay-Einführung von IXXAT [IXX06a] oder der FlexRay-Kurzeinführung von TZM [Tra06a], eine Möglichkeit zur Vorbereitung haben.

4.2 Präzisierung der Ziele

Entsprechend der Darstellungen aus Abschnitt 2.2.1 führe ich zunächst auf, was das von mir entwickelte Lehrmittel leisten soll:

- Anschaulichkeit: Die Darstellung der Medienzugriffskontrolle und der zugehörigen Elemente dürfen für den Lernenden nicht schwer zu erfassen sein.
- Erreichbarkeit: Alle wichtigen Parameter zur Beeinflussung der Medienzugriffskontrolle müssen dem Lernenden zugänglich sein.
- Steuerbarkeit: Um dem Lernenden die Beobachtung und dem Dozenten die Präsentation der Vorgänge zu erleichtern, ist eine weitreichende Steuerung des Simulationsvorganges notwendig.
- Übersichtlichkeit und Motivation: Die verwendeten Gestaltungsmittel dürfen nicht von dem eigentlichen Gegenstand ablenken. Dennoch ist eine ansprechende Gestaltung wünschenswert, um die Motivation zu fördern.
- Flexibilität und Vertiefung: Die Einstellungsmöglichkeiten sollen nicht auf bestimmte Szenarien beschränkt sein. Dadurch soll es möglich sein, die Eigenschaften der Medienzugriffskontrolle über die Präsentation hinaus erforschen zu können.

4.3 Vereinfachungen

In diesem Abschnitt beschreibe ich die von mir, ausgehend von der FlexRay-Protokollspezifikation [Fle05c, Fle05d], vorgenommenen Vereinfachungen. Diese Vereinfachungen bedeuten zunächst das Weglassen von nicht relevanten Inhalten:

Auf die Betrachtung des physischen Medienzugriffs habe ich weitestgehend verzichtet. Lediglich die durch den Medienzugriff vorgegebenen Parametergrenzen habe ich aufgegriffen. Die interaktive Visualisierung orientiert sich dabei an der vorhandenen Abstraktion, wie sie Bild 3.1 auf Seite 31 zeigt.

Die Spezifikation ist erwartungsgemäß sehr detailliert und enthält mehr als nur eine Beschreibung der Medienzugriffskontrolle. Die einzelnen Aspekte des Protokolls bauen teilweise aufeinander auf. So ist zum Beispiel die Spezifikation der Medienzugriffskontrolle eng an die Zeitsynchronisierung gekoppelt. Um die Arbeitsweise der Medienzugriffskontrolle darzustellen, sind nicht alle Details aus der Protokollspezifikation zu berücksichtigen und können reduziert oder ganz weggelassen werden. Im Folgenden beschreibe ich, um welche Details es dabei geht, und wie ich mit ihnen umgegangen bin.

Zu Beginn habe ich bestimmt, welche Teile der FlexRay-Spezifikation nicht für die von mir angestrebte Darstellung der Medienzugriffskontrolle wichtig sind. Im Einzelnen sind dies:

- *Protocol Operation Control* (Kapitel 2) Die Protokollsteuerung beschreibt das gesamte FlexRay-Protokoll. Teile daraus sind in der FlexRay-Protokollspezifikation

in eigenen Kapiteln beschrieben. Eines dieser Kapitel ist das Kapitel zur Medienzugriffskontrolle (Kapitel 5). Auf die anderen Kapitel gehe ich in den weiteren Punkten dieser Liste ein. Die restlichen Details zur Protokollsteuerung, die in Kapitel 2 der Spezifikation aufgeführt sind, sind für die getrennte Betrachtung der Medienzugriffskontrolle nicht relevant.Es handelt sich dabei um den Zustandsautomaten der Protokollsteuerung, um Protokollzustände, Protokollkommandos und die Beschreibung des Protokollverhaltens in besonderen Situationen, wie zum Beispiel beim Einschalten des Clusters.

- *Coding and Decoding* (Kapitel 3). Hier wird die Kodierung und Dekodierung beschrieben, die Umsetzung zwischen Kommunikationsstrukturen und zu übertragenden Bitströmen. Um die Medienzugriffskontrolle zu beschreiben, genügen die Kommunikationsstrukturen, weshalb ich die Inhalte dieses Kapitels nicht berücksichtige.

- *Frame Format* (Kapitel 4). Dieses Kapitel enthält die detaillierte Definition des FlexRay-Rahmenformats. Der Rahmen enthält nur eine Information, die für die Medienzugriffskontrolle von Bedeutung ist: den Rahmenbezeichner. Da der Bezeichner vom Knoten vorgegeben wird, kann er mit dem Rest der internen Struktur eines Rahmens zu einem Datenblock abstrahiert werden.

- *Frame and Symbol Processing* (Kapitel 6). Die hierin beschriebene Verarbeitung von Rahmen und Symbolen bezieht sich auf die Ebene zwischen dem *Coding and Decoding* und dem *Controller Host Interface*. Es geht dabei um eine protokollinterne Verarbeitung der Daten, bevor sie in Richtung der Anwendungsschicht weitergegeben werden. Dies beeinflusst die Medienzugriffskontrolle nicht.

- *Wakeup and Startup* (Kapitel 7). Das Kapitel beschreibt die Vorgänge beim Einschalten des Clusters und beim Eingliedern eines Knotens in den Cluster. Die Medienzugriffskontrolle des FlexRay-Protokolls ist zu diesem Zeitpunkt noch nicht aktiv. Die Simulation des Clusterstarts ist daher nicht notwendig. Ich setze für meine Simulation den Cluster als gestartet voraus.

- *Clock Synchronization* (Kapitel 8). Die hierin beschriebenen zeitlichen Strukturen, wie sie Bild 2.1 auf Seite 9 zeigt, sind für die Medienzugriffskontrolle wichtig, das genaue Verfahren zur Zeitsynchronisierung jedoch nicht. Ich setze eine funktionierende Synchronisierung als gegeben voraus und beschränke mich daher auf die Macroticks.

- *Controller Host Interface* (CHI) (Kapitel 9). Das CHI ist die Schnittstelle zwischen Protokoll- und Anwendungsebene. Auf dieser Ebene hat die Medienzugriffskontrolle längst stattgefunden. Das CHI ist für ihre Betrachtung daher nicht wichtig.

Zur näheren Betrachtung bleiben im Grunde das Kapitel über die Medienzugriffskontrolle sowie Teile der Kapitel über die Zeitsynchronisierung und die Protokollsteuerung übrig. Die Anhänge, die die Systemparameter und Konfigurationsbedingungen beschreiben, enthalten weitere Informationen, die gebraucht werden, um ein Simulationsmodell für die FlexRay-Medienzugriffskontrolle zu erstellen.

Die Beschreibung der Medienzugriffskontrolle enthält Zeitvorgaben, die notwendig sind, um den physischen Medienzugriff zu steuern. Anhand des Zeitverhaltens im statischen Segment erläutere ich diese Zeitvorgaben, um zu zeigen, weshalb sie für das Verständnis der Funktionsweise der Medienzugriffskontrolle keine Bedeutung haben (Bild 4.1).

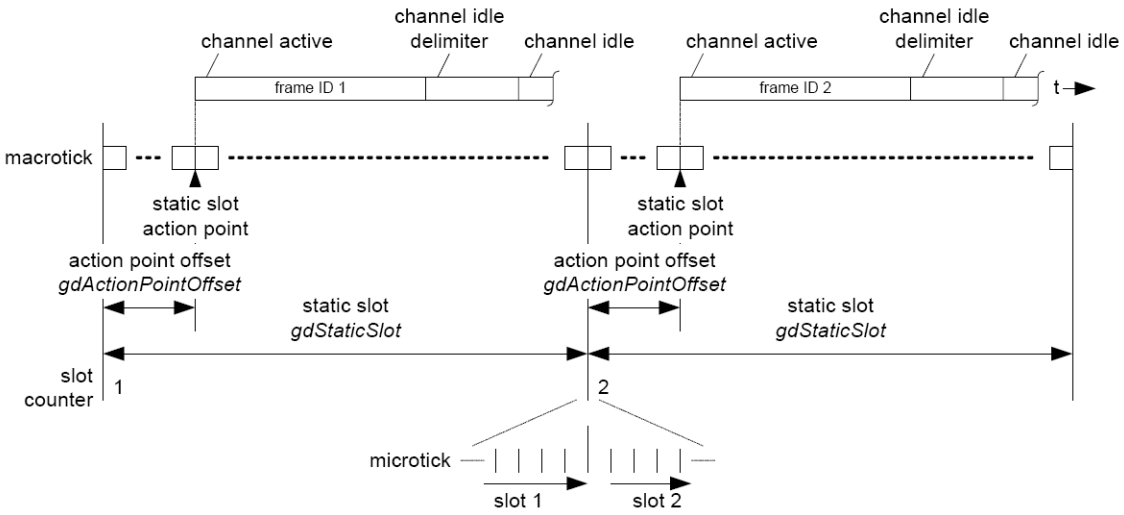


Bild 4.1: Zeitverhalten im statischen Segment. Quelle: [Fle05c].

Die ganz links liegende Linie stellt praktisch den Anfang eines statischen Slots dar. Der Slot-Zähler wurde auf eins gesetzt und es beginnt eine Wartezeit der Länge *gdActionPointOffset* bis zu dem bereits in Abschnitt 2.1.6 erwähnten *action point*, der denjenigen Zeitpunkt bestimmt, an dem der Kommunikationscontroller angewiesen wird, seine Übermittlung zu beginnen. Die Verzögerung zwischen Slot-Anfang und *action point* ist abhängig von den auf dem Bus auftretenden Signallaufzeiten und den angenommenen Schwankungen derselben (die *Präzision* des Clusters). Nach dem *action point* beginnt oben im Bild der mit *channel active* beschriftete Teil, der, passend zum Slot, die Bezeichnung *frame ID 1* (Rahmenbezeichner 1) enthält. Während dieses Teils findet die Datenübertragung statt. Sie endet mit dem *channel idle delimiter*, dem die *channel idle* Phase folgt, die den Rest der Zeit im statischen Slot belegt. Mit Hilfe der *channel idle* Phase erkennen die Knoten das Ende des aktuellen Kommunikationselements.

Wie bereits in Abschnitt 2.1.6 beschrieben, sind hierbei nur zwei Eigenschaften für die Medienzugriffskontrolle wichtig:

1. Slot-Nummer und Rahmenbezeichner müssen übereinstimmen.

2. Die Slot-Nummer kann von den Knoten gezählt werden.

Es ist deswegen unnötig, den detaillierten Zeitverlauf darzustellen oder sogar zu simulieren, nur um die Medienzugriffskontrolle vorzuführen. Aus demselben Grund nehme ich die Dauer der NIT als konstant an: Die variable Dauer der NIT ist, wie ich auf Seite 13 beschrieben habe, ausschließlich für die Zeitsynchronisierung wichtig. Eine Darstellung der für die Synchronisierung wichtigen Details sehe ich daher nicht vor. Diese Details würden zudem die Aufmerksamkeit des Lernenden von der Medienzugriffskontrolle ablenken.

Mit der alleinigen Darstellung von gesendeten Rahmen und Slot-Zuordnungen lässt sich die Medienzugriffskontrolle zwar bereits erläutern, jedoch ist dies noch relativ abstrakt. Die Bedeutung der Konsequenzen, die durch die Funktion der Medienzugriffskontrolle entstehen, kann durch die Einbettung in einen Anwendungskontext verdeutlicht werden (situiertes Lernen, Abschnitt 2.2.2). Ein Teil dieses Kontextes ist, wie auch bei den in der Lehre bereits eingesetzten Bildern, die vereinfachte Darstellung eines FlexRay-Clusters. Ein weiterer Teil, der insbesondere bei einer laufenden Simulation wichtig wird, ist die Anzeige der Datenübertragung, um die Konsequenzen der Medienzugriffskontrolle auf das Senden und Empfangen von Daten im dynamischen Segment zu verdeutlichen.

Das Fehlen eines Nachrichtenkonzeptes (das FlexRay-Protokoll überlässt dies einer höheren Schicht) erschwert diese Präsentation. Erst durch Nachrichten, die länger als FlexRay-Rahmen sein können, und die dadurch entstehende Möglichkeit der Fragmentierung, ist eine Darstellung der Priorisierung im dynamischen Segment möglich. Ohne solche Nachrichten würde die Priorisierung nur durch das fallweise Auslassen von Rahmen gezeigt werden können. Die variable Länge der Rahmen spielt bei der Priorisierung zwar ebenfalls eine Rolle, aber sie muss ohnehin gezeigt werden und macht die Bedeutung der Priorisierung für die Anwendung nicht deutlicher. Deshalb sehe ich vor, in meinem Programm Nachrichten darzustellen.

Das statische Segment ist bei FlexRay für zeitkritische Steuerdaten vorgesehen. Zum Einen nehme ich an, dass in der Praxis hier nur kurze Nachrichten versendet werden, die in einen Rahmen hinein passen. Zum Anderen wird der im statischen Segment durch Fragmentierung auftretende Effekt nicht durch die Funktion der Medienzugriffskontrolle beeinflusst, sondern nur durch die ihre Grundkonfiguration und die, nicht durch das Protokoll vorgegebene, Zusammenstellung der versendeten Nachrichten. Sei *staticSlots* die Anzahl der statischen Slots, die ein Knoten innerhalb des statischen Segments eines Kommunikationszyklusses zugewiesen bekommen hat. Nehmen wir nun an, die Cluster-Konfiguration sehe vor, dass der Knoten alle diese Slots

ausnutzt, um pro Kommunikationszyklus möglichst viele Fragmente derselben Nachricht zu übertragen. Dann entspricht dies aus der Sicht der Medienzugriffskontrolle der Übertragung von *staticSlots* vielen Nachrichten von der Länge je eines statischen Slots. Aus der Sicht der Anwendung verlängert sich die Dauer der Übertragung einer vollständigen Nachricht bei jeder Vergrößerung der Nachricht über eine Slot-Länge hinweg um den Abstand zwischen dem zuletzt verwendeten und dem nächsten verfügbaren Slot. Beispiel: Verwendet ein Knoten die Slots 3 und 5 im statischen Segment, dann verlängert sich die Übertragungsdauer einer drei Slots langen Nachricht gegenüber einer zwei Slots langen Nachricht um die Dauer eines Kommunikationszyklusses zuzüglich der Dauer eines Slots. Der Knoten kann im ersten Zyklus zwei Rahmen übertragen (in Slot 3 und 5), und im zweiten Zyklus – eine Zyklusdauer später – kann er den dritten Rahmen übertragen (in Slot 3), was eine weitere Slot-Dauer beansprucht. Die Übertragungsdauer einer vier Slots langen Nachricht ist gegenüber der drei Slots langen Nachricht um zwei Slot-Längen größer (Slot 4 wird übersprungen und Slot 5 wieder genutzt). Da das statische Segment von der Medienzugriffskontrolle während der Clusterlaufzeit nicht verändert wird, sind die Abstände zwischen den Slots konstant. Die Veränderung der Nachrichtenlänge wirkt sich daher nur auf die Anwendungsebene aus, ohne von der Medienzugriffskontrolle beeinflusst zu werden. Da ich in meinem Programm die Funktion der Medienzugriffskontrolle zeigen will, und nicht die praktischen Auswirkungen der Konfiguration auf Anwendungsebene, habe ich eine gesonderte Behandlung von Nachrichten im statischen Segment nicht vorgesehen. Mit Hilfe dieses Nachrichtenkonzeptes ist es mir möglich, auf eine eigene Darstellung für Rahmen zu verzichten: Im statischen Segment entsprechen die Nachrichten den Rahmen und im dynamischen Segment kommen noch die Fragmente hinzu.

Bei der Struktur des Kommunikationszyklusses lasse ich das Symbolfenster weg, weil eine Medienzugriffskontrolle darin nicht stattfindet [Fle05c, Abschnitt 5.1.5]. Da das Symbolfenster optional ist, ist ein Kommunikationszyklus ohne Symbolfenster immer noch vollständig.

Obwohl die NIT für die Medienzugriffskontrolle nicht wichtig ist, habe ich mich dafür entschieden, sie darzustellen. Der Kommunikationszyklus ist ohne NIT nicht vollständig. Aus Gründen der Visualisierung ist die Darstellung der NIT ebenfalls vorteilhaft: Dadurch wird ein ruhiger Blick auf den Sendefortschritt der eventuell fragmentierten Nachrichten im dynamischen Segment möglich. Und schließlich bietet die NIT dem Beobachter bei kontinuierlich ablaufender Simulation eine kurze Pause, in der er seine Aufmerksamkeit auf den Anfang des nächsten Zyklusses lenken kann.

Wie schon in der Beschreibung der Feldbusse unterscheide ich nicht zwischen dem Knoten und seinen Bestandteilen. Bei der interaktiven Visualisierung bedeutet dies weiter, dass die Knoten repräsentativ für die Geräte stehen, die den FlexRay-Netzwerkknoten verwenden. Dadurch werden keine für die Medienzugriffskontrolle unnötigen Zwischenschichten eingebaut und die Abstraktion von Rahmen auf Nachrichten ist einleuchtender.

4.4 Visualisierung

Die Aufgabe der interaktiven Visualisierung der Medienzugriffskontrolle besteht darin, dem Beobachter einen unsichtbaren Vorgang und seine Konsequenzen näherzubringen. Die in der Lehre bereits verwendeten abstrakten Darstellungen eignen sich als Grundlage für dieses Vorhaben. Ihre Abstraktion des zeitlichen Verlaufs eines Kommunikationszyklusses ist eine Momentaufnahme der Kommunikation aus der Perspektive eines Außenstehenden (Bild 3.1). Eine versetzte Darstellung des Kommunikationsinhaltes wie in Bild 3.2 lehne ich aufgrund ihres zu erwartenden Platzbedarfes ab. Der wachsende Umfang der Darstellung bei steigender Anzahl von Knoten verringert zudem ihre Übersichtlichkeit, die für das Verständnis aber notwendig ist.

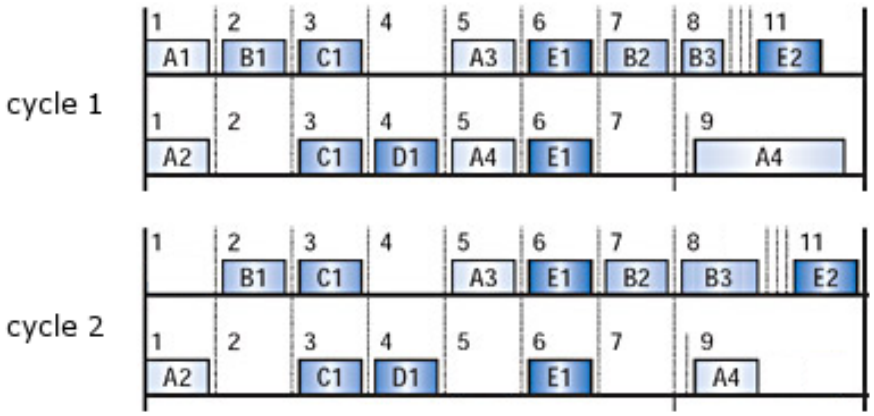


Bild 4.2: Einzelbild-Darstellung der Medienzugriffskontrolle. Erstellt aus [IXX06b].

Ein erster Ansatz für eine dynamische Darstellung des zeitlichen Verlaufs könnte sein, diese Darstellung wiederholt mit unterschiedlichen Medienzugriffssituationen zu zeigen (Bild 4.2). Dieser Ansatz ist aber zu einfach: Er würde dem Beobachter die eigentliche Arbeitsweise der Medienzugriffskontrolle während eines Kommunikationszyklusses vorenthalten und nur ihre Resultate je Zyklus anzeigen.

Meine nächste Idee hierzu war, dass der verhandelte Kommunikationszyklus je Kanal angezeigt, und der Verlauf der Zeit durch einen beweglichen Pfeil realisiert wird. Dies habe ich in einer ersten Konzept-Grafik festgehalten (Bild 4.3). In diesem Stadium der Konzeption war mir noch nicht klar, dass dieser Ansatz die Wirklichkeit nur ungenau wiedergibt und auch schwieriger zu einer Interaktion zu verarbeiten ist, da der Aufbau des dynamischen Segments gar nicht zu Beginn des Kommunikationszyklusses bekannt ist. Ich habe mich deshalb später dafür entschieden, den Zyklus aus Sicht eines zusätzlichen FlexRay-Knotens darzustellen, der nur als Beobachter fungiert und niemals sendet.

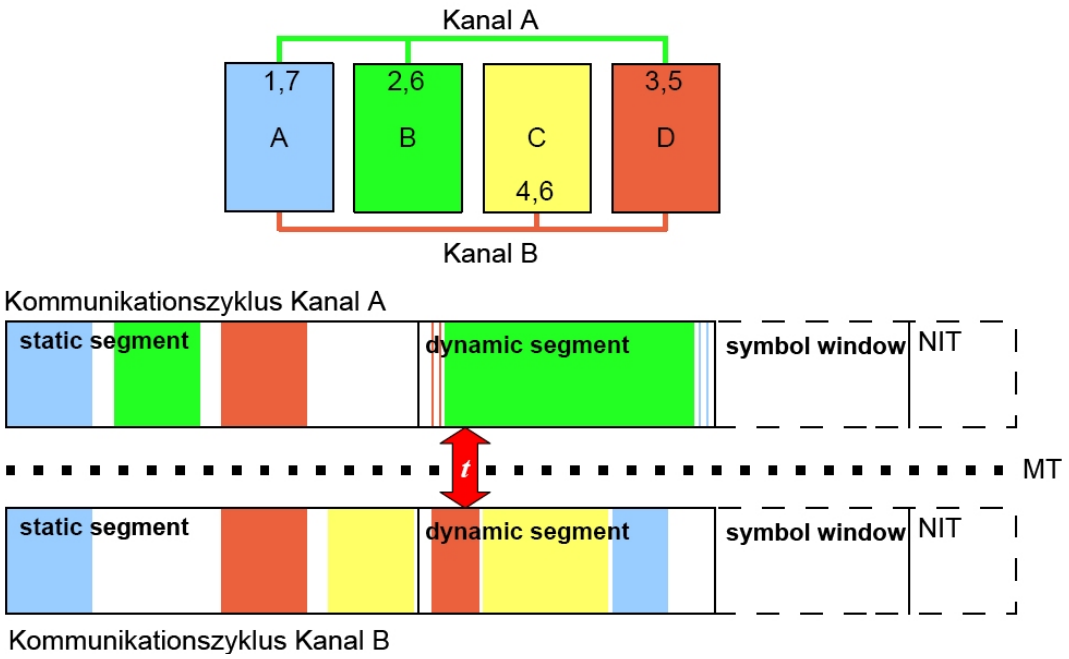


Bild 4.3: Konzeptgrafik der interaktiven Visualisierung

Ein FlexRay-Knoten „sieht“ den Kommunikationsfortschritt innerhalb eines Kommunikationszyklusses mit einer Genauigkeit von einem Macrotick. Dabei soll die Anzeige der dynamischen Slots stückweise aus Minislots zusammengesetzt werden, da dies dem Wissen der Knoten enspricht: Sie kennen die Dauer der Minislots, und bekommen das Ende eines dynamischen Slots mit. Während der Übertragung können sie daher die bisherigen Größen der dynamischen Slots mit der Genauigkeit von einem Minislot bestimmen. Die statischen Slots können aufgrund ihrer festen Größe stets komplett angezeigt werden. Es ist im statischen Segment zwar auch der Beginn eines jeden Slots bekannt, doch um die Dynamik des Kommunikationsvorgangs zu verdeutlichen, habe ich mich dafür entschieden, die statischen Slots auch immer erst bei Erreichen des zugehörigen Zeitabschnitts anzuzeigen.

Dem Bild ist auch die einheitliche Färbung der Slots und der zugehörigen Knoten zu entnehmen. Die Zahlen für die Slotnummern habe ich später noch entfernt, da die Zuordnung eindeutig ist und die konkreten Nummern für die Präsentation der Funktionalität keine Rolle spielen. Die Slots innerhalb des Kommunikationszyklusses sollen nicht nur während der Kommunikation, sondern auch beim Entwurf des Zugriffsplans in den Knotenfarben dargestellt werden.

Um den Zusammenhang zwischen Methode und Wirkung der Methode zu verdeutlichen, stelle ich den eigentlichen Medienzugriff in abstrakter Weise dar. Dies geschieht durch Markierung der Linien, die das Medium symbolisieren, in der Farbe desjeni-

gen Knotens, der gerade senden darf. So wird dem Beobachter gezeigt, was eigentlich kontrolliert wird.

Die auf Seite 45 eingeführten Nachrichten und ihre Fragmente sollen außerhalb der Darstellung des Kommunikationszyklusses, am besten in der Nähe des Knotens, zu dem die Nachricht gehört, angezeigt werden. Für die Betrachtung der Medienzugriffskontrolle kann sich dies auf die Darstellung der Dauer der Nachrichten und ihrer Fragmente beschränken. Eine direkte Anzeige der Nachrichten und ihren Transport innerhalb des Clusters, wie dies zum Beispiel bei den Animationen von Krumm [Kru06] der Fall ist, in denen sie durch Briefumschläge symbolisiert werden, die sich entlang der Verbindungen bewegen, ist für die FlexRay-Medienzugriffskontrolle nicht notwendig. Anders als bei den Animationen von Krumm zeige ich keine Algorithmen, die auf dem Laufzeitverhalten der Nachrichten auf dem Bus oder deren Eintreffen bei den Knoten aufbauen. Eine einfache Präsentation der Vollständigkeit der Nachricht genügt, um den Fragmentierungseffekt erkennbar zu machen. Eine Idee hierfür ist die Anzeige eines horizontalen Balkens, der die Nachrichtendauer repräsentiert. Innerhalb des Balkens sollen dann Markierungen für den Sendefortschritt und die Fragmente angebracht werden.

Um den Benutzer nicht mit einer Vielfalt an Ansichten zu überfordern, soll die Konfiguration der Simulation möglichst in ihre Darstellung eingebettet werden. Bei der Slot-Zuordnung kann dies direkt über die Anzeige des Kommunukationszyklusses erfolgen. Abstrakte Tabellenansichten für die Slot-Zuordnung, die visuell weit vom Kommunikationszyklus und dem Cluster entfernt sind (Bild 4.4), wollte ich vermeiden.

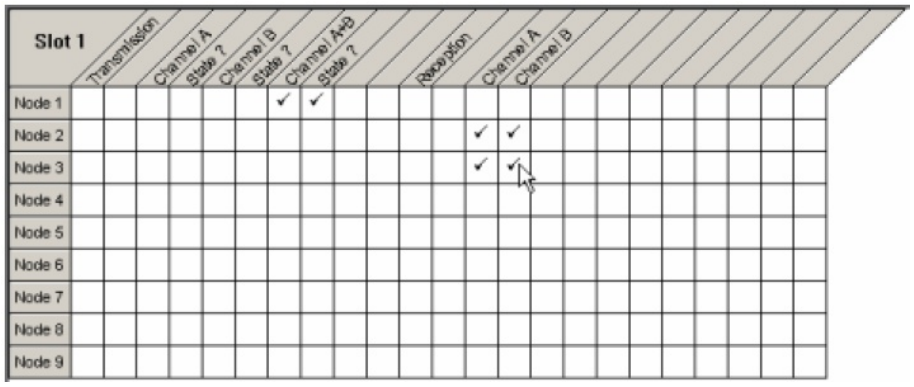


Bild 4.4: Slot-Zuordnung bei TZM FlexConfig. Quelle: [Tra06b].

Die Darstellung von verwendeten Slots durch in der Knotenfarbe ausgefüllte Kästchen, wie bei den existierenden Veranschaulichungen, wollte ich in mein Programm übernehmen. Die verfügbaren Bilder unterscheiden jedoch nicht zwischen „ein Slot hat keine Knotenzuordnung“ und „der Knoten, der diesem Slot zugeordnet ist, sendet

nicht“. Da ein leeres Rechteck keine Informationen enthält – in diesem Fall über einen Knoten – entscheide ich mich hierbei dafür, die leeren Stellen als „ein Slot hat keine Knotenzuordnung“ zu verwenden. Der andere Fall kann durch ein leeres, in der Knotenfarbe umrahmtes Rechteck dargestellt werden. In der ersten Konzeptgrafik (Bild 4.3) ist dies bereits im dynamischen Segment angedeutet.

Zuletzt sehe ich in meinem Konzept allgemein vor, dass die Darstellung ansprechend sein und Auflockerungen enthalten soll, um dem Lernenden einen gewissen „Spaß-Faktor“ zu bieten. Diese Dinge können einfache, die Ästhetik steigernde Gestaltungsdetails oder sparsam verwendete Effekte sein, damit das Programm sich ein klein wenig wie ein Spielzeug anfühlt. Dadurch soll der Lernende unterschwellig dazu ermuntert werden, mit dem Programm herumzuspielen. Wichtig ist dabei, dass die Effekte nicht zum Selbstzweck geraten und eventuell auch abschaltbar sind, falls sie störend werden.

4.5 Simulation

Durch die enge Kommunikationsstruktur bei FlexRay und die global etablierten Macroticks, die clusterweit eine kleinste gemeinsame Schrittgröße darstellen, auf der alle anderen Strukturen aufbauen, ist die zeitdiskrete Simulation für die Aufgabe geeignet. Zudem geht es nicht darum, die Simulation möglichst schnell ablaufen zu lassen, um danach die Ergebnisse zu visualisieren. Die Daten sollen während der Simulation für die grafische Repräsentation und das Feedback an den Benutzer eingesetzt werden. Die Simulation soll durch Einzelschrittsteuerung die einzelnen Schritte beobachtbar machen können, daher ist eine Simulation auf dieser Ebene ohnehin erforderlich.

4.5.1 Modell

Als Grundlage für die Implementierung des Protokolls enthält die FlexRay-Protokollspezifikation bereits eine modellartige Beschreibung der FlexRay-Medienzugriffskontrolle. Ich habe daher die Spezifikation als Modellgrundlage verwendet, wobei ich die in Abschnitt 4.3 besprochenen Anpassungen vorgenommen habe.

Dadurch, dass die Simulation nicht als Diagnose- oder Entwicklungswerkzeug eingesetzt werden soll, sondern als Antrieb für eine Visualisierung, war es mir auch möglich, auf eine detaillierte mathematische Modellierung des FlexRay-Clusters zu verzichten. Es genügt ein einfaches Modell, das direkt auf die syntaktischen und semantischen Strukturen der verwendeten Programmiersprache Java und den Möglichkeiten der visuellen Gestaltung aufbauen kann. Ich habe ein hierarchisches Modell gewählt, bei dem der FlexRay-Cluster als Behälter für die Knoten und als Verwaltungsinstanz für alle Cluster-globalen Daten dient. Bild 4.5 zeigt einen Überblick über diese Struktur. Im Folgenden verwende ich „Cluster“ als Bezeichnung für das in dem Bild gezeigte Cluster-Element ohne die zugehörigen Knoten, im Gegensatz zu einem FlexRay-Cluster, der eine Ansammlung von miteinander verbundenen FlexRay-Knoten ist. Die Nachrichten

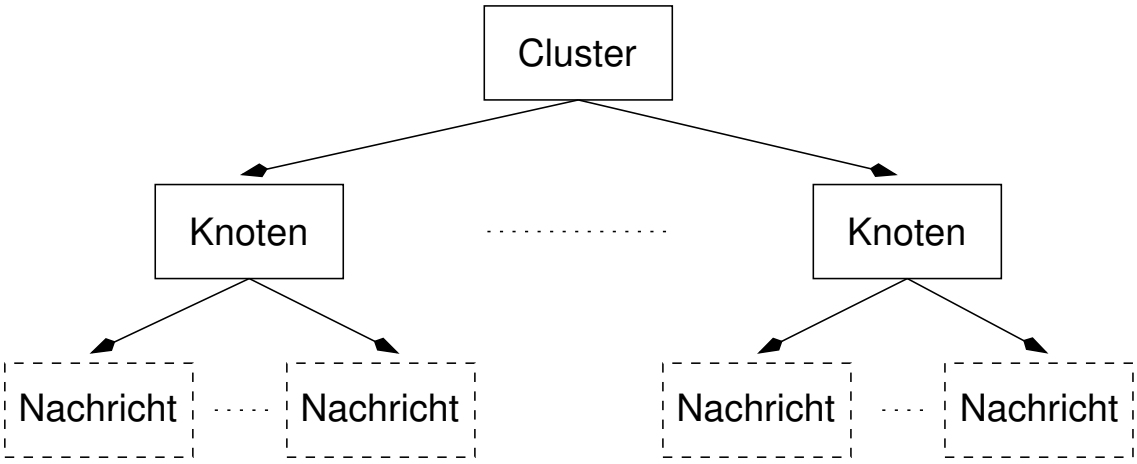


Bild 4.5: Modellstruktur für die Simulation

habe ich mit gestrichelten Kästchen umrandet, da die Beziehung zwischen Knoten und Nachrichten eine zeitliche ist, und keine materielle, wie dies bei Cluster und Knoten der Fall ist. Während die Knoten zu dem Cluster gehören und dadurch den FlexRay-Cluster repräsentieren, ist die Anzahl der hierarchisch unter den Knoten angeordneten Nachrichten variabel. Zu Beginn der Kommunikation gibt es keine und während der Kommunikation steigt die Anzahl der Nachrichten. Dennoch habe ich die Nachrichten in die Modellhierarchie aufgenommen, da sie selbst ihre Fragmente verwalten sollten.

Der Cluster verwaltet alle Knoten und sämtliche Cluster-globalen Daten und kann diese der Visualisierung zur Verfügung stellen. Dabei geht es um folgende Daten:

- Anzahl der Kanäle
- Anzahl der Knoten im Cluster
- Dauer eines Minislots
- Anzahl der Minislots
- Dauer eines statischen Slots
- Dauer der NIT
- Anzahl der Macroticks pro Zyklus
- Zähler für den aktuellen Macrotick
- Zähler für den Kommunikationszyklus
- Zähler für die aktuellen Slots in beiden Kanälen

- Status der Kommunikation im Kommunikationszyklus
- Zustand der Kanäle

Die Zähler für Macrotick, Kommunikationszyklus und Slotnummern gehören nach der Protokollspezifikation in die einzelnen Knoten. Bei meiner Simulation bin ich davon ausgegangen, dass die Macroticks vorgegeben sind. Die Macroticks müssen daher von den Knoten nicht selbst berechnet und infolgedessen auch nicht dort gespeichert werden. Der Cluster kann diese Funktion stellvertretend in meiner Simulation übernehmen. Das erlaubt mir auch, den Cluster als Datenlieferant für die Realisierung der Sichtweise des zusätzlichen Beobachter-Knotens zu verwenden (siehe Seite 47), ohne ein gesondertes Objekt hierfür einzuführen. Ähnlich verhält es sich mit den Zählern für Kommunikationszyklus und Slots. Diese sind von der Synchronisierung und vom Geschehen auf dem Bus abhängig. Letzteres ist durch den Kommunikationszyklus bestimmt, der bei einem FlexRay-Cluster für alle Knoten gleich ist. Eine Speicherung der diesbezüglichen Daten innerhalb jedes Knotens ist daher unnötig. Die Verwaltung durch den Cluster genügt. Anders als durch die FlexRay-Protokollspezifikation vorgegeben, zähle ich die Macroticks durchgängig über die Zyklen hinaus und setze sie nicht am Anfang jedes Zyklusses wieder auf Null. Dadurch entfällt der Bedarf für einen zusätzlichen Zähler, mit dem jeder Simulationsschritt identifiziert werden kann. Ein solcher Zähler ist erforderlich, um Zustandsinformationen für jeden einzelnen Simulationsschritt geordnet speichern zu können. Die Zustandsinformationen pro Schritt werden dazu benutzt, um die Simulation rückwärts und danach wieder in identischer Weise vorwärts laufen lassen zu können. Diese andere Zählweise der Macroticks kann durch die Visualisierung leicht angepasst werden, indem immer die Macroticks modulo der Länge des Kommunikationszyklusses angezeigt werden.

In den Knoten brauchen nur diejenigen Daten gespeichert zu werden, die auch in jedem Knoten verschieden sein können. Dies sind die Listen der Rahmenbezeichner pro Kanal, die für die Slot-Zuordnung gebraucht werden, die Information darüber, an welchen Kanälen der Knoten angeschlossen ist (Parameter *pChannels* in der Protokollspezifikation) und die gerade zu sendende Nachricht im dynamischen Segment.

Zur Verwaltung der Nachrichten habe ich Nachrichtenobjekte entworfen, die die nötigen Informationen hierzu selbst verarbeiten. Bei der Erstellung einer Nachricht bekommt diese die aktuelle Nummer des Kommunikationszyklusses, eine Nachrichtenkennung und eine Länge, gemessen in Minislots, zugewiesen. Die Nachrichtenobjekte enthalten Daten über den Sendefortschritt und pflegen diese während der Simulation selbst. Diese Daten sind: Die Anzahl der Fragmente und die Zeitpunkte der Fragmentgrenzen, der aktuelle Sendefortschritt der Nachricht in Macroticks und die Information darüber, ob die Nachricht bereits komplett übertragen wurde. Zeiten, in denen keine Übertragungen stattfinden (Untätigkeit im Cluster), modelliere ich mit speziellen leeren Nachrichten. Auf diese Weise wird mir ermöglicht, die Untätigkeit auf dem Bus zusammen mit den Nachrichten einheitlich zu verwalten.

4.5.2 Arbeitsweise

Grundlage für die Arbeitsweise ist eine Simulationssteuerung, mit der Einzelschritte vor und zurück durchgeführt werden können. Die Möglichkeit, kontinuierlich Vorwärtsschritte auszuführen, soll ebenfalls vorhanden sein. In jedem Schritt der Simulationssteuerung soll nacheinander ein Simulationsschritt des Clusters ausgeführt und dann die Visualisierung aktualisiert werden. Ein Schritt soll dabei einem Macrotick entsprechen. Zur Regelung der Geschwindigkeit bei der kontinuierlichen Generierung von Schritten muss eine veränderbare Verzögerung zwischen den einzelnen Schritten vorhanden sein. Mein Konzept sieht vor, dass die Macroticks – also die Simulationsschritte – in der Hierarchie des Modells heruntergereicht werden, das heißt, dass der Cluster zunächst notwendige Berechnungen anstellt und dann jeden Knoten einen Macrotick ausführen lässt. Ein Knoten stellt für jeden Schritt wiederum die für ihn notwendigen Berechnungen an, und lässt sein aktuelles Nachrichtenobjekt einen Macrotick ausführen. Bei der Vorwärtsbewegung soll dabei in jedem Schritt der Status der Simulation gespeichert werden, so dass ein Zurückschreiten und anschließendes Voranschreiten einfach und mit gleichem Ergebnis wiederholbar ist.

Die vom Cluster ausgeführten Berechnungen dienen der Aktualisierung der Clusterglobalen Variablen. Dieses sind die Nummer des aktuellen Macroticks und die Slot-Nummern je Kanal. Die Berechnungen im Knoten bestehen in der Verwaltung der Übertragungen pro Kanal. Die Knoten erzeugen bei Bedarf neue Nachrichtenobjekte. Die Nachrichtenobjekte aktualisieren mit jedem Macrotick den Fortschritt ihrer eigenen Übertragung, indem sie intern einen Zähler dafür erhöhen. Die Fragmentierung wird ebenfalls in den Nachrichtenobjekten vorgehalten.

4.5.3 Konfiguration

Die Konfiguration der Simulation besteht zunächst aus der Zusammenstellung eines Clusters. In einem grafischen Editor sollen mehrere Knoten und ein oder zwei Kanäle zu einem Cluster zusammengefügt werden. Der Kommunikationszyklus wird anfangs auf die in der FlexRay-Protokollspezifikation angegebenen Minimalwerte konfiguriert: Ein Kanal, zwei statische Slots, die je vier Macroticks dauern, Minislots der Länge zwei, kein dynamisches Segment und eine NIT der Länge zwei. Alle genannten Parameter sollen veränderbar sein. Darüber hinaus ist die Zuordnung zwischen Knoten und Slots konfigurierbar. Fehlkonfigurationen sollen aktiv vermieden werden. Dies soll hauptsächlich dadurch geschehen, dass alle Konfigurationen absolut sind. Das bedeutet zum Beispiel, dass eine Zuordnung zwischen Slot und Knoten eine bestehende Zuordnung aufhebt. Falls der Slot zuvor auf beiden Kanälen mit verschiedenen Knoten konfiguriert war, die je nur an einem Kanal angeschlossen sind, und der neue Knoten an beide Kanäle angeschlossen ist, dann sollen beide bestehenden Zuordnungen aufgehoben werden. Je nach Erkennbarkeit und Konsequenzen des Konfliktes soll der Lernende auf den Konflikt hingewiesen werden und seine Aktion bestätigen. Dies

ist ein Feedback wie in Abschnitt 4.6.3 beschrieben, auch wenn die Konsequenz nicht direkt sichtbar wird. Die Anzeige des Konfliktes genügt. Diese Entscheidung, keine Konfigurationskonflikte zuzulassen, habe ich getroffen, weil das Ziel des in dieser Arbeit entwickelten Programms darin besteht, die Funktionsweise des Verfahrens zur Medienzugriffskontrolle zu zeigen, und nicht die korrekte Anwendung des FlexRay-Protokolls.

4.6 Interaktion

Die Interaktion ist der dynamische Anteil des Modells, der in direktem Kontakt zum Benutzer steht. Wie ich in Abschnitt 2.2.3 ab Seite 22 bereits herausgestellt habe, ist bei der Bereitstellung von interaktiven Merkmalen, die das Lernen fördern sollen, die Beachtung ihrer Beschaffenheit notwendig. Bei dem im Rahmen dieser Arbeit entstandenen Programm gibt es mehrere Arten interaktiver Bestandteile, deren Konzeption ich im Folgenden einzeln darstelle. Dies sind das Tutorial, die Steuerung der Simulation und das Feedback für den Lernenden.

4.6.1 Tutorial

Eine der Interaktionen für den Benutzer besteht in einem einfachen Tutorial, das ihn in die Benutzung des Programms einführt. Das Tutorial führt den Lernenden schrittweise durch die Erstellung eines einfachen Modells eines FlexRay-Clusters bis hin zur ersten Simulation. Dies ist ein zielführendes Szenario, ähnlich den Methoden, die im Rahmen des situierten Lernens vorgeschlagen werden (siehe hierzu Seite 21). Die einzelnen Schritte sollen durch Textseiten mit animierten Bildern dargestellt werden. Dadurch, dass die Animationen dem Lernenden anfangs zeigen, wie Teilschritte auszuführen sind, soll der Lernende direkt auf die Programmbedienung schließen können. Mit Fortschreiten des Tutorials sollen diese Elemente in den Hintergrund treten, um den Lernenden nicht zu langweilen. Ich gehe davon aus, dass er nach wenigen Schritten eine eventuelle Unsicherheit überwunden hat. Um die Interaktion mit dem Lernenden motivierender zu gestalten, habe ich mich bei der Gestaltung der Inhalte des Tutorials dafür entschieden, sie so zu wählen, als wäre das Tutorial eine Person, die sich mit dem Lernenden unterhält. Dazu sind die Texte in Ich-Erzählweise formuliert.

Die Beschreibungen der einzelnen Schritte werden von konkreten Aufforderungen begleitet, etwas Bestimmtes zu tun. Sobald der Lernende die ihm aufgetragene Aufgabe erledigt hat, soll das Tutorial automatisch damit fortfahren, den nächsten Schritt zu erläutern. Dies soll die Anzahl an Bedienungsaktionen verringern, um nicht den Arbeitsfluss beispielsweise durch das Zielen auf einen „Weiter“-Knopf zu verlangsamen.

Bei der Entscheidung für diese Art der Interaktivität entsteht die Frage, ob diese problembehaftet ist: Der Lernende könnte die in der aktuellen Tutorial-Seite geforderten Aktionen ausführen, bevor er die Seite komplett gelesen hat. Für solche Benutzer

wäre eine andere Umsetzung mit „Weiter“-Knopf besser gewesen. Für dieses Problem und die allgemeine Verbesserung der Bedienung des Tutorials soll es eine Navigation entlang der einzelnen Schritte (Bild 5.15) erhalten. Damit erkennt der Benutzer den aktuellen Schritt des Tutorials und hat die Freiheit, jederzeit zu jeder Stelle innerhalb des Tutorials zu gelangen.

4.6.2 Steuerung der Simulation

Die Steuerung der Simulation besteht im Wesentlichen aus den üblichen Steuerungsfunktionen: Start, Pause, Stop, Schritt vor und Schritt zurück. Für die Regelung der Geschwindigkeit bietet sich ein Drehknopf an, da er auf kleinem Raum eine gute Repräsentation eines Einstellbereichs bietet.

Um die Simulation etwas aktiver zu gestalten, soll es eine Möglichkeit geben, von beliebigen Knoten aus spontan Nachrichten für das dynamische Segment zu versenden. Hierzu ist ein Knopf pro Knoten erforderlich.

4.6.3 Feedback für den Lernenden

Da das Programm kein klassisches Lernprogramm ist, welches Antworten auf Fragen analysiert und beantwortet, besteht das Feedback hauptsächlich aus den Reaktionen der Simulation, ihrer Konfiguration oder in Aktionen des interaktiven Tutorials.

Bei der Simulation gibt es kein Feedback im behavioristischen Sinn. Das Feedback erfolgt indirekt durch die sofortige Visualisierung der Simulationsergebnisse. Wird ein Schritt simuliert, sollen die Konsequenzen sofort sichtbar werden. Die in Abschnitt 4.4 ausgeführten Visualisierungen sollen mit jedem Schritt aktualisiert werden. Dies entspricht der gefahrlosen Erfahrung der *natürlichen Konsequenzen*, wie Niegemann sie für multimediale Lernumgebungen als alternatives Feedback vorschlägt [NHHM+04, Seite 234]. Sein Leitspruch dazu: „Schau halt, was passiert!“.

4.7 Zusammenfassung

Das Konzept für die interaktive Visualisierung der Medienzugriffskontrolle beim FlexRay-Protokoll besteht aus einer Simulation von Vorgängen während der Kommunikation in einem FlexRay-Cluster. Diese Vorgänge werden sichtbar gemacht, indem die verborgenen zeitlichen Zusammenhänge anhand ihrer logischen Strukturen in eine grafische Anzeige transformiert werden. Dazu soll der Lernende die Möglichkeit haben, ein einfaches Cluster-Modell grafisch zu erstellen. Dann soll er in der Lage sein, die für die Medienzugriffskontrolle notwendigen Parameter einzustellen und die Simulation der Kommunikation auszuführen. Das Modell der Simulation selbst soll der einfachen Struktur eines FlexRay-Clusters folgen. Sie soll schrittweise durchgeführt werden können und der Visualisierung als „Antrieb“ dienen. Die Visualisierung greift die bisher

in der Lehre verwendeten Bilder auf, und fügt ihnen mit Hilfe der Simulation die Manipulierbarkeit und die Bewegung hinzu. Ein interaktives Tutorial soll dem Lernenden helfen, selbst einen Einstieg in die Anwendung zu finden.

5 Umsetzung

5.1 Anforderungen

Das Programm zur interaktiven Visualisierung sollte in der Programmiersprache Java implementiert werden, um eine plattformübergreifende Verfügbarkeit zu gewährleisten. Ich wählte die zu Beginn der Arbeit aktuelle *Java^{TM} 2 Platform Standard Edition 5.0*. Einzige Anforderung an die Programmierweise war, dass die Simulation und die Visualisierung lose gekoppelt sind, ähnlich dem aus der Programmiersprache Smalltalk bekannten Model-View-Controller Entwurfsmuster (MVC), wie es in [GHJV94] oder für Java in [Ull05] beschrieben wird. Dabei sind Darstellung (View), Inhalt (Modell) und Steuerung (Controller) komplett voneinander getrennt. Dies erleichtert beispielsweise den Austausch der Darstellung, ohne die Steuerung oder das Modell ändern zu müssen.

Am Lehrstuhl XII wurden bereits andere Programme zur Unterstützung der Lehre entwickelt. Dabei wurde die Erfahrung gemacht, dass die Erfüllung der folgenden Eigenschaften sinnvoll ist:

- Vollbild-Präsentationsmodus
- Anpassungsmöglichkeit für die Farbeinstellungen, um eventuellen Problemen bei der Darstellung durch einen Projektor aus dem Weg gehen zu können
- Sprache: Menüs und Texte in Deutsch und Englisch
- Internationalisierung: Menüs und Hilfetexte sollen durch Texte in anderen Sprachen ersetzt werden können
- Speicherung von Modellen in einer XML-Datei zur späteren Wiederverwendung
- Vergleichbares Aussehen und Bedienung, wie sie andere Programme am Lehrstuhl XII zeigen. Das sind das in [NS06] entstandene Programm und die Nacharbeitung der Anwendung von [Pil04] durch Birgit Sirocic auf Grundlage ihres in Entstehung befindlichen Rahmenwerks *LearningVisualizations*

Bei der Umsetzung des Programms habe ich diese Anforderungen berücksichtigt, was durch die Verwendung des Rahmenwerks begünstigt wurde.

5.2 Allgemeines zur Programmierung

Für die Umsetzung habe ich das in den Anforderungen erwähnte Rahmenwerk *LearningVisualizations* genutzt. Es baut bei der Benutzerschnittstelle auf den Java Foundation Classes („JFC/Swing“, [SM06]) auf und bietet Basisklassen für einige der geforderten Funktionen. Es besteht aus

- **Grundkomponente:** Hierin ist die Basisklasse für ein Lernprogramm mit Gerüsten für die Lade- und Speicher-Funktionalität sowie der Grundkonfiguration und Internationalisierung enthalten
- **Avatar:** Schablone für interaktive Hilfefunktionen
- **Editor:** Basisklasse für einen grafischen Editor mit eigenständiger Darstellung
- **Grafische Bedienschnittstelle:** Dialoge und grafische Komponenten für verschiedene Aufgaben, zum Beispiel Dateiauswahl, Sprachwahl, Abfrage einer Bestätigung, Knöpfe, Menüs
- **Modell:** Schablone für das zu simulierende Modell im Speicher; hierin sind Gerüste von Funktionen zur Speicherung und Konsistenzprüfung des Modells enthalten
- **Simulator:** grundlegende Simulationssteuerung; schrittweise und als kontinuierliche Ausführung von Einzelschritten mit einstellbarer Verzögerung
- **Visualisierung:** Klassenhierarchie für den Aufbau der Darstellung des simulierten Modells

Die Struktur des Rahmenwerks ist offen. Es bestand für mich kein Zwang, alle Komponenten zu nutzen oder sie in der vorgeschlagenen Art und Weise zu nutzen. Teilweise lag dies auch daran, dass die Implementierung des Rahmenwerks noch am Anfang steht und die dadurch entstehenden Einschränkungen nicht die Implementierung meines Programms beeinflussen sollten. Durch die *modified MVC* Architektur von JFC/Swing, bei der ein starker Zusammenhang zwischen View und Controller besteht (siehe [Fow06]), ist eine komplette Separierung der Steuerung von ihrer Darstellung nur schwer möglich. Ich habe mich daher dafür entschieden, zugunsten der besseren Programmierbarkeit die durch JFC/Swing vorgegebene Struktur zu verwenden. Das bedeutet, dass ich zu jeder Modellkomponente eine eigene JFC/Swing-Komponente entwickelt habe, die die Daten der Modellkomponente darstellt. Gleichzeitig dient die JFC/Swing-Komponente als Eingabemedium für die Steuerung und Konfiguration. Für Daten, die ohne eigene Modellkomponente auskommen – das sind die Daten, die den Kommunikationszyklus und die Medien betreffen – habe ich ebenfalls eine eigene JFC/Swing-Komponente zur Darstellung entworfen.

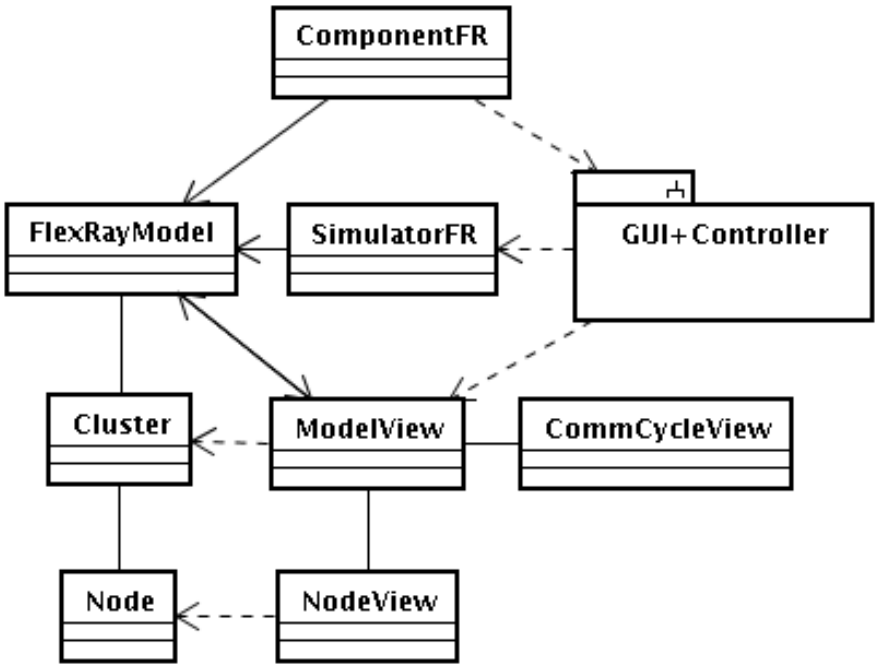


Bild 5.1: Einfaches frühes UML-Diagramm der Klassenstruktur

Entsprechend der Konzeption von Seite 49 entschied ich mich dafür, den Editor und die Visualisierung zu einer Einheit zusammenzufassen. Bild 5.1 zeigt einen groben Überblick über einen frühen Entwurf der Klassenstruktur meines Programms als UML-Diagramm[1]. Das Bild zeigt die übergreifende Programmkomponente *ComponentFR*, die die Hauptklasse der Anwendung darstellt. Darunter ist links die Umsetzung der in der Konzeption gezeigten Modellstruktur (Seite 51), ausgehend von dem übergeordneten *FlexRayModel* zu erkennen. Dieses stellt die Modellkomponente aus der Struktur des Rahmenwerks dar und verbindet in meinem Programm das Modell mit seiner Ansicht. Rechts neben den Modellkomponenten ist die Struktur der Visualisierung, ausgehend von *ModelView* zu erkennen, die direkt von dem *GUI+Controller*-Subsystem verwendet wird. *SimulatorFR* ist der Simulator, der direkten Zugriff auf das Modell hat und von *GUI+Controller* gesteuert wird. Die Kombination aus *ComponentFR*, *SimulatorFR* und *FlexRayModel* entspricht der durch das Rahmenwerk vorgegebenen Struktur.

[1] Aufgrund der schlechten Verfügbarkeit von erschwinglichen, funktionierenden UML-Werkzeugen, die mit den Besonderheiten von Java 5 umgehen können, sowie dem für eine Person vergleichsweise hohen Aufwand für die UML-basierte Entwicklung von JFC/Swing-Anwendungen habe ich auf die weitere Verwendung von UML verzichtet.

5.3 Ablauf der Simulation

Bei einem Vorwärtsschritt überprüft der Cluster, ob ein Simulationszustand zum aktuellen Macrotick gespeichert wurde. Das ist der Fall, wenn nach einem oder mehreren Rückwärtsschritten wieder vorwärts simuliert wird. Die überprüfung und Wiederverwendung von gespeicherten Zuständen bei Vorwärtsschritten ist notwendig, weil die Wiederholung des selben Zeitabschnitts die gleichen Ergebnisse bringen soll, auch wenn zufällige Nachrichten generiert wurden. Hat der Cluster einen gespeicherten Zustand gefunden, dann wird der Zustand wiederhergestellt. Andernfalls werden die folgenden Aktionen durchgeführt: Der Cluster aktualisiert die Zähler für den Macrotick und den Kommunikationszyklus. Danach holt er sich von den einzelnen Knoten die Information ab, ob im vorigen Schritt eine Übertragung (und damit ein Slot) beendet wurde. Mit dieser Information aktualisiert der Cluster dann die Zähler für die Slots je Kanal. Zuletzt durchläuft der Cluster in beiden Fällen, also nach Wiederherstellung oder aktiver Berechnung, die Knoten in einer Schleife und führt mit ihnen jeweils den Vorwärtsschritt durch. Im fall der aktiven Berechnung speichert der Cluster anschließend seinen aktuellen Zustand in der Liste der Simulationszustände. Da dieser Zustand bereits aktualisierte Zähler enthält, insbesondere den Macrotick, ist es im Falle der Wiederherstellung auch nicht notwendig den Macrotick vor der Wiederherstellung zu aktualisieren.

Bei einem Rückwärtsschritt stellt der Cluster lediglich den Zustand für den vorvorletzten Macrotick wieder her. Es muss der Zustand für den vor-vorletzten Macrotick sein, weil aufgrund des Vorgehens bei der Zustandsspeicherung dort der Zustand gespeichert ist, der die vorige Macrotick-Nummer enthält. Das ist der Zustand, der einen Schritt zuvor aktiv war. Danach lässt der Cluster die Knoten der Reihe nach die Berechnung für einen Rückwärtsschritt ausführen, die nur darin besteht, den passenden Knotenzustand wiederherzustellen.

Die Knoten verfahren bei einem Vorwärtsschritt ähnlich wie der Cluster: Nach der Überprüfung, ob für den Schritt ein gespeicherter Zustand vorliegt wird dieser entweder hergestellt, oder es wird weitersimuliert. Innerhalb der Knoten ist die Simulation in zwei Teile zerlegt: Ein Simulationsschritt für Kanal A und ein Simulationsschritt für Kanal B. Dieses Vorgehen habe ich gewählt, um leicht auf die verschiedenen Verbindungssituationen (nur A, nur B, A und B) reagieren zu können. Pro Kanal überprüft der Knoten, ob er daran angeschlossen ist. Ist dies der Fall, testet der Knoten, in welchem Segment er sich befindet. Im statischen Segment entscheidet er nur, ob er senden will oder nicht und tut dies entsprechend. Im dynamischen Segment prüft der Knoten, ob eine Nachricht bereits teilweise gesendet wurde. Ist dies der Fall, wird die Übertragung fortgesetzt. Ist dies nicht der Fall, wird gegebenenfalls eine neue Nachricht generiert und die Simulation der Übertragung begonnen. Ob eine Nachricht gesendet wird und welche Länge sie haben soll, ist dabei konfigurierbar. Neben der in vorgegebenen Grenzen zufälligen Generierung von Nachrichten ist es auch möglich, Nachrichten immer gleicher Länge und gleichmäßige Abstände zwischen den Nachrichten zu konfi-

gurieren. Zusätzlich erhält der Benutzer die Möglichkeit, während der Simulation die Übertragung einer Nachricht vorkonfigurierter Länge anzufordern.

Soll während eines Vorwärtsschrittes eine Übertragung stattfinden, dann weist der Knoten das zugehörige Nachrichtenobjekt an, seinen internen Macrotick-Zähler zu aktualisieren. Ist die Sendezeit für den Knoten abgelaufen, die Nachricht jedoch noch nicht komplett übermittelt, dann sorgt der Knoten dafür, dass die Nachricht ihre Fragmentzählung aktualisiert. Der Knoten stellt die aktuell übertragene Nachricht und damit den aktuellen Übertragungsstatus für den externen Zugriff zur Verfügung. Auf diese Weise kann der Cluster erfahren, ob eine Übertragung stattfindet, und die Visualisierungen von Knoten und Kommunikationszyklus erhalten die zur Darstellung erforderlichen Informationen.

5.4 Visualisierung und Steuerung

Die FlexRay-Geräte sind bei meiner Darstellung als bunte Kästchen realisiert. Die Kästchen sind mit Linien verbunden, die den Bus symbolisieren. Die Slots, in denen die Knoten senden, sind durch Kästchen in den Knotenfarben dargestellt, und im Kommunikationszyklus entsprechend der Zugriffsreihenfolge angeordnet. Die Breite eines Kästchens im dynamischen Segment zeigt die Dauer des Zugriffs an. Bild 5.2 zeigt das Programmfenster im Editormodus, bei englischer Spracheinstellung. Darin ist ein Beispiel-Cluster mit Stern-Topologie für beide Kanäle zu sehen. Innerhalb der Darstellung des Kommunikationszyklusses befindet sich eine Beschriftung, die dem Benutzer mitteilt, dass dies die Konfigurationsansicht ist. Der Slot unter dem Mauszeiger wird mit einem transparenten Rechteck in der *Highlightfarbe* durch Überdeckung hervorgehoben[2]. Dies ist die Umsetzung der Anzeige der integrierten Konfigurierbarkeit. Durch einen Mausklick auf den hervorgehobenen Slot kann dessen Knotenzuordnungskonfiguration erreicht werden. Die Highlightfarbe wird von dem Rahmenwerk als eine Standardfarbe für Hervorhebungen angeboten und ist durch den Benutzer konfigurierbar. Rechts sind die Knöpfe für die Steuerung des Simulationsablaufs zu erkennen – sie werden bis auf den Drehknopf ebenfalls vom Rahmenwerk angeboten.

Die Anzeige für die Nachrichten und deren Fragmentierung realisierte ich durch einen waagerechten, umrahmten Balken, der einem klassischen Fortschrittsbalken entspricht, wie er in grafischen Benutzeroberflächen verwendet wird: Die Länge des Balkens repräsentiert die Größe der zu übermittelnden Nachricht. Das Innere des Balkens wird mit fortschreitender Übermittlung der Nachricht gefüllt. Kann die Nachricht nicht vollständig im aktuellen Kommunikationszyklus übertragen werden, wird die aktuelle Stelle markiert und der Balken füllt sich erst im nächsten Kommunikationszyklus weiter. Diese Umsetzung ist in Bild 5.3 zu sehen.

[2]Den Begriff *Highlightfarbe* für die allgemeine Markierungsfarbe habe ich aus den Vorgaben des Rahmenwerks übernommen.

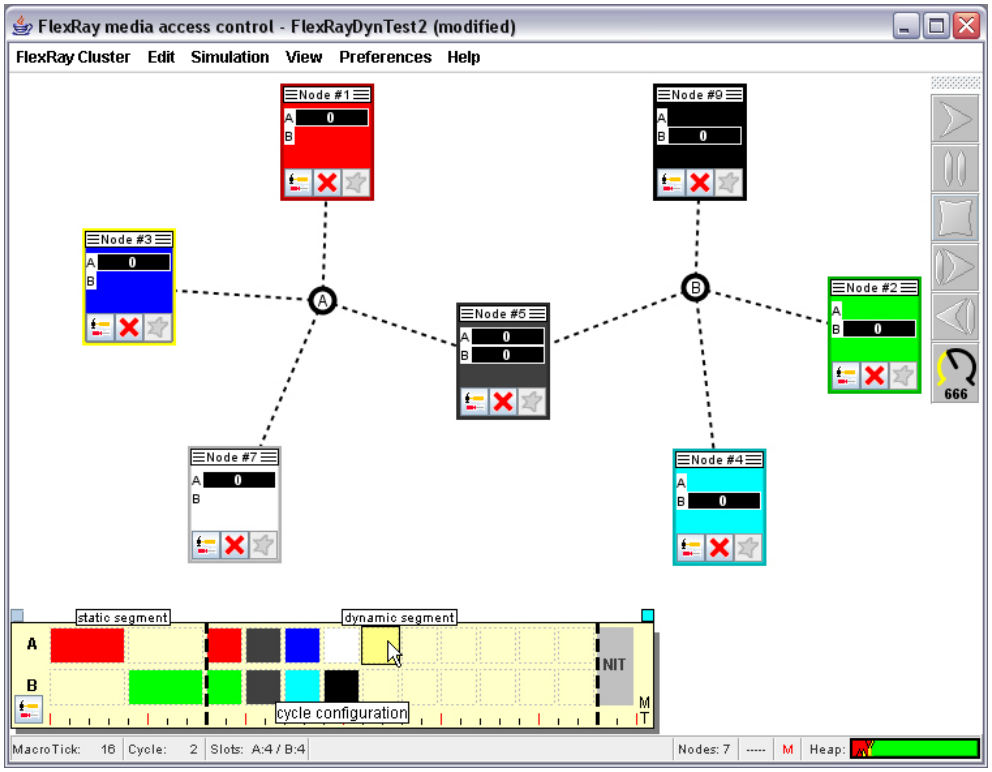


Bild 5.2: Die interaktive Visualisierung im Editor-Modus mit Spracheinstellung Englisch

Während der NIT soll diese Darstellung aktiv bleiben, damit der Benutzer die Fragmentierung der Nachrichten nach der letzten Übertragung in diesem Zyklus beobachten kann. Während des statischen Segmentes wird die Darstellung deaktiviert. Dies habe ich ausgenutzt, um die Anzahl der Nachrichten anzuzeigen, die im dynamischen Segment bisher vollständig übertragen werden konnten (Bild 5.4). Hierdurch wird die Bedeutung der Priorisierung insbesondere bei fragmentierten Nachrichten noch deutlicher.

Die Visualisierung der Slot-Zustände habe ich wie auf Seite 49 beschrieben umgesetzt. Bild 5.5 zeigt die Visualisierung des Kommunikationszyklusses zu einem Zeitpunkt, bei dem die Simulation bis zur Mitte des statischen Segments fortgeschritten ist. Die beiden blau gefärbten Slots links zeigen an, dass derselbe Knoten auf beiden Kanälen gesendet hat. Danach folgt auf Kanal A ein grün umrandetes leeres Feld, das anzeigt, dass ein Knoten hätte senden dürfen, dies aber nicht getan hat. Darunter ist auf Kanal B ein Slot zu sehen, der keine Knotenzuordnung besitzt. Die restlichen Slots, die in der Zukunft der Simulation liegen, sind durch leicht gestrichelte Kästchen angedeutet. Wie in dem Bild auch schon zu sehen ist, werden die statischen Slots immer

Bild 5.3: Bild eines FlexRay-Knotens aus der interaktiven Visualisierung. Der Fortschrittsbalken zeigt die Übertragung einer fragmentierten Nachricht an

Bild 5.4: Bild eines FlexRay-Knotens aus der interaktiven Visualisierung. Der Fortschrittsbalken zeigt hier die Anzahl der bisher vollständig versendeten Nachtichten im dynamischen Segment

vollständig gezeichnet, und nicht nur so weit wie die Macroticks vorangeschritten sind. Dies entspricht der Sichtweise des Beobachter-Knotens: Die Knoten haben Zugriff auf die Cluster-Parameter und kennen daher die Länge der statischen Slots. Wird in einem Macrotick eine Übertragung festgestellt, ist der gesamte Slot dieser Übertragung zuzuordnen.

Im dynamischen Segment erfolgt die Darstellung der Slot-Zustände genauso wie im statischen Segment. Die dynamischen Slots haben jedoch variable Länge. Aus Sicht eines Beobachterknotens können sie daher nicht sofort vollständig gezeichnet werden (wie auf Seite 48 beschrieben). Die Visualisierung baut die dynamischen Slots daher minislotweise auf. Bild 5.6 zeigt dies anhand der Darstellung eines Kommunikationszyklusses, wobei die Simulation über die Mitte des dynamischen Segments hinaus fortgeschritten ist. Bild 5.7 zeigt das Lernprogramm während der Simulation. Dort sind noch einmal die einzelnen Eigenschaften zusammen zu erkennen. Entgegen der Konzeption (siehe Bild 4.3 auf Seite 48) habe ich keinen Pfeil benutzt, um den zeitlichen Verlauf zu markieren. Anstelle des Pfeils habe ich ein transparentes Rechteck mit der Breite eines Macroticks über die Darstellung des Kommunikationszyklusses laufen lassen. Auf diese Weise wollte ich die Schrittgröße der Simulation deutlicher machen. Die Linien, die die Medien darstellen sind in der Farbe der Knoten gezeichnet, die gerade

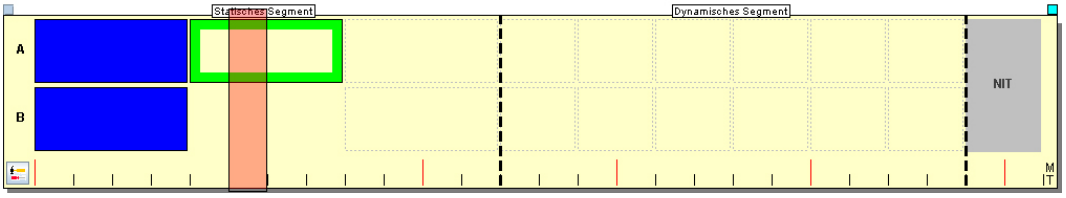


Bild 5.5: Kommunikationszyklus-Visualisierung, bei der die Simulation einige Schritte in das statische Segment fortgeschritten ist

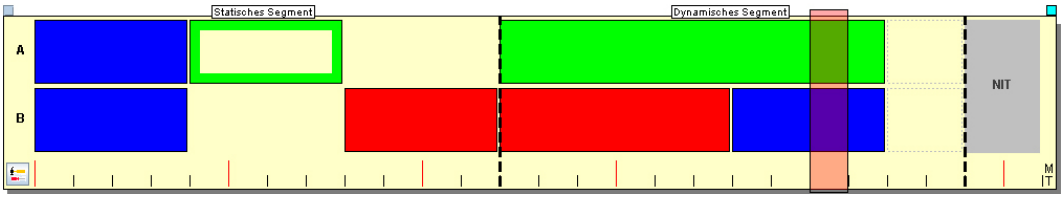


Bild 5.6: Kommunikationszyklus-Visualisierung, bei der die Simulation bis in das dynamische Segment fortgeschritten ist

eine Übertragung durchführen können. Zusätzlich ist der Titelbereich der momentan übertragenden Knoten in einer anderen Farbe dargestellt.

Ich habe für die Geschwindigkeitsregelung einen Drehknopf benutzen wollen. Da JFC/Swing kein solches Element anbietet, habe ich selbst einen Drehknopf entworfen (Bild 5.8). Dieser Drehknopf enthält die Anzeige der aktuell eingestellten Zeitverzögerung zwischen den Simulationsschritten in Millisekunden. Damit erhält der Benutzer eine Orientierung, die ihm den Einstellvorgang erleichtert. Position des Zeigers und Anzeige der Verzögerung erlauben dem Benutzer die Geschwindigkeit der Simulation einzuschätzen, noch bevor er den Start-Knopf betätigt.

Die FlexRay-Knoten haben einen Knopf, der dem Benutzer erlaubt, eine Sendeanforderung für das dynamische Segment zu generieren. In Bild 5.3 auf Seite 63 ist dieser Knopf zu erkennen: Es ist der mit dem sternförmigen Symbol. Mit der manuellen Generierung von Sendeanforderungen kann gezielt beobachtet werden, wie spontane Kommunikationsereignisse von der Medienzugriffskontrolle verarbeitet werden.

Die Konfigurationen der Knoten und des Kommunikationszyklusses werden jeweils durch die links unten angebrachten Knöpfe erreicht, die ein kleines Bild mit einem Hammer und einen Schraubendreher zeigen.

Im Editor-Modus ist für Tastatur- und Menükommandos immer ein Knoten ausgewählt. Das wird durch eine Umrandung in der Highlightfarbe angezeigt. Das ist in Bild 5.2 der Knoten „Node #3“. Gleichzeitig wird die Umrandung in der Highlightfarbe auch benutzt, um anzuzeigen, ob der Knoten der sich unter dem Mauszeiger befindet,

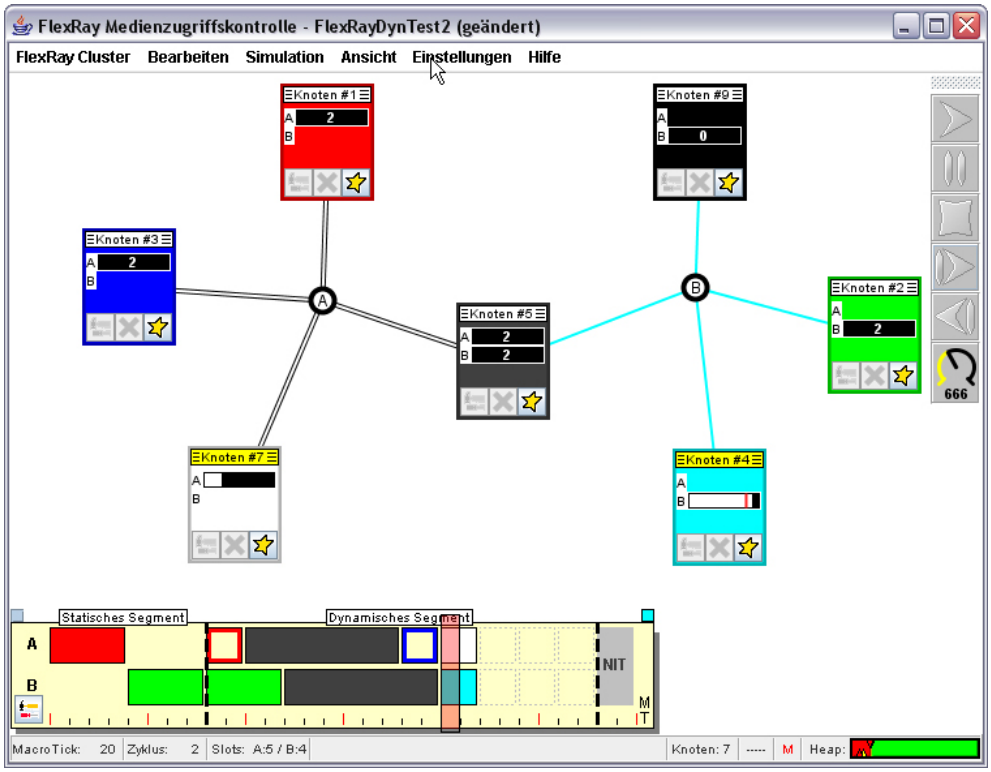


Bild 5.7: Die interaktive Visualisierung während der Simulation mit Spracheinstellung Deutsch

ausgewählt werden kann. Bild 5.9 zeigt dies: Knoten „Node #3“ ist ausgewählt, und der Mauszeiger befindet sich über „Node #7“ (weiß). In diesem Fall befindet sich der Mauszeiger gleichzeitig über dem Knotentitel. Durch Klicken und Ziehen des Knotentitels kann der Knoten verschoben werden, was dem Benutzer angezeigt wird, indem der Mauszeiger durch ein Vier-Wege-Symbol (Kreuz) ersetzt wird.

5.4.1 Feingestaltungen und Auflockerungen

Der Freiraum im Titelbereich der Knoten wurde mit Linien aufgefüllt, wie in allen Bildern zu erkennen ist. Dies ist eine Anlehnung an die Darstellung der Fenstertitel von grafischen Benutzerschnittstellen. Neben dem optischen Effekt der Aufgeräumtheit kann diese Markierung als Kennzeichnung dienen, dass der Knoten an dieser Stelle mit der Maus „angefasst“ werden kann. Ob diese Markierung intuitiv vom Benutzer erkannt wird, hängt davon ab, welche grafischen Benutzeroberflächen der Benutzer bereits kennt.

Für die Linien, die die Medien repräsentieren, habe ich eine abschaltbare Kantenglät-

Bild 5.8: Drehknopf zur Steuerung der Simulationsgeschwindigkeit

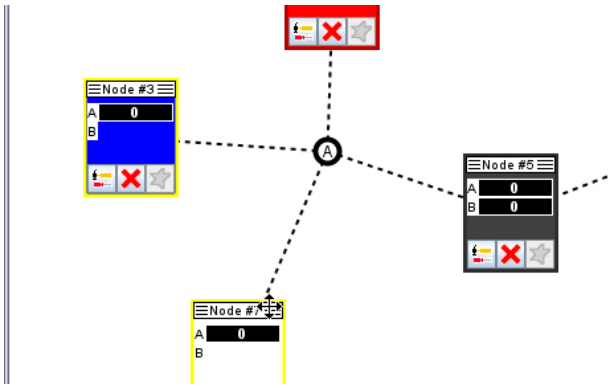


Bild 5.9: Markierungen, die an Knoten gezeigt werden, wenn der Mauszeiger darüber steht

tung realisiert, um die ästhetisch wenig ansprechenden, kantigen Linien durch eine glattere Darstellung zu ersetzen. Bild 5.10 zeigt den Unterschied beispielhaft an zwei kurzen Verbindungen.

Ein Spaßfaktor ist die (ebenfalls abschaltbare) Transparenz beim Verschieben von Knoten. Wird der Knoten mit Hilfe des Titelbereichs per Maus umpositioniert, dann findet eine weiche Ausblendung des Knotens bis hin zu etwa dreißig Prozent Durchlässigkeit statt. Bild 5.11 zeigt einen in der Umpositionierung befindlichen Knoten bei eingeschalteter Transparenz. Neben dem Spielzeugcharakter des Effektes erlaubt er zugleich, dass das gesamte Modell während der Verschiebung des Knotens sichtbar bleibt. Danach wird der Knoten wieder weich eingeblendet bis zur vollständigen Undurchlässigkeit.

Die Vergrößerung und Verkleinerung der Kommunikationszyklusdarstellung erfolgt bei voll gezeichnetem Kommunikationszyklus. Die von anderen Programmen bekannte Andeutung der Dimension des zu vergrößernden oder verkleinernden Objektes erschien mir hier nicht nur als unansehnlich, sondern auch als vorstellungshemmend. Wird der Zyklus während seiner Dimensionsänderung voll gezeichnet, erhält man sofort einen Überblick über sein zukünftiges Aussehen. Um keine Unregelmäßigkeiten in der Darstellung aufkommen zu lassen, werden horizontale Veränderungen erst dann durchgeführt, wenn der Benutzer die Maus weit genug bewegt hat, um eine gleichmäßige

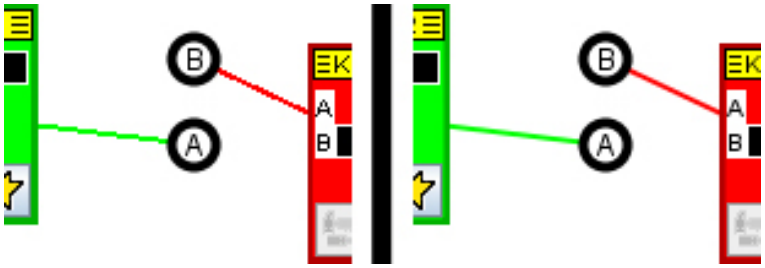


Bild 5.10: Kantenglättung: Links ausgeschaltet, rechts eingeschaltet

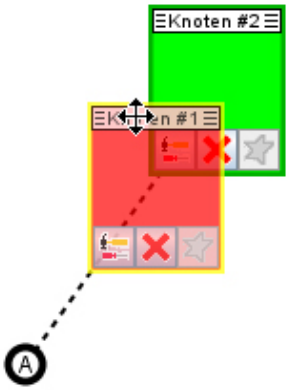


Bild 5.11: Transparenz beim Verschieben von Knoten

Darstellung zu gewährleisten. Bild 5.12 zeigt beispielhaft den Unterschied zwischen zwei Größenänderungen. Die Änderung muss groß genug sein, um jedem Macrotick mindestens eine Änderung von einem Bildpunkt zu erlauben. Bei vertikaler Anpassung der Größe ist dies nicht notwendig, da die präzise Anzeige der Simulation hiervon nicht abhängt. Das Bild zeigt auch die Änderung des Mauszeigers, ähnlich der Änderung für die Verschiebung der Knoten. Im oberen Teil ist auch die Kurzhilfe zu erkennen, die eingeblendet wird, wenn der Mauszeiger einen Moment über dem Knopf verweilt.

Ein einfaches und übliches Gestaltungsmerkmal ist die Verkürzung von Texten, falls der ihnen zur Verfügung stehende Raum zu klein wird. Bilder 5.5 und 5.6 zeigen noch die vollständigen Texte in der Darstellung des Kommunikationszyklusses, da dort genügend Platz für die Beschriftung übrig ist. Bild 5.13 zeigt einen Kommunikationszyklus, dessen Darstellung verkleinert wurde. Die Beschriftung des dynamischen Segments wurde hierbei zusammengeschoben, da sie sonst über das dynamische Segment hinausragen würde, wodurch die Markierung des dynamischen Segments nicht mehr eindeutig wäre. Zusätzlich könnten Überlappungen mit anderen Teilen der Darstellung deren Wahrnehmung und Bedienung erschweren. Die Verkürzung des Textes

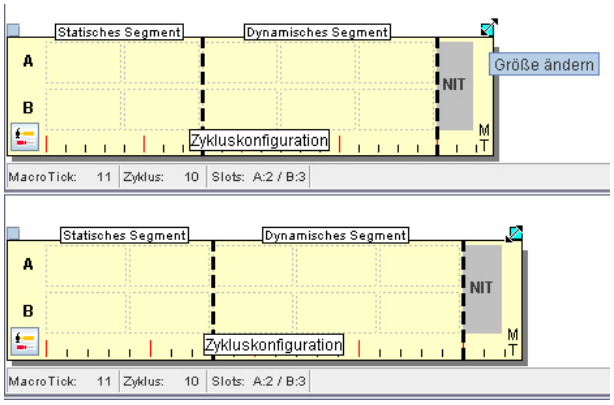


Bild 5.12: Größenänderung der Darstellung des Kommunikationszyklusses

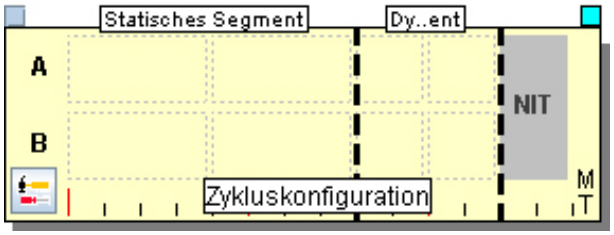


Bild 5.13: Automatische Verkürzung von Texten

besteht darin, einige Buchstaben aus der Mitte zu entfernen und durch zwei Punkte zu ersetzen, die das Fehlen von Buchstaben andeuten.

Die Darstellung des Kommunikationszyklusses wird grundsätzlich in der linken unteren Ecke der Anwendung positioniert, um automatisch am wenigsten im Weg zu sein. Bei einer Größenänderung des Anwendungsfensters wird die Darstellung automatisch mitverschoben. Da es vorstellbar ist, dass die fixe Positionierung relativ zum Programmfenster als Einschränkung wahrgenommen werden kann, ist die Darstellung des Kommunikationszyklusses auch verschiebbar. Der linke obere Knopf dient dabei ähnlich der Nadel einer Pinwand als Umschalter: Ist er aktiviert, bleibt die Darstellung links unten haften. Wird er deaktiviert, ändert sich die Anzeige und der Kommunikationszyklus wird verschiebbar. Bild 5.14 zeigt den Kommunikationszyklus in Bewegung. Daran ist die geänderte Darstellung mit einem gestreiften Titelbereich und dem geänderten Mauszeiger bei Überfahren dieses Titelbereichs zu erkennen. Dies ist konsistent zu der Darstellung und Funktionsweise des Titelbereichs der Knoten. Wie die Knoten wird auch der Kommunikationszyklus bei aktivierter Transparenz während der Umpositionierung durchsichtig dargestellt.

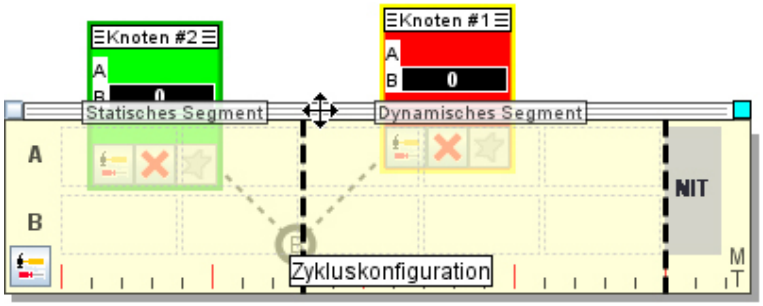


Bild 5.14: Verschieben der Kommunikationszyklusdarstellung

5.5 Tutorial und Hilfe

In der Konzeption habe ich erwähnt, dass das automatische Voranschreiten des Tutorials problematisch sein könnte (Seite 54). Dieses potenzielle Problem habe ich ausgenutzt, um die Personifizierung des Tutorials zu stärken und Sympathie zu erzeugen. Das Tutorial „sagt“ am Anfang:

> Ich bin etwas nervös und ungeduldig. Daher werde ich sofort zum nächsten Punkt des Tutorials springen, sobald sie alle Aufgaben der aktuellen Tutorialseite erledigt haben.

Die Umsetzung der Idee mit der Navigation entlang der Struktur des Tutorials von Seite 55 zeigt Bild 5.15. Aus Platzgründen und aufgrund des vergleichsweise geringen Umfanges des Tutorials habe ich nicht die Überschriften der Schritte, sondern nur ihre Nummern zur Navigation angeboten. Die eingeklammerte, fett gedruckte Nummer ist dabei die Nummer des aktuellen Schrittes im Tutorial. Um die Navigation nicht zu verlieren, wenn der Benutzer am Ende der aktuellen Seite angelangt ist, befindet sich dieselbe Navigation auch noch einmal dort.

Bild 5.15: Navigationspfad im Tutorial

Um den Benutzer auch außerhalb einer Übung zu unterstützen, kann in derselben Ansicht, in der das Tutorial dargestellt ist, eine einfache Hilfe angezeigt werden, die die Programmfunktionen erläutert. Innerhalb dieser Hilfe sind auch die Inhalte des Tutorials abrufbar, weshalb bei angezeigter Hilfe in der Statuszeile des Programms angezeigt wird, ob gerade das Tutorial aktiv ist oder die normale Hilfe.

5.6 Sonstiges

5.6.1 Speicherung der Modelldaten

Um ein Cluster-Modell zur Wiederverwendung zu speichern, müssen zwei getrennte Bereiche gesichert werden: Das eigentliche Modell mit seinen Parametern und die Details der Visualisierung, wie zum Beispiel die Objektpositionen. Ich habe mich dafür entschieden hier wie auch bei der Programmierung nach dem MVC-Entwurfsmuster beides getrennt zu halten. Dadurch ist es zum Beispiel leichter möglich, die Modelldaten durch eine andere Anwendung verarbeiten zu lassen, ohne dass umständlich die Ansichtsdaten umgangen werden müssen. Auch ist so die Manipulation der Daten direkt in der XML-Datei weniger problematisch: Die Änderung von Anzeigedaten zerstört das Modell nicht und umgekehrt.

5.6.2 Mehrere Modelle

Meine Anwendung unterstützt das gleichzeitige Laden mehrerer Modelle. Zwischen diesen kann per Tastendruck oder Menü schnell gewechselt werden. Diese Funktion ist nützlich für einen parallelen Vergleich ähnlicher Modelle mit geringfügigen Unterschieden.

5.7 Zusammenfassung

Die Konzeption zur interaktiven Visualisierung der Medienzugriffskontrolle beim FlexRay-Protokoll aus Kapitel 4 habe ich umgesetzt. Die Modellstruktur diente dabei als Grundlage für die Klassenstruktur des Modells im Programm. Visualisierungsdetails wurden konkretisiert, zum Beispiel die Darstellungen der Slot-Zustände oder der Drehknopf für die Geschwindigkeitsregelung. Die Gesamtdarstellung wurde um Details erweitert, damit die Anwendung leichter bedienbar ist und der „Spaß-Faktor“ zur Motivierung des Lernenden steigt. Hier sind als Beispiele die Markierungen der manipulierbaren Objekte oder die Transparenzeffekte zu nennen.

6 Fazit

Die Konzeption und Umsetzung selbst eines einfachen Lernprogrammes für einen bestimmten informatischen Sachverhalt kann eine sehr aufwändige Aufgabe sein. Neben Fachkenntnissen, die erforderlich sind, um den darzustellenden Sachverhalt verstehen und aufbereiten zu können, werden für eine gute Konzeption und Umsetzung auch Kenntnisse in der Didaktik, Mediengestaltung und Programmierung benötigt. Jeder dieser Bereiche stellt genügend Material für eine eigene Berufsgruppe zur Verfügung. Dies macht die Konstruktion von hoch qualitativen Lehrmitteln für bestimmte Sachverhalte sehr zeitintensiv. Darin liegt vermutlich ein Grund für die geringe Verfügbarkeit solcher Lehrmittel.

6.1 Bewertung des Lernprogramms

Das in dieser Arbeit entwickelte Programm erfüllt die Anforderung, als präsentations- und übungsunterstützendes Lehrmittel einsetzbar zu sein. Folgende Ziele wurden dabei erreicht:

- Präsentation der Funktionsweise der Medienzugriffskontrolle beim FlexRay-Protokoll
- Experimentieren mit der Medienzugriffskontrolle bei selbst zusammengestellten FlexRay-Cluster-Modellen
- Niedrige Komplexität von Präsentation und Bedienung im Vergleich zu existierenden Programmen
- Verbesserte Anschaulichkeit gegenüber unbewegten Bildern oder starren Animationen
- Durch die beiden vorigen Punkte: Förderung des Verständnisses der Medienzugriffskontrolle und ihrer Rahmenbedingungen beim FlexRay-Protokoll
- Lauffähig auf verschiedenen Plattformen durch Implementierung in Java
- Kostenlose Verfügbarkeit

Eine genaue Feststellung der Leistungsfähigkeit des Programms für die Lehre ist ohne eine ausführliche Untersuchung mit der Zielgruppe kaum möglich. In Anbetracht der Auswahl an didaktischen und gestalterischen Methoden sowie deren unterschiedliche Eignung für bestimmte Zielgruppen kann höchstens ein guter erster Entwurf möglich sein. Ich habe eine Auswahl aus diesen Methoden (Kapitel 2.2) bei der Konzeption und Umsetzung meines Programms berücksichtigt, wodurch die genannten Ziele als erreicht angenommen werden können. Ob das Programm jedoch seinen Zweck erfüllen kann, hängt auch von den persönlichen Vorgaben der Personen aus der Zielgruppe ab. Wenn beispielsweise ein Großteil der Zielgruppe die Gestaltung als hässlich empfindet, dann wäre dies eine Störung, die die Nutzbarkeit des Programmes stark mindern würde. Folgende erreichte Ziele der Konzeption und der Umsetzung sind daher meiner Ansicht nach nur unter Vorbehalt zu nennen:

- Leichte Zugänglichkeit für Jedermann, insbesondere für die Zielgruppe
- Motivierung des Lernenden durch ansprechende Gestaltung
- Geringe Belastung des Lernenden

6.2 Einordnung im didaktischen Kontext

Das in dieser Arbeit entstandene Programm bezeichne ich als Lernprogramm. Dies kann durch die vielfältige Nutzung des Wortes falsch verstanden werden. Es fragt keine Inhalte ab und hilft dem Lernenden auch nicht, sich Inhalte anzueignen. Es vermittelt dem Lernenden Zusammenhänge und lädt ihn ein, damit zu experimentieren und so das Verständnis anderweitig erworbener Inhalte zu verbessern. Aufgrund der Umsetzung auf einem Rechner ist diese Art Lehrmittel momentan dem Bereich des *E-Learning* zuzuordnen. Eine eigenständige Einordnung und Bezeichnung für solche rechnerbasierten Lehrmodelle gibt es noch nicht.

7 Ausblick

Eine allgemeine Idee für die Fortführung der Arbeit besteht in einer Evaluation der Anwendung. Nur durch die Beobachtung der Zielgruppe bei der Bedienung des Programms können die gemachten Entscheidungen des didaktischen und des visuellen Entwurfs bestätigt oder verworfen werden. Auch die angenommenen Voraussetzungen der Zielgruppe (siehe Abschnitt 4.1) können so überprüft werden. Um das Programm möglichst universell gestalten zu können wird es erforderlich sein, die Evaluation mit mehreren verschieden zusammengesetzten Personengruppen aus der Zielgruppe durchzuführen und die sich daraus ergebenden Anpassungen jeweils wieder mit allen Gruppen zu überprüfen. Eine Aufgabe, die zusammen mit der ersten Konzeption und Implementierung des Programms im Rahmen einer Diplomarbeit kaum möglich ist.

7.1 Simulation

Um den Realismus weiter zu steigern und den Lernenden besser auf die reale Anwendung von FlexRay vorzubereiten, könnten Fehlverhalten, Störungen und Fehlparametrisierungen simuliert werden. Auf diese Weise wird dem Lernenden nicht nur die Funktionsweise des Zugriffsverfahrens und die Konsequenzen der korrekten Anwendung präsentiert, sondern – wie auch in einem Testlabor – zusätzlich die Möglichkeit eingeräumt, schwerwiegende Fehler zu machen, die in der Simulation keine zerstörerischen Folgen haben. Vorstellbar wäre die Mehrfachbelegung von Slots oder die Nutzung von zu kleinen statischen Slots, was die gesonderte Konfiguration der Nachrichten bedingen würde. Ebenso wäre die Simulation von Knoten- und Medienausfällen denkbar.

Neben dem Aktionsknopf zum Versenden von Nachrichten im dynamischen Segment könnte ein weiterer Aktionsknopf an den Knoten angebracht werden, der eine Nachrichtenübermittlung im statischen Segment starten soll. Damit könnte die Leistungsfähigkeit des statischen Segmentes mit der des dynamischen Segmentes verglichen werden.

Die vorliegende Simulation ist grundsätzlich auf die Medienzugriffskontrolle ausgerichtet, jedoch nicht alleine auf diese beschränkt. Durch die Implementierung eines Nachrichtenkonzeptes und der objektorientierten Modellierung entlang der Protokollspezifikation ist die Erweiterung der Simulation über die Medienzugriffskontrolle hinaus denkbar.

Eine Erweiterung könnte die Simulation des für die Medienzugriffskontrolle nicht

erforderlichen Nachrichtenempfanges sein. Damit könnte die Wirkung der Fragmentierung im dynamischen Segment noch deutlicher gezeigt werden: Der empfangende Knoten kann dann signalisieren, dass er darauf wartet, eine Nachricht aus ihren Fragmenten zusammensetzen zu können.

Nach Abschluß der Spezifikation der Buswächter (siehe Seite 14) könnten diese in die Simulation aufgenommen werden, um ihre Auswirkung auf die Medienzugriffskontrolle zu zeigen. Insbesondere zusammen mit der vorgeschlagenen Erweiterung um Fehlverhalten und Störungen wäre dies sinnvoll.

Die Simulation könnte dann auch auf die Anwendungsebene ausgedehnt werden, wozu die Nachrichten mit Inhalten versehen werden müssen. Eine Simulation von konkreten sendenden und empfangenden Geräten könnte insbesondere bei der Präsentation für Nicht-Informatiker mit weniger Vorwissen anschaulicher sein. Wenn bei einem Video-Anzeigegerät aufgrund mangelhafter Priorisierung das Bild ausfällt, dann ist dies noch anschaulicher, als ein Kästchen, bei dem der Fortschrittsbalken noch nicht vollständig gefüllt ist.

7.2 Visualisierung

Das von mir hergestellte Lernprogramm ist noch immer vergleichsweise abstrakt. Eine sparsame Anreicherung mit auswechselbaren Bildern echter FlexRay-Geräte könnte den Kontext noch realistischer gestalten. Liegen zusätzlich Informationen über die Konfiguration des Systems vor, das als Quelle für die Fotos diente, kann die Authentizität noch gesteigert werden. Dadurch könnte das motivierende Gefühl entstehen, man würde mit einem Detail eines real existierenden, komplexen Gebrauchsgegenstandes experimentieren können.

Während diese Arbeit entstand waren noch keine Fahrzeuge mit FlexRay erhältlich, und wie ich auf Seite 10 bereits beschrieben habe, hat BMW keine veröffentlichungsfähigen Informationen über die Konfiguration eines kurz nach Abschluß dieser Arbeit verfügbaren Fahrzeugs zur Verfügung stellen können. Bei Verfügbarkeit solcher Informationen könnte der Realismus weiter gesteigert werden, indem man existierende Situationen nicht nur visuell sondern auch funktional Nachahmt.

7.3 Benutzerführung und Interaktion

Grundsätzlich gilt bei der Benutzerführung, dass sie nicht für alle geeignet sein kann. Die Umsetzung von kurzen Wegen für den Mauszeiger kann je nach Benutzergruppe auch die falsche sein (wie bereits auf Seite 54 angedeutet). Um derartige Mängel in Erfahrung zu bringen, ist es notwendig die Zielgruppe bei der Benutzung des Programmes zu beobachten.

Die Anwendung wird sich voraussichtlich nicht auf jeder Plattform, auf der sie ausgeführt werden kann, so verhalten, wie der Benutzer erwartet. Bestimmte Gestaltungen oder Verhaltensweisen von grafischen Elementen können von denen abweichen, die bei der Zielplattform üblich sind. Dies kann so etwas Einfaches sein, wie die Reihenfolge der Knöpfe in einem Dialog oder dessen Farben. Die Erweiterung der Benutzerführung um plattformspezifische Elemente könnte damit je nach Art und Häufigkeit der auftretenden Schwierigkeiten lohnenswert sein.

Das interaktive Tutorial, das den Lernenden in das Programm einführt, könnte durch ein Tutorial ersetzt werden, das auch bei Aktionen sinnvoll reagiert, die nicht den Vorschriften auf der Tutorial-Seite entsprechen. Eine weitere Möglichkeit wäre die Ergänzung des Tutorials um eine animierte Figur, die die Personifizierung des Tutorials vollendet. Diese Figur könnte auch benutzt werden, um zusätzliche Hilfestellungen während der Programmnutzung anzubieten. Wichtig ist hierbei, dass die Figur auf keinen Fall aufdringlich wird und dem Benutzer Informationen bietet, die er nicht haben will.

A Programmbedienung

Dieser Anhang enthält eine Beschreibung der Funktionen des Programms, die nicht bereits im Kapitel 5 im Rahmen der Umsetzung beschrieben wurden. Zur Veranschaulichung verwende ich Bildschirmfotos, die bei der Ausführung des Programms unter Linux entstanden sind.

A.1 Programmfenster

Das leere Hauptfenster des Programms ist in Bild A.1 zu sehen. Im oberen Bereich befindet sich die Menüleiste, rechts ist die Simulationssteuerung zu erkennen und am unteren Rand befindet sich eine Statuszeile. Der mittlere Bereich ist die Arbeitsfläche. Die Statuszeile ist in diesem Moment noch nicht weiter gefüllt. Links in der Statuszeile

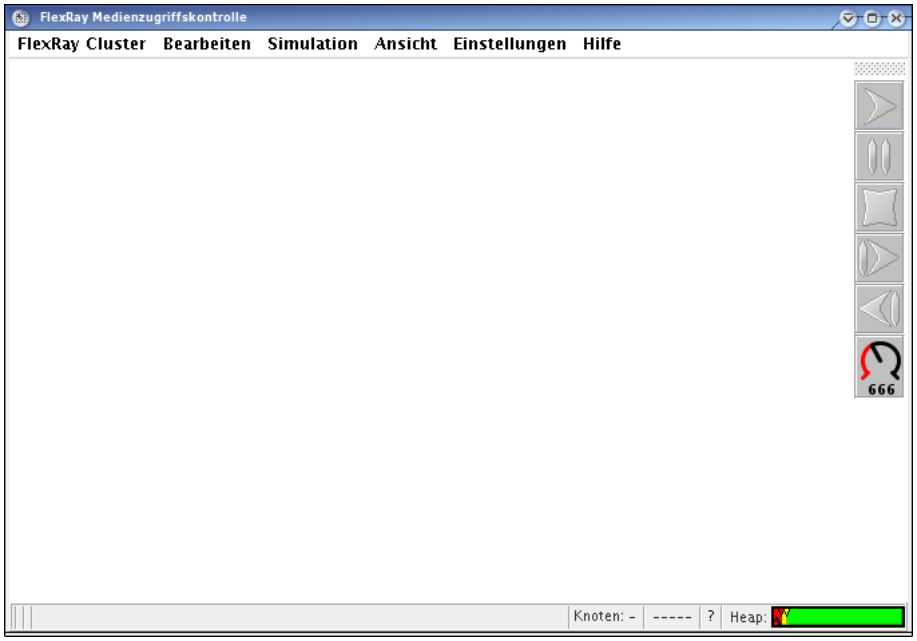


Bild A.1: Das leere Programmfenster

werden während der Simulation Informationen zum aktuellen Simulationsschritt angezeigt. Rechts beginnt die Information mit dem Wort *Knoten:*. Daneben wird später

die Anzahl der Knoten angezeigt. Es folgen fünf Bindestriche, die bei aktivierter Hilfe ersetzt werden: Durch *Tutorial*, wenn das interaktive Tutorial aktiv ist, und durch *Normal*, wenn die passive Hilfefunktion gewählt wurde. Das Fragezeichen rechts daneben ändert sich, sobald ein Modell erzeugt oder geöffnet wird. Ein rotes „M“ zeigt an, dass das Modell geändert (*modified*) wurde, ein grünes „S“ steht für die Information, dass das Modell seit der letzten Speicherung unverändert ist (*saved*). Der bunte Balken ganz rechts stellt den Speicher dar, der Java zur Verfügung steht („Heap“). Grün bedeutet verfügbaren Speicher, rot stellt den momentan belegten Speicher dar und gelb repräsentiert belegten, aber nicht genutzten Speicher. Die kleinen Pfeile oben zeigen die bisherigen Maximalwerte an, die Pfeile unten die bisherigen Minimalwerte. Dieser Balken ist für die Simulation selbst nicht wichtig, sondern erfüllt nur eine Diagnosefunktion. Bei Fehlverhalten des Programms kann man so erkennen, ob Speichermangel die Ursache ist.

A.2 Funktionen entlang der Menüstruktur

A.2.1 FlexRay-Cluster

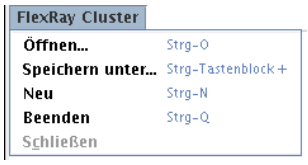


Bild A.2: Menü: FlexRay-Cluster

- **Öffnen...**: Hiermit lässt sich ein gepeichertes Modell wieder einlesen
- **Speichern unter...**: Das aktuelle Modell kann hierüber gespeichert werden
- **Neu**: Erzeugen eines neuen, leeren, Modells
- **Schließen**: Das aktuelle Modell wird geschlossen und das nächste erreichbare Modell angezeigt. Existieren in dem zu schließenden Modell noch ungespeicherte Änderungen, dann erscheint eine Nachfrage, ob diese gespeichert werden sollen.

A.2.2 Bearbeiten

- **Knoten** ▷
  - **Erzeugen**: Hinzufügen eines Knotens zum Modell

Bild A.3: Menü: Bearbeiten

  - **Entfernen**: Entfernen des markierten Knotens aus dem Modell

- **Kommunikationszyklus**: Einstellungen des Kommunikationszyklusses bearbeiten Bild A.9 auf Seite 82 zeigt den zugehörigen Dialog
- **Zwei Kanäle**: Auswahl zwischen einem und zwei Kanälen

A.2.3 Simulation

Bild A.4: Menü: Simulation

- **Start**: Kontinuierliche Simulation starten oder nach Pause fortsetzen
- **Schritt vorwärts**: Simulation einen Schritt vorwärts
- **Schritt zurück**: Simulation einen Schritt zurück
- **Pause**: Simulation pausieren
- **Reset**: Simulation anhalten und auf Ausgangswerte zurücksetzen
- **Geschwindigkeit** ▷
  - **Schneller**: Simulation schneller ablaufen lassen
  - **Langsamer**: Simulation langsamer ablaufen lassen

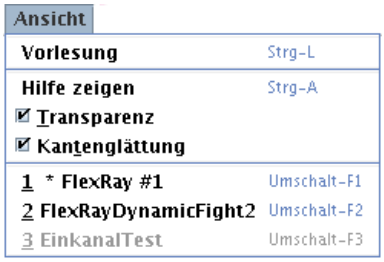


Bild A.5: Menü: Ansicht

A.2.4 Ansicht

- **Vorlesung**: Umschaltung in den Vollbild-Präsentationsmodus. Menü und Fensterdekorationen werden ausgeblendet. Das Menü ist über einen Rechtsklick auf die Arbeitsfläche erreichbar
- **Hilfe zeigen**: Einschalten des Anzeigebereichs für Tutorial und Hilfe. Nach dem Einschalten des Bereichs ändert sich der Text des Menüpunktes in **Hilfe verstecken**
- ☐ **Transparenz**: Aktivieren und Deaktivieren der Transparenzeffekte
- ☐ **Kantenglättung**: Kantenglättung ein- und ausschalten
- **(Modellnamen)**: Bis zu neun Modellnamen werden hier zur schnellen Umschaltung angeboten

A.2.5 Einstellungen

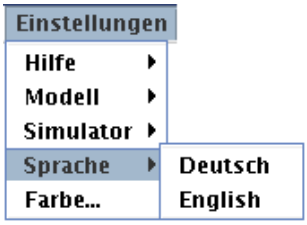


Bild A.6: Menü: Einstellungen

- **Hilfe** ▹: Der einzige aufklappende Punkt ist **Aktivieren beim Start** und bestimmt, ob beim Start des Programmes automatisch die Hilfe angezeigt wird

- **Modell** ▷: Der einzige aufklappende Punkt ist **Standardmodell auswählen...**. Hiermit kann das Modell ausgewählt werden, das beim Start des Programmes automatisch geladen werden soll
- **Simulator** ▷: Der einzige aufklappende Punkt ist **Geschwindigkeit übernehmen**. Die momentan eingestellte Simulationsgeschwindigkeit wird hiermit als Vorgabe gespeichert
- **Sprache** ▷:
  - **Deutsch**: Wählt die deutsche Sprache aus
  - **English**: Selects english language
- **Farbe...**: Ruft den Dialog des Rahmenwerks für allgemeine Farbeinstellungen auf. Damit können Hintergrund-, Vordergrund- und Highlightfarbe eingestellt werden

A.2.6 Hilfe

Bild A.7: Menü: Hilfe

- **Hilfe...**: Dieselbe Funktion wie **Hilfe zeigen** im Menü **Ansicht** – die Hilfe wird aktiviert
- **Über...**: Ein Informationsfenster über den Autor des Programms wird angezeigt

A.3 Erstellung und Konfiguration von Modellen

Ich beschränke mich bei dieser Beschreibung auf ein Modell mit einem Kanal. Die Vorgehensweisen sind für den zweiten Kanal analog. Ähnlich der Beschreibung im Tutorial führe ich die Erklärungen anhand eines Beispiel-Clusters mit Stern-Topologie durch. Mit dem Menüpunkt **Neu** aus dem Menü **FlexRay Cluster** kann ein neues Cluster-Modell angelegt werden. Das Modell ist zunächst leer, so dass nur ein einfacher Kommunikationszyklus angezeigt wird (Bild A.8). Die Marke „Zykluskonfiguration“ verdeutlicht, dass nicht die Simulation läuft, sondern die Konfigurationssicht aktiv ist. Durch Druck auf den Knopf links unten mit dem Hammer-und-Schraubendreher-

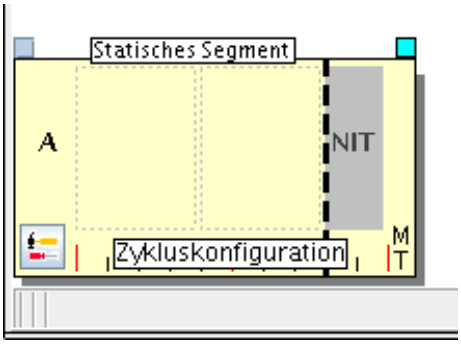


Bild A.8: Der erste Kommunikationszyklus

Symbol oder durch die Auswahl von **Kommunikationszyklus** aus dem Menü **Bearbeiten** kann die Konfiguration des Kommunikationszyklusses aufgerufen werden. Bild A.9 zeigt den erscheinenden Dialog. In diesem Dialog werden die grundlegenden Parameter

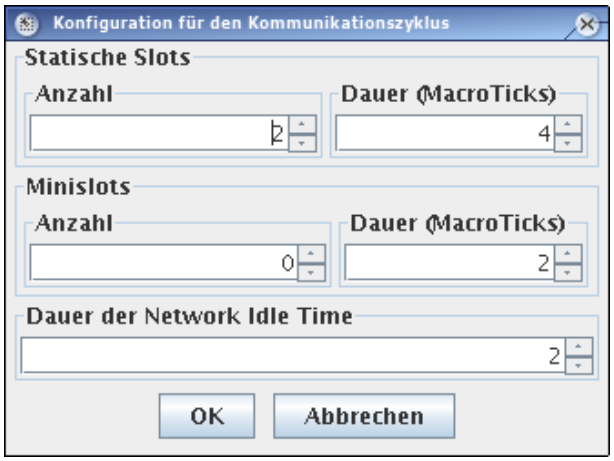


Bild A.9: Konfiguration des Kommunikationszyklusses

für den Kommunikationszyklus eingestellt. Das sind die Anzahl der statischen Slots, ihre Länge in Microticks, die Anzahl der Minislots, ihre Länge in Microticks und die Länge der NIT in Microticks. Für dieses Beispiel wähle ich vier statische Slots und fünf Minislots aus. Die NIT belasse ich auf ihrem vorgegebenen Wert.

Als nächstes müssen dem Cluster noch einige Knoten hinzugefügt werden. Mit der Funktion **Erzeugen** aus dem Untermenü **Knoten** des Menüs **Bearbeiten** oder der Tastenkombination **Strg+I** sind schnell vier Knoten erzeugt. Bild A.10 zeigt, wie sich das Modell dann präsentiert.

Mit Hilfe der Maus können die Knoten nun angeordnet werden, so dass sie sich nicht gegenseitig im Weg sind. Mit dem Konfigurationsknopf eines Knotens lässt sich

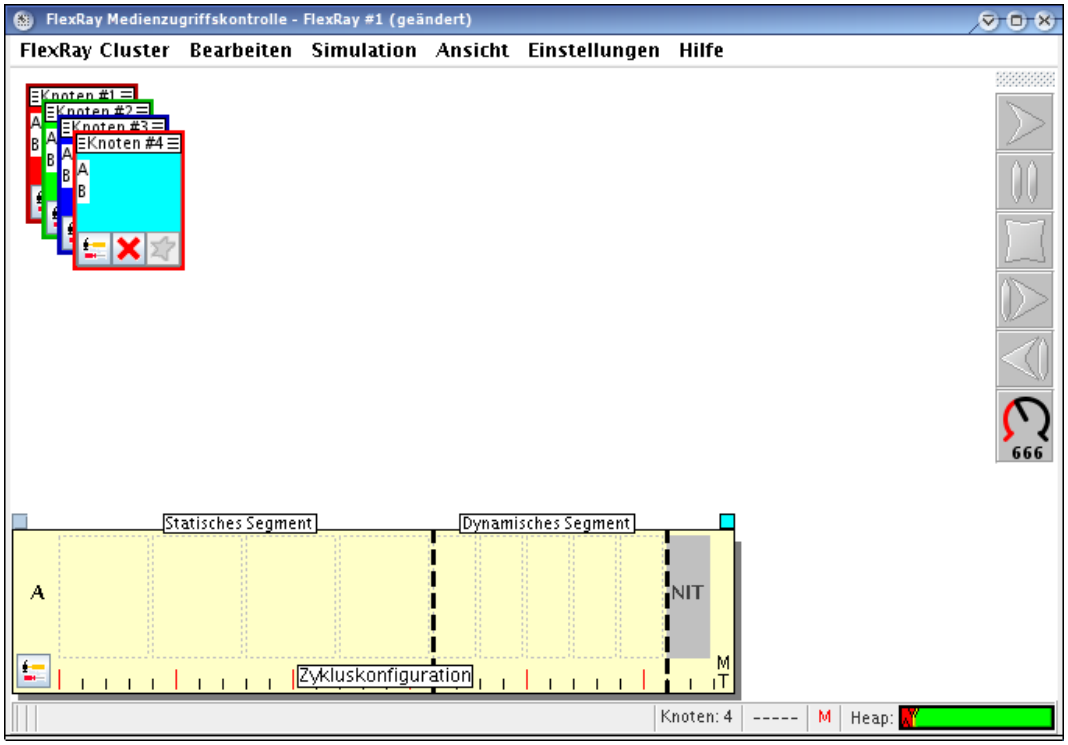


Bild A.10: Erstes Modell ohne Verbindungen

sein Konfigurationsdialog aufrufen. Bild A.11 zeigt den ersten Dialog von Knoten 1. Darin wird Kanal A zur Auswahl angeboten, der nicht vorhandene Kanal B bleibt unerreichbar. Ganz rechts ist der Knopf für die Auswahl der Farbe des Knotens.Darunter befindet sich noch ein Eingabefeld, um den Knoten zu benennen. Wird das Feld leer gelassen, dann heißt der Knoten automatisch „Knoten #“ gefolgt von seiner Knotennummer[1]. Einen der Knoten nenne ich für den Rest der Beispiele „Sensor X“.

Wird in die Auswahlbox für Kanal A nun ein Häkchen geklickt, erscheinen zusätzliche Optionen für die Knotenkonfiguration, und zwar die Parameter für das Sendeverhalten in Kanal A. Bild A.12 zeigt diese Einstellungen. In der ersten Zeile kann ein Bereich von Nachrichtenlängen angegeben werden, aus dem zufällig in jedem Kommunikationszyklus eine ausgewählt wird. Nach dem FlexRay-Protokoll bedeutet eine Nachricht der Länge 1 automatisch, dass der Knoten nicht sendet, denn die Mindestgröße für einen dynamischen Slot beträgt zwei Macroticks. Werden beide Werte gleich eingestellt, dann sendet der Knoten Nachrichten konstanter Länge.

Darunter befindet sich die Einstellung für das Aussetzen von Sendungen. Ähnlich wie bei der Nachrichtenlänge kann hier ein Bereich angegeben werden, aus dem zufällig gewählt wird, wie viele Zyklen der Knoten warten soll, bis er wieder im dynamischen

[1]Bei englischsprachiger Einstellung wird „Node #“ anstelle von „Knoten #“ angezeigt.

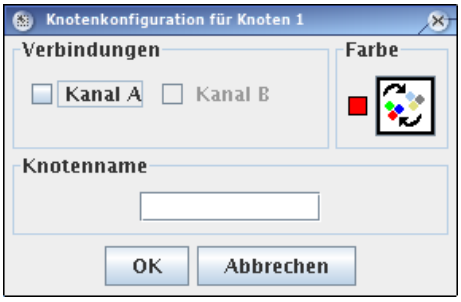


Bild A.11: Erster Knotendialog

Bild A.12: Erweiterter Knotendialog

Segment sendet. Auch hier gilt, dass bei gleicher Einstellung beider Werte immer diese Anzahl an Zyklen gewartet wird.

Die letzte Zeile in diesem Bereich der Knotenkonfiguration bezieht sich auf den Aktionsknopf, mit dem der Benutzer willkürlich das Senden einer Nachricht starten kann. Genau wie in der ersten Zeile wird hier ein Bereich angegeben, wie lang diese Nachricht sein soll. Bild A.13 zeigt das Anwendungsfenster, nachdem die Knoten alle mit Kanal A verbunden wurden.

Ein Mausklick auf einen Slot in der Darstellung des Kommunikationszyklusses bringt den Dialog für die Knotenzuordnung zum Vorschein. In diesem Dialog werden neben dem Zustand „unbenutzt“, der dazu dient, die Zuordnung aufzuheben, diejenigen Knoten angezeigt, die für diesen Slot in Frage kommen. Wie in Bild A.14 zu erkennen ist, wird, wenn vorhanden, der Name eines Knotens in diesem Dialog verwendet. Um die Zuordnung weiter zu erleichtern, sind die Auswahlfelder der Knoten in der jeweiligen Knotenfarbe umrahmt. Werden nun einige Knoten-Slot-Zuordnungen vorgenommen, kann der Kommunikationszyklus aussehen, wie in Bild A.15 gezeigt ist.

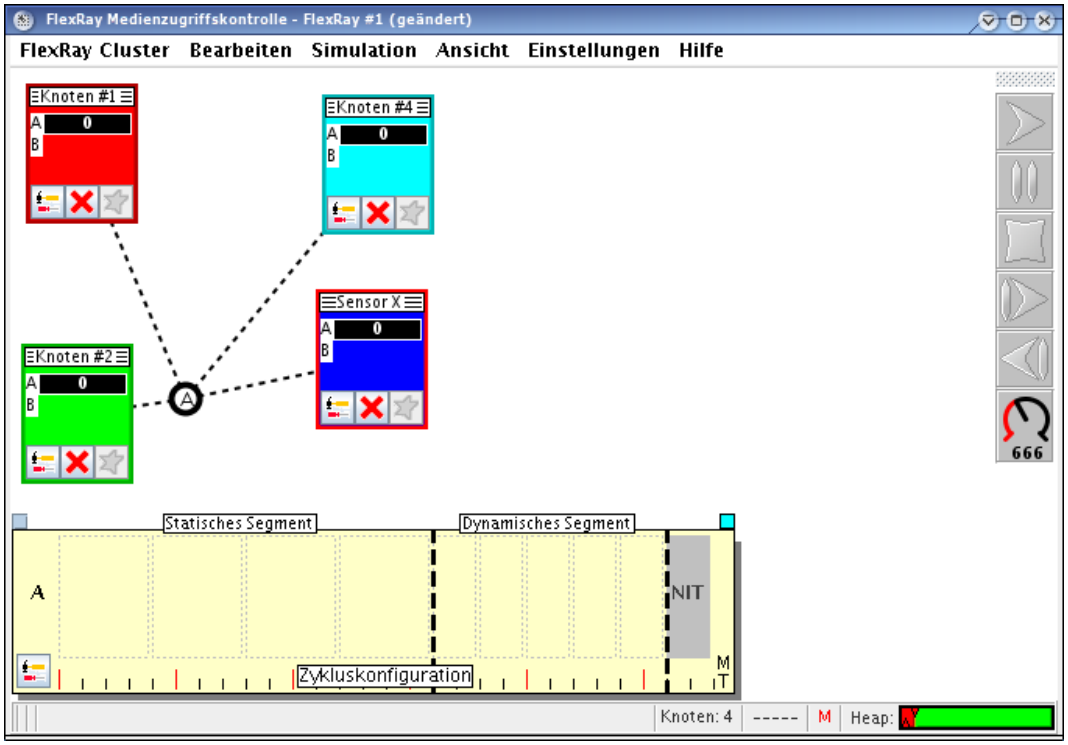


Bild A.13: Vier Knoten verbunden mit Kanal A

A.4 Simulation

Die Steuerung der Simulation geschieht über die Steuerungsknöpfe, wie sie in Bild A.1 und A.10 rechts unterhalb des Menüs zu sehen sind. Die Bedeutungen der Knöpfe von oben nach unten:

- **Start**: Starten der kontinuierlichen Simulation oder Fortsetzen der Simulation nach dem Pausieren
- **Pause**: Pausieren der Simulation. Sie kann dann schrittweise bedient oder mit dem Start-Knopf fortgesetzt werden
- **Stop**: Anhalten der Simulation und zurücksetzen auf den Anfangszustand.
- **Vor**: Ein Schritt vorwärts. Dies funktioniert während einer Pause oder aus dem Anfangszustand heraus
- **Zurück**: Ein Schritt zurück
- **Geschwindigkeit**: Der Drehknopf regelt die Verzögerungsdauer in Millisekunden zwischen den einzelnen Schritten während der kontinuierlichen Simulation

Bild A.14: Knotenauswahl für Slot 0, Kanal A

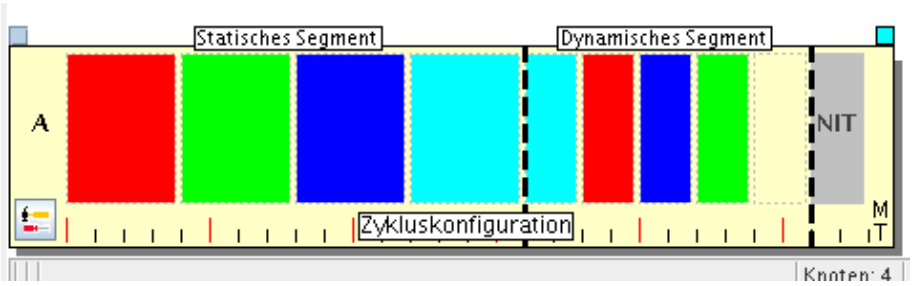


Bild A.15: Kommunikationszyklus nach Knotenzuordnung

Nach dem Start der Simulation werden die für die Konfiguration verwendeten Markierungen am Kommunikationszyklus ausgeblendet. Bild 5.7 auf Seite 65 zeigt das Programm während der Simulation. Wird die Simulation gestoppt, dann werden die Markierungen wieder eingeblendet.

B FlexRay-Protokollparameter

In diesem Anhang liste ich diejenigen in der FlexRay-Protokollspezifikation definierten Parameter auf, die für die interaktive Visualisierung und ihre Konzeption berücksichtigt wurden oder direkt damit zusammenhängen. Die Bedeutung der in den Präfixen der Bezeichner verwendeten Buchstaben ist in Tabelle B.1 aufgeführt, die verwendeten Maßeinheiten in Tabelle B.2 und die Werte sind schließlich in Tabelle B.3 zu finden.

Präfix	Typ	Beschreibung
c	Protokoll-konstante	Werte, die Eigenschaften oder Grenzwerte des Protokolls definieren. Diese Werte sind für das Protokoll vorgegeben und nicht einstellbar.
v	Knotenvariable	In den Knoten gespeicherte Werte, die abhängig von Zeit und Ereignissen variieren[a].
g	Clusterparameter	Parameter, die in jedem Knoten eines Clusters denselben Wert haben müssen.
p	Knotenparameter	Parameter, die in jedem Knoten unterschiedliche Werte haben können.
d	Dauer	Werte, die die Dauer zwischen zwei Zeitpunkten beschreiben.
t	Zeit	Werte, die einen Zeitpunkt beschreiben.

[a]Die FlexRay-Protokollspezifikation gibt noch „*etc.*“ als Variationsgrund an.

Tabelle B.1: Präfixe bei den Namen der FlexRay-Parameter. Auszug aus [Fle05c, Tabellen 1-2 und 1-3].

Einheit	Name
μT	Microtick
MT	Macrotick
μs	Mikrosekunde

Tabelle B.2: Verwendete Einheiten bei den FlexRay-Parametern.

Parameter	Bedeutung	Wertebereich [Einheit]
pChannels	Kanäle, mit denen der Knoten verbunden ist	A, B oder A&B
pdMicrotick	Dauer eines Microticks	*pSamplesPerMicrotick*-*gdSampleClockPeriod* [μs]
gdMinislot	Dauer eines Minislots	2 - 63 [MT]
gdStaticSlot	Dauer eines statischen Slots	4 - 661 [MT]
gdSymbolWindow	Dauer eines symbol window	0 - 142 [MT]
gdNIT	Dauer der Network Idle Time	2 - 805 [MT]
gdCycle	Dauer eines Kommunikationszyklusses	10 - *cdCycleMax*[μs]
cdCycleMax	Höchstmögliche Dauer eines Kommunikationszyklusses	10 - 16000 [μs]
gCycleCounterMax	Höchste Nummer eines Kommunikationszyklusses	63 [MT]
vCycleCounter	Zähler für Kommunikationszyklen	0 - *gCycleCounterMax* [MT]

Tabelle B.3: FlexRay-Parameter. Auszug aus [Fle05c, Anhänge A und B].

C Besonderheiten der Implementierung

C.1 Komponentenbauweise der Visualisierung

Um die Visualisierung modular und ressourcenschonend zu gestalten, habe ich mich dafür entschieden, sie entlang der JFC/Swing-Architektur zu gestalten. So habe ich nicht eine Zeichenfläche, auf der nacheinander alle Modellelemente gezeichnet werden, sondern jedes Modellelement hat eine eigene Visualisierungsklasse. Die Knoten werden durch eine spezielle *NodeComponent*-Klasse dargestellt, die eine JFC/Swing-Komponente ist. Der Kommunikationszyklus, die Stern-Kupplungen und das Medium haben ebenfalls eigene Anzeigekomponenten. Durch diese Organisation ist es einfach möglich, mit geringem Pflegeaufwand einige Eigenschaften von JFC/Swing auszunutzen:

- **Modularität.** Die Anpassung einer Komponente kann unabhängig von allen anderen Komponenten erfolgen. Neue generische Eigenschaften können leicht allen Komponenten hinzugefügt werden, zum Beispiel die Fähigkeit, mit der Maus verschoben zu werden.

- **Kompositionalität.** Komponenten können als Behälter für andere Komponenten dienen. Die Knoten und die Darstellung des Kommunikationszyklusses enthalten beispielsweise Knöpfe. Dies sind eigene JFC/Swing-Komponenten.

- **Selektive Zeichnung.** Durch die Komponentenbauweise kann JFC/Swing die Anzeige selbst verwalten und Komponenten bei Bedarf anweisen, sich selbst zu zeichnen. So muss nicht die komplette Berechnung des Zeichenflächeninhaltes durchgeführt werden, wenn nur wenige Objekte sich ändern.

- **Ereignisbehandlung.** Die Komponenten können als Quellen und Ziele für Ereignisse dienen. Ein Beispiel hierfür sind Maus-Ereignisse. Das Überstreichen eines Objektes mit dem Mauszeiger kann so direkt an dieses Objekt gemeldet werden. Aufwändige Berechnungen zur Mausposition und zugehörige Methodenaufrufe sind dadurch nicht erforderlich.

C.2 SpringUtilities

Zur einfacheren Programmierung der Knotenkonfigurationsdialoge habe ich die externe Klasse *SpringUtilities* aus dem Java-Tutorial [WC06], die die Verwendung des *SpringLayout* erleichtern soll, modifiziert und diese modifizierte Version verwendet. Ursprünglich erlaubt *SpringUtilities* die Anordnung von Komponenten in einem rechteckigen $x \times y$ Gitter, wobei die Anzahl der anzuordnenden Komponenten genau $x \cdot y$ sein muss. Meine Modifikation besteht nun darin, diese Einschränkung aufzuheben, und übriggebliebene Zellen des Gitters automatisch leer zu lassen.

C.3 Transparenz

Für einen einfachen wiederverwendbaren Transparenzeffekt habe ich eine eigene Klasse für JFC/Swing-Komponenten entwickelt, abgeleitet von der JFC/Swing-Komponentenklasse *JComponent*. Meine Klasse, *AlphaBlendingComponent*, kann *JComponent* ersetzen. Dadurch ist es möglich, in beliebigen eigenen Komponenten Transparenzeffekte zu verwenden, indem man von *AlphaBlendingComponent* ableitet anstelle von *JComponent*. Die anfängliche Transparenz der Komponente kann durch individuelle Einstellungen erfolgen. Um die Nutzung der Markierungseffekte zu erleichtern, können Transparenzstärken vordefiniert und ohne direkte Werteangabe umgeschaltet werden. Optional kann meine Klasse einen Thread starten, der bei diesen Umschaltungen das Umblenden weich durchführt, indem er die Transparenz schrittweise anpasst. Die Geschwindigkeit dieses Vorganges ist konfigurierbar.

C.4 Drehknopf

JFC/Swing kennt keinen Drehknopf. Da ich aus Platz- und Übersichtsgründen einen Drehknopf brauchte, musste ich selbst einen implementieren. Meine Klasse dazu habe ich *TurningKnob* genannt. Beim Entwurf des Knopfes standen mehrere Möglichkeiten für die Bedienung zur Auswahl: Eine Möglichkeit wäre gewesen, die Positionsmarkierung mit der Maus zu greifen und durch kreisförmige Bewegung der Maus bei gedrückter Taste die Positionsmarkierung zu verschieben, so als würde die Positionsmarkierung immer dort hin zeigen, wo sich der Mauszeiger befindet. Dies entspricht dem Verhalten von einigen Drehknöpfen, wie sie zum Beispiel in Steinberg Cubase SX [Ste06] oder Apple Logic [App06] zu finden sind. Ein anderes Bedienkonzept für einen solchen Drehregler gleicht dem eines universellen Schiebereglers: Bei gedrückter Maustaste regeln Mausbewegungen nach oben und unten oder nach links und rechts die Einstellung. Dieses Konzept findet bei einigen Komponenten von Apple Logic Anwendung oder bei Ableton Live [Abl06].

Aus technischen Gründen habe ich das zweite Bedienkonzept gewählt, denn die automatische Fokus-Konfiguration in JFC/Swing hat eine Umsetzung des ersten verhin-

dert: Die Beibehaltung des Fokusses auf den Drehknopf bei Bewegung des Mauszeigers außerhalb des Drehknopfs war leider nicht möglich. Um den Empfang von Mausbewegungen zu gewährleisten muss der Mauszeiger innerhalb der JFC/Swing-Komponente gehalten werden. Meine Drehknopf-Komponente verschiebt nach einer Mausbewegung den Mauszeiger zurück in ihren Ursprung. Um den Benutzer nicht durch einen zuckenden Mauszeiger zu verwirren, wird der Mauszeiger während des Regelvorganges ausgeblendet.

Die grafische Repräsentation des Drehknopfes ähnelt den Drehknöpfen von Ableton Live. Diese Darstellung ist auf natürliche Weise entstanden, als ich Versuche mit gezeichneten Skalen aufgrund ihrer Unübersichtlichkeit verworfen habe. Eine Andeutung des Knopfes mit Darstellung der Regelgrenzen und Anzeige des eingestellten Wertes sind bei einem geringen Platzangebot am einfachsten zu erfassen. Die Idee für die Färbung einer Hälfte des Regelbereiches habe ich schließlich von den Live-Knöpfen übernommen, nachdem das teilweise ausgefüllte Zeichnen des Knopfes unerwartete Schwierigkeiten brachte (siehe Anhang C.7.3).

C.5 Verschieben von Komponenten mit der Maus

Die Verschiebung von Komponenten per Maus habe ich generisch durch eine Klasse *ComponentDragger* realisiert. Zur Nutzung genügt es, eine Instanz dieser Klasse als *MouseListener* und *MouseMotionListener* einer beliebigen Komponente, die von *JComponent* abstammt, hinzuzufügen. Befindet sich diese Komponente in einem Behälter ohne Layout, dann kann sie durch den *ComponentDragger* innerhalb des Behälters bewegt werden. Dem *ComponentDragger* kann ein Rechteck übergeben werden, das den Bereich innerhalb der Komponente bestimmt, auf den er reagieren soll. Bei den Knoten in meinem Programm ist dies zum Beispiel der Titelbereich. Der *ComponentDragger* kann bei Komponenten, die von *AlphaBlendingComponent* (Abschnitt C.3) abstammen, automatisch ein weiches Aus- und Einblenden veranlassen.

C.6 Speicherung der XML-Daten

Über die Trennung von Modell- und Visualisierungsdaten hinaus (siehe Seite 70) habe ich jede Modell- und Anzeigekomponente mit der Fähigkeit ausgestattet, ihre XML-Repräsentation selbst als DOM-Dokument zurückzugeben. Ist eine Komponente dabei ein Behälter, so holt dieser Behälter die XML-Repräsentationen der in ihm enthaltenen Objekte ab und hängt diese in seine eigene XML-Repräsentation eine Ebene tiefer mit ein. Damit wird das XML-Dokument entlang der Objekthierarchie aufgebaut und die Struktur des fertigen XML-Dokuments entspricht genau der Struktur der Hierarchie im Modell.

Beim Einlesen der Daten wird ähnlich verfahren. An der Wurzel der Hierarchie

wird das gesamte Dokument entgegengenommen. Für die enthaltenen Unterstrukturen werden die entsprechenden Objekte erzeugt und ihnen die XML-Unterstruktur (die wieder Unterstrukturen enthalten kann) zur Verarbeitung übergeben.

Im Gegensatz zu einer zentralen Instanz zur Umsetzung zwischen XML und Objektstruktur verbessert diese Vorgehensweise die Wartbarkeit und die Erweiterbarkeit des Programms. So ist zum Beispiel bei fehlerhaften XML-Daten deren Ursprung sofort klar, und Modellkomponenten können ohne weitreichende Arbeiten an der Klassenhierarchie einfach ausgetauscht oder erweitert werden, ohne den XML-Datenaustausch zu gefährden.

C.7 Fehler in den Java-Bibliotheken

In diesem Abschnitt beschreibe ich kurz Probleme in den Java-Bibliotheken, auf die ich während meiner Umsetzung getroffen bin. Die Überschriften beginnen im Folgenden mit der Nummer, unter der der Fehler in der Fehlerdatenbank des Sun Developer Network [Ver06] recherchiert werden kann.

C.7.1 6358355: Monotonie bei float-Literalen

Bei der Verwendung von *float*-Literalen bleibt die Monotonie nicht bestehen. Während der Programmierung habe ich für das Zeichnen von Grafiken aufsteigende *float*-Werte im Quelltext verwendet, die Grafiken wurden jedoch nicht korrekt gezeichnet. Zwar hat das *float* Format eine definierte Ungenauigkeit, doch die Monotonie muss bei der Umrechnung erhalten bleiben, was beim Java-Compiler aber nicht der Fall ist. Als Alternative können die Werte in *double* angegeben und später nach *float* konvertiert werden. Das Problem ist allein auf die Compile-Zeit beschränkt: Die vom Compiler erzeugten Binärdateien enthalten bereits falsche Repräsentationen der *float*Werte. Während der Laufzeit eines Programms eingegebene Werte werden korrekt in *float*-Repräsentationen umgerechnet.

C.7.2 6296446: XML wird nicht eingerückt

Beim Schreiben der Modelldaten in XML-Dateien bin ich auf diesen Fehler gestoßen, der das manuelle Überprüfen und Editieren der Modelldaten erschwert hat. Alle Ausgaben werden an der ersten Textspalte ausgerichtet. Eine der Dokumentenstruktur entsprechende Einrückung findet nicht statt. Der Fehler ist angeblich in einer bestimmten Java-Version behoben worden, war jedoch in allen während meiner Arbeit verfügbaren Versionen von Java 5 enthalten. In der Fehlerdatenbank wird ein alternativer Weg zur Nutzung von *javax.xml.transform.Transformer* genannt, bei dem die Einrückung stattfindet.

C.7.3 6357387: Statt Kreissegmenten werden Vollkreise gezeichnet

Dieser Fehler trat im Zusammenhang mit Versuchen zu einer uhrenförmigen Darstellung der Macroticks und bei der Implementierung des Drehknopfes auf. Sollen Kreissegmente mit bestimmten Parametern gezeichnet werden, dann zeichnet die Java 2D API statt dessen Vollkreise, was zu unbrauchbaren Darstellungen führt.

C.7.4 6292739: Spracheinstellung wirkt sich nicht auf Tastenkürzeltexte aus

Wird zur Laufzeit die Lokalisierung umgeschaltet, dann bleiben in Menüs die Beschreibungen der Kurzbefehle in der vorigen Spracheinstellung bestehen. Wird zum Beispiel von Deutsch auf Englisch umgeschaltet, dann bleibt die Angabe *Strg* bestehen und wird nicht durch *Ctrl* ausgetauscht. Wird die Anwendung mit englischer Spracheinstellung neu gestartet, dann wird ordnungsgemäß *Ctrl* in den Menüs angezeigt. Bild C.1 zeigt dies am Beispiel meines Programms. In der Fehlerdatenbank ist eine Alterna-

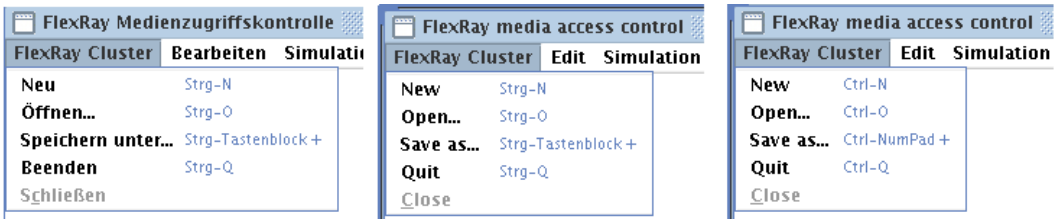


Bild C.1: Fehlerhafte Menüdarstellung bei Sprachumschaltung: Links vor der Sprachumschaltung von deutsch auf englisch, in der Mitte nach der Umschaltung. Rechts die korrekte englische Anzeige.

tive angegeben, um die Sprachumschaltung während der Laufzeit doch zu ermöglichen. Diese Alternative habe ich genutzt. Leider ist nicht garantiert, ob diese Möglichkeit mit zukünftigen Java-Versionen noch funktionieren wird.

C.7.5 4626632: null-Zeiger von Actions werden dereferenziert

Die genannte Fehlernummer ist nur der Ausgangspunkt für eine Reihe von Fehlern, die in diesem Fall zusammenhängen. Bei der Verwendung von *Actions* für die Menüprogrammierung von JFC/Swing kann es vorkommen, dass *null*-Zeiger innerhalb der JFC/Swing-Bibliotheken dereferenziert werden. Dies wirkte sich in meinem Programm bei der Umschaltung von Erklärungstexten für Menüpunkte aus. Es gibt keine alternative Möglichkeit, dies zu programmieren. Daher werden Erklärungstexte bei der

Sprachumschaltung nicht mit umgeschaltet. Angeblich ist dieser Fehler in Java 6 behoben.

Literaturverzeichnis

[Abl06] Ableton AG: *Live*. http://www.ableton.com/live/. Version: Juli 2006

[Alb06] Albotronic: *FlexRay Schulung*. http://www.albotronic.de/Schulung/Flexray.html. Version: August 2006

[And01] Anderson, John R.: *Kognitive Psychologie*. 3. Auflage. Spektrum Akademischer Verlag, 2001

[App06] Apple Computer International: *Logic Pro 7.2*. http://www.apple.com/de/logic/. Version: August 2006

[Bay06] Bayerische Motoren Werke AG: *Webseite*. http://www.bmw.de/. Version: August 2006

[BBSZ01] Böhringer, Joachim ; Bühler, Peter ; Schlaich, Patrick ; Ziegler, Hans-Jürgen: *Kompendium der Mediengestaltung für Digital- und Printmedien*. Springer, 2001

[Bei99] Beilner, Heinz: *Simulation. Folienkopien zur Vorlesung*. Version 5. Universität Dortmund, Fachbereich Informatik, 1999

[BFS87] Bratley, Paul ; Fox, Bennet L. ; Schrage, Linus E.: *A guide to simulation*. Springer, 1987

[BGS06] Böke, D. C. ; Grossman, D. ; Schuermans, R.: Effiziente Entwicklung von FlexRay-Systemen im Fahrzeug. In: *Elektronik Informationen* (2006), Nr. 1, S. 30–32

[BMW06a] BMW Group: *byteflight Webseite*. http://www.byteflight.com/homepage.htm. Version: April 2006

[BMW06b] BMW Group, EE21: *byteflight – A New High Speed Data Bus System for Safety-related Communication*. http://www.byteflight.com/presentations/introduction.pdf. Version: April 2006

[Bus96] Busse, Robert: *Feldbussysteme im Vergleich*. München : Pflaum, 1996

[CK04] Campione, Mary ; Kathy, Walrath: *The Java Tutorial*. Mai 2004 `http://java.sun.com/docs/books/tutorial/`

[Com04] Comer, Douglas E.: *Computernetzwerke und Internets*. 3., überarbeitete Auflage. Pearson Studium, 2004

[DEC06a] DECOMSYS Dependable Computer Systems, Hardware und Software Entwicklung GmbH: *DECOMSYS::SIMTOOLS*. `http://www.decomsys.com/htm/frs/3_flexray_simtools.htm`. Version: April 2006

[DEC06b] DECOMSYS Dependable Computer Systems, Hardware und Software Entwicklung GmbH: *FlexRay - Getting Started*. `http://www.decomsys.com/htm/frs/3_flexray_gettingstarted_ws.htm`. Version: August 2006

[DKR+05] Dengel, G. ; Krammer, J. (BMW AG) ; Ruh, J. (DaimlerChrysler AG) ; Schwaiger, E. ; R. (DeComSys GmbH), Nossal D. ; Oberhofer, K. (IXXAT Automation GmbH) ; Lucas, R. (Robert Bosch GmbH) ; Kuttenkeuler, S. (Sulzer GmbH) ; Quecke, H. (Vector Informatik GmbH): *FIBEX - Field Bus Exchange Format (MCD-2[FBX] Version 1.2)*. ASAM: Association for Standardisation of Automation and Measuring Systems, 2005

[dSP06] dSPACE GmbH: *Webseite*. `http://www.dspace.de/`. Version: April 2006

[Dud03] Dudenredaktion: *Duden. Das große Fremdwörterbuch*. 3. überarbeitete Auflage. Dudenverlag, 2003

[Dud06] Dudenredaktion: *Duden Band 1 - Die deutsche Rechtschreibung*. 24., völlig neu bearbeitete und erweiterte Auflage. Dudenverlag, 2006

[Eul92] Euler, Dieter: *Didaktik des computergestützten Lernens*. BW Bildung und Wissen, 1992

[Fle05a] FlexRay Consortium: FlexRay Communications System Prelimiary Central Bus Guardian Specification Version 2.0.9. (2005)

[Fle05b] FlexRay Consortium: FlexRay Electrical Physical Layer Specification v2.1, Revision A. (2005)

[Fle05c] FlexRay Consortium: FlexRay Protocol Specification v2.1, Revision A. (2005)

[Fle05d] FlexRay Consortium: FlexRay Protocol Specification v2.1, Revision A Errata Sheet Version 1. (2005)

[Fle06a] FlexRay Consortium: *FlexRay Frequently Asked Questions.* `http://www.flexray.com/faq.php`. Version: April 2006

[Fle06b] FlexRay Consortium: *FlexRay Webseite.* `http://www.flexray.com/`. Version: März 2006

[Fow06] Fowler, Amy: *A Swing Architecture Overview.* `http://java.sun.com/products/jfc/tsc/articles/architecture/`. Version: Mai 2006

[GBP00] Griessbach, Robert ; Berwanger, Josef ; Peller, Martin: byteflight – neues Hochleistungs-Datenbussystem für sicherheitsrelevante Anwendungen. In: *ATZ/MTZ „Automotive Electronics“* (Januar 2000)

[GHJV94] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design patterns: elements of reusable object-oriented software.* Reading, MA, USA : Addison-Wesley, 1994

[GL00] Ghosh, Sumit ; Lee, Tony S.: *Modeling and Asynchronous Distributed Simulation: Analyzing Complex Systems.* IEEE Press, 2000

[Gor87] Gordon, Geoffrey: *System Simulation.* 2nd edition. Prentice-Hall, 1987

[GWGK05] Gagné, Robert M. ; Wagner, Walter W. ; Golas, Katharine C. ; Keller, John M.: *Principles of Instructional Design.* 5th edition. Wadsworth/Thomson Learning, 2005

[Hol04] Holtwick, Peter: *Aufbau einer Simulationsumgebung für Medienübertragung über WLAN-Netzwerke*, Universität Dortmund, Diplomarbeit, 2004

[IP02] Issing, Ludwig (Hrsg.) ; Paul, Klimsa (Hrsg.): *Informationen und Lernen mit Multimedia und Internet.* 3. Auflage. Weinheim : Beltz PVU, 2002

[IXX06a] IXXAT Automation GmbH: *Einführung in FlexRay.* `http://www.ixxat.de/introduction_flexray_de,16541,8584.html`. Version: Juli 2006

[IXX06b] IXXAT Automation GmbH: *Bild zum FlexRay-Kommunikationszyklus.* `http://www.ixxat.de/portal/pics/flexray_uebersicht_bild1.jpg`. Version: März 2006

[IXX06c] IXXAT Automation GmbH: *Multibus-Analyser für FlexRay.* `http://www.ixxat.de/index.php?seite=flexray_analyser_de`. Version: März 2006

[KCH01] KRUTSCH, Kenneth F. ; CARGO, David S. ; HOWLETT, Virginia: *Professional Java Custom UI Components*. Birmingham, UK : WROX Press Ltd., 2001

[Ker01] KERRES, Michael: *Multimediale und telemediale Lernumgebungen*. 2. Auflage. München : Oldenbourg, 2001

[Kes04] KESELING SOFTWARE: *SKS Vokabel 1.4.1*. `http://www.keseling.de/vok.htm`. Version: Mai 2004

[KM06] KOHM, Markus ; MORAWSKI, Jens-Uwe: *KOMA-Script*. 2. überarbeitete und erweiterte Auflage. Lehmanns Fachbuchhandlung, 2006

[KMOS72] KÖCHER, D. ; MATT, E. ; OERTEL, C. ; SCHNEEWEISS, H.: *Einführung in die Simulationstechnik*. Deutsche Gesellschaft für Operations Research, 1972

[Koc94] KOCHER, Hartmut: *Entwurf und Implementierung einer Simulationsbibliothek unter Anwendung objektorientierter Methoden*. Stuttgart, Germany, Universität Stuttgart, Elektrotechnik, IND, Diss., 1994

[Kre86] KREUTZER, Wolfgang: *System Simulation: Programming Styles and Languages*. Addison-Wesley, 1986

[Kri05] KRITZENBERGER, Huberta: *Multimediale & interaktive Lernräume*. München : Oldenbourg Wissenschaftsverlag, 2005

[Kru02] KRUMM, Heiko: *Vorlesungsfolien Rechnernetze und verteilte Systeme*. `http://ls4-www.informatik.uni-dortmund.de/RVS/RvSfolien.zip`. Version: 2002

[Kru06] KRUMM, Heiko: *Animationen zu Rechnernetze und verteilte Systeme*. `http://ls4-www.informatik.uni-dortmund.de/RVS/MA/hk/OrdnerVertAlgo/VertAlgo.html`. Version: Mai 2006

[Lam88] LAMPRECHT, Günther: *Einführung in die Programmiersprache SIMULA*. Vieweg, 1988

[Lea97] LEA, Doug: *Concurrent Programming in Java*. Rading, MA, USA : Addison-Wesley, 1997

[Lüt04] LÜTZENKIRCHEN LEHRSYSTEME: *Kombinatorische Funktionen 1 (Demo)*. `http://home.t-online.de/home/LLehrsysteme/Lernprogramm2.htm`. Version: Mai 2004

[Man06] Manske, Thilo: *Wireless Local Area Networks.* http://einstein.informatik.uni-oldenburg.de/lehre/semester/seminar/02ss/wlan/WLANnode10.html#SECTION10044300000000000000. Version: Mai 2006

[Mar03] Marwedel, Peter: *Embedded System Design.* Kluwer Academic Publishers, 2003

[Mar06] Marwedel, Peter: *Folien zur Vorlesung „Eingebettete Systeme“.* Universität Dortmund, Fachbereich Informatik, Lehrstuhl XII, April 2006

[MC06] Microcomputer-Consulting, HILF!GmbH: *FlexRay Schulung.* http://www.hilf.de/de/site/schulung/kurse/flexray_schulung.html. Version: August 2006

[Mic06] Microsystems, Sun: *J2SE Name and Version Change.* http://java.sun.com/j2se/j2se-namechange.html. Version: September 2006

[Mir06] Mirabilis Design Inc.: *FlexRay Bus and Network Topology Architecture Modeling.* http://www.mirabilisdesign.com/WebPages/mdi_demonstration/automotive/FlexRay/FlexRay.html. Version: März 2006

[MOS05] MOST Cooperation: MOST Specification, Rev. 2.4. (2005)

[MOS06a] MOST cooperation: *MOST Network.* http://www.mostcooperation.com/technology/network/index.php. Version: April 2006

[MOS06b] MOST cooperation: *Webseite.* http://www.mostcooperation.com/. Version: April 2006

[NHHM+04] Niegemann, Helmut M. ; Hessel, S. ; Hochscheid-Mauel, D. ; Aslanski, K. ; Deimann, M. ; Kreuzberger, G.: *Kompendium E-Learning.* Berlin : Springer, 2004

[Nie01] Niegemann, Helmut M.: *Neue Lernmedien konzipieren, entwickeln, einsetzen.* Bern : Huber, 2001

[NS06] Nolte-Spicker, Alexandra: *Interaktive Visualisierung von Weg-Zeit-Diagrammen.* Diplomarbeit an der Universität Dortmund, Fachbereich Informatik, Lehrstuhl XII, 2006

[Pho97] Phoenix Contact Gmbh & Co.: *Grundkurs Sensor/Aktor Feldbustechnik.* Vogel, 1997

[Pil04] PILZ, Markus: *Interaktive Simulation von Prozeßnetzwerken.* Diplomarbeit an der Universität Dortmund, Fachbereich Informatik, Lehrstuhl XII, 2004

[Pro93] PROJEKTGRUPPE DOLL: *Entwicklung eines Hypermedia-Systems zur Gestaltung und Präsentation von Unterrichtseinheiten.* Universität Dortmund, Fachbereich Informatik, LS1, 1993

[Rec05] RECHENBERG, Peter: *Technisches Schreiben.* Carl Hanser Verlag, 2005

[Rei98] REISSENWEBER, Bernd: *Feldbussysteme.* Oldenbourg Verlag, 1998

[RM05] RINN, Ulrike (Hrsg.) ; MEISTER, Dorothee M. (Hrsg.): *Medien in der Wissenschaft.* Bd. 21: *Didaktik und neue Medien.* Münster : Waxmann, 2005

[Rob06a] ROBERT BOSCH GMBH: *Controller Area Network (CAN).* `http://www.can.bosch.com/`. Version: April 2006

[Rob06b] ROBERT BOSCH GMBH: *Time Triggered CAN (TTCAN).* `http://www.ttcan.com/`. Version: April 2006

[Sch02] SCHULMEISTER, R: *Grundlagen hypermedialer Lernsysteme: Theorie-Didaktik-Design.* Oldenbourg Wissenschaftsverlag, 2002

[Sho82] SHOEMAKER, S. (Hrsg.): *Computer Networks and Simulation II.* North-Holland, 1982

[SM00] SCHUHMANN, H. ; MÜLLER, W.: *Visualisierung. Grundlagen und allgemeine Methoden.* Berlin : Springer, 2000

[SM06] SUN MICROSYSTEMS, Inc: *Java Foundation Classes (JFC/Swing).* `http://java.sun.com/products/jfc/`. Version: September 2006

[Som06] SOMMER, Egon: *Vorlesungsskript „Kommunikation und Netzwerkmanagement“.* `http://www.e-technik.fh-muenchen.de/fb/lab/lsa/aut/docs/carbus/komnet0.4.pdf`. Version: Mai 2006

[Spr95] SPRAVE, Uwe: *Eine Lernkomponente für den Hypermedia-Prototyp HyperDoLL*, Universität Dortmund, Diplomarbeit, 1995

[Ste06] STEINBERG MEDIA TECHNOLOGIES GMBH: *Cubase SX 3.* `http://www.steinberg.de/35_0.html`. Version: Juli 2006

[The06] THE MATHWORKS: *Webseite.* `http://www.mathworks.com/`. Version: April 2006

[Thi06] THIELE, Lothar: *Folien zur Vorlesung „Eingebettete Systeme“*. ETH Zürich, Fachgruppe Technische Informatik, August 2006 http://www.tik.ee.ethz.ch/tik/education/lectures/ES/slides/8_Communication2.pdf

[Tra06a] TRANSFERZENTRUM MIKROELEKTRONIK: *Kurzeinführung in FlexRay*. http://www.tzm.de/FlexRay/FlexRay_Introduction.html. Version: April 2006

[Tra06b] TRANSFERZENTRUM MIKROELEKTRONIK: *FlexConfig manual for software version S2V7-F*. Mai 2006

[TTA06a] TTA-GROUP: *Legal Notice (for TTP specification request)*. http://www.ttagroup.org/technology/notice.htm. Version: April 2006

[TTA06b] TTA-GROUP: *Webseite*. http://www.ttagroup.org/. Version: April 2006

[TTA06c] TTAUTOMOTIVE SOFTWARE GMBH: TTXPlan. http://ttautomotive.com/products/doc/TTAutomotive-TTX-Plan-Flyer.pdf. Version: Juli 2006

[TTT04] TTTECH COMPUTERTECHNIK AG: *Sicherheit beim Bremsen und Lenken*. 2004

[TTT06a] TTTECH COMPUTERTECHNIK AG: *Lecal Notice (for TTP specification request)*. http://www.tttech.com/technology/legalnotice.htm. Version: April 2006

[TTT06b] TTTECH COMPUTERTECHNIK AG: *Webseite*. http://www.tttech.com/. Version: April 2006

[Tys99] TYSZER, Jerzy: *Object-Oriented Computer Simulation of Discrete-Event Systems*. Kluwer Academic Publishers, 1999

[Ull05] ULLENBOOM, Christian: *Java ist auch eine Insel*. 5. Auflage. Galileo Computing, 2005

[Vec06a] VECTOR INFORMATIK GMBH: *FlexRay Workshop*. http://www.vector-cantech.com/vi_class_flexray_de,,1296.html. Version: August 2006

[Vec06b] VECTOR INFORMATIK GMBH: *Media Oriented Systems Transport (MOST)*. http://www.vector-cantech.com/vi_most_de,,2816.html. Version: August 2006

[Vec06c] VECTOR INFORMATIK GMBH: *Webseite*. http://www.vectorinformatik.de/. Version: August 2006

[Ver06] VERSCHIEDENE AUTOREN: *Sun Developer Network: Bug Database*. http://bugs.sun.com/bugdatabase/index.jsp. Version: August 2006

[Vie03] VIEWEG, Detlef (3SOFT GmbH): IDB1394 oder MOST? In: *automotive* (2003), Nr. 5, S. 47 – 49

[Völ06] VÖLKEL, Frank: BMW X5: Erstes Auto mit FlexRay-Datenbus. In: *Tom's Hardware Guide Business* (14. August 2006). http://business.thgweb.de/2006/08/16/bmw_x5_flexray_datenbus_neu/index.html

[WC06] WALRATH, Kathy ; CAMPIONE, Mary: *SpringUtilities*. http://java.sun.com/docs/books/tutorial/uiswing/layout/example-1dot4/SpringUtilities.java. Version: Juli 2006

[Whe06] WHEELER, Paul: Audio-Genuss auf vier Rädern. In: *Elektronik automotive* (August 2006). http://www.elektroniknet.de/index.php?id=478

[WIZ06] WIZARD EDUCATION SYSTEMS WES: *Hydrostatik-Lernprogramm HYDRO.EXE*. http://www.physikcd.ch/cal22.html. Version: August 2006

[WL88] WELCH, Jennifer L. ; LYNCH, Nancy: A New Fault-Tolerant Algorithm for Clock Synchronization. In: *Information and Computation* 77 (1988), April, Nr. 1, S. 1–36

[WP06] WINKLER, Manfred ; PALLIERER, Roman (DeComSys GmbH): Modellbasiertes Systemdesign für FlexRay. In: *automotive* (2006), Nr. 5-6, S. 30 – 33