

Prüfer: Prof. Dr. rer. nat. Kurt Rothermel

Betreuerin: Dr. rer. nat. Cora Burger

Betreuer: Dipl.-Inf. Rolf Mecklenburg

begonnen am: 18.05.1998

beendet am: 18.11.1998

CR-Nummer: K.3.1, K.3.2, H.5.1, I.3.2, C.2.2, C.2.4

Studienarbeit Nr. 1715

**"Erweiterung des Java-Baukastens zur
Visualisierung von Protokollen:
Erweiterung für die Visualisierung
von Endlichen Automaten"**

Jürgen Rau

Fakultät Informatik
Institut für Parallele und Verteilte
Höchstleistungsrechner
Universität Stuttgart
Breitwiesenstr. 20-22
70565 Stuttgart

Inhaltsverzeichnis

INHALTSVERZEICHNIS.....	1
KAPITEL 1:	
EINLEITUNG.....	3
1 TEILAUFGABEN DER STUDIENARBEIT	3
2 ÜBERBLICK.....	4
KAPITEL 2:	
BESCHREIBUNG DES ALTEN BAUKASTENS.....	6
1 DARSTELLUNG UND EINBINDUNG.....	6
2 KONTROLL-INTERFACE	7
3 SCHNITTSTELLE ZUM BAUKASTEN.....	7
3.1 <i>Prozesse</i>	7
3.2 <i>Nachrichten</i>	8
3.3 <i>Darstellungssteuerung</i>	9
4 ZIELSETZUNG DES BAUKASTENS	10
5 DEFIZITE DES BAUKASTENS.....	10
6 VORGENOMMENE ÄNDERUNGEN	10
KAPITEL 3:	
GRUNDLEGENDE KONZEPTE DES NEUEN BAUKASTENS	12
1 TERMINOLOGIE.....	12
2 STRUKTUREN.....	12
KAPITEL 4:	
BESCHREIBUNG DER NEU ERSTELLTEN BAUKASTENKOMPONENTEN.....	15
1 GRUNDSTRUKTUR.....	15
2 ZUSTÄNDE.....	16
3 ÜBERGÄNGE.....	17
4 PRÄDIKATE.....	18
5 AUTOMATENDARSTELLUNG	20
6 VARIABLEN	21
7 TIMER.....	22
8 WARTESCHLANGEN.....	23
9 STATUSBAR	24
10 SAMMELFENSTER	25
11 ABLAUFBEISPIEL	26
KAPITEL 5:	
BESCHREIBUNG EINES UMFANGREICHEN ANWENDUNGSBEISPIELS.....	29
1 GRUNDKONZEPTE FÜR DEN EINSATZ DES NEUEN BAUKASTENS.....	29
2 BESCHREIBUNG DES „STOP AND WAIT“-PROTOKOLLS	30
2.1 <i>Grundlagen</i>	30
2.2 <i>Prinzip</i>	31
2.3 <i>Sender</i>	33
2.4 <i>Empfänger</i>	36
3 DAS DEMONSTRATIONSAPPLET	38

4	IMPLEMENTIERUNGSASPEKTE.....	41
5	ERKENNTNISSE	42
KAPITEL 6:		
IMPLEMENTATIONSBESCHREIBUNG DES NEUEN BAUKASTENS		43
1	BUFFERING	43
2	AUTOMATENDARSTELLUNG	43
2.1	<i>Zustände</i>	44
2.2	<i>Automatische Zustandsanordnung</i>	45
2.3	<i>Übergänge</i>	45
2.4	<i>Prädikate</i>	46
3	VARIABLEN	46
4	TIMER.....	47
5	WARTESCHLANGEN	47
6	EINBINDUNG DES ALTEN BAUKASTENS.....	47
7	WEITERENTWICKLUNG	47
KAPITEL 7:		
VERBESSERUNGS- UND ERWEITERUNGSVORSCHLÄGE		49
1	VERBESSERUNGEN AM GRAFIKBAUKASTEN	49
2	KONZEPTIONELLE ERWEITERUNGEN	50
ZUSAMMENFASSUNG		51
LITERATURVERZEICHNIS.....		52

Kapitel 1:

Einleitung

Im Bereich der verteilten Systeme und der Rechnernetze, aber auch in anderen Bereichen in denen rechnerbasierte Kommunikation stattfindet, werden Protokolle in großem Umfang eingesetzt, um einen geordneten, den jeweiligen Anforderungen entsprechenden Ablauf zu gewährleisten. Leider sind die meisten praxisrelevanten Protokolle sehr komplex und damit schwer zu überblicken. Dies macht sich vor allem in Bereichen bemerkbar, in denen Menschen mit Protokollen in Berührung kommen, mit denen sie noch keine bzw. keine ausreichenden Erfahrung besitzen. In Vorträgen und ganz besonders in der Lehre ist dies der Fall. Aber auch für Experten kann eine übersichtliche, von den Implementierungsdetails abstrahierende Darstellung von Nutzen sein.

Im Rahmen des HiSAP-Projektes (früheres ProtoVis-Projekt) soll deshalb ein Baukasten entwickelt werden, der die Darstellung von Protokollen vereinfacht. Sogar eine Entwicklung bis hin zu einem Protokollsimulator ist denkbar. Ein erster Schritt dazu wurde in einer früheren Studienarbeit von Peter W. Schurr [Schu97] gemacht, in der eine Visualisierung von Time-Sequence-Diagrammen entwickelt wurde. Leider ermöglichen diese nur eine High-Level Darstellung, in der im wesentlichen nur der Nachrichtenaustausch zwischen Kommunikationspartnern veranschaulicht werden kann. Das Problem ist somit, daß der interne Ablauf der einzelnen Kommunikationspartnern nicht sichtbar ist. Dieses Problem soll nun im Rahmen dieser Studienarbeit in Angriff genommen werden, indem für diese ein Visualisierungsbaukasten entwickelt wird. Da die gängige grafische Repräsentation endlicher Automaten intuitiv verständlich und zumindest im Informatikbereich weithin bekannt ist, wurde diese für die Darstellung gewählt.

Als Ergebnis dieser Studienarbeit soll somit ein Visualisierungsbaukasten entstehen, der die Darstellung aller Details eines Protokolls ermöglicht und verschiedene Abstraktionsebenen darstellen kann. Aus diesem Grund werden die von Peter W. Schurr erstellten Komponenten zur Visualisierung der Time-Sequence-Diagramme mit den in dieser Studienarbeit erstellten Komponenten zur Visualisierung von endlichen Automaten zu einem neuen Baukasten verschmolzen.

1 Teilaufgaben der Studienarbeit

Um einen Einblick in den Ablauf der Studienarbeit zu geben seien deren Teilaufgaben kurz vorgestellt. Die Studienarbeit gliederte sich in folgende sieben Teilaufgaben:

Zuerst war eine Einarbeitung in das Themengebiet und in die Programmiersprache Java ([Flan98], [CoHo97] und [CoHo98]) verlangt. Mit Hilfe der vorangegangenen Studienarbeiten [Schu97], [Kons98] und [Gort98], sollte ein Überblick von der Problemstellung gewonnen werden. Außerdem sollte eine Einarbeitung in den schon bestehenden Visualisierungsbaukasten stattfinden, um dessen Vor- und Nachteile auszuloten. Anschließend fand eine Einarbeitung in SDL ([Turn93] und [Hoge89]) und in Visualisierungstechniken mit dem Augenmerk auf endliche Automaten statt.

Als zweite Aufgabenteil war die Portierung des bestehenden Baukastens auf die aktuelle Version von Java vorgesehen, zu diesem Zeitpunkt war das die Version 1.1.6. Der unter der Version 1.0 erstellte alte Baukasten wäre zwar dank Abwärtskompatibilität unter der Version

1.1 selbstverständlich verwendbar gewesen, allerdings wäre ein einheitliches Erscheinungsbild durch das Vermischen mehrerer Java Versionen doch stark beeinträchtigt worden. Dies träfe vorallem auf das beim Übergang von Java 1.0 auf Java 1.1 erneuerte und verbesserte Event-Handling zu. Außerdem waren vor allem die zahlreichen als "Deprecatetd" gekennzeichneten Methoden zu ersetzen.

Um die Fähigkeiten des bestehenden Baukastens besser zu verstehen, sollte im dritten Aufgabenteil eine Beschreibung des Baukastens mit seinen Fähigkeiten und Defiziten angefertigt werden. Um dies leisten zu können, war es nötig, mehrere kleine Beispielanwendungen mit diesem zu erstellen und somit mit seinen Fähigkeiten zu experimentieren.

Als vierter Aufgabenteil war eine Spezifikation des zu implementierenden Baukastens zu schreiben und diesen auf Grund der Spezifikation zu implementieren und in den alten Baukasten so gut wie möglich zu integrieren.

Um den neuen Baukasten und dessen Fähigkeiten zu veranschaulichen, sollte im fünften Aufgabenteil die Visualisierung eines Protokolls im Rahmen einer Beispielanwendung erstellt werden. Als zu visualisierendes Protokoll war das "Stop and Wait" Protokoll vorgesehen.

Im sechsten Aufgabenteil sollte eine schriftliche Ausarbeitung erstellt werden und abschließend im siebten Aufgabenteil war eine multimediale Präsentation zu erstellen. Letzte sollte im Rahmen des Verteilten Systeme Kolloquiums vorgetragen werden, um die Ergebnisse der Studienarbeit einem interessierten Publikum vorzustellen.

2 Überblick

Im folgenden wird eine Übersicht über die verschiedenen Teile und die Struktur dieser Ausarbeitung gegeben.

Im Anschluß an diese Einleitung, wird in Kaptitel 2 zunächst der alte Baukasten, auf dem diese Studienarbeit basiert, in Bezug auf seine Fähigkeiten und Defizite vorgestellt. Daran schließt sich in Kapitel 3 eine Vorstellung der grundlegenden Konzepte des neuen Baukastens an.

Danach folgt eine detaillierte Beschreibung der Entwicklungen dieser Studienarbeit. Dafür werden in Kapitel 4 die neu erstellten Bestandteil im neuen Baukasten und deren Benutzerschnittstelle vorgestellt. In den Abschnitten dieses Kapitels werden die einzelnen Bausteine mit ihren Fähigkeiten und Schnittstellen beschrieben. Im nachfolgenden Kapitel 5 wird dargelegt wie aus den einzelnen Bausteinen des neuen Baukastens eine Anwendung entwickelt werden kann und ein rudimentäres Konzept für eine „Logikebene“ vorgestellt, die die Steuerung des Baukastens übernehmen soll. Für diesen Zweck wird eine Beispielanwendung für das „Stop and Wait“-Protokoll beschrieben. Um diese besser verständlich zu machen, wird eine Einführung in das „Stop and Wait“-Protokoll gegeben. In Kapitel 6 wird dann auf Implementierungsaspekte des neuen Baukastens eingegangen. Außerdem wird die Einbindung des alten Baukastens beschrieben.

Abschließend wird in Kapitel 7 auf möglich Verbesserungs- und konzeptionelle Erweiterungsvorschläge eingegangen, die sich aus den erworbenen Erfahrungen mit dem neuen Baukasten ergeben haben.

Nachdem nun einführend die Ziele der Studienarbeit und der Aufbau dieser Ausarbeitung vorgestellt wurden, kann nun zur detaillierten Beschreibung des Baukastens übergegangen werden. Dazu wird nachfolgend zunächst der alte Baukasten vorgestellt. Dies ist notwendig, da der neue Baukasten auf ihm basiert und er im wesentlichen unverändert als ein Bestandteil des neuen Baukastens weiter verwendet wird. Aus diesem Grund werden auch die vorgenommenen Änderungen erläutert.

Kapitel 2: Beschreibung des alten Baukastens

Im folgenden soll zunächst der Funktionsumfang des Visualisierungsbaukastens in seiner ursprünglichen Version, wie er in der Studienarbeit von Peter W. Schurr [Schu97] erstellt wurde, beschrieben werden. Auf Ergänzungen, die in anderen Arbeiten gemacht wurden, wird hier nicht näher eingegangen. Abschließend werden die an ihm vorgenommenen Änderungen erläutert, die für seine Weiterverwendung im neuen Baukasten notwendig waren.

1 Darstellung und Einbindung

Wie in der als Beispiel gedachten Anwendung, die in der Abbildung 2.1 zu sehen ist, werden im Baukasten von Peter W. Schurr Time-Sequence-Diagramme zur Veranschaulichung von Kommunikationsvorgängen zwischen Prozessen verwendet. Diese Diagramme sind aus der Lehre wohl bekannt und bei nicht zu komplexen Beispielen erfahrungsgemäß intuitiv verständlich und übersichtlich.

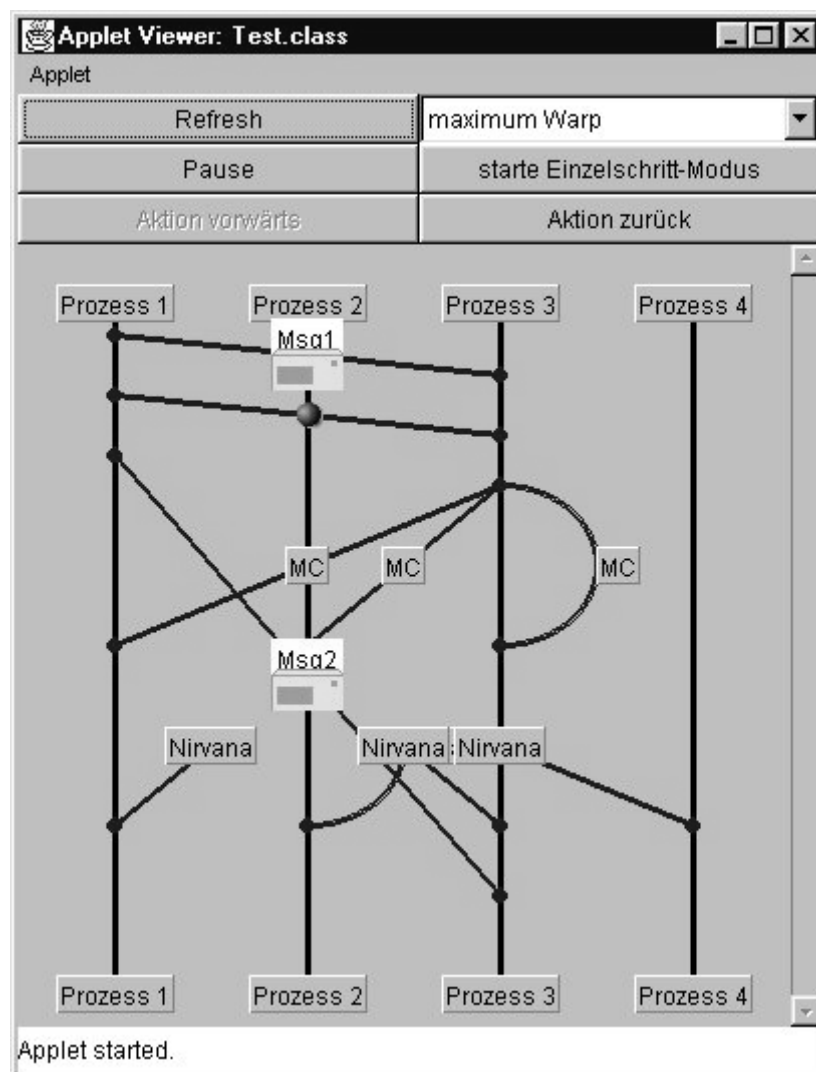


Abbildung 2.1

In Time-Sequence-Diagrammen werden die verschiedenen Prozeßinstanzen als senkrechte, von oben nach unten verlaufende, Linien dargestellt und stellen gleichzeitig die Zeitachse dar. Die Nachrichten, die zwischen den Prozessen ausgetauscht werden, sind tendentiell als horizontale Verbindungslinien zwischen den Prozeßlinien dargestellt. Die vertikale Differenz zwischen dem Punkt an dem die Nachricht abgesandt und an dem Punkt an dem sie empfangen wurde, gibt die zur Nachrichtenübermittlung verbrauchte Zeit an. Diese Zeitangabe entspricht keinem realen Zeitwert, sondern gibt nur einen qualitativen Wert an.

Das Einbinden des Baukastens in eine Anwendung geschieht durch Erzeugen eines "Assistent" Objekts. Dieses Objekt stellt ein Fenster (genauer ein Panel) zur Darstellung des Time-Sequenz-Diagramms mit dazugehörigen Steuerbuttons zur Verfügung, in das durch die Anwendung Prozesse und Nachrichten eingefügt werden können. Dies bedeutet, daß die Initialisierung und Steuerung des Baukastens von der Anwendung aus durch Methodenaufrufe erfolgt. Der "Assistent" hingegen übernimmt die Darstellung des Time-Sequenz-Diagramms und visualisiert den zeitlichen Ablauf. Beides läuft, von gewissen Einstellungsmöglichkeiten abgesehen, selbständig und parallel zur Anwendung ab. Beispielsweise wird die Darstellung automatisch auf die Größe des Fensters angepaßt. Steuerbar sind hingegen die Ablaufgeschwindigkeiten und die Abstände zwischen Vorgängen, wie z.B. dem Versenden zweier Nachrichten.

2 Kontroll-Interface

Das Kontroll-Interface ist zur Ablauf- und Geschwindigkeitssteuerung durch den Anwender nötig. Der Benutzer kann aus einer vorgegeben Menge an Ablaufgeschwindigkeiten wählen, die allerdings nicht näher spezifiziert sind, sondern in zueinander relativen Werten angegeben werden. Als Default ist eine mittlere Geschwindigkeit eingestellt.

Der Ablauf der Darstellung kann unterbrochen und anschließend wieder fortgesetzt werden. Außerdem wird ein Einzelschritt-Modus geboten, der es ermöglicht den Ablauf nicht fließend, sondern in vom Betrachter ausgelösten Schritten zu gestalten. Darauf basiert die wohl wichtigste Funktion, die Möglichkeit eines Undos. Der Benutzer wird somit in die Lage versetzt, den Ablauf nach Belieben vor und zurück zu verfolgen. Es sei allerdings noch angemerkt, daß kein echtes Undo stattfindet, sondern jeweils nur Teile der Grafik übermalt werden.

Sollte die Darstellung aus einem, in der Regel nicht vom Baukasten selbst verursachten Grund beschädigt sein, kann der Benutzer einen „Refresh“ der Anzeige durchführen.

3 Schnittstelle zum Baukasten

3.1 Prozesse

Prozesse können zu jeder Zeit von der Anwendung aus erzeugt und gelöscht werden. Es können beliebig viele Prozesse eingefügt werden. Prozesse werden standardmäßig durch zwei dreidimensional wirkende Rechtecke dargestellt, in denen jeweils der Prozeßname steht. Die beiden Rechtecke sind vertikal angeordnet und werden durch eine schwarze Linie verbunden, von der Nachrichten "ausgehen" und an der Nachrichten "ankommen" können (Abbildung 2.1: Prozess 1 bis Prozess 4). Anstatt der Rechtecke ist auch ein beliebiges Bild möglich, das von der Anwendung aus gesteuert, geladen werden muß und von ihr beim Erzeugen des Prozesses angegeben wird. Der zeitliche Ablauf wird dargestellt, indem sich das untere Linienende mitsamt dem Rechteck weiter nach unten bewegt. Wird das darzustellende Bild zu

groß für das Fenster, dann wird automatisch das Bild nach oben hinaus geschoben, um immer den aktuellen Vorgang sehen zu können. Will der Betrachter den hinausgeschobenen Bildteil sehen, dann kann er mit Hilfe von Scrollbars zu diesen Bereichen gelangen.

3.2 Nachrichten

Die Anwendung löst das Versenden von Nachrichten aus. Nachrichten werden durch blaue Linien zwischen Prozessen dargestellt. Die Linien werden zusätzlich mit einem Symbol versehen, das verdeutlichen soll, daß sie Nachrichten repräsentieren (Abbildung 2.1: Msg1). Nachrichten können von einem beliebigen Prozeß zu einem beliebigen Prozeß geschickt werden. Nachrichten an sich selbst sind also auch erlaubt.

Von einem Prozeß kann ein Broadcast-Nachricht an alle anderen Prozesse gesendet werden, außerdem ist dabei auch eine Nachricht an sich selbst möglich. Von einem Prozeß kann eine Nachricht an eine beliebige Menge von Prozessen gesendet werden (Multicast), auch hier kann eine Nachricht an sich selbst geschickt werden (Abbildung 2.1: MC). Somit existieren also schon "High-Level" Prozeduren, die nicht mehr durch die Anwendung realisiert werden müssen, die allerdings aufgrund der parallel ablaufenden Darstellung auch nicht von dieser implementiert werden könnten.

Es stehen für die oben angegebenen Arten des Nachrichtenversendens jeweils Methoden zur Verfügung, die diese Nachrichten vollständig, das bedeutet ohne Störung, darstellen. Weil aber gerade die Veranschaulichung von Fehlerfällen möglich sein soll, wurden analog zu diesen weitere Methoden implementiert, die nur die erste Hälfte der Nachrichtenübertragung der jeweiligen Nachricht(en) anzeigen. Die zweite Hälfte kann dann mit Hilfe zusätzlicher Methoden erzeugt werden. Für Multi- und Broadcastnachrichten kann der Verlauf für jeden Empfänger anders dargestellt bzw. gesteuert werden.

Jede Methode die eine Nachricht versendet und nur deren ersten Hälfte darstellt, gibt einen eindeutigen Namen für die Nachricht zurück, über welche die fortzusetzende Nachricht eindeutig identifiziert werden kann. Bei Multi- bzw. Broadcast-Nachrichten wird eine Menge von Namen zurückgegeben, damit jede einzelne Nachricht individuell behandelt werden kann.

Die zweite Hälfte einer, auf obige Art versandten Nachricht, kann individuell gehandhabt werden. Eine Nachricht kann unverändert zum Empfänger übertragen werden. Sie kann verloren gehen, was bedeutet das sie keinen zweiten Teil hat. Außerdem kann sie verändert oder verzögert werden (Abbildung 2.1: Msg2). Die letztgenannte Option ermöglicht das Überholen von nacheinander abgesandten Nachrichten und kann somit eine häufige Eigenschaft von Netzen mit Paketvermittlung darstellen, in denen die Reihenfolge der Pakete beim Übertragen vom Sender zum Empfänger nicht garantiert beibehalten wird.

Nachrichten sind standardmäßig durch ein Briefsymbol mit einem Text an der Oberseite markiert. Auch eine Darstellung ohne dieses Symbol ist möglich, in diesem Fall wird ein 3D-Rechteck mit dem Text im inneren, analog zu dem bei der Darstellung von Prozessen, verwendet. Es gibt also zwei Default-Darstellungen für Nachrichten, zwischen denen für jede einzelne Nachricht oder auch für Gruppen von Nachrichten umgeschaltet werden kann. Die Größe des Symbols bzw. des Rechtecks, wird bei den Default-Darstellungen automatisch an die Textgröße angepaßt.

Wie bei den Prozessen, kann auch hier ein von der Anwendung bestimmtes Bild verwendet werden (Abbildung 2.1: Blauer Ball). Soll auch ein Text angezeigt werden, dann muß dieser schon Teil des Bildes sein. Das Bild ist für jede Nachricht und auch für jeden der beiden Teile

der Nachrichtenübermittlung individuell einstellbar. Auf diese Art sind die Nachrichten bzw. Nachrichtengruppen leicht zu unterscheiden und Nachrichtenverfälschungen klar erkennbar.

Eine weitere Möglichkeit ist es, sogenannte "Nirvana"-Nachrichten auftauchen zu lassen. Dies sind Nachrichten, die aus dem Nichts kommen. Sie könnten z.B. von einem Eindringling in die Kommunikation kommen oder von einem, in einem Protokoll eventuell möglichen, fremden Sender stammen. Diese Nachrichten haben also keinen ersten Teil. "Nirvana"-Nachrichten können auch an Gruppen von Prozessen (Multicast) bzw. an alle Prozesse (Broadcast) gerichtet sein (Abbildung 2.1: Nirvana). Die Darstellung von "Nirvana"-Nachrichten kann auf die gleiche Art und Weise wie die anderer Nachrichten eingestellt werden.

Nachrichten können vom Benutzer mit der Maus angeklickt werden und die den Baukasten benutzende Anwendung kann diese Information dann durch Nachfragen erhalten. Als Antwort wird durch den Baukasten geliefert, ob und, wenn ja, welche Nachricht seit der letzten Nachfrage angeklickt wurde. Mit Hilfe der mitgeteilten ID der Nachricht, kann die Anwendung dann Änderungen im Ablauf oder in der Art der Darstellung vornehmen.

3.3 Darstellungssteuerung

Wie schon in den vorangegangenen Abschnitten deutlich wurde, kann die Darstellung auf vielfältige Weise von der Anwendung aus beeinflusst werden. An dieser Stelle wird nun, im Gegensatz zu den zuvor erläuterten Einflußmöglichkeiten, die im wesentlichen von konkreten Details des Ablaufs der Anzeige abstrahierten, auf die zeitliche Steuerung durch die Anwendung näher eingegangen. Aus ihr erfolgt indirekt auch eine veränderte räumliche Aufteilung.

Durch das Einfügen einer Pause, kann z.B. ein größerer Abstand zwischen dem Versenden zweier Nachrichten eingeschoben werden. Dies macht sich nicht nur durch eine zeitliche Verzögerung bemerkbar, sondern wird auch in der Grafik sichtbar, weil die nach unten laufende Prozeßdarstellung weiter fortschreitet. Weiterhin kann durch sie eine Synchronisation unter Prozessen vorgenommen werden, deren Anzeige gleichzeitig beginnen bzw. enden sollen. Dies ist nötig, da die Anwendung Methoden nun mal nicht exakt gleichzeitig aufrufen kann.

Um dies zu vereinheitlichen, wurden Methoden vorgesehen, die vor und nach dem Erzeugen bzw. dem Beenden von Prozessen ebenfalls Pausen einfügen können. Weiterhin gibt es Methoden zum Einfügen einer Pause vor dem Versenden einer Nachricht bzw. nach dem Empfangen einer Nachricht.

Die Zeit wird im Baukasten in Zeiteinheiten gemessen. Die Defaulteinstellung etwa, für die soeben beschriebenen Pausen, beträgt 5 Zeiteinheiten, was dem Fortschreiten der Anzeige um 5 Pixel entspricht. Die Zeitdauer der Nachrichtenübermittlung kann eingestellt und auch abgefragt werden. Ebenfalls ist es möglich die aktuelle Zeit im Baukasten abzufragen.

Die Ablaufgeschwindigkeit kann, analog, wie bei den Steuerbuttons beschrieben, auch von der Anwendung aus gesetzt werden.

Abschließend seien noch zwei Möglichkeiten vorgestellt, die Darstellung direkt zu beeinflussen. Damit die Darstellung nicht springt, wenn ein neuer Prozeß eingefügt wird, kann die minimale Prozeßzahl angegeben werden. Da nun die Raumeinteilung schon vorab entsprechend vorgenommen werden kann, muß das Bild beim Einfügen eines Prozesses nicht

neu aufgeteilt werden. Außerdem ist es möglich, den Baukasten durch einen Reset in seinen Ausgangszustand zurück zu versetzen, damit der Aufbau der Anzeige von neuem beginnen kann und auch ein veränderter Ablauf möglich wird.

4 Zielsetzung des Baukastens

Wie man aus dem Umfang der obigen Abschnitte klar erkennen kann, wurde auf die Nachrichtenübermittlung das Hauptaugenmerk gelegt. Die Darstellung der Prozesse wurde dabei zu deren Gunsten zurückgestellt. Dadurch wird, wie schon erwähnt wurde, eine sehr übersichtliche und intuitiv verständliche Darstellung möglich gemacht. Nicht unbedingt nötige Details werden vor dem Betrachter verborgen, ohne das die Verständlichkeit darunter leidet. Die Funktionalität des Baukastens ist klar eingegrenzt und er kann leicht in einer Anwendung verwendet werden.

5 Defizite des Baukastens

Die Stärke des Baukastens ist gleichzeitig auch seine Hauptschwäche. Da die Prozesse nicht näher beschrieben werden, bekommt der Betrachter keine Informationen darüber, was mit den empfangenen Nachrichten passiert und was das Absenden von Nachrichten auslöst, bzw. welche Nachrichten wann verschickt werden. Außerdem werden keine Kommunikationskanäle modelliert, was ebenfalls der Verständlichkeit bei komplexeren Beispielen abträglich ist oder manche Probleme nicht vollständig darstellen läßt.

Weiterhin ist für die Erstellung von Anwendungen problematisch, daß im Baukasten keinerlei Logik integriert ist und dieser Teil komplett auf die Anwendung abgeschoben wird. Eine noch vorzunehmende Erweiterung, des bislang auf die grafische Repräsentation konzentrierten Baukastens um eine „Logikschicht“, wird diesen Umstand voraussichtlich zu beseitigen wissen. Dazu aber mehr im nächsten Kapitel.

Leider ist die Unterstützung durch den Baukasten zum Setzen einer beliebigen Anzeige von der Anwendung aus nicht besonders ausgeprägt, so daß sich dies negativ auf seine Verwendbarkeit für komplexere oder gar interaktive Anwendungen auswirkt. Der Baukasten selbst ist nur in geringem Maße interaktiv steuerbar.

Da der ganze Baukasten und die Programmierschnittstelle sehr „prozedural“ ausgefallen ist, ist eine leichte Erweiterbarkeit, wie schon in anderen Beschreibungen zu Peter W. Schurr's Baukasten angemerkt wurde, wohl nicht ohne weiteres zu erwarten. Eine Überarbeitung des Baukastens, wobei ein rein objektorientierter Entwurf die Zielsetzung ist, wird sich deshalb wohl nicht vermeiden lassen.

6 Vorgenommene Änderungen

Der zuvor beschriebene Baukasten von Peter W. Schurr wurde praktisch unverändert im neuen Baukasten übernommen. Änderungen waren dafür nur in Details nötig und werden nachfolgend erläutert.

Der alte Baukasten ist in Java 1.0 geschrieben worden, um mit dem neuen Baukasten konform zu sein, wurde er auf Java 1.1 portiert. Dabei waren die als „deprecated“ gekennzeichneten Methoden durch ihre neuen Pendanten zu ersetzen. Dies kam im wesentlichen einem Umbenennen gleich. Wichtiger war allerdings die Umstellung auf das neue Event-Handling von Java 1.1, damit eine einheitliche Schnittstelle erreicht werden konnte. Dazu waren, auf

Grund der neuen Konzeption im Event-Handling, kleinere Änderungen in der Struktur der Implementierung notwendig. Es wurde allerdings darauf geachtet, daß möglichst wenige Veränderungen vorgenommen werden mußten, und vor allem, daß keine Veränderungen an der grundsätzlichen Art der Implementierung vorgenommen wurden. Dies bedeutet, daß im wesentlichen nur Methoden entfernt und deren Inhalt auf andere Methoden verteilt wurde.

Am Funktionsumfang selbst, wurde bis auf eine Ausnahme, keine Änderung vorgenommen. Die Ausnahme ist die Möglichkeit Bilder zu laden, die wie zuvor beschrieben wurde, in der Darstellung verwendet werden können. Diese Option mußte entfernt werden, da in ihrer Implementierung die zwingende Voraussetzung getroffen worden war, daß der Baukasten innerhalb eines Applets verwendet wird. Innerhalb des neuen Baukastens stehen somit nur die Standarddarstellungen zu Verfügung.

Der neue Baukasten soll gleichermaßen in Applets und Applikationen einsetzbar sein. Da allerdings der alte Baukasten, wie gerade festgestellt wurde, lediglich für die direkte Verwendung in Applets gedacht war, mußte noch ein Fenster implementiert werden, in dem dessen Darstellung erfolgen kann, unabhängig davon ob er in einem Applet oder einer Applikation verwendet wird. Dieses Fenster bietet neben der Darstellung des alten Baukastens lediglich die Funktion, ein durch den Benutzer über eine Menüleiste ausgelöstes „Beenden-Event“ zu generieren. Auf dieses Event kann dann beispielsweise von einer Anwendung mit deren Beendung reagiert werden. Das Fenster wird, in Analogie zu einer in Kapitel 4 noch vorzustellenden neu erstellten Komponente im neuen Baukasten, als Sammelfenster bezeichnet, auch wenn diese Bezeichnung in der momentanen Form des alten Baukastens noch nicht viel Sinn macht.

Bei der Implementierung des Sammelfensters, wurde ein Fehler im alten Baukasten entdeckt, der bei der direkten Verwendung in Applets nicht aufzutreten scheint. Die Scrollbar an der Darstellung des Time-Sequence-Diagramm wird nicht aktiviert, wenn diese größer als die zu Verfügung stehende Fläche für die Darstellung wird. Damit sind Visualisierungen nur zu einem Teil verfolgbar.

Nachdem die ursprüngliche Form des Baukastens und somit auch ein wesentlicher Bestandteil des neuen Baukastens beschrieben worden ist, kann sich nun gänzlich auf den in dieser Studienarbeit erstellten neuen Baukasten konzentriert werden. Dazu ist es allerdings zunächst unerlässlich auf grundlegende Konzepte einzugehen, die dem neuen Baukasten zugrunde liegen. Ohne deren Verständnis ist der Einsatz oder gar eine Weiterentwicklung des neuen Baukastens nicht denkbar.

Kapitel 3:

Grundlegende Konzepte des neuen Baukastens

Bevor in den folgenden Kapiteln detailliert auf die neu erstellten Komponenten des neuen Baukastens und deren Implementierung eingegangen werden kann, ist es für ein besseres Verständnis unerlässlich auf grundlegende Strukturen hinzuweisen und konzeptionelle Sachverhalte vorzustellen. Dazu ist allerdings zunächst eine Begriffsklärung unerlässlich.

1 Terminologie

Um im folgenden keine Mißverständnisse aufkommen zu lassen, von welchem Baukasten bzw. welchen Baukastenteilen die Rede ist, seien kurz die verwendeten Begriffe definiert.

Der von Peter W. Schurr implementierte Baukasten in seiner ursprünglichen Form, wie er ausführlich im letzten Kapitel vorgestellt worden ist, wird als alter Baukasten bezeichnet. Der in dieser Studienarbeit erstellte Baukasten wird als neuer Baukasten oder auch kurz als Baukasten bezeichnet. Der neue Baukasten beinhaltet sowohl den alten Baukasten, der die Time-Sequence-Diagramme visualisiert und deshalb mit TSD-Baukasten benannt wird, als auch die neu erstellten Teile, die für die Darstellung endlicher Automaten (Finite-State-Machines) u.ä. verantwortlich sind und zusammenfassend als FSM-Baukasten betitelt werden.

2 Strukturen

Der alte Baukasten bestand aus Benutzersicht im Sinne der objektorientierten Programmierung nur aus einer Objektart, welche die Darstellung von Time-Sequence-Diagrammen übernimmt und in denen, im Rahmen einer im wesentlichen vorgegebenen Ablaufstruktur, nur Aktionen angestoßen werden können.

Der neue Baukasten besteht hingegen aus einer Sammlung von verschiedenartigen Objekten, die sich aus den Bestandteilen des TSD- und des FSM-Baukastens zusammensetzen, aus denen sich der Benutzer die für seine Zwecke nötigen Grafikbausteine herausnehmen kann. Somit wird hierbei eine strikte Trennung von grafischer Darstellung und der jeweiligen Steuerung durch die Anwendung vorgenommen. Dadurch wird ein Schichtenmodell realisiert.

Das Auslösen von Aktionen erfolgt eventbasiert und wird durch Events, sofern dies nötig und sinnvoll ist, nach Beendigung der Aktionen bestätigt. Allgemein findet die Kommunikation zwischen den Schichten eventbasiert statt. Durch die Bestätigungsevents kann eine Synchronisation unter den Schichten erreicht werden.

Das zuvor vorgestellte Schichtenmodell soll in späteren Arbeiten noch erweitert werden, um das einfachere Erstellen einer Anwendung besser zu unterstützen (Abbildung 3.1). Dabei soll nach dem MVC-Pardigma, wie es etwa in [GHJV95] beschrieben ist, vorgegangen werden. MVC steht für Model-View-Control und bedeutet, daß eine Trennung zwischen der grafischen Darstellung, deren Steuerung und dem darzustellenden „Modell“ (hier ist dies z.B. das Protokoll) durchgeführt wird.

Um dies auf die hier gegebene Problemstellung umzusetzen, wird eine Logikebene (Control) eingeführt, welche die Steuerung der in dieser Studienarbeit erstellten Grafikebene (View) übernimmt. Außerdem stellt die Logikebene für die Anwendung bzw. die Anwendungsebene

(Model) Funktionen bereit, mit denen ein Protokoll auf abstrakte Weise angegeben werden kann. Beispielsweise könnte eine auf Tabellen basierende Eingabeform gewählt werden (vgl. Kapitel 5), aus der die Logikebene dann die Ablaufsteuerung des Protokolls ermittelt und mit deren Hilfe dann die Grafikebene angesteuert wird. Auf diese Weise wird die Anwendung von jedwedem Problem mit der grafischen Darstellung entbunden, sie beinhaltet im wesentlichen nur noch die Spezifikation des jeweiligen Protokolls.

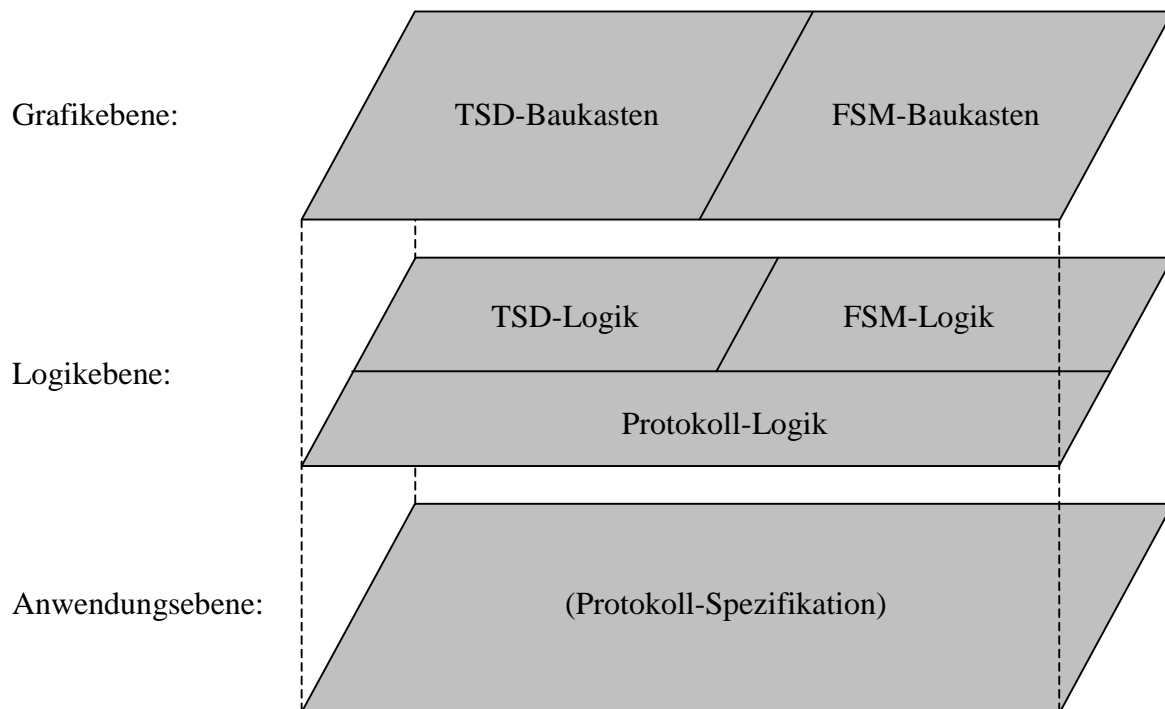


Abbildung 3.1

Wie sich aus dem vorigen erahnen läßt, wird eine weitere Unterteilung der Logikschicht vonnöten sein, um deren Komplexität zu handhaben. Eine Trennung in Teile, welche die direkte Steuerung der Komponenten der Grafikebene (TSD- und FSM-Logik) übernehmen und einen Teil der für die Übersetzung, einer von der Anwendungsebene angegebenen Protokollspezifikation, in eine Ablaufsteuerung verantwortlich ist, würde beispielsweise nahe liegen.

Die Logikschicht ist, wie schon gesagt wurde, nicht Bestandteil dieser Studienarbeit und wird deshalb noch Gegenstand weiterer Arbeiten sein. Da diese Schicht also noch nicht implementiert ist, mußte in der später vorgestellten Beispielanwendung die Funktionalität der Logikschicht mit derer der Anwendungsschicht gemischt werden, eine allgemeine Verwendbarkeit ist somit nicht gegeben.

Das zuvor beschriebene Schichtenmodell stellt die Struktur der Implementierung des Baukastens dar, sagt aber nichts über eine Hierarchie zwischen den dargestellten Komponenten der Visualisierung aus. Da eine solche aber existiert und von grundlegender Bedeutung ist, sei diese einführend erläutert.

Die vom TSD-Baukasten dargestellten Protokollteile, stellen die oberste Ebene dar. Es werden die Kommunikationspartner (Prozesse) und die dazwischen ausgetauschten Nachrichten visualisiert. Die Kommunikationskanäle zwischen Kommunikationspartnern werden je nach Anschauungsweise nur implizit bzw. gar nicht dargestellt. Der FSM-Baukasten visualisiert die Internas der Prozesse und ist somit aus Sicht der Visualisierung des Protokolls unterhalb des TSD-Baukastens anzuordnen.

Bezüglich der Implementation sind beide Baukastenteile hingegen gleichberechtigt, aber streng getrennt. Beide liegen, wie aus dem vorher gesagten schon deutlich wurde, in der Grafikebene des Schichtenmodells. Ob auch eine Trennung deren Steuerung auf Logikebene sinnvoll ist, wie sie zuvor vorgeschlagen wurde und sie in Abbildung 3.1 dargestellt ist, wird sich erst bei der Implementierung der Logikebene zeigen. In der Beispielanwendung wurde eine solche Trennung vorgenommen.

Da nun die grundlegenden Konzepte des Baukastens und seine alten Komponenten ausführlich vorgestellt worden sind, kann nun auf dessen neu erstellten Bestandteile näher eingegangen werden. Der neu erstellte Teil, also der FSM-Baukasten, wird im nächsten Abschnitt zunächst aus der Sicht des Baukastenbenutzers beschrieben, wie dies auf analoge Weise auch schon im zweiten Kapitel mit dem alten Baukasten, also dem TSD-Baukasten, geschehen ist. Dazu werden der Reihe nach die verschiedenen Komponenten und deren Funktionsumfang vorgestellt.

Kapitel 4:

Beschreibung der neu erstellten Baukastenkomponenten

Da nun der alte Baukasten, aus dem in nahezu unveränderter Form der TSD-Baukasten hervorgegangen ist, und die dem neuen Baukasten zugrunde liegenden Konzepte beschrieben worden sind, soll nun mit dem FSM-Baukasten die eigentliche Neuentwicklung dieser Studienarbeit vorgestellt werden. Dabei wird unter anderem die geänderte Zielsetzung im Vergleich zum alten Baukasten ins Auge fallen. Beim neuen Baukasten wird vor allem Wert auf eine leichte Änderbarkeit und Erweiterbarkeit gelegt, wo hingegen beim alten Baukasten hauptsächlich auf eine komfortable Schnittstelle für den Anwendungsentwickler geachtet wurde. Der Grund für diesen Wandel ist, daß der neue Baukasten in seiner jetzigen Form der erste Teil einer Entwicklung sein soll, in der seine konkrete Ausprägung noch nicht in allen Details erkennbar ist und deshalb noch Veränderungen zu erwarten sind.

Bevor mit der Beschreibung des FSM-Baukastens begonnen werden kann, sei nochmals unterstrichen, daß der FSM-Baukasten nur für die grafische Darstellung zuständig ist, nicht aber für Logik oder Steuerung. Durch die gewählten Bezeichnungen für die Komponenten könnte eine Verwechslung mit dem Modell der endlichen Automaten nahe liegen, hier ist aber lediglich die reine Darstellung gemeint. Auch sei darauf hingewiesen, daß einige der Komponenten des FSM-Baukasten Funktionen bieten, die für die Verwendung auf Logikebene mißbraucht werden können, wie etwa das Uminterpretieren von Statusinformationen. Es sei nochmals daran erinnert, daß dies dem in Kapitel 3 vorgestellten Schichtenmodell zuwiderläuft und deshalb vermieden werden sollte.

1 Grundstruktur

Der FSM-Baukasten besteht im Sinne der objektorientierten Programmierung aus einer Sammlung von Objekten zur Darstellung der Internas der Prozesse, die in einem Protokoll miteinander kommunizieren. Als Basis wird die übliche Darstellung endlicher Automaten verwendet. Diese besteht einerseits aus Knoten, die Zustände repräsentieren, und Kanten, die Übergänge darstellen.

Um anzugeben, wann ein Zustandsübergang erlaubt ist, werden Prädikate verwendet. Befindet sich der Automat in einem Zustand, von dem ein bestimmter Übergang ausgeht und ist das, diesem Übergang zugeordnete Prädikat erfüllt, dann kann der damit erlaubte Zustandsübergang durchgeführt werden. Prädikate werden durch Beschriftung der jeweiligen Kante visualisiert, die den jeweiligen Übergang repräsentiert.

Sowohl für Zustände und Übergänge wie auch für Prädikate existieren jeweils Objekte, die diese darstellen können. Um aber dem Benutzer einen kompletten Überblick über den Ablauf des darzustellenden Protokolls zu geben, wurden weitere Visualisierungselemente der Darstellung des endlichen Automaten zur Seite gestellt, die vor allem ergänzende Informationen über die Durchführbarkeit von Zustandsübergängen liefern und weitere Aktionen zusätzlich zum Zustandsübergang darstellen können. Da diese Visualisierungselemente allerdings nicht direkt zur Darstellung des Automaten gehören, werden sie von dieser getrennt dargestellt.

Der Hauptteil des Baukastens besteht somit aus einer Kollektion von Fensterteilen (jeweils ein Objekt), welche die Darstellung für den Benutzer übernehmen. Das wichtigste davon ist die Automatarstellung, welche die Visualisierung des endlichen Automaten koordiniert und somit die Hauptschnittstelle für die Nutzung des Baukastens bildet.

Da der FSM-Baukasten nach dem Schichtenmodell ausschließlich die grafische Darstellung übernimmt, also die Grafikebene darstellt, bleibt die Logikebene bzw. eine Anwendung, die den Baukasten direkt benützen möchte oder im momentanen Zustand des gesamten Baukastens auch muß, im wesentlichen noch dafür verantwortlich, die gewünschte Darstellung aus den Baukastenkomponenten aufzubauen und die Visualisierung zu steuern. Dabei werden durch den TSD-Baukasten keine Einschränkungen hinsichtlich dem Sinngehaltes des von ihm dargestellten gemacht, da nach dem Schichtenmodell auf der Grafikebene keinerlei Logik vorhanden sein darf.

2 Zustände

Zustände sind Elemente, die ihre grafische Darstellung selbst in einen beliebigen Grafikbereich zeichnen können. Wie ein Zustand dargestellt wird, hängt allerdings von der jeweiligen Zustandsklasse ab, da der TSD-Baukasten objektorientiert entwickelt wurde. In analoger Weise trifft dies auch auf alle anderen Komponenten des TSD-Baukastens zu. Bei den Zuständen gibt es, im Gegensatz beispielsweise zu den in Abschnitt 3 dieses Kapitels vorgestellten Prädikaten, bislang nur eine Standardklasse.

Ein Zustand wird als farbig ausgefüllten Kreis dargestellt, wie in Abbildung 4.1 zu sehen ist. Jeder Zustand besitzt eine individuelle Position. Sie gibt an, an welche Stelle der Zustand gezeichnet werden soll. Dabei entspricht die Position dem Mittelpunkt des Kreises. Selbstverständlich sollten die Positionen der verschiedenen Zustände in einem gemeinsamen Koordinatensystem liegen, da sie ja gemeinsam dargestellt werden sollen.

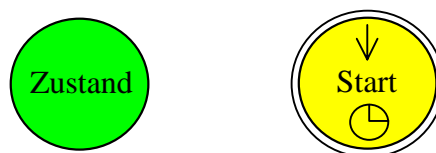


Abbildung 4.1

Alle Zustände sollten einen Namen haben, der ihre Bedeutung widerspiegelt. Dieser Name wird von der Anwendung vergeben. Der Name wird im Inneren des Kreises angezeigt. Die Größe des Kreises wächst mit der Länge des Zustandsnamens. Ist der Name eines Zustands sehr kurz kann es sein, daß seine Darstellung zu klein wird, um optisch ansprechend zu wirken. Deshalb gibt es eine untere Grenze für die Größe eines Zustands, die auch eingehalten wird, wenn kein Name angegeben wurde. Aus dem letztgenannten Fall läßt sich somit erkennen, daß die Darstellung des Namens auch abgeschaltet werden kann.

Zustände haben selbst einen Zustand, sie können nämlich aktiv oder inaktiv sein. Der Zustand, in dem sich der Automat gerade befindet, ist aktiv. Dies darf natürlich nur ein Zustand zu jedwedem Zeitpunkt sein. Bei einem Zustandsübergang muß dies beachtet werden, aber dazu später mehr bei den Prädikatobjekten.

Der aktive Zustand wird mit grünem Hintergrund dargestellt, alle anderen Zustände hingegen mit gelbem Hintergrund (Abbildung 4.1). Die Information, ob ein Zustand aktiv ist, kann von diesem erfragt werden. Dies ist ein Beispiel für die in der Einleitung zu diesem Kapitel erwähnten Möglichkeiten, in denen Statusinformationen der Grafik für die Steuerung des Automaten verwendet werden könnten. Wie dort schon erwähnt wurde, ist dies gemäß dem Schichtenmodell unzulässig.

Zustände können außerdem als Start- oder Endzustände spezifiziert werden, dies hat eine veränderte Darstellung zur Folge. Die Darstellungsform ist an die in endlichen Automaten übliche angelehnt. Startzustände haben oberhalb ihres Namens, aber innerhalb des Kreises, einen nach unten gerichteten Pfeil und sollten außerdem zu Beginn der Visualisierung als aktiv dargestellt werden. Endzustände hingegen werden mit einem doppelten Kreis gezeichnet. In der Abbildung 4.1 auf der rechten Seite ist zu sehen, daß ein Zustand sowohl Startzustand, als auch Endzustand sein kann. Abgesehen von der Art der Darstellung, verhält sich ein Start- bzw. ein Endzustand genau gleich wie jeder normaler Zustand.

Sind mit einem Zustand ein oder auch mehrere Timer verknüpft (siehe Abschnitt 7. Timer), d.h. beispielsweise ein Übergang, der von diesem Zustand ausgeht, wartet auf das Beenden des Timers, dann wird je nach Anwendung dies durch ein kleines Uhrensymbol am Rand der Zustandsrepräsentation angezeigt. Festzustellen, ob zu einem Zustand ein Timer zugeordnet ist, obliegt der Anwendung. Dies bedeutet somit aber auch, daß die zuvor gegebene Definition nicht bindend ist und je nach Bedarf angepaßt werden kann.

Jeder Zustand sollte von der Anwendung einen Infotext erhalten, der die Funktion des Zustands im zu visualisierenden Protokoll beschreibt. Jede dargestellt Komponente im FSM-Baukastens bietet diese Möglichkeit. Diese Informationen können auf Wunsch dem Betrachter der Visualisierung angezeigt werden (siehe Abschnitt 5 über die Automatendarstellung). Der Infotext kann beliebig lang sein und muß nicht besonders formatiert zu sein. Es können allerdings auch die in einigen Programmiersprachen, unter anderem in Java, üblichen Steuerzeichen in Strings verwendet werden. Das Einfügen von beispielsweise `\n`, spezifiziert einen Zeilenumbruch.

3 Übergänge

Übergänge sind ebenfalls Objekte. Sie modellieren die gerichtete Kanten. Übergänge haben somit einen Anfang und ein Ende, dies ist jeweils ein Zustand. Sie werden deshalb als Pfeile dargestellt. Übergänge können sich ebenso wie Zustände selber zeichnen.

Als Default werden sie als einfache Linie mit einer Pfeilspitze in schwarzer Farbe dargestellt (Abbildung 4.2). Die Darstellung des Übergangs beginnt und endet an der Darstellung der zu ihm gehörenden Zustände. Als Alternative soll in folgenden Arbeiten anstatt der Darstellung mit der Linie eine Darstellung mit einem Polygon angeboten werden, das automatisch so angeordnet wird, daß eine optisch ansprechende und übersichtliche Darstellung auch bei größeren Automaten ermöglicht wird.

Wenn der Fall auftritt, daß zwischen zwei Zuständen in beide Richtungen Übergänge vorhanden sind, werden diese automatisch so angeordnet, daß keine Überlappungen stattfinden. Es ist allerdings zu empfehlen, daß für eine Richtung auch nur ein Übergang verwendet wird, was durch zusammenfassen erreichbar ist.

Ein weiterer Sonderfall ist, daß ein Übergang im selben Zustand endet, wie der von dem er ausgegangen ist. Auch dies wird automatisch erkannt und ist für den Einsatz des Baukastens

transparent. Ein solcher Übergang wird als Schlinge mit einer Pfeilspitze auf einer Seite dargestellt. Da im Gegensatz zu normalen Übergängen keine bestimmte Positionierung für die Darstellung am Zustand nahe liegt, wird diese praktisch willkürlich festgelegt. Dabei wird sich an den vier Achsen eines im Mittelpunkt des Zustand liegenden Koordinatensystems orientiert.

Wird ein Zustandsübergang ausgelöst, dann beginnt eine kleine Animation abzulaufen. Vom Ausgangszustand des Übergangs an, beginnt die Darstellung des Übergangs eingefärbt zu werden (Abbildung 4.2). Als Farbe wird die gleiche wie bei einem aktiven Zustand verwendet, mit welcher der Ausgangszustand eingefärbt sein sollte, da nur von einem aktiven Zustand ein Zustandsübergang ausgehen darf. Dies wird allerdings vom FSM-Baukasten nicht geprüft.

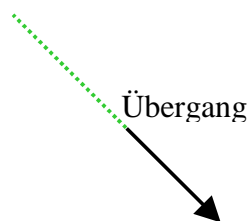


Abbildung 4.2

Die Einfärbung läuft solange den Übergang entlang, bis dieser komplett eingefärbt ist, dann wird der Ausgangszustand als inaktiv und der Endzustand als aktiv dargestellt. Kurz darauf wird der Übergang wieder Schwarz dargestellt. Während eines Zustandsübergangs sollten keine sonstigen Aktionen in der Darstellung des Automaten stattfinden, damit die Übersichtlichkeit gewahrt bleibt. Um dies der Steuerung des Baukastens zu ermöglichen, wird ein Event nach der Beendigung des Zustandsübergangs ausgelöst.

Wie bei den Zuständen kann durch einfaches Erzeugen einer neuen Übergangsklasse eine andere bzw. erweiterte Darstellung implementiert werden, ohne das weitere Änderungen am Baukasten notwendig werden. Dies erleichtert beispielsweise die zuvor erwähnte Implementierung der Übergangsdarstellung als Polygon.

Allen Übergängen ist gemein, daß ihnen ein Prädikat zugeordnet ist, das eine Bedingung für den Zustandsübergang visualisiert. Das Prädikat kann ein logischer Ausdruck sein oder aber auch einen mehr informellen Charakter haben. Zusätzlich kann wie bei den Zuständen, einem Übergang ein Infotext mitgegeben werden, der nähere Beschreibungen für den Benutzer enthält.

4 Prädikate

Prädikate sind wie Zustände und Übergänge auch Objekte. Jedes Prädikat ist einem Übergang zugeordnet und wird ebenfalls an diesem angezeigt. Ihre Darstellung besteht aus einem Text der einen logischen Ausdruck beschreibt. Die atomaren Teile des logischen Ausdrucks sollten entweder in grüner Farbe geschrieben werden, wenn sie wahr sind, oder in roter Farbe, wenn sie falsch sind. Um den Wahrheitswert eines zusammengesetzten Ausdrucks zu kennzeichnen, sollte das zu ihm gehörende Klammernpaar farbig markiert werden (Abbildung 4.3). In einer

alternativen Darstellung könnte auch der Hintergrund der Textabschnitte eingefärbt werden, wobei der Text selber Schwarz bleibt. Um dies zu erreichen steht ein Defaultprädikat und ein Erweitertesprädikat zur Verfügung.

Im Defaultprädikat wird die Einfärbung vom Benutzer des FSM-Baukastens vorgenommen. Die Anwendung wird lediglich dahingehend unterstützt, daß sie den String für das Prädikat mit Hilfe von Teilblöcken in verschiedenen, nicht vorbestimmten Farben setzen kann. Dazu steht ein Objekt zur Verfügung, das einen farbigen String modelliert. Die Einfärbung des Strings kann sich auf den Text oder auf den Texthintergrund beziehen. Dies wird für den ganzen String einheitlich gehandhabt und kann umgeschaltet werden. Somit wird eine weitgehende Freiheit für die Darstellungsart gegeben.

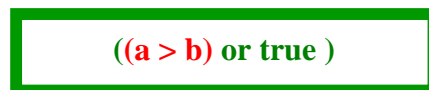


Abbildung 4.3

Der Wahrheitswert des Prädikats, wird durch eine farbige Umrahmung dargestellt. Die Farben sind, analog zu denen bei der Darstellung der atomaren Ausdrücke empfohlenen, rot für wahr und grün für falsch. Dies kann im Gegensatz zur Einfärbung des Strings nicht verändert werden. Wird ein Prädikatcheck ausgelöst, dann beginnt die Umrahmung in rot und grün zu blinken. Währenddessen, wird nacheinander eine Folge von farbigen Strings, die vom Baukastenbenutzer angegeben werden müssen, dargestellt. Jeder dieser Strings wird für eine kurze Zeitspanne angezeigt, bevor der nächste an die Reihe kommt. Ist der Prädikatcheck beendet, bleibt der letzte String erhalten, der Rahmen hört auf zu blinken und nimmt die, dem für das Prädikat neu zu vergebenen Wahrheitswerts, entsprechende Farbe. Ein Event wird ausgelöst und die Animation des Zustandsübergangs sollte daraufhin begonnen werden.

Der neue Wahrheitswert des Prädikats wird vom Baukastenbenutzer festgelegt, d.h. es werden keine logischen Auswertungen im Prädikat selber vorgenommen. Soll der Ablauf eines Prädikatenchecks animiert dargestellt werden, dann muß, wie zuvor beschrieben wurde, dieses durch schrittweises Verändern des Strings, durch den Baukastenbenutzer selbst, vorgenommen werden, das Defaultprädikat liefert dazu keinerlei weitergehende Unterstützung.

Die zuvor vorgestellte Vorgehensweise läßt dem Baukastenbenutzer große Freiheit bei der Darstellung des Prädikats und des Prädikatenchecks, hat allerdings auch den Nachteil, das diesem die ganze Arbeit überlassen wird. Um dieses Problem zu umgehen, wird ein Erweitertesprädikat im Baukasten bereit gestellt, das etwas Logik enthält und die Animation des Prädikatenchecks übernimmt.

Der Baukastenbenutzer baut einen logischen Ausdruck mittels einer festgelegten Vorgehensweise aus Operatorobjekten auf, die logische Operatoren modellieren, und übergibt diesen dem erweiterten Prädikat. Zuerst werden die atomaren Teile des Gesamtausdruckes mit booleschem Wert und dem darzustellenden String erzeugt. Dann werden jeweils zwei atomare Ausdrücke bzw. schon gebildete Ausdrücke zu einem neuen Ausdruck zusammengefügt. Dabei muß jeweils eine logischer Operator verwendet werden. Zur Verfügung stehen als Operatoren "AND", "OR", "XOR" und "NOT". Durch diese Vorgehensweise können alle

möglichen Prädikate erzeugt werden, es können aber keine inkorrekten Prädikate eingegeben werden. Die Klammerung der Teilausdrücke wird dabei durch die Reihenfolge der Erzeugungsschritte des Gesamtausdrucks festgelegt.

Ist der Gesamtausdruck erst einmal erzeugt und dem Erweitertenprädikat übergeben worden, dann wird die Auswertung der Teilprädikate und des Gesamtprädikats automatisch vorgenommen, auch der anzuzeigende String mit allen Einfärbungen, Operatoren und Klammern wird automatisch erstellt. Dabei wird sich an die bei den Defaultprädikaten empfohlene Darstellungsweise gehalten. Die Anwendung kann nur noch den String und den Wahrheitswert von atomaren Ausdrücken beeinflussen. Dies genügt, um die Prädikate voll steuerbar zu machen. Ansonsten besteht nur noch die Möglichkeit die Logik eines Erweitertenprädikats komplett auszutauschen.

Um die zuvor beschriebene Vorgehensweise der Prädikaterstellung zu verdeutlichen, wird das Beispiel in Abbildung 4.3 näher betrachtet. Das dort dargestellte Prädikat enthält zwei atomare Ausdrücke "(a > b)" und "true". Bei der Erzeugung eines atomaren Ausdrucks muß dessen Wahrheitswert und der darzustellende String angegeben werden. Die Einfärbung des Strings wird automatisch vorgenommen, die Farbe wird dabei durch den Wahrheitswert bestimmt. Im Beispiel hat der Ausdruck "(a > b)" den Wahrheitswert „falsch“ und wird somit rot eingefärbt. Wie man anhand der beiden Ausdrücke im Beispiel sehen kann, werden um den String eines atomaren Ausdrucks nicht automatisch Klammern gesetzt, sondern sie müssen manuell in den String eingefügt werden, wenn sie erwünscht sind. Die beiden erzeugten atomaren Ausdrücke werden jetzt mit Hilfe des "OR"-Operators zu einem zusammengesetzten Ausdruck verknüpft. Der Wahrheitswert und der String, inklusive Klammern und Einfärbungen, wird für diesen Ausdruck automatisch erzeugt. Im Beispiel ist somit der Gesamtausdruck erstellt, es könnten aber auch noch größere Ausdrücke durch weiters Verknüpfen mit zusammengesetzten bzw. atomaren Ausdrücken gebildet werden.

Wenn ein Prädikatcheck angestoßen werden soll, dann muß lediglich noch ein atomarer Ausdrucks des Prädikats und dessen neuer Wahrheitswert angegeben werden. Dies bedeutet, daß alle Teilausdrücke des Prädikats bis hin zum Gesamtausdruck ausgewertet werden und eine Animation dieses Vorgangs für den Benutzer abläuft. Die Animation des Prädikatenchecks besteht aus der schrittweisen Änderung der Einfärbungen von Teilen des Strings, von den atomaren Ausdrücken beginnend, hin zum Gesamtausdruck. Analog zum Defaultprädikat blinkt während des Prädikatchecks der Rahmen und nach Beendigung des Prädikatchecks wird ein Event ausgelöst.

Wie man erkennen kann, wäre die gleiche Darstellung des Prädikats und der gleiche Ablauf des Prädikatchecks auch mit dem Defaultprädikat zu erreichen, dies würde aber für den Baukastenbenutzer einen viel größeren Aufwand bedeuten, da er die diversen farbigen Strings selbst erstellen müßte. Der Nachteil der erweiterten Prädikate ist hingegen, daß die Art der Darstellung und die Visualisierung des Prädikatchecks deutlich eingeschränkt ist bzw. man sich mit dem Vorgegebenen zufrieden geben muß.

5 Automatendarstellung

Die Automatendarstellung nimmt Zustands-, Übergangs- und Prädikatobjekte entgegen und verwaltet die Darstellung des aus ihnen gebildeten endlichen Automaten. Alle Events die an den Automaten und seine Bestandteile gerichtet sind, gehen über die Automatendarstellung. Beispielsweise muß der Ausführungswunsch eines Prädikatchecks oder eines Zustandsübergangs an die Automatendarstellung gerichtet werden.

Um die Darstellung des Automaten zu ermöglichen müssen zunächst die Zustände, Übergänge und Prädikate an die Automatendarstellung durch ein Event übergeben werden. Die Zustände werden dann an den Positionen angezeigt, die vom Baukastenbenutzer diesen mitgegeben werden können, wie schon bei der Beschreibung der Zustände beschrieben wurde. Die Übergänge und Prädikate werden automatisch passend angeordnet, da sie die Übergänge an den Zuständen und die Prädikate wiederum an den Übergängen orientieren.

Danach kann durch den Baukastenbenutzer mittels eines Events eine automatische Anordnung der Zustände ausgelöst werden. Sie werden dann so verteilt, daß es möglichst wenig Überkreuzungen zwischen den die Übergänge darstellenden Kanten gibt. Damit kann die zuvor beschriebene aufwendige Positionierung der Zustände durch den Baukastenbenutzer entfallen und er kann die Positionen der Zustände gleich auf den voreingestellten Werten belassen. Außerdem wird die Größe der Automatendarstellung bei der automatischen Ausrichtung der Zustände auf den jeweiligen endlichen Automaten angepaßt.

Da der Mensch allerdings häufig bessere bzw. für vorher nicht genau absehbare Anforderungen passendere Ergebnisse erzielen kann, wenn er die Zustände selbst verteilt, wird das interaktive verschieben der Zustände durch den Anwender ermöglicht. Soll ein Zustand zum verschieben selektiert werden, so wird dem Anwender durch eine andere Darstellung des Mauspfils angezeigt, wenn sich der Mauspfil über einem Zustand befindet. Der Zustand kann dann bei gedrückter linker Maustaste verschoben werden.

Befindet sich der Mauspfil über der Darstellung eines Zustands, eines Übergangs oder eines Prädikats, dann kann durch Drücken der rechten Maustaste ein Fenster aufgerufen werden, in dem der Infotext steht, der dem jeweiligen Objekt zugeordnet ist. Ist einem Objekt kein Infotext zugeordnet, wird der Mausklick ignoriert und es passiert keine weitere Aktion. Wie in den nächsten Abschnitten noch zu sehen sein wird, ist das Anzeigen eines Infotextes bei allen Bestandteilen der Visualisierung auf gleiche Weise möglich.

Da eine übersichtliche Automatendarstellung größer werden kann, als die zur Verfügung stehende Fläche im Fensters, sollten Scrollbars verwendet werden, um es zu ermöglichen die Anzeige auf den interessanten Bereich der Automatendarstellung zu fokussieren. Mehr dazu im Abschnitt 10, bei der Beschreibung des Sammelfensters.

6 Variablen

Die Variablen sind der erste beschriebene Bestandteil des FSM-Baukasten, der nicht direkt zur Darstellung des endlichen Automaten gehört. Sie werden deshalb in einem separaten Fensterteil aufgelistet. Jede Variable mit einem festen Namen versehen und besitzen einen veränderbaren Wert. In der konkreten Darstellung steht der Name links und der Wert rechts. Optisch wird eine Trennung beider, durch eine unterschiedlich Art der Darstellung erreicht (Abbildung 4.4). Der Name steht innerhalb eines Buttons und der Wert in einem Textfeld.

Der Wert einer Variable kann, sofern dies erlaubt ist, interaktiv durch den Benutzer editiert werden. Durch Drücken des Buttons kann ein Event erzeugt werden, in dem der neue Wert der Variable enthalten ist. Dies sollte allerdings nur in Ausnahmefällen möglich sein, z.B. wenn eine Variable einer Konstanten (maximale Wiederholungszahl) entspricht und der Anwender interaktiv die Protokollvisualisierung beeinflussen können soll.

Da das Ändern von Variablen im Normalfall nicht erlaubt ist, wird die Darstellung verändert, wenn eine Variable verändert werden kann, so daß dies klar ersichtlich ist (Variable 6 und 7 in Abbildung 4.4). Jede Variable kann hierbei getrennt behandelt werden.

Selbstverständlich kann der Wert aller Variable jederzeit vom Baukastenbenutzer geändert werden. Dies kann durch ein Event veranlaßt werde, in dem die betreffende Variable und der neue Wert mitgeteilt werden. Die zuletzt geänderten Variablen werden farbig unterlegt dargestellt, um diese für den Benutzer kenntlich zu machen (Variable 4 und 7 in Abbildung 4.4). Diese Markierung bleibt für mehrere Sekunden erhalten, danach wird sie automatisch wieder entfernt.

Variable 1	3,87	Variable 2	false
Variable 3	25	Variable 4	0
Variable 5	Text	Variable 6	84
Variable 7	true	Variable 8	47
Variable 9	47		

Abbildung 4.4

Variablen können durch die Anwendung eingefügt und entfernt werden. Die Variablendarstellungen werden dabei übereinander gestapelt und automatisch mehrspaltig aufgeteilt. Die Variablen werden zeilenweise, in der Reihenfolge in der sie eingefügt wurden, angeordnet.

Durch Anklicken mit der rechten Maustaste in die Darstellung einer Variable, kann in einem Fenster ein Infotext angezeigt werden, der dieser zugeordnet ist. Der Infotext sollte Aufschluß darüber geben, was für eine Funktion die Variable hat und ihren Zusammenhang mit der Automatendarstellung erklären.

Wird die Variablenauflistung zu groß für die im Fenster verfügbare Fläche, dann wird ein Teil der Grafik verdeckt. Der Benutzer des FSM-Baukastens sollte, wie auch schon bei der Automatendarstellung, eine Scrollbar zu Verfügung stellen, um jede Variable betrachten zu können.

7 Timer

Neben den normalen Variablen, die im vorigen Abschnitt beschrieben wurden, gibt es noch zwei spezielle Typen von Variable, die eine besondere grafische Darstellungsform erhalten und somit separat dargestellt werden. Dies sind zum einen die nun im folgenden vorgestellten Timer und die im nächsten Abschnitt näher beschriebenen Nachrichten.

Ein Timer wird in Form einer Digitaluhr dargestellt, die auf einen bestimmten Anfangswert gesetzt werden kann und dann vom Baukastenbenutzer gesteuert in Einzelschritten nach unten zählt (Abbildung 4.5). Alternativ kann auch eine bestimmte Zeit angegeben werden, in welcher der Timer ablaufen soll. Dieser Zeitwert wird in realer Zeit angegeben. Außerdem ist es möglich zwischen beiden Modi zu wechseln.

Nach erzeugen des Timers wird der Wert in gelber Farbe angezeigt. Dies bedeutet, daß der Timer in einem Wartezustand ist. In diesen Zustand kann man auch durch zurücksetzen des

Timers kommen, wobei ein verändern des Anfangswertes möglich ist. Wird das automatische Zählen gestartet oder der Wert vom Baukastenbenutzer erniedrigt, wird für die Anzeige des Wertes auf grün umgeschaltet. Ist der Timer abgelaufen, d.h. der Wert 0 wird angezeigt, dann wird ein Event ausgelöst. Außerdem erhält die Anzeige des Timers eine andere Farbe, in diesem Fall rot.

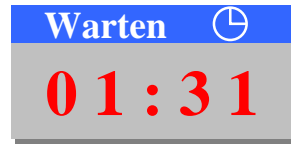


Abbildung 4.5

Wie man aus dem zuvor beschriebenen Eigenschaften erkennen kann, besitzt ein Timer drei verschiedenartige Zustände: auf seinen Start wartend, ablaufend und beendet.

Um eine (logische) Zuordnung des Timers zu einem Zustand im Automaten für den Benutzer erkenntlich zu machen, kann der Name des Zustands in der oberen Hälfte der Darstellung des Timers ausgegeben werden. Ist diese Zuordnung nicht erwünscht, kann anstatt dem Namen eines Zustands auch ein Text angezeigt werden, der Aufschluß über die Bedeutung des Timers gibt.

Da nur ein Begriff in der Regel nicht ausreicht, um die Bedeutung des Timers zu erklären, kann jedem Timer ein erklärender Infotext mitgegeben werden. Jeder Timer zeigt durch Klicken mit der rechten Maustaste in seinen Darstellungsbereich ein Infofenster mit dem Infotext an.

8 Warteschlangen

Es gibt grafisch dargestellte Warteschlangen für eingehende und ausgehende Nachrichten. Eine Warteschlange für eingehende Nachrichten ist wahrscheinlich in den meisten Prozessen nötig, eine für ausgehende Nachrichten hingegen nicht unbedingt, da deren Funktion eventuell von der eingehenden Warteschlange eines anderen Prozesses übernommen werden soll.

Nachrichten sind Objekte, die allerdings nicht bis ins letzte Detail spezifiziert sind. Als einziges steht fest, daß sie einen String beinhalten, der bei ihrer Darstellung in der Warteschlange verwendet wird und ihre jeweilige Bedeutung widerspiegeln sollte. Außerdem besitzt jede Nachricht ein Ziel und einen Inhalt. Was dies jeweils ist, muß erst durch den Baukastenbenutzer festgelegt werden und ist für den FSM-Baukasten selbst unerheblich.

Jede Nachricht wird mit einem Briefsymbol dargestellt, das mit dem Name der Nachricht beschriftet ist. Da nicht jeder Name gleich lang ist, wird die Größe des Briefsymbols automatisch angepaßt. Ist der Name sehr kurz, dann wird das Briefsymbol nicht unter eine Mindestgröße verkleinert.

Die Warteschlangendarstellung besteht aus einem Nachrichtenstapel (Abbildung 4.6). Die neu einzufügenden Nachrichten, werden bei der Warteschlange für ausgehende Nachrichten einfach am Boden des Stapels eingefügt, bei der für eingehende Nachrichten an der Oberseite

des Stapels. Im letzteren Fall läuft eine kleine Animation ab, die eine hereinfliegende Nachricht veranschaulicht (Abbildung 4.6 rechtes). Ist das Einfügen der Nachricht beendet, wird bei beiden Arten der Warteschlangen ein Event erzeugt.

Analog läuft dies bei der Entnahme von Nachrichten ab. Bei der Warteschlange für eingehende Nachrichten wird die Nachricht einfach am Boden des Stapels entfernt. Bei der Warteschlange für ausgehende Nachrichten läuft beim Herausnehmen der Nachricht eine kleine Animation ab, bei der ein Nachrichtensymbol heraus fliegt (Abbildung 4.6 links). Anschließend wird ein Event ausgelöst.

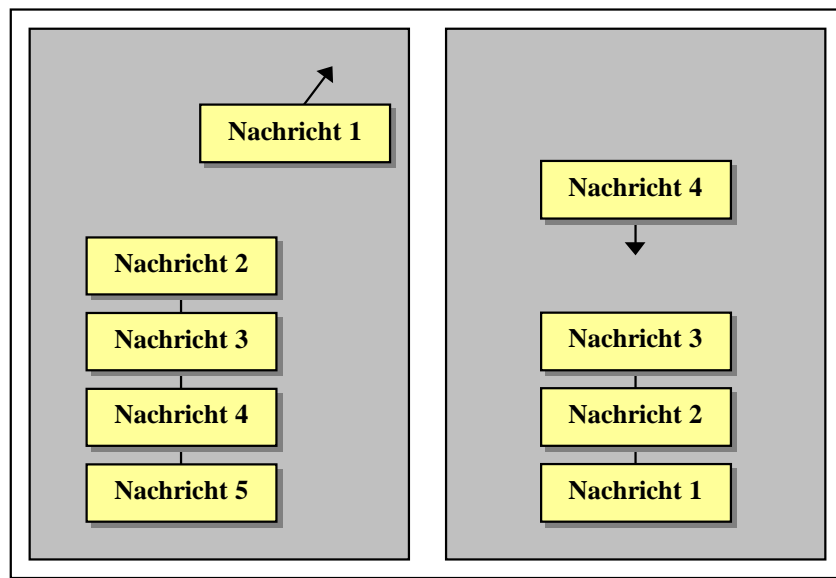


Abbildung 4.6

Da Warteschlangen sehr lang werden können und deshalb eventuell nur Teile von ihr auf einmal dargestellt werden können, sollten Scrollbars eingesetzt werden, um dem Anwender die ganze Darstellung zugänglich zu machen.

Warteschlangen können beliebig lang sein oder eine begrenzte Länge haben. Nachrichten die bei einer gefüllten Warteschlange eintreffen gehen in der Regel verloren. Wenn die Länge einer Warteschlange begrenzt werden soll, muß dies allerdings auf Logikebene gehandhabt werden, da dies keine Aufgabe für die grafische Darstellung ist.

9 Statusbar

Für die Informationen des Benutzers über Aktionen steht eine Statusbar zur Verfügung. Die üblicherweise auftretenden Aktionen sind Zustandsübergänge, Änderungen der Werte von Variablen, Zustandsänderungen von Timern und das Ankommen bzw. Ausgehen von Nachrichten. Der Anwendung steht es allerdings frei auch in beliebigen anderen Situationen die Statusbar für die Kommunikation mit dem Benutzer zu verwenden.

In die Statusbar können einzelne Textzeilen eingefügt werden, die übereinander aufgelistet werden (Abbildung 4.7). Der Text selber ist nicht weiter eingeschränkt, sollte aber komplett in den Darstellungsbereich der Statusbar passen. Außerdem sollte ein zusammengehörender Text nicht auf mehrere Zeilen verteilt werden. Eine Numerierung der Zeilen kann automatisch erzeugt und angezeigt werden.

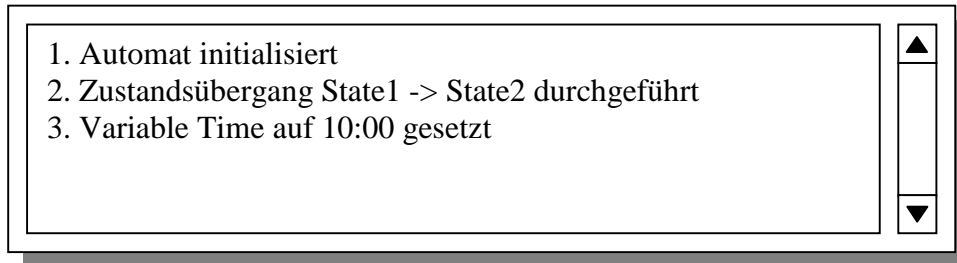


Abbildung 4.7

Da nur ein kleiner Teil der Zeilen auf einmal in der Statusbar angezeigt werden kann, ist eine Scrollbars vorhanden. Standardmäßig wird immer auf die zuletzt eingefügte Zeile fokussiert. Somit werden nur die neusten Zeilen dauerhaft angezeigt, die älteren Zeilen sind nur über die Scrollbars erreichbar.

10 Sammelfenster

Es wird ein Sammelfenster bereitgestellt, das die zuvor beschriebenen einzelnen Fensterteile kompakt zusammen darstellt. Die Verwendung dieses Fensters ist keine zwingende Voraussetzung beim Einsatz des FSM-Baukastens, es stellt nur eine mögliche Erleichterung bei der Erstellung einer Anwendung dar.

Jedes Sammelfenster enthält mindestens eine Automatenarstellung und eine Statusleiste. Alle andern Komponenten können je nach Bedarf eingesetzt werden. In Abbildung 4.8 ist die Aufteilung des Sammelfensters zu sehen. Wie man sehen kann, sind ein Bereich mit Variablen, zwei Warteschlangen und neun Timer darstellbar. Wird eine dieser Komponenten nicht gebraucht, wird der von ihr ansonsten belegte Platz durch die andern Komponenten ausgefüllt.

Die optionalen Komponenten können jederzeit, also auch zur Laufzeit, entfernt werden. Dies ist vor allem für Timer interessant, die nur dann angezeigt werden sollten, wenn der aktive Zustand mit ihnen verknüpft ist.

Da für einen Betrachter nicht unmittelbar klar ist, was die verschiedenen Teile im Sammelfenster bedeuten sollen, kann jeder Bereich des Fensters mit einer Beschriftung versehen werden. Sie wird in weißer Schrift auf blauem Hintergrund oberhalb der jeweiligen Komponente angezeigt. Dabei ist zu beachten, daß nicht verwendete Beschriftungen auch nicht angezeigt, als auch kein blauer Balken ohne Schrift erscheint.

Im Sammelfenster werden die in den vorigen Abschnitten geforderten Scrollbars mit den verschiedenen Fensterteilen verknüpft. Dies bedeutet, daß die Automatenarstellung, die Warteschlangen und der Bereich mit den Variablen jeweils vertikale und horizontale

Scrollbars erhalten. Die Scrollbars werden allerdings nur dann angezeigt, wenn sie notwendig sind, also nur wenn sie größer sind als der ihnen zu Verfügung gestellte Fensterbereich.

Das Sammelfenster unterstützt eine interaktive Steuerung durch den Anwender. Dazu wird eine Menüleiste eingesetzt. Es gibt eine Menü, das immer vorhanden ist, in der feste Menüpunkte eingetragen sind. Bislang sind dies ein Menüpunkt zum beenden der Anwendung, ein Menüpunkt der das Ausrichten des Automaten in der Automaten-darstellung auslöst und ein weiteren Menüpunkt, der die Numerierung in der Statusbar an- bzw. abschaltet.

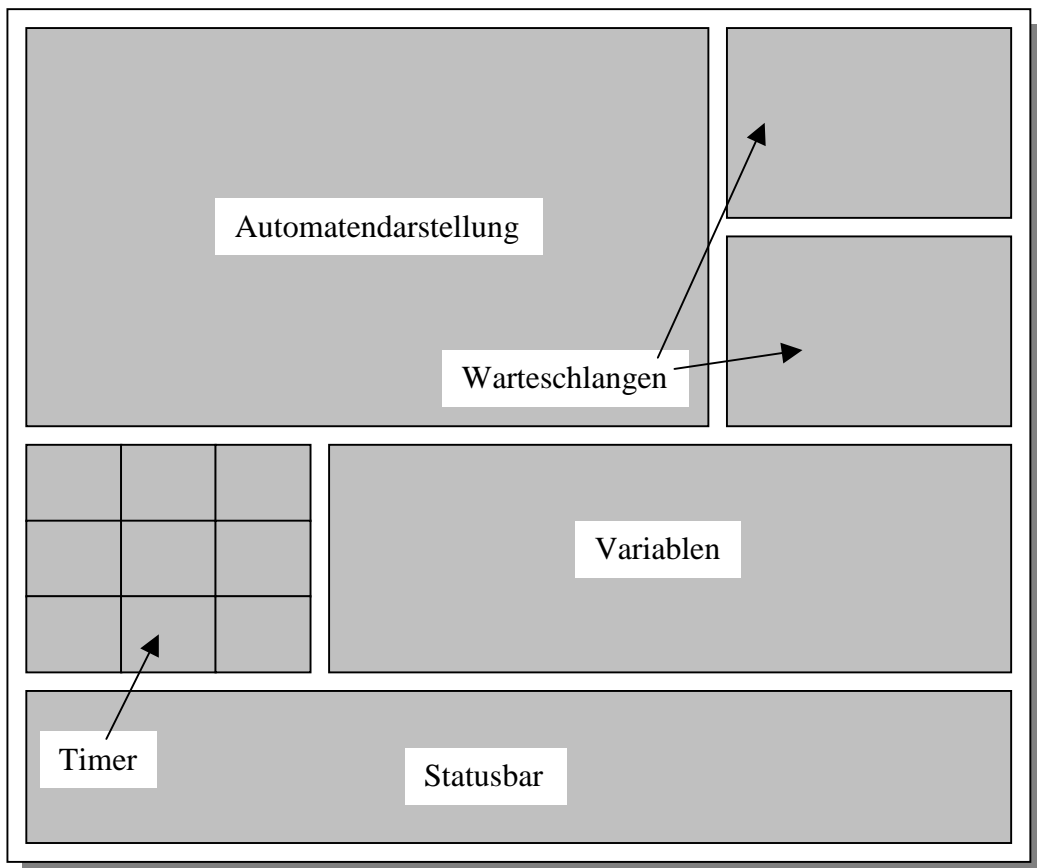


Abbildung 4.8

Weiterhin gibt es eine dynamische Menüleiste, in die der Baukastenbenutzer eigene Menüpunkte eintragen kann. Für diese Menüpunkte muß ein eindeutiger Name vergeben werden. Wird einer dieser Menüpunkte ausgewählt, dann wird ein Event ausgelöst, in dem der Name des Menüpunktes enthalten ist.

11 Ablaufbeispiel

Um einen besseren Eindruck davon zu geben, wie sich das Zusammenspiel der zuvor einzeln beschriebenen Komponenten gestaltet, soll im folgenden ein kompletter Ablauf aus Anwendungssicht umrissen werden. Dazu wird ein Ablaufbeispiel rund um den kleinen endlichen Automat entworfen, der in Abbildung 4.9 zu sehen ist, der das Protokoll eines Weckers darstellt. Angemerkt sei noch, daß auf die Ausgaben in der Statusbar verzichtet wird. Man kann diese sich implizit bei den Aktionen erzeugt vorstellen.

Als erster Schritt muß der endliche Automat vom Baukastenbenutzer aufgebaut werden. Zunächst werden die beiden Zustände erzeugt und der Zustand „Warten“ als erster aktiver Zustand, also als Startzustand, gekennzeichnet. Der Zustand „Warten“ sei gleichzeitig noch Endzustand. Zwischen den beiden Zuständen werden anschließend die drei Übergänge erzeugt. Jedem Übergang wird nun ein Prädikat zugeordnet. Im Beispiel sollen Erweiterteprädikate verwendet werden. Jedes Prädikat wird deshalb, wie zuvor beschrieben wurde, aus atomaren Ausdrücke und booleschen Operatoren aufgebaut, hier also nur atomare Ausdrücke. Die atomaren Ausdrücke werden auf ihre Ausgangsbelegungen mit Wahrheitswerten gesetzt, d.h. alle auf „falsch“. Die Zustände, Übergänge und Prädikate werden an die Automatendarstellung, im nun zu erzeugenden Automatenfenster, per Event übergeben. Außerdem wird die automatische Ausrichtung der Zustände durch ein Event ausgelöst.

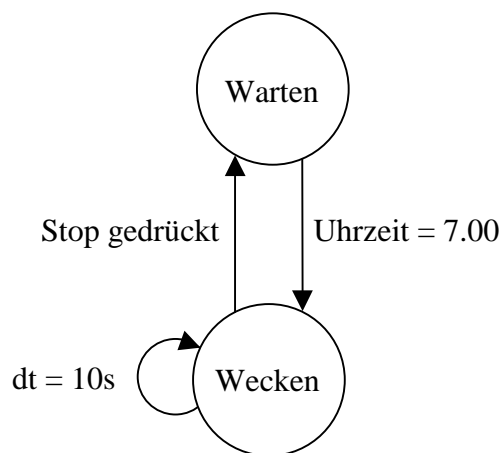


Abbildung 4.9

Abschließend werden zwei Warteschlangen erzeugt. Eine für eingehende und eine für ausgehende Nachrichten. Dann wird ein Variablenbereich mit zwei Variable erzeugt. Die erste enthält die aktuelle Uhrzeit, die Zweite enthält die Weckzeit 7.00 Uhr. Beide sehen nicht editierbar. Die Warteschlangen und der Variablenbereich werden per Events in das Sammelfenster eingefügt. Damit ist die Ausgangsdarstellung aufgebaut.

Nun wird die Protokollvisualisierung gestartet. Alle nun an die Grafikkomponenten gesendeten Anweisungen seien weiterhin mittels Events übertragen. Zunächst treffe eine Nachricht in der Eingangwarteschlange ein, was durch eine Animation verdeutlicht wird. Nach beenden der Animation wird ein Event ausgelöst, worauf die Nachricht ausgewertet wird. Sie enthält die neue aktuelle Uhrzeit, dies sei 7.00 Uhr. Der Wert wird in die Variable für die aktuelle Uhrzeit eingetragen und für fünf Sekunden gelb hinterlegt. Nun wird ein Prädikatcheck am Prädikat „Uhrzeit = 7.00“ ausgelöst. Dazu wird ein Event an die Automatendarstellung geschickt, welches den zugehörigen atomaren Operator und den neuen Wahrheitswert „wahr“ enthält. Nach beenden des Prädikatchecks, was durch ein Event angezeigt wird, wird der Zustandsübergang vom Zustand „Warten“ in den Zustand „Wecken“ durchgeführt und animiert. Was wiederum durch ein Event abgeschlossen wird.

Nun werden zwei Aktionen durchgeführt. Zuerst wird eine Nachricht mit Hilfe der ausgehenden Warteschlange, was eine kleine Animation mit Event am Ende bedeutet, an eine Klingel gesandt. Dann wird ein Timer mit dem Startwert 10 Sekunden erzeugt und in das

Sammelfenster eingefügt. Dies verkleinert die Darstellung des Bereichs für die Variablen. Der Timer wird gestartet und beginnt automatisch zu laufen. Bevor der Timer abgelaufen ist, trifft eine Nachricht in der Eingangswarteschlange ein, was wiederum animiert wird. Die Nachricht sagt aus, daß auf die Stoptaste gedrückt worden ist. Daraufhin wird ein Prädikatcheck, auf analoge Weise wie zuvor erklärt wurde, für das Prädikat „Stop gedrückt“ angestoßen, an dessen Ende das Prädikat den Wert Wahr hat. Nachdem das Event am Ende des Prädikatchecks versandt wurde, wird der Zustandsübergang „Wecken“ nach „Warten“ durchgeführt, nach dessen Ende der Timer gestoppt und aus dem Sammelfenster entfernt wird.

Da sich der Automat nun im Endzustand befindet, ist ein kompletter Durchlauf vollzogen und die Protokollvisualisierung kann damit beendet werden.

Da nun alle Komponenten des TSD- und des FSM-Baukastens vorgestellt worden sind, wird im nächsten Kapitel nun eine größere Beispielanwendung betrachtet, in der das „Stop and Wait“-Protokoll visualisiert wird. Im Gegensatz zu dem hier besprochenen Ablaufbeispiel, wird dort vor allem das Zusammenspiel der verschiedenen Komponenten und der beiden Baukästen betrachtet, aufgrund der erhöhten Komplexität kann dort allerdings nicht mehr auf die kleinen Details eingegangen werden, wie es im Ablaufbeispiel noch möglich war.

Kapitel 5:

Beschreibung eines umfangreichen Anwendungsbeispiels

Um einen Eindruck davon zu gewinnen, wie die zuvor vorgestellten Komponenten des neuen Baukastens in der Praxis eingesetzt werden können, betrachten wir nun ein Anwendungsbeispiel. Als zu visualisierendes Protokoll wurde das „Stop and Wait“-Protokoll gewählt. Dieses Protokoll ist noch einfach genug, um übersichtlich dargestellt werden zu können, benötigt zu Visualisierung aber zugleich alle wesentlichen Komponenten des Baukastens und ist hinreichend praxisrelevant.

Im folgenden wird, bevor wir zur Beschreibung des erstellten Demonstrationsapplets kommen, zunächst ein Einblick in grundlegende Konzepte für den Einsatz des Baukastens gegeben und das „Stop and Wait“-Protokoll vorgestellt. Die grundlegenden Konzepte beziehen sich hierbei, im Gegensatz zu den vorhergehenden Kapiteln, nicht direkt auf die einzelnen Komponenten des Baukastens, sondern vielmehr auf deren Zusammenspiel.

1 Grundkonzepte für den Einsatz des neuen Baukastens

Im momentanen Entwicklungsstand des neuen Baukastens ist, wie schon erwähnt wurde, die Logikebene noch nicht implementiert, die Anwendung muß also deren Aufgaben mit übernehmen. Diese bestehen aus dem Modellieren des darzustellenden Protokolls und dem Steuern der Visualisierung.

Der auf die Grafiksteuerung bezogene Teil einer Anwendung gliedert sich allgemein in zwei Phasen. In der ersten Phase wird die Darstellung aufgebaut. Dies bedeutet, daß die benötigten Zustands-, Übergangs- und Prädikatobjekte erzeugt werden und die von ihnen erzeugten grafischen Darstellungen in den jeweils gewünschten Ausgangszustand versetzt werden. Analog wird mit den gewünschten Variablen, Timer und Nachrichtenwarteschlangen verfahren.

Ist die Darstellung aufgebaut, können in der zweiten Phase Visualisierungen von Aktionen angestoßen werden. Dabei sind lediglich durch die vorhandenen grafischen Elemente und deren Fähigkeiten Beschränkungen vorgegeben. Dies bedeutet, daß z.B. ein Zustandsübergang nicht rückwärts ablaufend dargestellt werden kann. Keine Einschränkungen bestehen allerdings hinsichtlich etwaiger Abläufe, die aus Sicht der Logik eigentlich nicht möglich wären. Beispielsweise ist es möglich, einen Zustandsübergang zu animieren, der von einem Zustand ausgeht, der nicht aktiviert ist. Somit ist es also auch möglich, einen Automaten mit mehreren aktiven Zuständen zu erzeugen, was eigentlich im Modell der endlichen Automaten nicht gestattet ist. Dergleichen zu verhindern, obliegt derzeit noch der Anwendung.

Wie in den vorhergehenden Kapitel beschrieben wurde, findet die Kommunikation mit dem Baukastens eventbasiert statt. Das Problem ist dabei vor allem, die zeitliche Steuerung der Darstellung und ein eventuell paralleler Ablauf von Aktionen. Falls eine Operation eine länger Zeitspanne beansprucht, was vor allem bei einer Animation der Fall ist, antwortet der Baukasten nach deren Beendigung mit einem Event. Dies ist beispielsweise bei

Zustandsübergängen der Fall. Möchte die Anwendung eine Synchronisation erzielen, muß sie auf dieses Event warten, oder es zumindest beachten und je nach Anforderung darauf reagieren. Die Anwendung kann somit auch mehrere parallel ablaufende Animationen, wie Prädikatchecks oder Zustandsübergänge handhaben.

Für die Anwendung wird es problematischer, wenn durch den Grafikbaukasten asynchron Events ausgelöst werden können. Dies ist etwa beim Ablauf eines Timers oder bei Benutzereingaben der Fall. Die Verfahrensweise in dieser Situation kann von Anwendung zu Anwendung verschieden sein. Eine Möglichkeit ist die Einführung einer Warteschlange, wie es etwa im Demoapplet implementiert wurde. Mit ihrer Hilfe werden die eintreffenden Events der Reihe nach abgearbeitet.

2 Beschreibung des „Stop and Wait“-Protokolls

Im Demoapplet wird das „Stop and Wait“-Protokoll (auch „Idle RQ“-Protokoll genannt) visualisiert, wie es in [Hals96] beschrieben ist, dabei wird sich streng an die dort gegebene Spezifikation gehalten.

Damit der interessierte Leser direkt einen Vergleich der Spezifikation mit der Implementation durchführen kann, werden die wesentlichen Teile der Spezifikation des Senders und des Empfängers zusammengefaßt. Dies soll weniger ein Ersatz für das Studium der einschlägigen Literatur sein, als viel mehr eine kompakte Übersicht.

2.1 Grundlagen

Das „Stop and Wait“-Protokoll ist im Bereich der Rechnernetze anzusiedeln. Deren Ziel ist es, den Datenaustausch zwischen verschiedenen Rechnern zu ermöglichen. Da dies im allgemeinen eine komplexe und schwierige Aufgabe ist, wurde sie in mehrere Bereiche aufgeteilt. Um diese Bereiche sauber zu trennen und deren konkrete Implementierungen voneinander unabhängig zu machen, wurde ein Schichtenmodell eingeführt. Dieses ist vom Prinzip her mit dem zuvor vorgestellten Schichtenmodell des Baukasten vergleichbar, allerdings weit umfangreicher.

Der zweit untersten Schicht ist das „Stop and Wait“-Protokoll zuzuordnen. Sie wird mit Sicherungsschicht oder auch mit (Data) Link Layer bezeichnet. Die Aufgabe dieser Schicht ist es, aus der untersten Schicht, die Bitübertragungsschicht oder Physical Layer genannt wird und für die physikalischen Probleme der Datenübertragung zuständig ist, aber nicht sicherstellen kann, daß die Daten immer korrekt übertragen werden, eine zuverlässige Datenübertragung für die übergeordneten Schichten sicher zu stellen.

Das „Stop and Wait“-Protokoll kontrolliert die Übertragung von Daten zwischen einem Sende- und einem Empfängerrechner. Auf beiden wird der entsprechende Teil des „Stop and Wait“-Protokolls ausgeführt. Der Sendeteil wird auch mit Primary bezeichnet und deshalb mit P abgekürzt. Der Empfängerteil heißt analog Secondary und wird mit S abgekürzt.

Die Daten werden der Reihe nach vom Sender zum Empfänger übertragen. Dabei werden die nächsten Daten erst dann übertragen, wenn gesichert ist, daß die gerade gesendeten Daten beim Empfänger korrekt empfangen wurde.

2.2 Prinzip

Im folgenden, soll der Teil des „Stop and Wait“-Protokoll, der für das Senden verantwortlich ist, mit Sender und analog der Empfangsteil mit Empfänger, bezeichnet werden. Man kann sich somit für ein bessere Verständlichkeit eine Aufteilung auf einen sendenden und einen empfangenden Rechner vorstellen, die allerdings in Wirklichkeit so nicht existiert, da in der Regel alle Kommunikationspartner die Fähigkeit des Sendens und des Empfangens haben.

Der Sender erhält der Reihe nach Daten bzw. Datenpakete, die zum Empfänger übertragen werden sollen. Aus dem jeweils ersten Datenpaket erstellt er einen Daten-Frame, der neben den Daten unter anderem auch die Nummer des Frames in der Sendefolge enthält (send sequence number, kurz $N(S)$). Die Nummer wird aus einer Variable ausgelesen, die jeweils die Folgenummer für den als nächstes übertragenen Frame enthält (send sequence variable, kurz $V(S)$). Ist der Daten-Frame erstellt, wird er zum Empfänger übertragen. Der Sender merkt sich außerdem den Daten-Frame und stellt vorläufig das Senden bzw. Erstellen weiterer Daten-Frames ein. Für den Daten-Frame wird im folgenden die allgemeinere Bezeichnung Information-Frame (kurz I-Frame) verwendet, da nicht nur Daten, sondern auch Steueranweisungen u.ä. enthalten sein können.

Hat der Empfänger den I-Frame korrekt empfangen, überprüft er ob dies der von ihm als nächstes erwartete I-Frame ist. Dies leitet er aus der im I-Frame enthaltenen $N(S)$ und einer Variable ab, die die $N(S)$ des letzten korrekt empfangenen und in die Empfangsreihenfolge passenden I-Frames enthält (receive sequence variable, kurz $V(R)$). Ist dies der Fall (bzw. ist der I-Frame noch älter), sendet der Empfänger eine Bestätigung an den Sender. Die Bestätigung enthält die aktuelle Nummer in der Empfangsfolge (receive sequence number, kurz $N(R)$). Wurde der I-Frame zuvor noch nie korrekt empfangen, ist dies seine $N(S)$.

Sobald der Sender die Bestätigung vom Empfänger bekommt, deren $N(R)$ dem gemerkten I-Frame mit $N(S)$ entspricht, löscht der Sender den gemerkten I-Frame und kann nun wieder neue I-Frame erzeugen, also neue Daten senden. Über die ausgetauschten Nachrichten ist somit mit Hilfe der Folgenummern eine Synchronisation zwischen Sender und Empfänger vorgenommen worden. Der Sender kann also den Empfänger nicht mit Daten überschwemmen.

In dem zuvor beschriebenen Ablauf wurde davon ausgegangen, daß alle Nachrichten ihr Ziel auch erreichen. Kritischer sind allerdings die Fälle, in denen Nachrichten verloren gehen. Ist dies der Fall, kann die Kommunikation zwischen Sender und Empfänger zum Erliegen kommen, da beide gleichzeitig auf eine Nachricht vom anderen warten. Ein Beispiel dafür ist beispielsweise, wenn eine Bestätigung vom Empfänger verloren geht. Wenn dies passiert, tritt die Situation ein, daß der Sender auf die verlorengegangene Bestätigung wartet und deshalb nicht weiter sendet. Gleichzeitig verschickt der Empfänger keine weiteren Bestätigungen, da er keine I-Frames mehr vom Sender erhält, auf die er reagieren könnte. Wie man leicht sehen kann, entsteht die gleiche Situation, wenn ein I-Frame vom Sender verloren geht.

Um dieses Problem zu beseitigen, überträgt der Sender im „Stop and Wait“-Protokoll nach einer bestimmten Zeitspanne seit dem letzten Empfang einer Nachricht vom Empfänger den gemerkte I-Frame einfach erneut. Somit kann die Kommunikation nicht dauerhaft zum Erliegen kommen, sofern die Verbindung zwischen Sender und Empfänger nicht total unterbrochen ist. Um feststellen, wann die Zeitspanne verstrichen ist, wird beim Sender ein Timer eingesetzt.

Mit dem bislang gesagten, sind prinzipiell alle Fehlerfälle abgedeckt, allerdings gibt es noch einige Verbesserungen, damit das Protokoll effizienter wird. Eine ist beispielsweise, daß Versenden einer „negativen“ Bestätigung durch den Empfänger, wenn ein durch die Übertragung beschädigter I-Frame bei ihm eintrifft. Erhält der Sender die „negative“ Bestätigung, überträgt er den I-Frame sofort noch mal, ohne auf den Ablauf des Timers durch das Ausbleiben der („positiven“) Bestätigung zu warten.

Eine weitere Möglichkeit ist das Einfügen von Begrenzungen, wie etwa für die maximale Anzahl an Übertragungen des selben I-Frames vom Sender zum Empfänger. Dies soll hier aber nur am Rande erwähnt sein, da dies für das prinzipielle Verständnis der Beispielanwendung nicht unbedingt nötig ist und außerdem wahrscheinlich auch intuitiv verstanden werden kann.

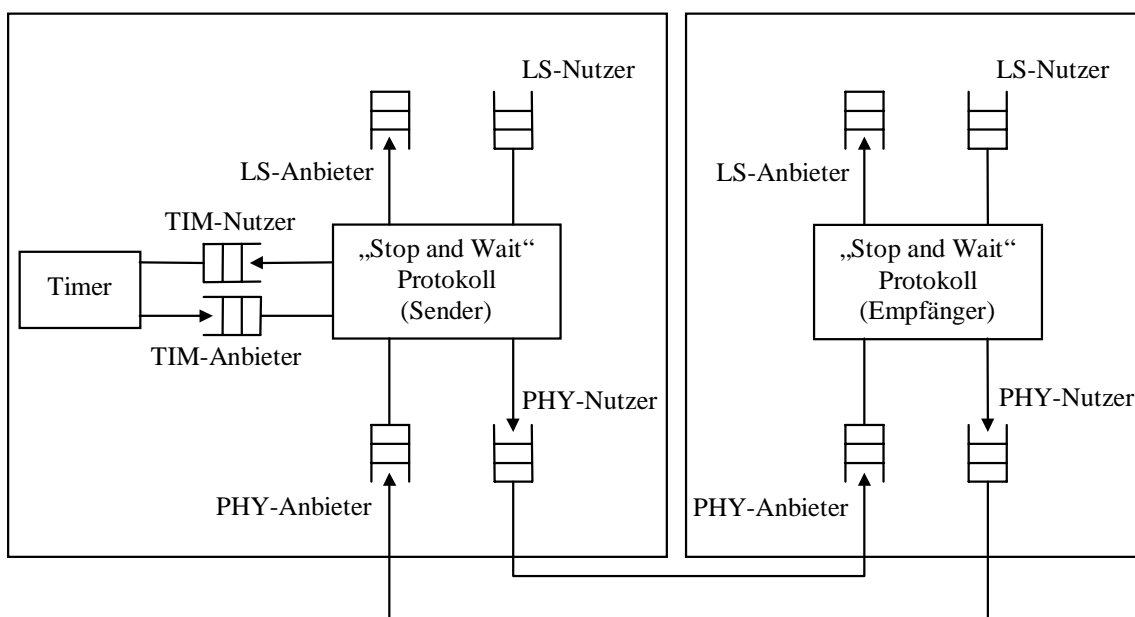


Abbildung 5.1

Viel wichtiger für das Verständnis der folgenden Abschnitte, in denen Sender und Empfänger im Detail spezifiziert werden, sind die in Abbildung 5.1 dargestellten Schnittstellen und Verknüpfungen der Komponenten. Wie man sehen kann, findet die Kommunikation zwischen den Komponenten über Warteschlangen statt. In der Abbildung ist das im vorigen Abschnitt vorgestellte Schichtenmodell angedeutet. An der LS-Schnittstelle wird mit den übergeordneten Schichten kommuniziert, denen ein sicherer Übertragungskanal geboten werden soll. Analog nutzt die Schicht auf der das „Stop and Wait“-Protokoll abläuft ihrerseits die PHY-Schnittstelle, über der ihr ein unzuverlässiger Übertragungskanal geboten wird, mit dem der Rechner auf Senderseite mit dem Rechner auf Empfängerseite Daten austauschen kann. Außerdem ist in Abbildung 5.1 der zuvor beschriebene Timer dargestellt, der auf Senderseite benötigt wird.

2.3 Sender

In diesem und im nächsten Abschnitt werden die wesentlichen Punkte, wie sie in der Spezifikation des „Stop and Wait“-Protokolls von [Hals96] dargestellt worden sind kurz zusammengefaßt. Dies ist nötig, damit die in der Beispielanwendung gewählte Darstellung verstanden werden kann. Auf eine ausführliche Erklärung wird allerdings mit einem Verweis auf die Literatur verzichtet.

In diesem Abschnitt wird zunächst auf den Sender eingegangen. Für diesen werden vorausschickend einige Begriffserklärungen in tabellarischer Form gegeben:

Eingehende Events

<i>Name</i>	<i>Schnittstelle</i>	<i>Bedeutung</i>
LDATAreq	LS-Nutzer	Datenübertragung angefordert
ACKRCVD	PHY-Anbieter	ACK-Frame von Empfänger erhalten
TEXP	TIM-Anbieter	Auf ACK wartender Timer ist abgelaufen
NACKRCVD	PHY-Anbieter	NACK-Frame von Empfänger erhalten

Zustände

<i>Name</i>	<i>Bedeutung</i>
IDLE	Keine laufende Nachrichtenübertragung
WTACK	Warten auf eine Bestätigung

Ausgehende Events

<i>Name</i>	<i>Schnittstelle</i>	<i>Bedeutung</i>
TxFRAME	PHY-Nutzer	Formatiert und überträgt einen I-Frame
RetxFRAME	PHY-Nutzer	Erneute Übertragung des auf seine Bestätigung wartenden I-Frames
ERRORind	LS-Nutzer	Fehlermeldung: Frame aus einem angegebenen Grund verworfen

Prädikate

<i>Name</i>	<i>Bedeutung</i>
P0	$N(S)$ in wartendem I-Frame = $N(R)$ in ACK-Frame
P1	Blocksummenüberprüfung von ACK- bzw. NACK-Frame hat keinen Fehler im Frame erkannt

Zustandsvariablen

<i>Name</i>	<i>Bedeutung</i>
Vs	Variable für die Nummer in der Sendefolge
PresentState	Momentaner Zustand, in dem sich die Protokollinstanz befindet
ErrorCount	Anzahl der empfangen Frames, die fehlerhaft waren
RetxCount	Anzahl der Neuübertragungen für den aktuellen Frame

Aktionen

-
- [1] = Starte Timer
 - [2] = Inkrementiere $V(S)$
 - [3] = Stoppe Timer
 - [4] = Inkrementiere RetxCount
 - [5] = Inkrementiere ErrorCount
 - [6] = Setze RetxCount auf 0 zurück
-

Abbildung 5.2 zeigt die Spezifikation des Senders als endlichen Automat. Die gleiche Darstellung wurde auch im Demoapplet verwendet.

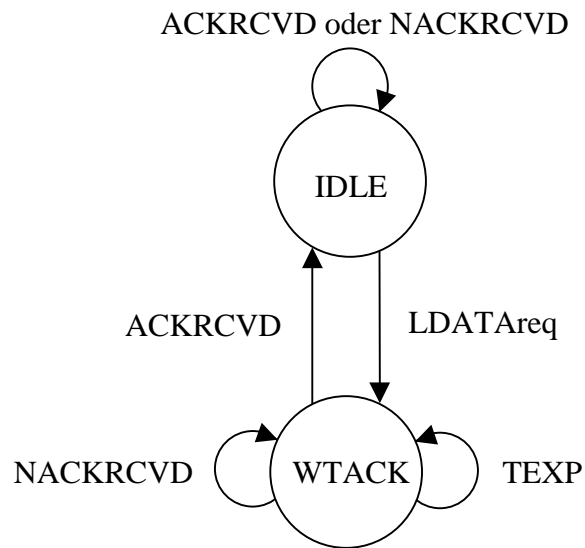


Abbildung 5.2

In der nachfolgenden Tabelle wird der in Abbildung 5.2 dargestellte endliche Automat in tabellarischer Form präsentiert und außerdem sind hier zusätzlich die Aktionen aufgeführt, die den Zustandsübergängen zugeordnet sind:

Eingehendes Event: Aktueller Zustand:	LDATAreq	ACKRCVD	TEXP	NACKRCVD
IDLE	1	0	0	0
WTACK	0	2	3	3

0 = [5]; IDLE

1 = TxFrame, [1], [2]; WTACK

2 = P0 und P1: [3], [6]; IDLE

P0 und nicht P1: RetxFrame, [1], [4]; WTACK

Sonst: [5]; IDLE

3 = RetxFrame, [1], [4]; WTACK

Aus der vorstehenden Beschreibung läßt sich nun folgender Pseudo-Code für die Steuerung des Senders ableiten, der aus [Hals96] übernommen wurde und aus dem direkt die Beispielanwendung entwickelt werden konnte (siehe hierzu auch den Abschnitt über die Implementierungsaspekte):

```

program      IdleRQ_Primary;
const       MaxErrorCount;
           MaxRetxCount;
type       Events = (LDATAreq, ACKRCVD, TEXP, NACKRCVD);
           States = (IDLS, WTACK);
var        EventStateTable = array [Events, States] of 0..3;
           PresentState : States;
  
```

```

        Vs, ErrorCount, RetxCOUNT : integer;
        EventType : Events;
procedure Initialize;
procedure TxFrame;
procedure RetxFrame;
procedure LERRORind;
procedure Start_timer;
procedure Stop_timer;
function P0 : boolean;
function P1 : boolean;

begin
    Initialize; /* Zustandsvariablen und EventStateTable */
    repeat
        Warte auf das Eintreffen eines Events;
        EventType := type of event;
        case EventStateTable[EventType, PresentState] of
        0 :   begin ErrorCount := ErrorCount + 1;
                PresentState = IDLE;
                if (ErrorCount = MaxErrorCount) then LERRORind;
            end;
        1 :   begin TxFrame;
                Start_timer;
                Vs := Vs + 1;
                PresentState := IDLE
            end;
        2 :   begin if (P0 and P1) then
                begin Stop_timer;
                    RetxCOUNT := 0;
                    PresentState := IDLE;
                end;
                else if (P0 and not P1) then
                begin RetxFrame;
                    Start_timer;
                    RetxCOUNT := RetxCOUNT - 1;
                    PresentState := WTACK;
                end;
                else
                begin PresentState := IDLE;
                    ErrorCount := ErrorCount + 1;
                    if (ErrorCount = MaxErrorCount) then
                    begin LERRORind;
                        Initialize;
                    end;
                end;
            end;
        3 :   begin RetxFrame;
                Start_timer;
                RetxCOUNT := RetxCOUNT + 1;
                PresentState := WTACK;
                if (RetxCOUNT = MaxRetxCOUNT) then
                begin LERRORind;
                    Initialize;
                end;
            end;
        end;
    until false;
end.
```

2.4 Empfänger

Auf analoge Weise zum vorigen Abschnitt wird hier die Spezifikation des Empfängers kurz umrissen. Es wird wiederum mit einer Begriffserklärung in tabellarischer Form begonnen:

Eingehende Events

<i>Name</i>	<i>Schnittstelle</i>	<i>Bedeutung</i>
IRCVD	PHY-Anbieter	I-Frame vom Sender empfangen

Zustände

<i>Name</i>	<i>Bedeutung</i>
WTIFM	Wartet auf einen neuen I-Frame von Sender

Ausgehende Events

<i>Name</i>	<i>Schnittstelle</i>	<i>Bedeutung</i>
LDATAind(X)	LS-Anbieter	Zeigt den Empfang des I-Frames mit $N(S) = X$ an und übergibt dessen Inhalt
TxACK(X)	PHY-Nutzer	Formatiert und überträgt einen ACK-Frame mit $N(R) = X$
TxNACK(X)	PHY-Nutzer	Formatiert und überträgt einen NACK-Frame mit $N(R) = X$
LERRORind	LS-Anbieter	Gibt eine Fehlermeldung mit Angabe des Grundes aus

Prädikate

<i>Name</i>	<i>Bedeutung</i>
P0	$N(S)$ in I-Frame = V_r
P1	Blocksummenüberprüfung von I-Frame hat keinen Fehler erkannt
P2	$N(S)$ in I-Frame = $V_r - 1$

Zustandsvariablen

<i>Name</i>	<i>Bedeutung</i>
V_r	Variable für die Nummer in der Empfangsfolge
ErrorCount	Anzahl der empfangen Frames, die fehlerhaft waren

Aktionen

- | |
|--------------------------------|
| [1] = Inkrementiere V_r |
| [2] = Inkrementiere ErrorCount |

Abbildung 5.3 zeigt die Spezifikation des Empfängers in grafischer Form. Hierbei steht das Minuszeichen in IRCVD- dafür, daß TxNACK ausgeführt wird. Analog bedeutet das Pluszeichen in IRCVD+, daß TxACK ausgeführt wird.

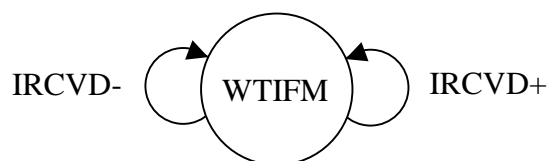


Abbildung 5.3

In der nachfolgenden Tabelle ist die vorige grafische Darstellung des endliche Automaten für den Sender nochmals in tabellarische Form zu sehen und um die Aktionen beim Zustandsübergang ergänzt worden:

Eingehendes Event:	IRCVD
Aktueller Zustand:	1
WTIFM	1

1 = Nicht P1: TxNACK; WTIFM
 P1 und P2: TxACK; WTIFM
 P0 und P1: LDATAind, TxACK, [1], [2]; WTIFM

Nachfolgend ist der Pseudo-Code für den Empfänger zu sehen, wie er in [Hals96] aus der vorstehenden Beschreibung abgeleitet wurde und aus dem die Steuerung des Empfängers in der Beispielanwendung entwickelt wurde:

```

program      IdleRQ_Secondary;
const       MaxErrorCount;
type        Events = IRCVD;
            States = WTIFM;
var         EventStateTable = array [Events, States] of 1;
            EventType : Events;
            PresentState : States;
            Vr, X, ErrorCount : integer;

procedure   Initialize;
procedure   LDATAind(X);
procedure   TxACK(X);
procedure   TxNACK(X);
procrdure  LERRORind;
function    P0 : boolean;
function    P1 : boolean;
function    P2 : boolean;

begin       Initialize; /* Zustandsvariablen und EventStateTable */
            repeat
                Warte auf das Eintreffen eines Events;
                EventType := type of event;
                case EventStateTable[EventType, PresentState] of
                1 :   begin X := N(S) from I-frame;
                        if (not P1) then TxNACK(X);
                        else if (P1 and P2) then TxACK(X);
                        else if (P0 and P1) then
                            begin LDATAind(X);
                                TxACK(X);
                                Vr := Vr + 1;
                            end;
                        else
                            begin ErrorCount := ErrorCount + 1;
                                if (ErrorCount = MaxErrorCount) then
                                    begin LERRORind;
                                        Initialize;
                                    end;
                            end;
                        end;
                end;
            until false;
end.
    
```

3 Das Demonstrationsapplet

Die erstellte Demonstration des „Stop and Wait“-Protokolls ist sowohl als Applet im Rahmen einer WWW-Seite als auch als Applikation lauffähig. Zwischen beiden Arten der Ausführung bestehen keine Unterschiede. Nach dem Start erscheinen drei Fenster, in denen die ganze Darstellung stattfindet. Dies bedeutet aber auch, daß der Darstellungsbereich für das Applet in einer WWW-Seite nicht genutzt wird und somit leer bleibt.

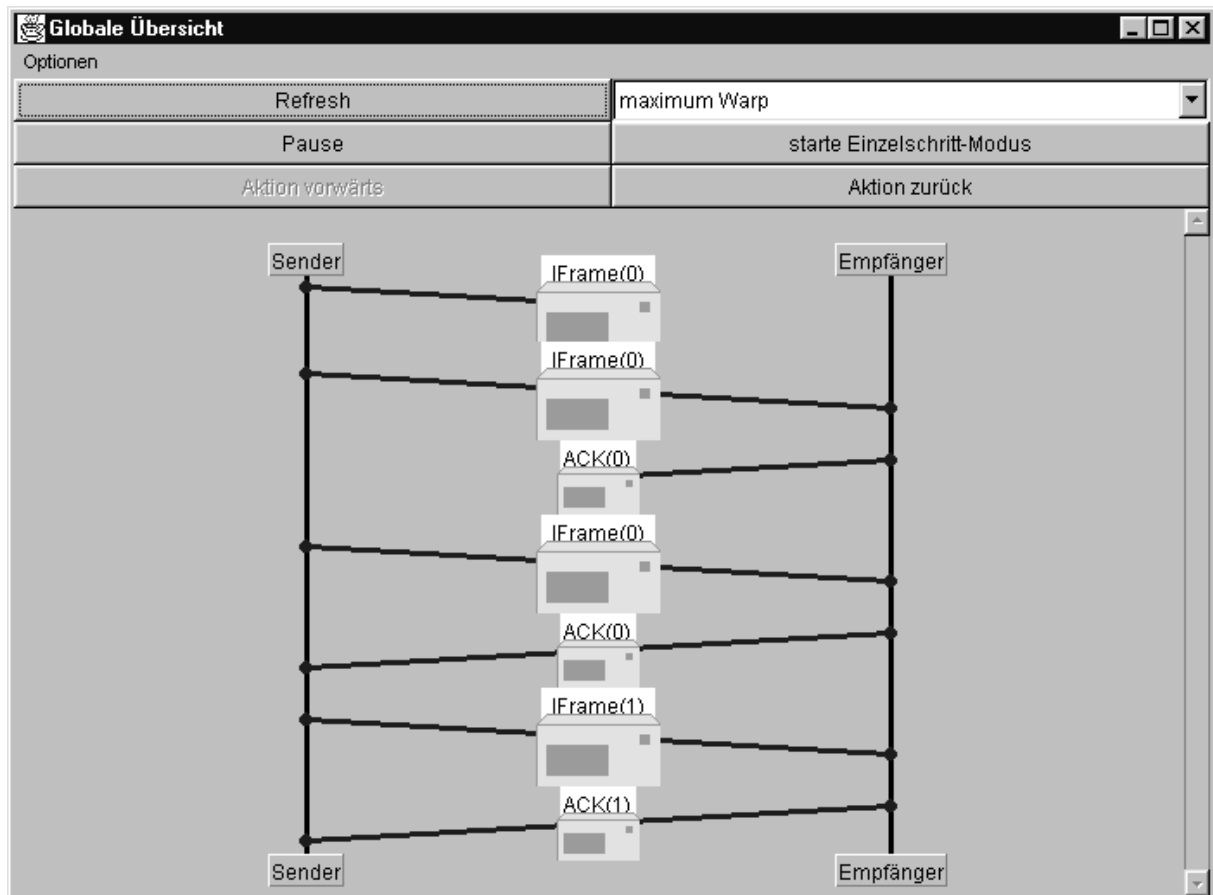


Abbildung 5.4

Die Fenster sind die in den vorigen Kapiteln vorgestellten Sammelfenster. Ein Sammelfenster des TSD-Baukastens und zwei des FSM-Baukastens. Letztere stellen den internen Ablauf des Senders bzw. des Empfängers dar. Das Sammelfenster des TSD-Baukastens visualisiert den Austausch der diversen Nachrichten zwischen Sender und Empfänger, auf implizite Art wird somit auch der Kommunikationskanal zwischen beiden dargestellt. Man kann somit sagen, daß dies die Übersichtsansicht des Kommunikationsablaufs ist und somit die höchste Ebene der Visualisierung repräsentiert.

In Abbildung 5.4 ist eine Momentaufnahme des Time-Sequence-Diagramms zu sehen. Auf der linken Seite wird der Sender dargestellt, auf der rechten Seite der Empfänger. Zwischen Sender und Empfänger wurden Nachrichten ausgetauscht. Vom Sender zum Empfänger werden die zu übermittelnden Daten bzw. Teile davon geschickt. Sie werden in der Darstellung mit IFrame bezeichnet und ihre jeweilige Nummer steht in Klammern dahinter. Der Empfänger antwortet mit Bestätigungen (ACK) oder Ablehnungen (NACK). Die

Nummer des IFrames auf den Bezug genommen wird, steht analog in Klammern hinter der jeweiligen Bezeichnung.

Um den Nachrichtenaustausch in der Beispielanwendung nicht eintönig zu gestalten, geht im Mittel jede fünfte Nachricht verloren, d.h. nur 80% der Nachrichten kommen durch. Welche Nachricht verloren geht, wird per Zufallsgenerator ermittelt. Der Visualisierungsablauf ist also jedesmal verschieden und nicht vorhersagbar, wie es bei einer Nachrichtenübertragung in der Realität auch der Fall ist.

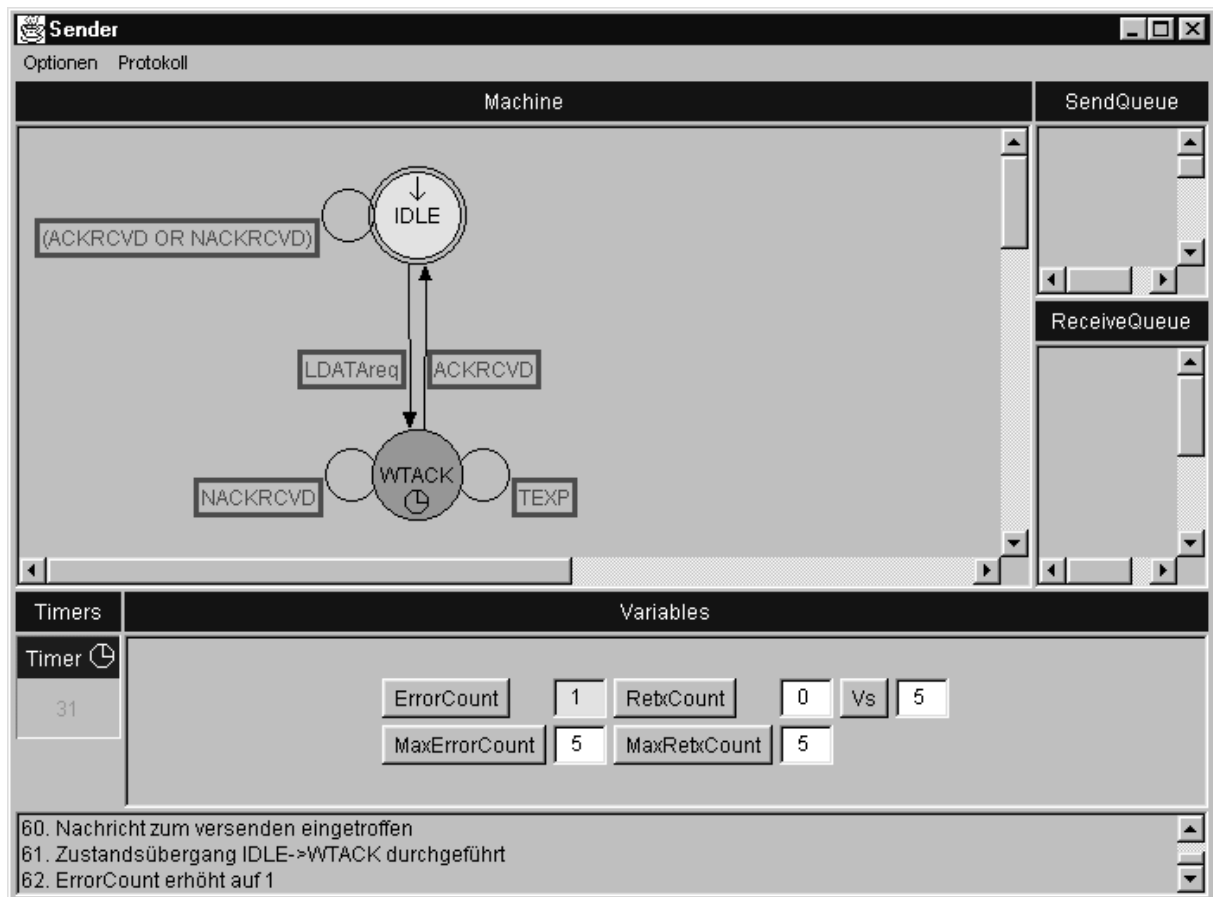


Abbildung 5.5

Das Fenster, das die Internas des Senders visualisiert ist in Abbildung 5.5 zu sehen. Wie man sehen kann, ist der dargestellte endliche mit dem zuvor beschriebenen vollständig identisch. Die vorigen Beschreibungen zum „Stop and Wait“-Protokoll können somit ohne Abstriche übernommen werden.

Rechts neben der Darstellung des Automaten sind die Warteschlangen zu sehen. Sie stellen die PHY-Schnittstelle dar, vergleiche dazu Abbildung 5.1. Die LS-Schnittstelle wird nicht explizit dargestellt. Allerdings kann der Benutzer die Ankunft von Übermittlungswünsche beim Sender interaktiv steuern. Dazu steht das Protokoll-Menü zur Verfügung. Der Benutzer kann Übermittlungswünsche selbst einzeln erzeugen oder diese in einem bestimmten Zeittakt automatisch an den Sender schicken lassen. Damit kann der Benutzer zwischen einer endlos fortlaufenden Simulation des Protokolls wählen oder den Ablauf nach seinen Bedürfnissen

steuern. Er hat somit die Möglichkeit sich die Übertragung schrittweise anzusehen, d.h. jeden Übertragungswunsch einzeln, da der Simulationsablauf nach dessen Darstellung anhält. Außerdem kann er selber Fehlersituationen herbei führen, in dem er den Sender mit Sendewünschen überschwemmt.

Unterhalb der Automatendarstellung ist der zuvor beschriebene Timer und die Variablen zu sehen. Letztern beinhalten nicht nur die bei der Spezifikation des „Stop and Wait“-Protokolls beschriebenen Variablen, sondern stellen auch die Konstanten dar, die die maximale Anzahl an Fehlern und wiederholten Übertragungen angeben. Die Konstanten sind so angeordnet, daß ein direkter Vergleich mit den Variablen deren Wert sie begrenzen ins Auge fällt.

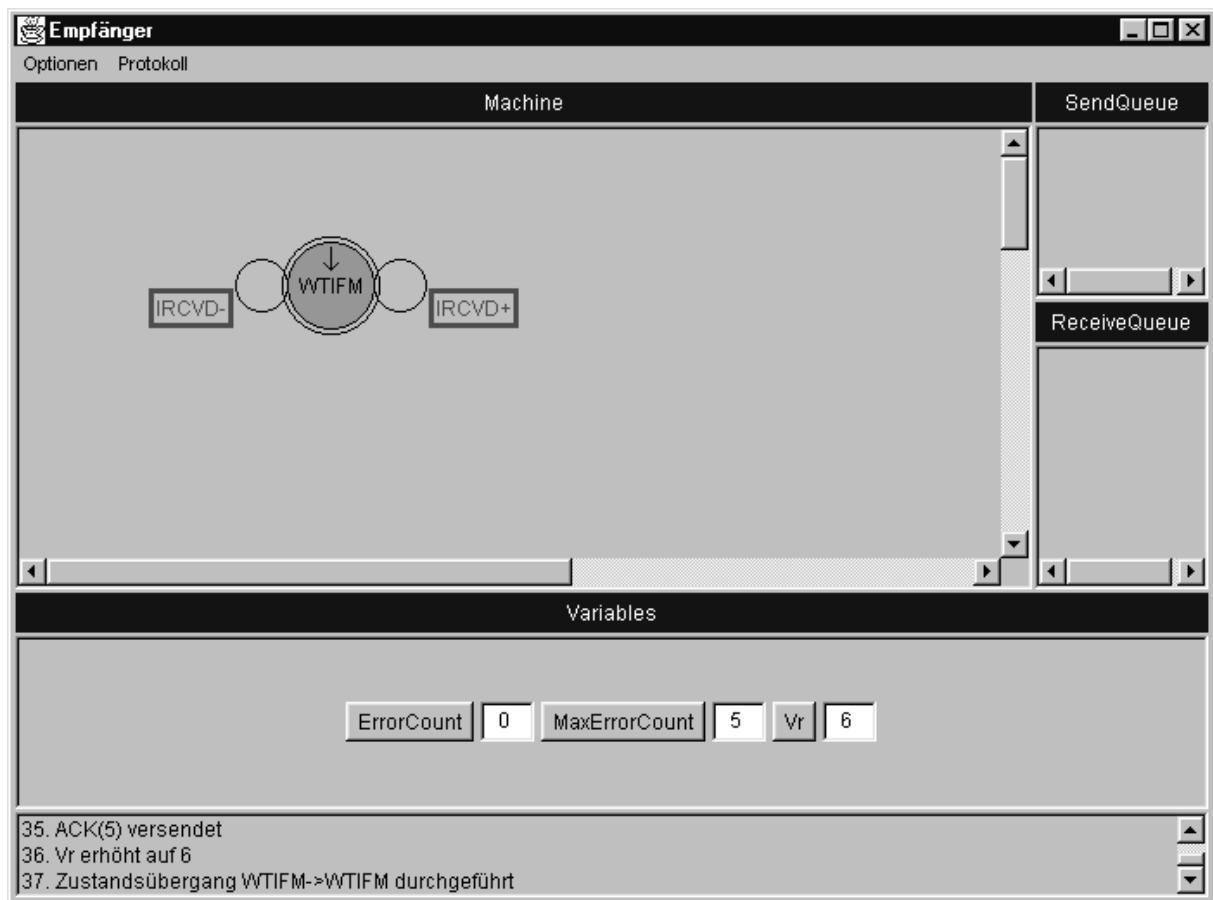


Abbildung 5.6

In Abbildung 5.6 ist die Visualisierung des Empfängers zu sehen. Für die Darstellung des Empfängers gelten, mit kleinen Einschränkungen, die gleichen Aussagen, die für die Darstellung des Senders gemacht wurden.

Auch hier ist die Automatendarstellung mit jener in der Spezifikation identisch. Die PHY-Schnittstelle wird analog als zwei Warteschlangen dargestellt, die rechts neben der Automatendarstellung zu sehen ist. Auch die Variablen und Konstanten sind zusammen unterhalb der Automatendarstellung zu sehen.

Es gibt allerdings zwei Detail, in denen sich die Darstellung des Empfängers zu der des Senders unterscheidet. Beim Empfänger gibt es laut Spezifikation keine Timer, deren

Darstellungsbereich wird somit von der Darstellung der Variablen mit benutzt. Außerdem wird beim Empfänger die LS-Schnittstelle überhaupt nicht dargestellt und der Benutzer hat hier im Gegensatz zum Sender auch keinen Einfluß auf sie.

Zum Abschluß sei noch angemerkt, daß für alle Komponenten im Sender- und Empfängerfenster Infotexte vorhanden sind, die durch anklicken mit der rechten Maustaste betrachtet werden können. Eine Ausnahme bildet hierbei allerdings die Statusbar, in der die jeweils ausgeführten Aktionen beim Sender bzw. Empfänger protokolliert werden.

4 Implementierungsaspekte

Bei der Implementierung des Demoapplets, wurde darauf geachtet, möglichst einfach und durch methodisches Vorgehen aus der Spezifikation des „Stop and Wait“-Protokolls, wie sie in [Hals96] gegeben ist, zu einer lauffähigen Anwendung zu kommen.

Zunächst wurde die Spezifikation des Senders und des Empfängers, die dort auch schon in Pseudo-Code gegeben waren, übernommen. Der Pseudo-Code ist zuvor im Abschnitt über den Sender bzw. den Empfänger vorgestellt worden. Wäre dieser nicht gegebene gewesen, hätte er zunächst aus einer in anderer Form gegebenen Spezifikation hergeleitet werden müssen.

Nach dem der Pseudo-Code für den Sender und den Empfänger vorliegen, kann daraus eine lauffähige Anwendung erzeugt werden. Ein Ablauf zur Anwendungserstellung für die Visualisierung des Protokolls könnte somit so aussehen:

1. Die gegebene Spezifikation des Protokolls die in Pseudo-Code vorliegt auf der Syntax von Java übersetzen, sofern diese nicht schon für den Pseudo-Code verwendet worden ist. Der entstandene Code wird in die run-Methode eines Threads eingefügt. Es entstehen also neue Klassen, die jeweils einen Unterklasse der Klasse Thread sind.
2. Alle Zuweisungen und sonstige Aktionen werden durch Methodenaufrufe ersetzt. Der durch den Methodenaufruf ersetzte Code wird in die jeweils geschaffene Methode ausgelagert.
3. In der erstellte Menge an Methoden werden jeweils zusätzlich der Code eingefügt, der die zur Steuerung der Visualisierung nötigen Events an die Komponenten des Baukastens ausgelöst. Was konkret in einer Methoden geschieht, hängt von der Bedeutung ab, die ihr über die jeweilige Protokollspezifikation zugeordnet ist.
4. Es wird eine Methode ergänzt, die den Aufbau der Darstellung übernimmt. Sie muß einmal am Anfang der run-Methode aufgerufen werden.

Der Ablauf muß sowohl für den Sender wie auch für den Empfänger durch geführt werden. Womit zwei verschiedene Klassen von Objekten entstehen. Damit wäre die Steuerung des FSM-Baukastens in der Anwendung erstellt.

Jetzt ist noch ein Objekt für die Steuerung der Darstellung des Time-Sequence-Diagramms zu erstellen. Dieses Objekt verwaltet gleichzeitig die Nachrichtenübertragung zwischen den Objekten die die Sender- und Empfängerdarstellung steuern und visualisiert diese sozusagen als „Nebeneffekt“. Stellt als einen Nachrichtenkanal zwischen den Kommunikationspartnern bereit. Um die Nachrichtenübertragung zu synchronisieren werden Warteschlangen

verwendet. Es sei darauf hingewiesen, dass diese nichts mit der grafischen Darstellung von Warteschlange zu tun haben.

Beide Kommunikationspartner legen ihre zu versendenden Nachrichten in einer Ausgangswarteschlange ab. Das Objekt, das die Steuerung des Time-Sequence-Diagramms übernimmt, holt diese im Polling-Verfahren ab und legt diese in die Eingangswarteschlange des jeweiligen Kommunikationspartners. Dieser kann dann die Nachricht verarbeiten, sobald er dazu bereit ist.

Wie schon erwähnt wurde, wird also auch intern in den Objekten, welche die Darstellung des Senders und des Empfängers steuern, eine Warteschlange verwendet um asynchron eintreffende Events von den Grafikkomponenten, sowie vom Kommunikationspartner zu handhaben.

5 Erkenntnisse

Aus dem erstellten Anwendungsbeispiel konnten schon die ersten Erkenntnisse über die Fähigkeiten und Defizite des neuen Baukastens gewonnen werden. Dabei zeichnet sich schon die schlechte Einsetzbarkeit für große Protokolle ab. Hauptgrund dafür ist, daß der Betrachter der Visualisierung, bei der Vielzahl an Komponenten die er gleichzeitig im Überblick behalten sollte, auch schon bei dem recht kleinen Protokoll im Anwendungsbeispiel teilweise überfordert ist. Dies ist vor allem der Fall, wenn mehrere Aktionen gleichzeitig ablaufen. Da es aber Aktionen gibt, die quasi asynchron zum Rest der Visualisierung auftreten können, wie etwa der Ablauf von Timern oder das Eintreffen von Nachrichten, ist dies nicht ganz zu vermeiden.

Dieses Problem wird zwar durch das Protokollieren der Aktionen in der Statusbar abgemildert, dies wird aber bei größeren Protokollen nicht mehr ausreichen um dem Betrachter einen vollständigen Überblick zu ermöglichen. Auch die Betrachtung des Time-Sequence-Diagrammes, das den vorangegangenen Ablauf ebenfalls noch darstellt, kann dabei nur begrenzt hilfreich sein, da es nur die globale Sicht zeigt und deshalb auf Detailfragen keine ausreichende Antwort geben kann.

Für die Visualisierung kleiner Protokolle ist der neue Baukasten allerdings trefflich einsetzbar, vor allem wenn der Benutzer mit ihm interaktiv arbeitet und somit praktisch spielerisch ein Verständnis für das visualisierte Protokoll gewinnen kann. Der Hauptvorteil im Gegensatz zum alten Baukasten ist hierbei, daß der Betrachter sich den ihn interessierenden Teil bzw. Abstraktionsgrad selber auswählen kann. Somit kann er jede für sein Verständnis nötige Detailinformation bekommen.

Nachdem die verschiedenen Komponenten des neuen Baukastens in den vorangegangenen Kapiteln bzw. deren Zusammenspiel in diesem Kapitel vorgestellt wurde und die Implementierung des Demonstrationsapplets beleuchtet worden ist, somit also ein vollständiger Überblick aus Benutzersicht über den neuen Baukasten vorliegt, wird nun auf die Implementation der Baukastenteile eingegangen. Der Schwerpunkt liegt hierbei natürlich auf den in dieser Studienarbeit erstellten Komponenten.

Kapitel 6:

Implementationsbeschreibung des neuen Baukastens

Nachdem in Kapitel 2 die Bestandteile des TSD-Baukastens bzw. in Kapitel 4 die des FSM-Baukastens vorgestellt wurden, und in Kapitel 5 gezeigt wurde, wie der neue Baukasten in einer Anwendung eingesetzt werden kann, wollen wir uns nun auf die Aspekte seiner Implementierung konzentrieren. Dieses Kapitel richtet sich also im Gegensatz zu den vorhergehenden, die sich vor allem an den Baukastenbenutzer wendeten, hauptsächlich an diejenigen, die den Baukasten weiterentwickeln wollen und sich deshalb in seine Implementierungsdetails einarbeiten müssen.

Im folgenden werden wichtige Strukturen und Algorithmen vorgestellt. Dabei wird allerdings nicht bis auf Codeebene hinuntergegangen, sondern es werden, von den Details abstrahierend, Verfahren und Zusammenhänge erläutert. Wenn detailliertere Informationen über die Implementierung benötigt werden, sei auf die mit JavaDoc erstellten WWW-Seiten und vor allem auf den Quellcode verwiesen. Werden im folgenden Namen von Klassen aus dem Quellcode bzw. aus Java selbst verwendet, dann werden diese *kursiv* geschrieben, um dies kenntlich zu machen.

1 Buffering

Beim Visualisieren eines dynamischen Ablaufs in einem Grafikbereich, tritt ein Problem mit der Darstellung auf, das Bild beginnt zu flackern. Dieses Phänomen ist dadurch zu erklären, daß das Bild zunächst gelöscht werden muß, bevor es neu gezeichnet werden kann. Ein direktes Zeichnen ist nicht möglich, da z.B. beim Verschieben eines Zustandes, dessen alte Darstellung zunächst mit dem von ihm verdeckten Hintergrund überschrieben werden müßte, was allgemein nicht ohne weiters möglich ist oder ebenfalls zu unerwünschten Nebeneffekten führen kann.

Eine allgemein anwendbare und einfache Lösung des Problem ist das Buffering. Dabei wird nicht direkt auf den Bildschirm gezeichnet, sondern in einen Puffer. Erst das fertig erstellte Bild wird durch das Kopieren des Puffers auf den Schirm gebracht. Dem Betrachter bleiben somit unerwünschte Zwischenschritte beim erstellen der Grafik verborgen, wie etwa das gelöschte Bild, das den Eindruck des Flackerns hervorgerufen hat.

Das Buffering wird deshalb bei allen Darstellungsbereichen eingesetzt, in die gezeichnet wird. Dies sind die Darstellungen der endlichen Automaten, der Timer und der Warteschlangen.

2 Automatendarstellung

Im folgenden ist die Rede von den Elementen der Darstellung und nicht von abstrakten Begriffen aus der Automatentheorie. Damit keine begriffliche Unklarheit auftritt, sei darauf hingewiesen, daß dies alles Klassen im objektorientierten Sinne sind, was im folgenden nicht explizit erwähnt wird.

Die Bestandteile der Automatendarstellung werden unter Zuhilfenahme der Vektorrechnung gezeichnet. Auch deren Anordnung wird mit deren Hilfe bestimmt. Auf Implementierungsfragen, die mit der Vektorrechnung zusammenhängen wird im folgenden nicht näher eingegangen. Sollten sich diesbezüglich Fragen ergeben, wird auf die einschlägige Literatur verwiesen, wie etwa [MeWi93].

Die Darstellung des endlichen Automaten wird von der *MachineCanvas* koordiniert. Da sich Zustände, Übergänge und Prädikate selbst in jeden Grafikbereich zeichnen können der ihnen mitgeteilt wird, was sie somit universell einsetzbar macht, muß dies nur veranlaßt werden. Diese Aufgabe obliegt der *MachineCanvas*, in die gleichzeitig auch gezeichnet werden soll.

Außer der Koordinierung der Darstellung hat die *MachineCanvas* noch zwei weitere wichtige Aufgaben zu erfüllen. Sie nimmt zentral die Events, die für die Komponenten der Automatendarstellung bestimmt sind entgegen und reicht sie an diese weiter. Noch wichtiger aber ist, daß sie für diese den Infotext in einem extra Fenster ausgibt und das Verschieben der Zustände durch den Anwender ermöglicht. Kurz gesagt, sie verwaltet die Events die von der Maus kommen.

Um dies leisten zu können, wird sie von den Zuständen, Übergängen und Prädikaten unterstützt, in dem diese feststellen können, ob ein Punkt innerhalb ihrer Darstellung liegt. Damit dies einleuchtender wird, soll kurz der Fall besprochen werden, in dem der Anwender einen Infotext zu einer Komponente aufrufen will.

Wenn der Anwender mit der rechten Maustaste auf die ihn interessierende Komponente klickt, wird ein Event erzeugt, daß die Position des Mauspeils enthält. Die *MachineCanvas* verarbeitet dieses Event. Sie fragt alle Zustände, Prädikate und Übergänge der Reihe nach, ob die Position in ihrer Darstellung liegt. Die erste Komponente die darauf mit ja antwortet, wird nach ihrem Infotext gefragt, der dann in einem separaten Fenster ausgegeben wird. Auf analoge Weise wird vorgegangen, wenn ein Zustand verschoben werden soll.

2.1 Zustände

Zustände werden um die ihnen zugeordnete Position gezeichnet, diese gibt also ihren Mittelpunkt an. Um sich selbst darstellen zu können, muß von ihnen die Größe ihres Namens, einem *String*, im Grafikbereich ermittelt werden können, da dieser nicht konstant ist.

Die wichtigste Aufgabe die den Zuständen zukommt ist die Anordnung der ihnen zugeordneten Übergänge herbeizuführen. Dazu geben sie auf Anfrage eines Übergangs dessen „Schnittpunkt“ mit ihrer Darstellung zurück. Um dies möglich zu machen, muß der Übergang den zweiten Zustand angeben mit dem er Verknüpft ist. Wenn dies ein anderer Zustand ist, wie der Zustand selbst, dann wird der „Schnittpunkt“ aus den Positionen der beiden Zustände berechnet. Ist es der gleiche Zustand, wird eine Richtung bezüglich der Position des Zustands ermittelt, in die der Übergang als Schlinge dargestellt werden soll.

Da jeder Zustand die an ihm endenden Übergänge kennt, kann der Zustand erkennen, wenn mehrere Übergängen zwischen ihm und einem bestimmten anderen Zustand existieren. In diesem Fall werden die „Schnittpunkte“ verschoben, damit eine Darstellung ohne Überlappungen erreicht wird. Eine analoge Aussage gilt ebenfalls für mehrere Übergänge, die am gleichen Zustand beginnen und enden.

2.2 Automatische Zustandsanordnung

Für die automatische Anordnung der grafischen Darstellungen der Zustände, wird der „Mincut“-Algorithmus verwendet, wie er in [Bait98] beschrieben worden ist. Der Begriff „Mincut“ leitet sich von minimaler Schnitt ab. Dies bedeutet, daß die Zustände auf eine Art angeordnet werden, die Zustände mit vielen Verbindungen (Übergängen) möglichst benachbart legt. Wenn man jedem Zustand ein begrenztes Gebiet zuordnet, folgt daraus eine minimale Anzahl an Überkreuzungen von Verbindungen und Gebietsgrenzen.

Zunächst muß eine sogenannte Netzliste erstellt werden. Dies ist eine Datenstruktur, die die Verbindungen zwischen den Zuständen enthält, also die Übergänge, wobei Schlingen nicht berücksichtigt werden. Anschließend wird die zur Verfügung stehende Darstellungsfläche durch einen Schnitt in zwei Teilflächen A und B aufgeteilt.

Anschließend kommt der „Mincut“-Algorithmus zum Einsatz.:

1. Man ordnet einen beliebigen Zustand der Teilfläche A zu.
2. Anhand der Netzliste wird die mittlere Anzahl der Verbindungen zwischen den zu platzierenden Zuständen als Trennwert definiert.
3. Bestehen zwischen dem Anfangsobjekt und dem als nächsten zuzuordnenden Objekt laut Netzliste mehr Verbindungen als der Trennwert, dann wird es ebenfalls A zugeordnet sonst B.
4. Alle weiteren Zustände werden den bereits einer der beiden Teilflächen A oder B zugehörigen Zuständen zugeordnet. Dies ist jeweils die Zustandsmenge, zu denen sie die zahlreicheren Verbindungen besitzen. Bei gleichen Verbindungszahlen wird ein Zustand so zugeordnet, daß eine möglichst gute Gleichverteilung erreicht wird.

Wenn alle Zustände zugeordnet sind, erfolgt eine Rekursion.

5. Die beiden Teilflächen A und B werden mit je einem Schnitt versehen, der quer zum bisherigen liegt. Die jeweils zugeordneten Teilmengen von Zuständen werden unabhängig von einander auf die jeweils neuen Teilflächen aufgeteilt.

Die Rekursion wird solange durchgeführt, bis jede Teilfläche nur noch je ein Zustand enthält. Die ist somit das Abbruchkriterium. Anschließend wird die Position der Zustände auf die Mitte der für sie ermittelten Teilfläche gesetzt. Damit ist die Ausrichtung der Zustände vollzogen.

2.3 Übergänge

Übergänge sind im Normalfall einfache Linien, die die Zustände verbinden. Den Anfangs- und den Endpunkt bekommen sie von den ihnen zugeordneten Zuständen mitgeteilt. Außerdem wird an einem Ende mit Hilfe der Vektorrechnung ein Pfeilspitze gezeichnet.

Dies wäre weiter nicht bemerkenswert, wenn sie sich ihre Darstellung nicht automatisch daran anpassen würde, wenn sich der gleiche Zustand an ihrem Anfang und Ende befindet. In diesem Fall werden sie als Schlingen dargestellt, deren Lage sie vom diesem Zustand erfahren.

Übergänge sind eine Unterklasse der Klasse *Thread*, damit sie einen Zustandsübergang selbständig animieren können. Die Animation besteht darin, daß ihre Darstellung in zwei Teilen mit verschiedenen Farben gezeichnet wird. Der prozentuelle Anteil der jeweiligen Farbe wird dabei variiert, wobei eine schrittweise Verschiebung von der einen zur anderen Farbe stattfindet.

Eine wichtige Aufgabe die den Übergängen obliegt, ist die Bestimmung der Positionen, an denen die ihnen zugeordneten Prädikate gezeichnet werden sollen. Dafür wird ebenfalls auf die Vektorrechnung zurückgegriffen.

2.4 Prädikate

Prädikate benutzen einen sogenannten *ColoredString* um sich zeichnen zu können. *ColoredStrings* sind ebenfalls in der Lage sich selbst zu zeichnen und stellen normalen Strings dar, die allerdings stückweise Einfärbungen besitzen. Prädikate ergänzen die Darstellung des *ColoredStrings* lediglich noch um eine farbige Einrahmung, die ihren Wahrheitswert widerspiegelt.

Die Position an der ein Prädikat gezeichnet wird, ist nicht fest einstellbar, sondern muß von mal zu mal neu angegeben werden. Hierbei muß außerdem immer spezifiziert werden, auf welche Stelle der Prädikatdarstellung sich die Position bezieht. Zur Auswahl stehen die vier Ecken der Einrahmung. Mit Hilfe dieser Option können die Übergänge eine überlappungsfreie Darstellung der Prädikate erzeugen.

Prädikate sind eine Unterklasse der Klasse *Thread*, da sie einen Prädikatcheck animieren können müssen. Dazu werden eine Reihe von *ColoredStrings* jeweils für eine kurze Zeitspanne angezeigt. Außerdem blinkt der Rahmen in den zwei Farben, die sonst die beiden Wahrheitswerte repräsentieren.

Erweiterte Prädikate funktionieren auf ähnliche Art wie die normalen Prädikate. Die einzige Ausnahme bildet dabei der *ColoredString*, der für die Darstellung verwendet wird. Die erweiterten Prädikate ermitteln diesen nämlich automatisch mit Hilfe von Operatoren, die einen logischen Ausdruck modellieren. Dabei wird der zu zeichnende *ColoredString* von den atomaren Operatoren bis hin zu dem Operator zusammengesetzt, der den Gesamtausdruck repräsentiert und der als einziger dem erweiterten Prädikat bekannt ist.

Aus dem zuvor gesagten ist erkennbar, daß bei erweiterten Prädikaten keine Animation mit einer Reihe von vorgegebenen *ColoredStrings* möglich ist. Die Animation des Strings wird viel mehr durch den „Operatorenbaum“ selber erzeugt, indem der Reihe nach unterschiedliche *ColoredStrings* dem Prädikat geliefert werden.

3 Variablen

Eine Variable enthält außer dem Wiesen über ihren Namen, ihrem Wert und ihrem Infotext praktisch keine weitere Funktionalität. Abgesehen davon, daß sie ein Event erzeugt, wenn ihr Name oder ihr Wert geändert wird.

Die Visualisierung wird somit ganz dem Variablenbereich überlassen. Dieser übernimmt allerdings nur die Anordnung der Darstellungen der verschiedenen Variablen. Deren Darstellung selbst überläßt er wiederum einem Button und einem Textfeld. Der Button stellt den Namen der Variablen dar und ist für die Anzeige des Infotextes verantwortlich.

Außerdem kann er ein Event erzeugen, in dem der Wert der Variablen enthalten ist. Dieses Event wird ausgelöst, wenn der Anwender mit der linken Maustaste auf den Button klickt.

Das Textfeld ist für die Darstellung des Wertes der Variable verantwortlich. Wird der Wert der Variable geändert, färbt es seinen Hintergrund für einige Sekunden gelb ein. Um die Zeitspanne zu überwachen, wird der Taktgeber verwendet, der im nächsten Abschnitt näher vorgestellt wird.

4 Timer

Die Timerdarstellung selber ist, wie man es vielleicht erwarten könnte, keine Unterklasse der Klasse *Thread*. Sie ist lediglich für die Darstellung verantwortlich. Um trotzdem einen automatisch ablaufenden Timer darstellen zu können, benutzt sie einen reinen Taktgeber, der nach einem einstellbaren Intervall ein Event auslöst.

Das Event des Taktgebers bewirkt eine erneute Zeichnung der Timerdarstellung, allerdings mit dem aktualisierten Wert. Der Taktgeber kann auch unabhängig vom der Timerdarstellung verwendet werden, er hat dann allerdings nicht zwingend etwas mit der Visualisierung zu tun.

5 Warteschlangen

Warteschlangen sind vom Prinzip her ähnlich aufgebaut wie die Automatendarstellung. Sie koordinieren die Darstellung von Nachrichten, die sich allerdings selber zeichnen können. Außerdem verwalten sie die Events von der Maus und erzeugen beispielsweise die zu den Nachrichten gehörenden Fenster, mit den jeweiligen Infotexten.

Es gibt allerdings einen wichtigen Unterschied zu der Automatendarstellung. Die Animation von eintreffenden bzw. ausgehenden Nachrichten wird nicht von der Nachricht selber, sondern von der Warteschlange erstellt. Die Animation wird durch ständiges verändern der Position der Nachricht und jeweiligem neu zeichnen der Warteschlange erzeugt. Die Warteschlangen sind somit eine Unterklasse der Klasse *Thread*.

6 Einbindung des alten Baukastens

Der alte Baukasten wurde, wie schon in Kapitel 2 bei dessen Beschreibung festgestellt worden ist, praktisch unverändert in den neuen Baukasten übernommen. Dies hat die Konsequenz zur Folge, daß der neue Baukasten quasi zweigeteilt ist, nämlich in die alten und in die neuen Komponenten.

Eine bessere Verschmelzung bzw. eine Vereinheitlichung der beiden Teile des Baukastens erscheint wünschenswert, war aber im Rahmen dieser Studienarbeit nicht durchzuführen. Wer sich deshalb für eine detaillierte Implementationsbeschreibung des alten Baukastens interessiert, sei auf die Ausarbeitung zur Studienarbeit von Peter W. Schurr [Schu97] verwiesen.

7 Weiterentwicklung

Da im neuen Baukasten auf eine leichte Änderbarkeit das Augenmerk gelegt wurde, ist er streng objektorientiert entworfen worden, was aus den vorigen Erläuterungen schon hervorgegangen sein sollte. Jede Komponente kann somit an geänderte Anforderungen leicht

angepaßt werden. Auch der komplette Austausch von Komponenten sollte ohne Probleme möglich sein.

Dies ist vor allem bei der Darstellung des endlichen Automaten interessant. Beispielsweise könnte für die Zustände eine komplett andere Darstellung implementiert werden, ohne das an anderen Bestandteilen der Automaten-Darstellung eine Änderung vorgenommen werden muß. Dies ist durch einfaches Implementieren einer neuen Zustandsklasse möglich. Sogar eine noch drastischere Umstellung, wie z.B. die Ersetzung der Darstellung der endlichen Automaten durch die Darstellung von Petri-Netze, ist ohne Schwierigkeiten implementierbar.

Da nun alle Aspekte des neuen Baukastens beleuchtet wurden, kann nun im nächsten Kapitel auf Verbesserungs- und Erweiterungsvorschläge eingegangen werden. Dabei wird vor allem auf konzeptionelle Punkte eingegangen. Kleine Detailverbesserungen werden nur am Rande erwähnt und es wird deshalb auch nicht auf Vollständigkeit geachtet.

Kapitel 7: Verbesserungs- und Erweiterungsvorschläge

1 Verbesserungen am Grafikbaukasten

Ein großes Manko des neuen Baukastens, ist die nicht vollständig gegebene Anpassung an umfangreiche Protokolle. Da noch keine Erfahrungen mit diesen vorliegen, ist noch nicht ganz klar wo die Schwierigkeiten in allen Einzelheiten liegen, daß Probleme existieren ist aber abzusehen, da z.B. Teile des Automaten nicht zu sehen sein werden und auch die Zahl der Variablen, Timer usw. steigen wird und damit die Darstellung für den Betrachter unübersichtlich wird.

Somit ist klar, daß ergänzende Konzepte für eine übersichtliche Darstellung zu erarbeiten sind, da der Benutzer durch die verschiedenartigen Informationen leicht überfordert wird. Dieser Effekt ist schon in der noch recht kleinen Beispielanwendung für das "Stop and Wait" Protokoll ersichtlich. Die Zahl der vom Menschen gleichzeitig aufnehmbaren und verarbeitbaren Eindrücke ist, wie man leider festgestellt hat, auf die doch recht kleine Zahl von ungefähr sieben Dingen beschränkt. Die Protokollierung in der Statusbar ist dafür wohl nur eine unzureichende Lösung.

An diesen Bereich knüpft direkt die noch fehlende übersichtliche Anordnung der Elemente im endlichen Automaten an. Die Zustände des endlichen Automaten können zwar übersichtlich angeordnet werden, was für kleine Automaten auch zu einer weitgehend überlappungsfreien bzw. zumindest übersichtlichen Darstellung der Übergänge und Prädikate führt, dies ist in größeren Automaten allerdings keineswegs zu erwarten.

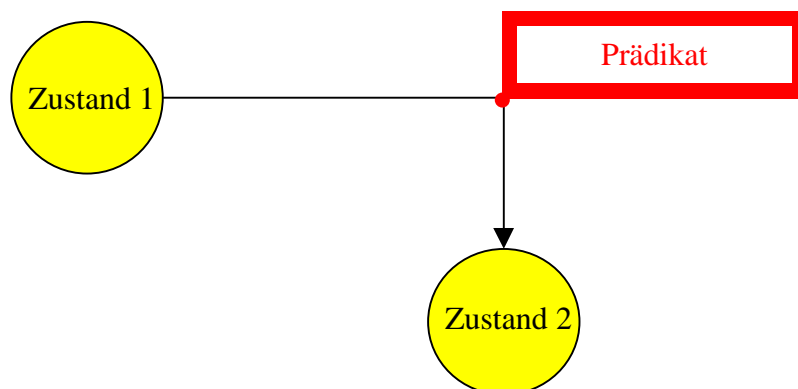


Abbildung 6.1

Es wäre nötig die Darstellung des endlichen Automaten mit Algorithmen für eine bessere Platzierung der einzelnen Elemente zu erweitern. Dabei sind verschiedenartige Probleme zu lösen. Wie etwa sollen Übergang mit identischem Anfangs- und Endzustand (Schlingen) an diesem Zustand angeordnet werden oder wie sollen Übergänge verlaufen, an welcher Stelle sollen Prädikate angezeigt werden und wie soll deren Verknüpfung zu einem Übergang angezeigt werden (Abbildung 6.1).

Dies alles hätte allerdings den Rahmen der Studienarbeit gesprengt und mußte somit vernachlässigt werden oder mit einfachsten Algorithmen implementiert werden. Zu erwähnen ist dabei noch, daß für ähnliche Problemstellungen schon interessante und erprobte Algorithmen aus anderen Bereichen, etwa dem Elektro-CAD, existieren. Beispiele dafür sind etwa Algorithmen für die "Plazierung" der Zustände und das "Routen" der Übergänge.

Die verschiedenen Bausteine des Baukastens könnten noch komplett auf Java Bean Technologie umgestellt werden. Damit könnte eine leichtere Einbindung in andere interaktive Programme erreicht werden und auf zukunftsweisende Programmier Techniken gesetzt werden. Die Bausteine wurden zwar schon in diese Richtung implementiert, wie an der vorgenommen Eventbasierung zu erkennen ist, aber sie wurden noch nicht auf diesen Aspekt getestet und teilweise sind diesbezüglich auch noch konzeptionelle Ergänzungen nötig.

2 Konzeptionelle Erweiterungen

Aus den vorhergehenden Kapitel, vor allem aus der Beschreibung der Beispielanwendung für das "Stop and Wait" Protokoll, sollte ersichtlich geworden sein, daß die leichte Einsatzbarkeit des Baukastens in der Praxis durch die noch fehlende Separierung bzw. Implementierung der Logikebene stark behindert wird. Abgesehen von der vereinfachten Erstellung von Anwendungen, seinen stellvertretend noch zwei Aspekte genannt.

Die Unterstützung einer grafisch veranschaulichten Undo-Funktionalität wird im neuen Baukasten nur unzureichend geboten. Allerdings müssen hierzu erst noch die Konzepte auf der Logikebene erarbeitet werden, um die Anforderungen für die Grafikebene klar zuerkennen und spezifizieren zu können.

Eine automatische Erstellung einer Dokumentation oder das Aufzeichnen und wieder Abspielen eines Ablaufs sind eine interessante Ergänzungen für die noch zu implementierende Logikebene.

Ein anderer problematischer Punkt ist der TSD-Baukasten. Er ist nicht objektorientiert programmiert und nicht eventbasiert anzusteuern. Noch kritischer ist allerdings, daß er nicht genügend Statusinformationen nach außen gibt. Kurz gesagt, der TSD-Baukasten ist eine nach außen abgeschlossener Block, somit stehen der neue und der alte Baukastenteil nun quasi unabhängig gegenüber. Leider hat dies die unangenehme Konsequenz, daß viele wünschenswerte Funktionen, wie etwa eine globale Undo-Funktionalität, eine globale Zeitbasis und eine globale Ablaufdokumentation nicht oder nur unzureichend implementiert werden können. Es wäre wünschenswert, den TSD-Baukasten zu überarbeiten und mit dem FSM-Baukasten zu verschmelzen, wobei die Trennung zwischen Logik- und Grafikebene auch dort strikt durchzuhalten ist.

Zusammenfassung

Die Studienarbeit ist im Themengebiet der Visualisierung von Protokollen anzusiedeln, zu dem im Rahmen des HiSAP-Projektes (früheres ProtoVis-Projekt) schon verschiedene Konzepte in diversen Arbeiten erprobt und verschiedenartig erweitert wurden. Ein Hauptaugenmerk wurde dabei auf den Einsatz in der Lehre gelegt.

Zielsetzung dieser Studienarbeit war es, den Visualisierungsbaukasten für Time-Sequence Diagramme von Peter W. Schurr, um eine Repräsentation für die dort nicht näher darstellbaren Prozesse zu ergänzen. Für diesen Zweck sollten endliche Automaten verwendet werden, welche um Darstellungen von Timern, Variablen und Warteschlangen ergänzt werden mußten. Daraus ergab sich eine Sammlung von einzelnen Bausteinen aus denen je nach Bedarf eine Darstellung zusammengebaut werden kann. Der neue Baukasten wurde aus diesem Grund streng objektorientiert gehalten.

Für den Baukasten sollte weiterhin die Sprache Java verwendet werden, da diese auf beinahe jeder Plattform ausführbare Programme erzeugt und vor allem die Programme in den gängigen WWW-Browsern lauffähig sind. Damit ist gewährleistet, daß die erstellten Anwendungen, z.B. die Demonstration eines Protokolls, überall eingesetzt werden können und nicht zuletzt überall gleich präsentiert werden. Etwa die Verwendbarkeit bei Vorträgen einer Tagung, ohne sich Gedanken machen zu müssen, ob und wenn ja wie die Präsentation darstellbar ist, stand dabei im Vordergrund.

Um die Fähigkeiten des neuen Baukastens zu veranschaulichen, wurde eine Beispielanwendung erstellt, die das "Stop and Wait" Protokoll visualisiert und auch eine Simulation dieses Protokolls ermöglicht. Durch die Erstellung der Beispielanwendung wurden schon einige erste Erkenntnisse gewonnen, wo noch Verbesserungs- und Erweiterungsmöglichkeiten des Baukastens bestehen.

Literaturverzeichnis

- [Schu97] Peter W. Schurr
Visualisierung von Lehrinhalten mittels Java Applets
Studienarbeit Nr. 1608
Universität Stuttgart, 1997
- [Kons98] Papoulidis Konstantinos
**Erweiterung des Java-Baukastens zur Visualisierung von Protokollen:
Weiterentwicklung der Benutzerschnittstelle, Undo-Funktionalität**
Studienarbeit Nr. 1670
Universität Stuttgart, 1998
- [Gort98] Ralph Gorth
**Erweiterung des Java-Baukastens zur Visualisierung von Protokollen:
Erweiterung des Baukastens zur Visualisierung von SDL-Konstrukten**
Studienarbeit Nr. 1671
Universität Stuttgart, 1998
- [Tane97] Andrew S. Tanenbaum
Computernetzwerke
Prentice Hall, 1997
- [Hals96] Fred Halsall
Data Communications, Computer Networks and Open Systems
Fourth Edition
Addison-Wesley Publishers Ltd., 1996
- [Turn93] Kenneth J. Turner
**Using formal description techniques
An Introduction to Estelle, Lotos and SDL**
University of Stirling
John Wiley & Sons Ltd., 1993
- [Hoge89] Dieter Hogrefe
**Estelle, Lotos und SDL
Standard-Spezifikationsprachen für verteilte Systeme**
Springer Verlag, 1989
- [MeWi93] Gerhard Merziger, Thomas Wirth
Repetitorium der Höheren Mathematik
Binomi Verlag, 1993
- [Bait98] Prof. Dr. U.G. Baitinger
Vorlesungsskript: Entwurfsautomatisierung
Universität Stuttgart, 1998

- [GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
Design Patterns
Elements of Reusable Object-Oriented Software
Addison-Wesley Publishers Ltd., 1995
- [Flan98] David Flanagan
Java in a nutshell
Deutsche Ausgabe für Java 1.1
O'Reilley, 1998
- [CoHo97] Gary Cornell, Cay S. Horstmann
Core Java 1.1
Volume I-Fundamentals
Sun Microsystems Press / Prentice Hall, 1997
- [CoHo98] Gary Cornell, Cay S. Horstmann
Core Java 1.1
Volume II-Advanced Features
Sun Microsystems Press / Prentice Hall, 1998

Erklärung

Ich versichere, daß ich diese Arbeit selbständig verfaßt und nur die angegebenen Hilfsmittel verwendet habe.

Ostfildern, den

(Jürgen Rau)

Ein Musterexemplar liegt bei der die Arbeit entgegennehmenden Stelle zur Einsicht aus.