Prüfer: Prof. Dr. Rothermel

Betreuerin: Dr. Cora Burger

Begonnen am: 1.11.1996

Beendet am: 30.4.1997

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

Studienarbeit Nr. 1608

Visualisierung von Lehrinhalten mittels Java Applets

Peter W. Schurr

Kurzfassung

Die Verbesserung der Lehre durch den Einsatz von Rechnern ist ein relativ neues Anwendungsfeld im Bereich der Informatik. Um Erfahrungen in diesem Anwendungsfeld zu sammeln, wurde am Lehrstuhl für Verteilte Systeme der Universität Stuttgart eine Studienarbeit mit dem Titel „Visualisierung von Lehrinhalten mit JAVA Applets“ ausgegeben.

Ausgangspunkt waren die in der Vorlesung „Kooperation in Verteilten Systemen“ behandelten Kommunikationsprotokolle. Die Funktionsweise dieser Protokolle sollte durch Animation der ausgetauschten Nachrichten leichter verständlich gemacht werden.

Damit die Animation nicht für jedes Protokoll extra programmiert werden muß, wurde im Rahmen dieser Studienarbeit ein Baukastensystem zur Animation von Nachrichten und Prozessen erstellt.

Die Implementierung geschah, wie in der Aufgabenstellung gefordert, in der Programmiersprache Java in Form von einigen Klassen und zugehörigen Methoden.

Neben dem Entwurf und der Implementierung dieses Baukastensystems wird in dieser Ausarbeitung auch die Verwendung des Baukastens beschrieben. Hierfür wurden für einige Protokolle aus der oben genannten Vorlesung JAVA Applets geschrieben, die mit Hilfe des Baukastens den Ablauf der Protokolle animiert darstellen.

Inhaltsverzeichnis

Kapitel 1
Einleitung 1
1.1 Motivation 1
1.2 Aufgabenstellung 1
1.3 Überblick 2

Kapitel 2
Anforderungsanalyse 3
2.1 Entwurfsszenarium 3
2.2 Umfeld des Baukastensystems 4
2.3 Berücksichtigung von unzuverlässigen Kommunikationskanälen. 4
2.4 zentrale und dezentrale Funktionalitäten 5
2.5 Spezifikation der nötigen Funktionalitäten 5

Kapitel 3
Entwurfsentscheidungen 7
3.1 Abfrage- versus Erweiterungsansatz 7
3.2 Playback versus „Trial and Error“ History-Arten 9
3.3 Überlegungen zur Darstellung 10
3.4 Unterstützung von Nachrichtenmanipulationen 12
3.5 Frühe Entwurfsphase und erster Prototyp 13

Kapitel 4
Implementierung des Baukastens 14
4.1 Gesamtarchitektur des Baukastens 14
4.2 Sichtbare Graphik-Elemente 15
4.2.1 Das Display-Objekt 16
4.2.2 Die Scrollbar 16
4.2.3 Das Steuer-Panel 17
4.3 Verwendung von Images 19
4.3.1 Das statischeObjekteImage 19
4.3.2 Das antiFlackerImage 21
4.3.3 Sonstige kleinere Images 22
4.4 Schnittstellen zum Java Laufzeitsystem 24
4.4.1 Zeitsystem des Baukastens 24
4.4.2 Threads und deren Notwendigkeit 25
4.4.3 Eventhandling 26
4.4.4 Die Paint() und Update() Laufzeitmethoden 26
4.5 Die Handhabung von Nachrichten 27
4.5.1 Unterstützte Nachrichtentypen 27
4.5.2 Schritte einer Nachrichtenübermittlung 28
4.5.3 Methoden-Wrapping und Overloading 30
4.5.4 Rückgabewerte der Nachrichten-Methoden 32

Kapitel 5
Beispielanwendungen 33
5.1 Fakultätsberechnung nach Actors 33
5.2 Einfaches RPC-Beispiel 36
5.3 Trader- und Broker-Beispiele 37
5.4 Beispiel von interagierenden Objekten 40
5.5 Terminaushandlungs-Beispiel 42
5.6 Beispiel für unzuverlässige Kommunikationskanäle 44

Kapitel 6
Probleme mit der Sprache Java 46
6.1 Transparenzprobleme 46
6.2 Fehlende Popup-Menüs in Applets 47
6.3 Fehlerhafte InputFocusPolicy 48
6.4 Überlastung des Laufzeitsystems 49
6.5 Einschätzung der Verwendbarkeit von Java 50

Zusammenfassung 51

Ausblick 52

Literaturverzeichnis 53

Anhang A
Beschreibung der Schnittstelle des Baukastens 54
A.1 Allgemeine Verwaltungsmethoden 54
A.2 Methoden zur Verwaltung von Prozessen 58
A.3 Methoden für normale Nachrichten 59
A.4 Methoden für Nachrichtenverfälschungen 62

Anhang B
Programmcode einer Beispielanwendung 68

Kapitel 1

Einleitung

1.1 Motivation

Im Bereich der Verteilten Systeme existieren eine Vielzahl von verwendeten Protokollen. Das Vermitteln dieser Protokolle an Studierende geschieht bisher mit Hilfe von Momentaufnahmen der bei den Protokollen ausgetauschten Nachrichten. Um aus diesen Momentaufnahmen ein klares Bild über den Ablauf eines Protokolls zu erlangen, bedarf es eines guten Vorstellungs- und Abstraktionsvermögens auf Seite der Studierenden.

Im Rahmen dieser Studienarbeit „Visualisierung von Lehrinhalten mit Java Applets“ soll das Vorstellungsvermögen durch rechnergestützte Visualisierung der Protokolle soweit unterstützt werden, daß der Betrachter sein Hauptaugenmerk auf die Funktionalität der Protokolle lenken kann. Die Animation dient hierbei als Hilfsmittel, um das zu übermittelnde Wissen in eine ansprechendere Form zu bringen.

Die Animation übernimmt die Aufgabe, die scheinbar unzusammenhängenden Momentaufnahmen zu einem kontinuierlichen und zusammenhängenden Bild zu verbinden. Dieses zusammenhängende Bild führt zu einer besseren Vorstellbarkeit und einem besseren Verständnis.

Des weiteren soll der Zugang zu den Protokollen dadurch erleichtert werden, daß durch die Interaktivität und die sichtbare Bewegung am Bildschirm an die Experimentierbereitschaft des Lernenden appelliert wird. Er soll den Ablauf der Protokolle, in der für ihn optimalen Geschwindigkeit, spielerisch erfahren können.

Die Verwendung von Java Applets ist dabei ein weiterer Schritt, die Protokolle einem breiteren Publikum zugänglich zu machen. Die Verbreitung von Java nimmt momentan ständig zu, und zukünftig werden Java Applets wahrscheinlich auf allen gängigen Rechnern unabhängig vom verwendeten Betriebssystem dargestellt werden können.

Um nicht für jedes zu visualisierende Protokoll die Darstellung auf dem Bildschirm neu programmieren zu müssen, soll eine Menge von wiederverwendbaren Objekten und Methoden entwickelt werden. Dieser Baukasten an Objekten und Methoden soll es demjenigen, der das Protokoll vermitteln will, ermöglichen, sich auf das Protokoll und dessen Implementierung zu konzentrieren und die Darstellung möglichst weitgehend dem Baukasten zu überlassen.

Das im Rahmen dieser Studienarbeit zu entwickelnde System soll somit sowohl den Lernenden als auch den Lehrenden unterstützen, sich auf das Eigentliche, die Funktionalität des Protokolls, zu konzentrieren.

1.2 Aufgabenstellung

Die Aufgabenstellung dieser Studienarbeit bestand aus fünf Teilaufgaben.

Teilaufgabe Eins war die Einarbeitung in die Programmiersprache Java und der dazugehörenden Graphik-Klassen. Diese Einarbeitung war nötig, da vor der Studienarbeit noch keine Java-Kenntnisse vorlagen, die Studienarbeit jedoch in Java zu implementieren war.

Die zweite Teilaufgabe bestand in der Einarbeitung in den Inhalt der Vorlesung „Kooperation in Verteilten Systemen“ um daraus die Anforderung an das Baukastensystem zu erstellen.

Teilaufgabe Drei umfaßte den Entwurf und die Implementierung des Baukastensystems. Es wurden dabei folgende Funktionalitäten gefordert: Es sollen Komponenten erzeugt und vernichtet werden können. Diese Komponenten sollen Nachrichten austauschen können. Die Nachrichten sollen unterschiedliche Typen haben können, und es soll eine Unterstützung für unzuverlässige Transportmedien in Form von verschiedenen Nachrichtenmanipulationen geben. Die Nachrichtenmanipulationen sahen dabei folgende Fälle vor: Nachrichten können bei der Übertragung verloren gehen oder es können zusätzliche Nachrichten auftauchen. Auch der Inhalt und die Reihenfolge, in der die Nachrichten ankommen, sollen verändert werden können.

Neben diesen Anforderungen an die Komponenten und Nachrichten, ist auch eine interaktive Geschwindigkeitssteuerung und eine Unterbrechung der Darstellung gefordert. Darüberhinaus sollen auch Vorwärts- und Rückwärtssprünge in der Darstellung möglich sein.

Teilaufgabe Vier forderte die Erstellung einiger Beispielprogramme, die mit Hilfe des Baukastensystems verschiedene Protokolle visualisieren sollen. Die darzustellenden Protokolle reichten dabei von einfachen Remote Procedure Calls über verteilte, interagierende Objekte und verschiedene Vermittlungsverfahren bis hin zur Fakultätsberechnung mit Actors (siehe Kapitel 5 für eine genauere Erklärung dieser Protokolle).

Die letzte Teilaufgabe verlangte eine schriftliche Ausarbeitung. In dieser sollen die verwendeten Konzepte und die Implementierung des Baukastensystems beschrieben werden. Auch die Anwendungsbeispiele sollen dokumentiert und eine Beurteilung gemacht werden.

1.3 Überblick

Kapitel 1 beschreibt neben der Aufgabenstellung auch die Motivation für diese Studienarbeit. Kapitel 2 gibt die im Rahmen der Anforderungsanalyse gewonnenen Erkenntnisse wieder. Das Kapitel 3 widmet sich anschließend den beim Entwurf getroffenen Entscheidungen. Die Implementierung des Baukastensystems wird in Kapitel 4 beschrieben. Die Beispielanwendungen werden in Kapitel 5 beschrieben. Kapitel 6 erläutert einige der im Verlauf dieser Studienarbeit aufgetretenen Probleme mit Java. Mit der Zusammenfassung und dem Ausblick endet die eigentliche Ausarbeitung. Anhang A listet die von der Anwendung aufrufbaren Methoden des Baukastens mit ihren Parametern und ihrer Funktion auf. Im Anhang B findet man den kompletten Code eines Anwendungsprogramms, das als Beispiel für die Anwendung des Baukastensystems dient.

Kapitel 2

Anforderungsanalyse

Dieses Kapitel dokumentiert die Anforderungen an das Baukastensystem, die direkt aus der Aufgabenstellung abgeleitet werden können. Abschnitt 2.1 erklärt einleitend ein Szenarium, aus dem erste Anforderungen abgeleitet wurden. Abschnitt 2.2 stellt kurz das Umfeld, in dem das Baukastensystem eingesetzt werden soll, dar. In Abschnitt 2.3 werden die benötigten Funktionalitäten für die geforderten Nachrichtenverfälschungen aufgeführt. Abschnitt 2.4 widmet sich der Verteilung der Aufgaben auf die einzelnen Objekte. Abschnitt 2.5 präsentiert abschließend eine Liste der benötigten Funktionalitäten des Baukastens.

2.1 Entwurfsszenarium

Der Entwurf der Objekte und Methoden erfolgte anhand eines Szenariums, das die wesentlichen Merkmale der zu implementierenden Funktionen enthält. Dieses Szenarium ist in Abbildung 2.1 dargestellt.

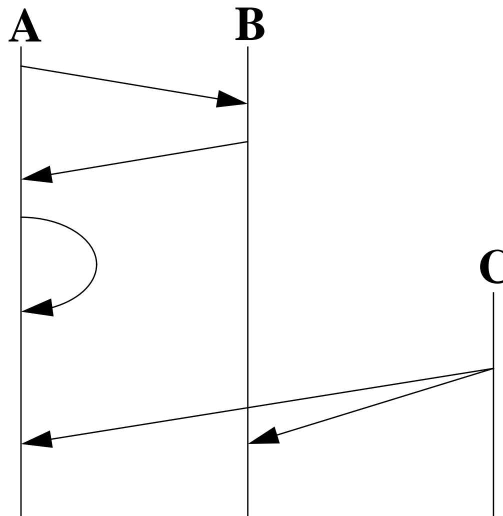

Abbildung 2.1: Entwurfszenarium

Das Szenarium stellt eine fiktive Kommunikation zwischen drei Komponenten, dargestellt durch die drei senkrechten Striche, dar. Anfangs existieren nur die zwei Komponenten A und B. Als erstes schickt A eine Nachricht an B. B antwortet kurze Zeit später mit einer weiteren Nachricht. Anschließend schickt A eine Nachricht an sich selbst. Als letztes kommt eine weitere Komponente C hinzu und verschickt eine Broadcastnachricht.

Hieraus lassen sich schon einige benötigte Funktionalitäten ableiten:

* Es müssen mehrere Komponenten dargestellt werden können.
* Komponenten können im Verlauf der Zeit dynamisch hinzukommen oder verschwinden.
* Es muß verschiedene Arten von Nachrichten geben. Im Szenarium werden z.B. Nachrichten von einer Komponente an eine andere Komponente, von einer Komponente an alle anderen Komponenten und von einer Komponente an sich selbst benötigt.

Es wird prinzipiell keine Aussage gemacht, was diese Komponenten genau darstellen. Es können Prozesse auf einem Rechner, Agenten in einem Agentensystem (zu Agenten siehe [Hohl]) oder Personen in einem Arbeitsablaufmodell sein. Eine Darstellung von Komponentengruppierungen aufgrund von räumlichen oder inhaltlichen Unterschieden wird nicht vorgenommen, da dies die Darstellung zu unübersichtlich machen würde. Der Einfachheit halber und aufgrund des direkten Bezugs zu Rechnern, wird im folgenden statt von Komponenten meist von Prozessen gesprochen.

2.2 Umfeld des Baukastensystems

Zur Erklärung des Baukastenumfelds und für den späteren Gebrauch müssen hier zuerst die beteiligten Personengruppen vorgestellt werden. Die Wissensvermittlung wird allgemein mit dem Bild eines Lehrer - Schüler oder des Professor - Studierender Verhältnisses assoziiert. In Anlehnung hieran wird in dieser Studienarbeit das allgemeinere Bild eines Vortragenden - Lernenden Verhältnisses verwendet. Der Vortragende soll dabei dem Lernenden einen bestimmten Sachverhalt vermitteln. Hierzu stellt er den Sachverhalt mit Hilfe des Rechners animiert dar.

Die Darstellung soll im Rahmen dieser Studienarbeit über Java Applets geschehen. Applets werden in HTML-Seiten eingebunden und mit diesen zusammen von einem Browser dargestellt. Hieraus ergibt sich folgendes Umfeld für das Baukastensystem: Der Browser des Lernenden stellt eine HTML-Seite mit dem Applet des Vortragenden dar. Das Applet des Vortragenden wiederum erzeugt eine Instanz des Baukastens und benutzt diese zur Visualisierung eines Protokolls. Graphisch stellt sich dies wie in Abbildung 2.2 gezeichnet dar.:

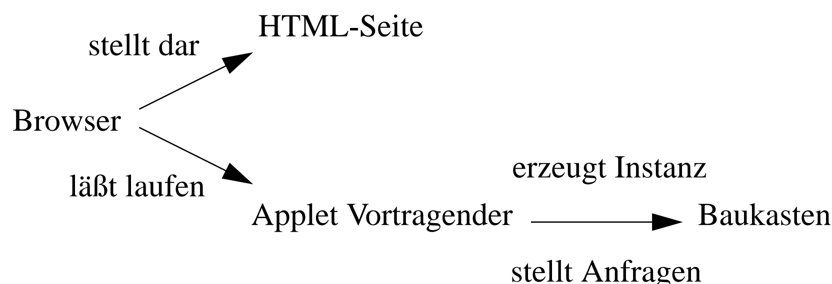

Abbildung 2.2: Umfeld des Baukastens

Daraus ergibt sich die Anforderung, daß der Baukasten eine Schnittstelle in Form eines Objekts anbieten muß, welches sich in den normalen Darstellungsablauf von Java Applets integrieren läßt.

2.3 Berücksichtigung von unzuverlässigen Kommunikationskanälen.

Der Baukasten soll die Darstellung von zuverlässigen Kommunikationskanälen ermöglich. Bei diesen kommen abgesandte Nachrichten garantiert unverändert beim Empfänger an und der Lernende kann keine Nachrichtenmanipulationen auslösen. In diesem Fall arbeitet die Anwendung im Applet des Vortragenden einfach ihr Protokoll ab und reagiert auf keine Benutzereingaben.

Über dies hinausgehend soll der Baukasten eine Unterstützung von allgemeinen Kommunikationsprotokollen bieten. Diese Protokolle beinhalten die Fehlerbehandlung bei Nachrichtenverlusten oder Nachrichtenfälschungen. Zwecks Darstellung des Verhaltens dieser Protokolle beim Eintritt solcher Fehler soll der Lernende die Möglichkeit haben, interaktiv zu entscheiden, ob eine Nachrichtenmanipulation stattfinden soll oder nicht. Hierzu muß die Anwendung auf Benutzereingaben reagieren und abhängig von der jeweiligen Eingabe unterschiedliche Nachrichtenveränderungen vornehmen.

Der Baukasten muß deshalb neben Methoden für eine normale Darstellung einer Nachrichtenübermittlung auch Methoden anbieten, die den Nachrichtenverlust, die Nachrichtenverfälschung, die Nachrichtenvertauschung oder das Einstreuung von zusätzlichen Nachrichten entsprechend auf dem Bildschirm darstellen.

2.4 zentrale und dezentrale Funktionalitäten

Um die Objektorientierung von Java sinnvoll zu nutzen, sollten die einzelnen Prozesse und Nachrichten eigenständige Instanzen von einer Prozess- und einer Nachrichtenklasse sein.

Die in der Aufgabenstellung geforderten Funktionalitäten der Geschwindigkeitswahl samt der Unterbrechung der Darstellung lassen sich jedoch nur sehr schwer über eine variable Anzahl von Prozessen und Nachrichten verteilen. Auch die Implementierung der geforderten Vorwärts- und Rückwärtssprünge ist verteilt nur schwer möglich. Deshalb muß es ein zentrales Objekt geben, das diese nicht verteilbaren Funktionalitäten zentral erbringt. Eine Kombination mit dem in Abschnitt 2.2 geforderten Schnittstellenobjekt erscheint dabei sinnvoll.

2.5 Spezifikation der nötigen Funktionalitäten

Die benötigten Funktionalitäten sehen somit folgendermaßen aus:

Es wird ein zentrales Objekt benötigt, dieses soll Assistent genannt werden und folgende Aufgaben übernehmen:

* Er soll Schnittstelle zur Anwendung sein und alle Baukastenzugriffe von der Anwendung sollen nur über ihn ablaufen. Dies erfordert eine Umsetzung der von der Anwendung aufgerufenen Methoden in Methodenaufrufe in den Prozess- und Nachrichtenobjekten des Baukastens
* Er kann von verschiedenen Objekten der Anwendung (z.B. von einem Server und einem Client-Objekt) aufgerufen werden und muß deshalb eine Synchronisation vorsehen
* Er soll möglichst keine Rückwirkung auf Anwendungs-Objekte haben. Alle Methoden sollen immer okay/true zurückliefern.
* Er soll der Anwendung symbolische Namen für erzeugte Prozesse und Nachrichten anbieten und diese intern in die Objektreferenzen umwandeln können.
* Er soll die Verwaltung aller Prozesse und Nachrichten übernehmen
* Er soll das Layout der Prozesse und Nachrichten auf dem Bildschirm steuern.
* Er soll die Darstellung durch das Aufrufen von Methoden in den Prozessen und Nachrichten auslösen.
* Er soll einen Zeitbegriff bieten, um die Reihenfolge der Nachrichten festzulegen.

* Er soll eine Variierung der Ablaufgeschwindigkeit ermöglichen. Dabei sollte eine mittlere Geschwindigkeit der Standard sein, und sowohl schnellere als auch langsamere Animationen möglich sein.
* Er soll Vorwärts- und Rückwärtssprünge in der Historie ermöglichen. (siehe Abschnitt 3.2)

Die Prozesse sollen durch Instanzen einer Prozess-Klasse implementiert werden und folgende Funktionalitäten bieten:

* Sie sollen über Namen ansprechbar sein und diesen Namen auf dem Bildschirm darstellen.
* Sie sollen sich auf Anforderung des Assistenten selbst darstellen.
* Sie sollen ihre Darstellung auf ein Zeitintervall zwischen dem Erzeugen und dem Beenden des Prozesses einschränken können.

Die Nachrichten sollen durch Instanzen einer Nachrichten-Klasse implementiert werden und folgende Funktionalitäten bieten:

* Sie sollen die Nachrichtenübertragung animiert darstellen.
* Sie sollen auf Aufforderung des Assistenten den Weg der Nachrichtenübermittlung vom Sender zum Empfänger durch Linien und die Nachricht selbst in Form eines Piktogramms auf dem Bildschirm darstellen. Anfangs- und Endpunkte sollen zwecks Übersichtlichkeit durch Punkte gekennzeichnet werden.
* Sie sollen Nachrichten von einem Sender an einen Empfänger und Multicast-/Broadcastnachrichten an mehrere Empfänger darstellen können. Auch die Darstellung von Nachrichten von Prozessen an sich selbst soll möglich sein.
* Sie sollen folgende Fehlerfälle darstellen können: Paketverluste, Auftauchen zusätzlicher Pakete, Nachrichteninhaltsveränderungen und Nachrichtenüberkreuzungen
* Sie sollen die Nachrichtenübermittlung in Teilschritten zulassen, damit eine Manipulation durch den Benutzer vorgenommen werden kann.

Kapitel 3

Entwurfsentscheidungen

Im Verlauf des Entwurfs des Baukastens mußten einige Entscheidungen getroffen werden, die sowohl das Erscheinungsbild, als auch die Implementierung beeinflußten und deshalb im folgenden dargestellt werden sollen.

Abschnitt 3.1 widmet sich dabei der Frage, wie der Kontrollfluß zwischen Anwendung und Baukasten verteilt wird. Abschnitt 3.2 beschäftigt sich mit der in der Aufgabenstellung geforderten History-Funktionalität. Abschnitt 3.3 beschreibt die in der Literatur verwendeten Darstellungen und erklärt, warum nur eine davon vom Baukasten unterstützt wird. Abschnitt 3.4 behandelt die Frage, wer für die Implementierung der Nachrichtenmanipulationen durch den Benutzer, zuständig ist. Der Abschnitt 3.5 schildert die Ergebnisse der ersten Prototypentwicklung.

3.1 Abfrage- versus Erweiterungsansatz

Für das Zusammenspiel zwischen dem Programm des Vortragenden und dem Baukasten ergaben sich beim Entwurf zwei mögliche Ansätze.

Der erste Ansatz wurde als „Abfrageansatz“ bezeichnet, da bei diesem der Baukasten beim Programmcode des Vortragenden erfragt, was als nächstes angezeigt werden soll. Der Assistent übernimmt hier die aktive Rolle, während das Programm des Vortragenden nur reagiert.

Der zweite Ansatz wurde als „Erweiterungsansatz“ bezeichnet, da bei diesem ein eventuell schon bestehendes Programm des Vortragenden um Darstellungsfunktionsaufrufe „erweitert“ wird. Hier ist das Programm des Vortragenden der aktive Teil, und der Assistent reagiert nur auf die gestellten Anfragen.

Der Hauptunterschied zwischen den beiden Ansätzen ist der unterschiedliche Kontrollfluß. Beim Abfrageansatz ist die Kontrolle des Ablaufs vollständig im Assistenten implementiert. Dort läuft eine Art zentraler Schleife ab, in der jedesmal beim Programmcode des Vortragenden gefragt wird, was als nächstes passiert und dargestellt werden soll. Nach Beantwortung dieser Frage wird die Darstellung eingeleitet und danach mit dem nächsten Schleifendurchlauf fortgefahren.

Beim Erweiterungsansatz hingegen liegt die Kontrolle ganz auf der Seite des Programms des Vortragenden. Dieses stellt das Hauptprogramm dar, welches bei Bedarf Methoden des Assistenten aufruft um die Darstellung machen zu lassen.

Neben diesem Hauptunterschied existieren noch eine Reihe von weiteren Unterschieden, die teilweise Folgen dieses Hauptunterschiedes sind.

Ein für den Vortragenden sicher wichtiger Punkt ist die Frage, wie intuitiv oder gewöhnungsbedürftig die Ansätze sind. Der Abfrageansatz ist in diesem Sinne der gewöhnungsbedürftigere Ansatz, denn der Vortragende muß sich an den vom Assistenten gegebenen Ablauf anpassen und muß versuchen, das, was er darstellen will, in eine dem Ablauf konforme Form zu bringen. Der Erweiterungsansatz hingegen ist intuitiver, da das Hauptaugenmerk beim darzustellenden Protokoll bleiben kann und die Aufrufe der Darstellungsfunktionen praktisch nebenher gemacht werden können.

Der Abfrageansatz ist quasi die Umkehrung des Baukasten-Prinzips, hier stellt der Assistent den Rahmen, der vom Vortragenden ausgefüllt werden muß. Der Erweiterungsansatz hingegen entspricht dem Baukasten-Prinzip, bei dem das Programm des Vortragenden bei Bedarf Bausteine in Form von Objekten oder Methoden aus dem Baukasten holt.

Ein weiterer Unterschied ist die Flexibilität und die Frage, wie leicht diese erreicht werden kann. Im Falle des Abfrageansatzes können höchstwahrscheinlich nicht alle Java-Funktionen verwendet werden, da diese mit dem Assistenten im Konflikt stehen können. Beispielsweise kann nicht einfach ein weiterer Thread gestartet werden, da dies ohne Wissen des Assistenten und außerhalb dessen Kontrolle, zu undefinierten Zuständen führen könnte.

Da beim Erweiterungsansatz die Kontrolle komplett im aufrufenden Programm liegt, stehen in diesem Fall alle von der Programmiersprache Java angebotenen Möglichkeiten zur Verfügung. Dies bedeutet eine hohe Flexibilität die nur durch die Hostsprache Java beschränkt wird.

Ebenfalls problematisch ist beim Abfrageansatz die Programmierung im Programm des Vortragenden. Dieses muß abhängig vom aktuellen Zustand dem Baukasten sagen, welche Aktionen als nächstes ausgeführt werden sollen. Dies führt zu einer großen Anzahl von

```
if <zustand> then <action> else <action> fi
```

Konstrukten. Dies würde sicherlich sehr schnell ineffizient und unübersichtlich werden, besonders da die Anzahl der Zustände im Falle einer Rekursion sehr groß werden kann.

Der Erweiterungsansatz ist somit flexibel und gleichzeitig vergleichsweise leicht zu implementieren, während der Abfrageansatz starrer ist, sofern eine vergleichbare Flexibilität nicht durch deutlich höheren Implementierungsaufwand erkauft wird.

Der Abfrageansatz stellt eine Erweiterung der Parametrisierung von Programmen dar und hat sicher seine Existenzberechtigung für Probleme, in denen vergleichsweise einfache Abläufe als Parameter in einem Rahmenprogramm eingebunden werden sollen. Für die gestellte Aufgabe dieser Studienarbeit ist der Erweiterungsansatz jedoch sinnvoller, da die Abläufe komplex werden können und der Implementierungsaufwand gleichzeitig nicht unendlich steigen soll. In Tabelle 3.1 sind die genannten Argumenten noch einmal zusammengefaßt.

Tabelle 3.1: Vergleich von Erweiterungs- und Abfrageansatz

Erweiterungsansatz	Abfrageansatz
flexibel	starr
intuitiv	gewöhnungsbedürftig
Kontrolle beim Programm des Vortragenden	Ablaufkontrolle im Assistenten
Assistent reaktiv	Assistent aktiv
bestehende Programme sind erweiterbar zu sich selbst visualisierenden Programmen	Algorithmus des bestehenden Programms muß in Form des Assistenten gebracht werden
Baukastenprinzip	Umkehrung des Baukastenprinzips
leicht implementierbar	komplex, wenn unterschiedliche Situationen machbar sein sollen

3.2 Playback versus „Trial and Error“ History-Arten

Bereits früh in der Entwurfsphase stellte sich die Frage, wie die geforderten Rückwärtssprünge implementiert werden können. Recht schnell war klar, daß dazu eine History-Funktion nötig ist, die genau protokolliert, welche Aktionen ablaufen. Rückwärtssprünge können dann auf zwei Arten implementiert werden. Entweder es wird zu jeder Aktion eine Gegenaktion definiert, die die Wirkung der Aktion aufhebt, oder alternativ wird ein Rückwärtssprung ersetzt durch die Simulation aller Aktionen bis zu dem Zeitpunkt, zu dem zurückgesprungen wird. Das Baukastensystem verwendet den zweiten Ansatz.

Interessanter ist die Frage, wie nach einem Rückwärtssprung weitergemacht werden darf. Im einfachsten Fall kann nach einem Rücksprung nur weiter zurück oder wieder in die Richtung, aus der der Sprung kam, vorwärts gesprungen werden. Dies entspricht einem einfachen Playback ohne Veränderung des Ablauf

Es ist jedoch auch ein zweiter Fall denkbar, daß nach einem Rücksprung ein Parameter geändert und dann in eine neue Richtung vorwärts gesprungen wird. Dieser Fall ist sicher sinnvoll für den Vortragenden, da somit unterschiedliche Abläufe bei unterschiedlichen Eingaben vergleichend dargestellt werden können. Dies ist jedoch dann kein einfaches Playback mehr, sondern eine Art von „Trial and Error“ bei dem nach einem „Error“ ein Stück zurückgegangen wird und ein anderer Versuch gestartet wird.

Abbildung 3.1 stellt die beiden Ansätze graphisch gegenüber.

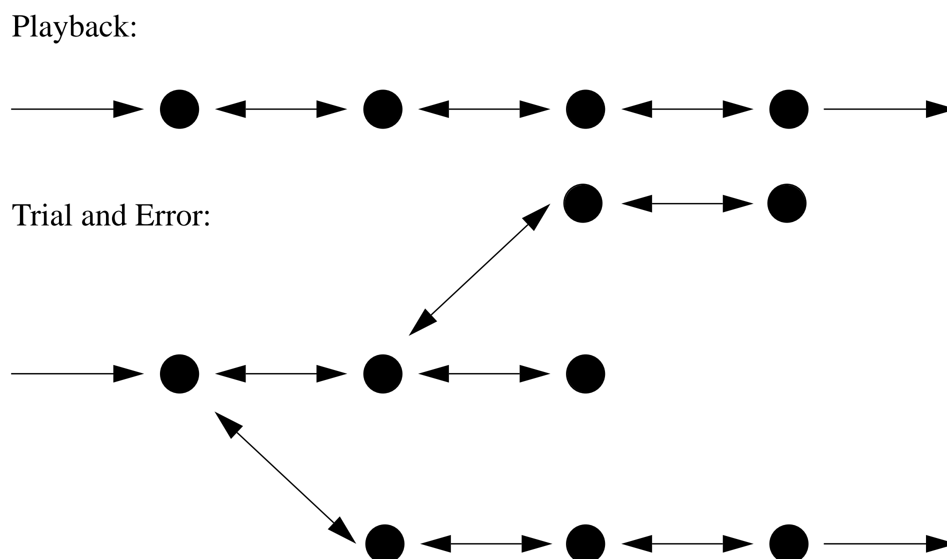

Abbildung 3.1: Vergleich von Playback und Trial and Error

Die Implementierung hiervon ist jedoch sehr kompliziert. Es reicht nicht mehr aus, daß die History-Funktion sich nur die Zustände der Anzeige merkt, sondern sie muß sich auch die Zustände des Programms des Vortragenden merken, damit bei einem Rücksprung auch der Zustand des Programms des Vortragenden wieder auf den alten Zustand gesetzt werden kann. Dies ist jedoch praktisch unmöglich, da die History-Funktion nicht wissen kann, welche Objekte und Variablen des Programms den Zustand darstellen und somit gesichert werden müssen. Diese Art des Rücksprungs muß deshalb vom Programm des Vortragenden unterstützt werden, indem es die Sicherung der Zustände selber implementiert. Ein Großteil der History-Funktionalität muß somit vom Programm des Vortragenden selbst erbracht werden

Aufgrund dieses Problems wurde im Rahmen dieser Studienarbeit nur ein History-Mechanismus für den einfachen Fall des Playbacks implementiert.

3.3 Überlegungen zur Darstellung

Neben den Fragen, wieweit History-Funktionalität geboten werden kann und welche Ansätze beim Kontrollfluß möglich sind, gab es noch eine dritte zu treffende Entscheidung, die Entscheidung, welche Art der Darstellung auf dem Bildschirm gewählt werden soll. Auch hier gab es zwei Alternativen, die auf die Darstellungen in der Literatur der zu visualisierenden Protokolle zurückgehen.

Zum einen gibt es bei Actors (siehe [Borghoff]) eine Darstellung, in der die Prozesse und Nachrichten so angeordnet werden, daß die Überschneidungen möglichst gering sind und ein optisch ansprechendes Gesamtbild entsteht (siehe Abbildung 3.2).

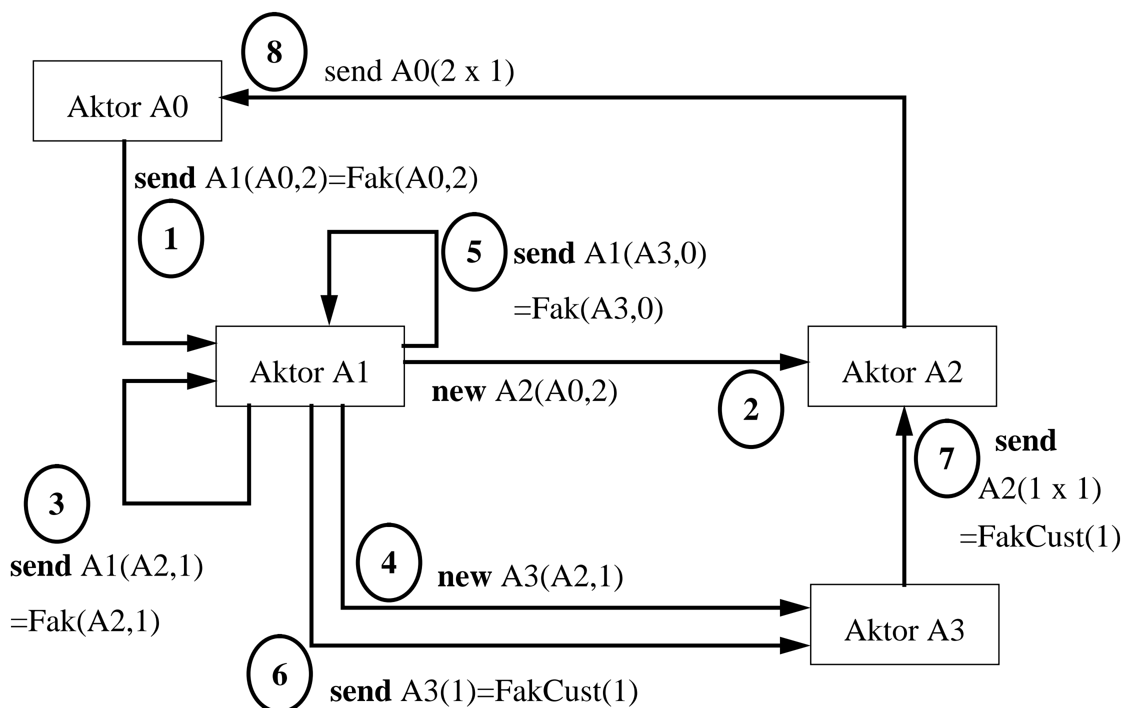

Abbildung 3.2: Fakultätsberechnung mit Actors

Ein solches Bild läßt sich jedoch nicht ohne weiteres automatisch durch einen Rechner erzeugen, da die Entscheidungen, die ein Mensch bei der Verteilung der Objekte trifft, nicht deterministisch und somit weitgehend zufällig und von den künstlerischen Fähigkeiten des Menschen abhängig sind.

Wenn man eine solche Darstellung von einem Rechner erstellen lassen will, muß man die intuitiven Entscheidungen, die ein Mensch trifft, durch berechenbare Entscheidungen ersetzen.

In Netzwerkmonitoren, die vor ähnlichen Problemen stehen, wenn sie ein Rechnernetz darstellen sollen, wird dann oftmals eine Verteilung der Objekte anhand von geometrischen Aspekten vorgenommen. Meistens werden die Objekte dann kreis- oder ellipsenförmig um einen Mittelpunkt herum angeordnet. Der interessierte Leser kann sich dies z.B. bei dem im

Internet frei verfügbarem tkined [ftp://ftp.ibr.cs.tu-bs.de/pub/local/tkined] Paket anschauen. Dieses verwendet eine erweiterte Form dieses Ansatzes, indem die Objekte auf mehreren Kreisen um einen Mittelpunkt herum verteilt werden.

Das Ergebnis der über diesen Ansatz erzeugten Darstellungen entspricht jedoch praktisch nie einer Darstellung wie der von Actors in Abbildung 3.2.

Der zweite Darstellungsansatz der in der Literatur häufig verwendet wird, ist der einer Zeitachse, entlang derer die Objekte verteilt sind. Abbildung 3.3 zeigt eine solche Darstellung..

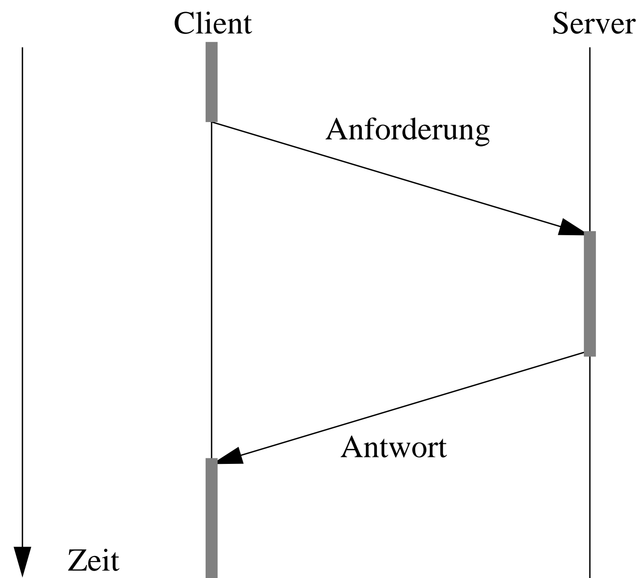

Abbildung 3.3: Einfacher RPC-Call entlang einer Zeitachse

Vorteil dieser Darstellung ist, daß sie sehr leicht automatisch erzeugt werden kann, da die Prozesse immer nebeneinander angeordnet sind und sich parallel zur Zeitachse beliebig lang ausdehnen lassen. Auf diese Weise kann auch der Fall, daß viele Nachrichten von einem Prozess verschickt oder empfangen werden, dargestellt werden, ohne daß vor lauter Nachrichten die Prozesse nicht mehr zu sehen sind.

Ein weiterer Vorteil ist der, daß die Darstellung aufgrund des Zeitbezugs für Menschen intuitiv gut verständlich ist.

Aufgrund der genannten Argumente wurde für diese Studienarbeit die Darstellung anhand einer Zeitachse gewählt. Dies stellt keine Einschränkung dar, da die Abläufe von Actors auch entlang einer Zeitachse dargestellt werden können. Siehe hierzu auch das Actors-Anwendungsbeispiel in Abschnitt 5.1.

Die Zeitachse verläuft in der Darstellung senkrecht von oben nach unten. Die Prozesse werden als Linien parallel zu der Zeitachse dargestellt. Sie müssen dabei nicht über den kompletten dargestellten Zeitraum existieren, sondern können zu einem beliebigen Zeitpunkt beginnen und enden. Dargestellt werden sie dann nur in dem zwischen diesen beiden Zeitpunkten liegenden Zeitintervall. Die Nachrichten werden als schräg zwischen den Prozessen verlaufende Linien dargestellt. Dabei liegt der Anfang, der das Absenden der Nachricht darstellt, zeitlich vor, darstellungsmäßig also oberhalb des Endpunktes, der dem Empfang der Nachricht entspricht.

3.4 Unterstützung von Nachrichtenmanipulationen

Im Verlauf des Entwurfs stellte sich die Frage, wo die zusätzlich nötigen Benutzerinteraktion für den Fall der unzuverlässigen Nachrichtenübermittlung, implementiert werden sollten. Zur Wahl standen zwei Möglichkeiten, die die Implementierung der Benutzerinteraktion verschieden zwischen Anwendungsprogramm und Baukasten verteilen.

Möglichkeit Eins sah dabei vor, daß die komplette Einflußnahme des Benutzers vom Anwendungsprogramm verarbeitet wird. Dies schließt die Ausgabe der möglichen Manipulationen auf dem Bildschirm und die Interpretation der Benutzereingaben ein. Der Baukasten muß in diesem Fall dann nur für die nach der Benutzermanipulation möglichen Zustände eine entsprechende Darstellung auf dem Bildschirm anbieten

Vorteil hierbei ist, daß die Schnittstelle zwischen Baukasten und Anwendung sehr einfach bleibt, da die Informationen immer nur in eine Richtung fließen. Der Baukasten muß keine Informationen über die vom Benutzer gemachten Manipulationen an das Anwendungsprogramm weitergeben, sondern dient rein der Darstellung der vom Anwendungsprogramm geforderten Aktionen.

Möglichkeit Zwei hingegen sah vor, daß die Interaktion mit dem Benutzer vom Baukasten übernommen wird.

Dies erfordert eine erweiterte Schnittstelle, damit dem Baukasten gesagt werden kann, welche Manipulationen jeweils möglich sind und wie er die Manipulation des Benutzers an das Anwendungsprogramm melden soll. Die jeweils möglichen Manipulationen können von Fall zu Fall unterschiedlich sein, je nachdem was das Anwendungsprogramm an Fehlerfällen kennt. Die hierzu nötige Erweiterung der Schnittstelle zwischen Anwendungsprogramm und Baukasten dürfte recht aufwendig sein.

Auch die Rückmeldung der ausgewählten Manipulation würde dabei die Schnittstelle vergrößern. Die Rückmeldung kann prinzipiell auf drei unterschiedliche Arten geschehen:

* durch Auslösen einer Exception
* durch Rückgabe eines Status-Wertes oder
* durch Aufruf einer Methode im Anwendungsprogramm.

Welche dieser drei Möglichkeiten der Entwickler des Anwendungsprogramms, also der Vortragende, bevorzugt, hängt wohl sehr stark davon ab, wie weit er einen prozeduralen oder einen objektorientierten Ansatz bevorzugt. Für eine maximale Flexibilität müßte der Baukasten deshalb alle drei Möglichkeiten unterstützen. Dies bedeutet aber, daß zusätzlich abgeklärt werden muß, wie der Baukasten die Rückmeldung machen soll, oder es muß für die drei Möglichkeiten jeweils verschiedene Methoden geben. Beides würde umfangreiche Änderungen an der Schnittstelle bedeuten.

All dies erscheint ein unnötig hoher Aufwand zu sein, nur damit das Anwendungsprogramm die Interaktion mit dem Benutzer nicht machen muß. Da das Anwendungsprogramm jedoch höchstwahrscheinlich auch an anderen Stellen Eingaben des Benutzers annehmen muß, beispielsweise die Eingabe von Startwerten für irgendwelche Berechnungen, also die notwendigen Methoden zur Eingabe bereits implementieren muß, erscheint der Aufwand ungerechtfertigt.

Da die Auswirkungen der Manipulation auf das ablaufende Protokoll sowieso vom Anwendungsprogramm vorgenommen werden muß, da nur dieses weiß, wie das Protokoll darauf reagiert, hat der erste Ansatz zusätzlich den Vorteil, daß die Behandlung der Manipulation an einer Stelle geschieht und nicht über Anwendung und Baukasten verteilt ist.

Der Baukasten ist deshalb auf den ersten Ansatz ausgelegt.

3.5 Frühe Entwurfsphase und erster Prototyp

Die Entwurfsphase wurde von der Entwicklung eines Prototyps begleitet. Dies hatte hauptsächlich zwei Gründe. Zum einen sollte die Einarbeitung in die Programmiersprache Java durch praktische Arbeit mit Java vorangetrieben werden. Zum anderen sollten die beim Entwurf zu treffenden Entscheidungen auf Realisierbarkeit getestet werden. Gerade dieser zweite Punkt erwies sich als besonders wichtig. Grundlegende Entscheidungen der frühen Entwurfsphase mußten nämlich revidiert werden, weil eine Implementierung in der ursprünglich geplanten Art und Weise von Java nicht unterstützt wird.

Die frühe Entwurfsphase sah beispielsweise vor, daß möglichst viel Funktionalität durch Verwendung bereits existierender Objekte und Methoden erbracht werden sollte. Es wurde deshalb versucht, mit Standard-Graphikobjekten aus java.awt.* wie Panel, Container oder Frame ein Minimalsystem für diese Studienarbeit aufzubauen. Verwendet wurde besonders die automatische Verwaltung von Graphikobjekten über die Standard-Layoutmanager. Diese sorgen zum einen dafür, daß die Objekte nach vorgegebenen Kriterien auf der Darstellungsfläche verteilt werden, und zum anderen übernehmen sie auch die nötigen Aufgaben, wenn ein Teil der Darstellungsfläche neu gezeichnet werden muß. Dies ist zum Beispiel nötig, wenn andere Fenster die Darstellungsfläche zeitweise verdeckt haben und diese nach Beendigung der Überdekkung neu dargestellt werden muß. Die Darstellung sollte durch Überlagerung der einzelnen Prozess- und Nachrichtenobjekte realisiert werden. Jedes Objekt sollte sich dafür selbst in Form von Linien und Images darstellen. Die Überlagerung der Selbstdarstellungen hätte dann das gewünschte Gesamtbild ergeben.

Die Prototypenentwicklung zeigte jedoch, daß die Methoden/Objekte von java.awt.* nicht auf ein Problem wie dieses hin ausgelegt sind. Zum einen sind alle Graphikobjekte in Java nicht transparent, so daß ein Zusammensetzen der Bildschirmausgabe aus einzelnen Graphikobjekten durch Überlagerung dieser nicht möglich ist. Zum anderen wird die Überlagerung von Objekten durch die Standard-Layoutmanagern auch nicht unterstützt. Deren Hauptziel ist das genaue Gegenteil, nämlich die Objekte überschneidungsfrei nebeneinander zu plazieren (siehe auch Abschnitt 6.1 hierzu).

Aufgrund dieser beim ersten Prototypen aufgedeckten Unzulänglichkeiten von Java mußte von dem Ansatz der möglichst weitestgehenden Verwendung von bestehenden Objekten und Methoden etwas Abstand genommen werden. So werden für diese Studienarbeit die meisten Objekte nicht als eigenständige Graphikobjekte, also nicht als Subklassen von java.awt.container/component, sondern als Subklassen von java.lang.Object angelegt. Nur die zentralen Objekte des Assistenten und des Displays sind Graphikobjekte. Innerhalb dieser Objekte werden alle Layout- und Darstellungsfunktionen selber implementiert.

Kapitel 4

Implementierung des Baukastens

In diesem Kapitel wird beschrieben, aus welchen Objekten der Baukasten besteht und nach welchen Mechanismen diese zusammenarbeiten. Neben der Gesamtarchitektur werden auch die verwendeten grundlegenden Ideen und Mechanismen erläutert. Eine Beschreibung der einzelnen Methoden wird dabei nicht gegeben, hierfür ist die Aufstellung in Anhang A zuständig.

Abschnitt 4.1 erklärt die Gesamtarchitektur des Baukastensystems. Im Abschnitt 4.2 wird die Oberfläche des Assistenten und die Funktion der einzelnen Elemente vorgestellt. In Abschnitt 4.3 werden die aus Performancegründen verwendeten Images und deren Sichtbarkeit zu verschiedenen Zeitpunkten erklärt. Abschnitt 4.4 beschreibt dann die Teile des Baukastens, die mit dem Laufzeitsystem von Java eng verbunden sind: Die Ein- und Ausgabe und die benutzten Threads. Im Abschnitt 4.5 werden die Mechanismen zur Erzeugung, den Versand und den Empfang von Nachrichten erläutert.

4.1 Gesamtarchitektur des Baukastens

Der Baukasten besteht aus vier verschiedenen Klassen: eine Prozess-, eine Nachrichten-, eine Display- und eine Assistentenklasse. Von der Prozess- und der Nachrichtenklasse werden im Verlauf einer Animation beliebig viele Instanzen erzeugt. Von der Display- und der Assistentenklasse existiert immer nur eine Instanz. Die Instanz der Assistentenklasse, im folgenden kurz Assistent genannt, stellt die Schnittstelle zwischen Anwendung und Baukastensystem dar. Die Instanz der Displayklasse ist ein Hilfsobjekt, dessen Funktion in Abschnitt 4.2.1 erklärt wird. Die Instanzen der Prozess- und Nachrichtenklassen repräsentieren die darzustellenden Prozesse und Nachrichten.

Die Verwendung des Baukastensystems aus dem Programm des Vortragenden geschieht ausschließlich durch das Aufrufen von Methoden des Assistentenobjekts. Das Assistentobjekt wiederum ruft in den Prozess-, Nachrichten- und Displayobjekten Methoden auf, um seine Aufgaben zu erfüllen. Graphisch sieht dies wie in Abbildung 4.1 gezeigt aus.

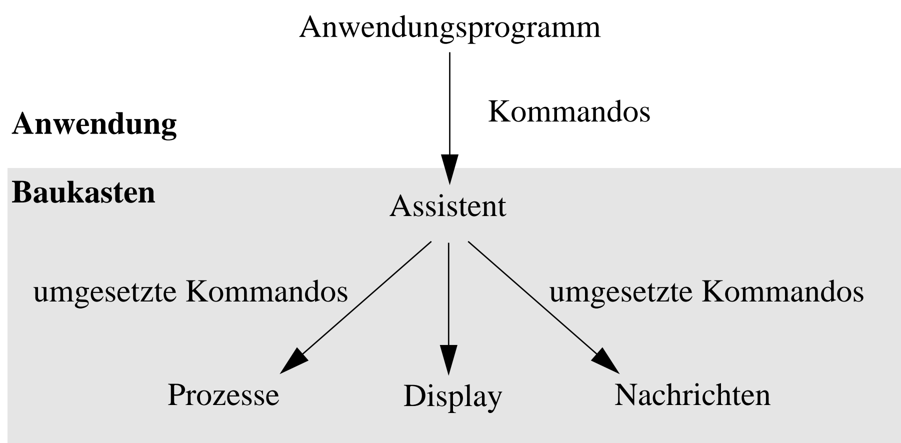

Abbildung 4.1: Kommandos zwischen Anwendung und Baukasten

Das Assistentenobjekt stellt die Schnittstelle zwischen den eigentlichen Prozess-/Nachrichten-Objekten und dem Anwendungsapplet dar. Ihm obliegen die zentralen Aufgaben des Baukastens: (siehe auch Abschnitt 2.5)

* die Verwaltung der Nachrichten/Prozessobjekte,
* deren Layout auf dem Bildschirm,
* die Steuerung der Darstellung der Objekte entsprechend dem Layout und
* die nötigen History-Funktionen.

Die Prozess- und Nachrichtenobjekte haben im wesentlichen nur die Aufgabe, sich selbst darzustellen, wenn dies vom Assistenten gefordert wird. Die Darstellung umfaßt sowohl eine statische Darstellung der Prozesse und der in der Vergangenheit verschickten Nachrichten, als auch eine dynamisch Darstellung der Animation einer Nachrichtenübermittlung.

Alle Prozesse und Nachrichten werden innerhalb eines einzigen Applets dargestellt und nicht durch ein Applet pro Prozess oder Nachricht. Grund hierfür ist, daß der Browser einzelne Applets abhängig von äußeren Einflüssen wie Breite des Browser-Fensters und Schriftgröße auf dem Bildschirm unterschiedlich verteilen würde. Es könnte somit nicht garantiert werden, daß die Prozesse in der gewünschten Form nebeneinander und die Nachrichten zwischen den Prozessen dargestellt werden. Auch ist das Layout bei unterschiedlichen Browsern unterschiedlich, so daß das Erscheinungsbild abhängig vom verwendeten Browser unterschiedlich wäre.

4.2 Sichtbare Graphik-Elemente

Das zentrale Objekt des Baukastens, der Assistent, ist eine Subklasse von java.awt.Panel und somit von der Anwendung als normales Panel in dessen Darstellung integrierbar. Das Panel dient als Container für die im folgenden beschriebenen Graphik-Objekte, die den sichtbaren Teil des Baukastens ausmachen. Alle in den folgenden Unterabschnitten beschriebenen Objekte werden im Konstruktor des Assistenten erzeugt und in den normalen Java-Darstellungsmechanismus eingefügt. Abbildung 4.2 zeigt einen Screenshot der in diesem Abschnitt beschriebenen Oberfläche des Baukastens.

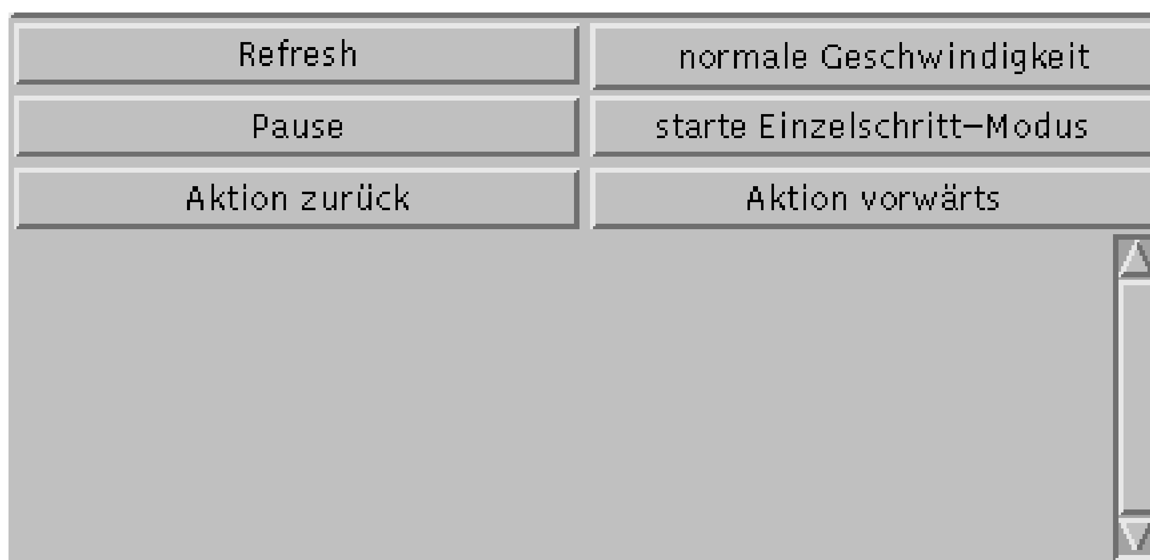

Abbildung 4.2: Screenshot der Baukastenoberfläche

4.2.1 Das Display-Objekt

Die eigentliche Darstellungsfläche des Assistenten für alle Nachrichten und Prozesse liegt in Form des Display-Objekts vor. Das Display-Objekt ist eine Subklasse von java.awt.Canvas, hat selbst relativ wenig Aufgaben und existiert primär aus Symmetriegründen. Gemeint ist damit, daß sich das Display-Objekt, die Scrollbar und das Steuer-Panel die komplette Fläche des Assistenten-Panels aufteilen und somit nichts vom Assistenten-Panel selbst sichtbar ist. Würde das Display-Objekt nicht existieren, dann würden Teile des Assistenten-Panels durch die Scrollbar und das Steuer-Panel überdeckt und der Assistent müßte entsprechend aufpassen, daß er in diese überdeckten Flächen nichts zeichnet, da der Benutzer dies durch die Überdeckung ja nicht sehen würde. Die folgende Abbildung 4.3 soll den Sichtbarkeitsunterschied etwas verdeutlichen

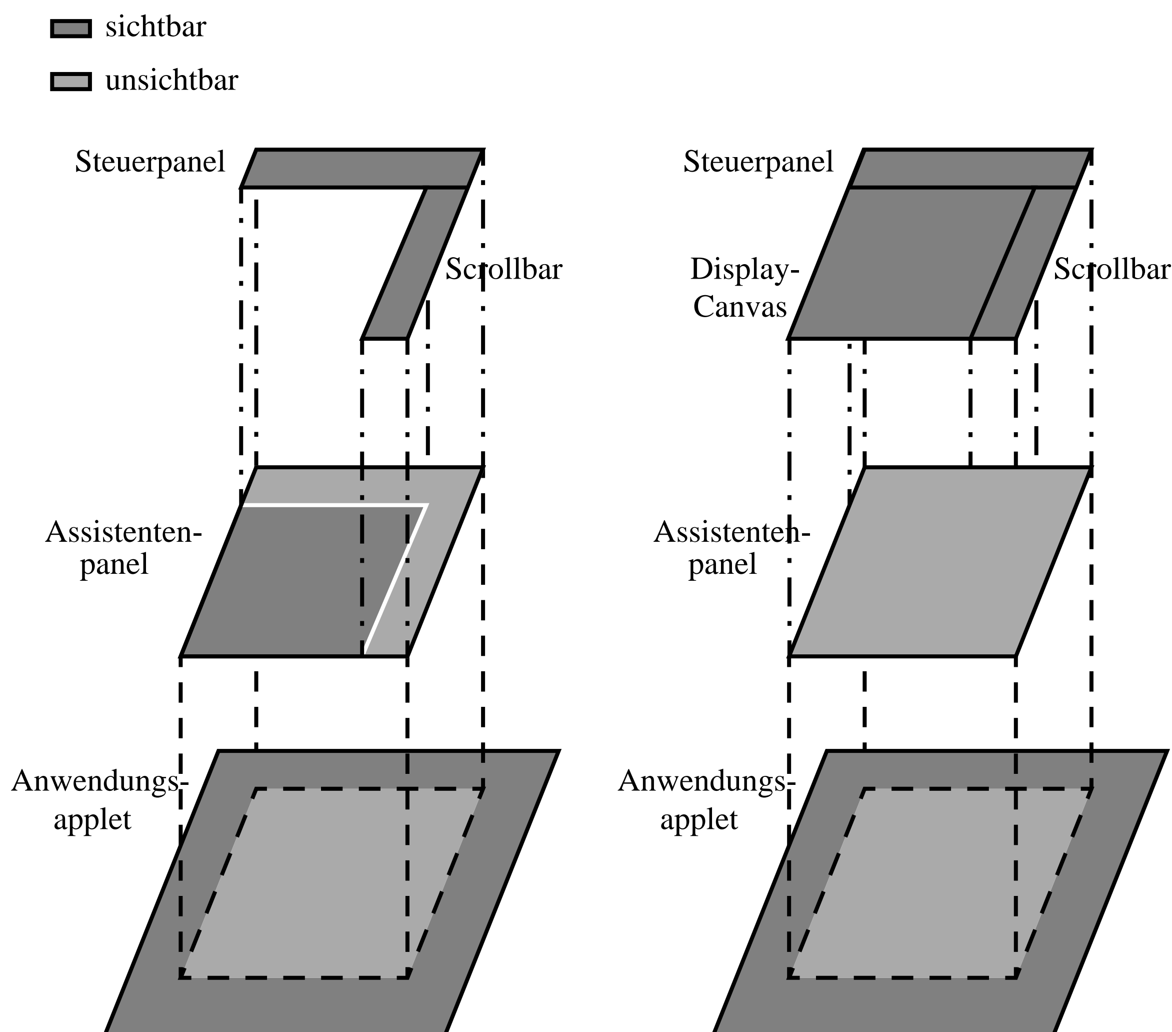

Abbildung 4.3: Sichtbarkeitsunterschied

4.2.2 Die Scrollbar

Neben dem Display-Objekt enthält das Assistenten-Panel die Scrollbar, die vertikales Scrollen ermöglicht. Die Scrollbar bestimmt dabei, welcher Bereich der verwendeten virtuellen Darstellungsfläche jeweils dargestellt wird. Die virtuelle Darstellungsfläche ermöglicht eine über-

sichtliche Darstellung auch für Protokolle, die aus vielen Nachrichten bestehen. Entsprechend könnten durch eine horizontale Scrollbar mehr Prozesse nebeneinander dargestellt werden. Durch eine gleichzeitig horizontal und vertikal ausgedehnte Darstellungsfläche würde jedoch die Verständlichkeit nicht vergrößert sondern verkleinert. Dies liegt vorallem daran, daß der Lernende durch das komplexe Navigieren in der virtuellen Darstellungsfläche zu leicht den Überblick verliert. Aus diesem Grund wurde auf eine horizontal ausgedehnte virtuelle Darstelungsfläche und damit auf eine horizontale Scrollbar verzichtet.

4.2.3 Das Steuer-Panel

Als drittes Graphik-Objekt, welches zur kompletten Überdeckung des Assistenten-Panels beiträgt, existiert ein Container für die Buttons und den Choice für die Geschwindigkeitswahl. Für diesen Container wurde ein Panel verwendet, welches durch die enthaltenen Graphikobjekte wiederum vollständig überdeckt wird. Dieses scheinbar unnötige Panel dient nur dem leichteren Layout der Graphikobjekte. Das Steuer-Panel enthält eine Reihe von Standard-Graphikobjekten mit deren Hilfe der Betrachter einige Funktionen im Assistenten auslösen kann. Die Abbildung 4.4 zeigt ein um das Steuer-Panel und dessen Inhalt erweiterte Schichtstruktur der Graphikobjekte. Dieselbe Schichtstruktur ist auch noch einmal dreidimensional dargestellt

Als erstes ist ein Refresh-Button vorhanden, mit dessen Hilfe der Benutzer einen kompletten Neuaufbau der Darstellung der Prozesse und Nachrichten erzwingen kann. Normalerweise sollte die Darstellung ohne Verwendung des Refresh-Buttons jederzeit korrekt sein, jedoch wurde bei der Entwicklung festgestellt, daß die Darstellung aufgrund einer Überlastung des Java-Laufzeitsystems unter Umständen durcheinandergeraten kann (siehe auch Abschnitt 6.4 hierzu).

Das rechts neben dem Refresh-Button platzierte Choice-Objekt dient der Wahl der Geschwindigkeit, mit der die Animation der Nachrichtenübermittlung und der Fortschritt der Zeit ablaufen soll. Hauptzweck hiervon ist, daß der Benutzer die Darstellungsgeschwindigkeit nach seinen Wünschen und Bedürfnissen regeln kann. Unter anderem kann somit der Leistungsunterschied zwischen verschiedenen Rechnern etwas ausgeglichen werden. Benutzer, die nur einen relativ langsamen 386er PC besitzen, können durch Wahl einer hohen Geschwindigkeitsstufe die fehlende Leistung ihres Rechners kompensieren.

Unter diesen beiden Objekten sind zwei Buttons angeordnet, die ihre Beschriftung je nach aktuellem Zustand verändern. Der linke Button trägt normalerweise die Beschriftung „Pause“ und dient dann zum sofortigen Anhalten der Animation. Nach Betätigung dieses Buttons wird dessen Beschriftung zu „beende Pause“ verändert und ein nochmaliges anklicken dieses Buttons, setzt die Darstellung fort. Auf diese Weise kann an beliebiger Stelle eine beliebig lange Pause zwecks Erklärungen des Vortragenden eingefügt werden.

Entsprechend ermöglicht der rechte Button, der in Normalzustand die Aufschrift „starte Einzelschritt-Modus“ trägt, ein schrittweises Ablaufen des Protokolls (Single-Step). Hierbei wird im Gegensatz zum Pause-Button nicht an beliebiger Stelle angehalten, sondern immer am Ende einer Animation, also z.B. am Ende einer Nachrichtenübermittlung oder am Ende der Darstellung eines neuen Prozesses.

Nach drücken des „starte Einzelschritt-Modus“-Buttons läuft die Animation weiter bis zum Ende eines Animationsschritt und wartet dann auf weitere Benutzereingaben. Die Beschriftung des linken Buttons lautet während dieser Zeit „beende Einzelschritt-Modus“ und dient zur Rückkehr in den Normalzustand. Die Beschriftung des rechten Buttons ist gleichzeitig

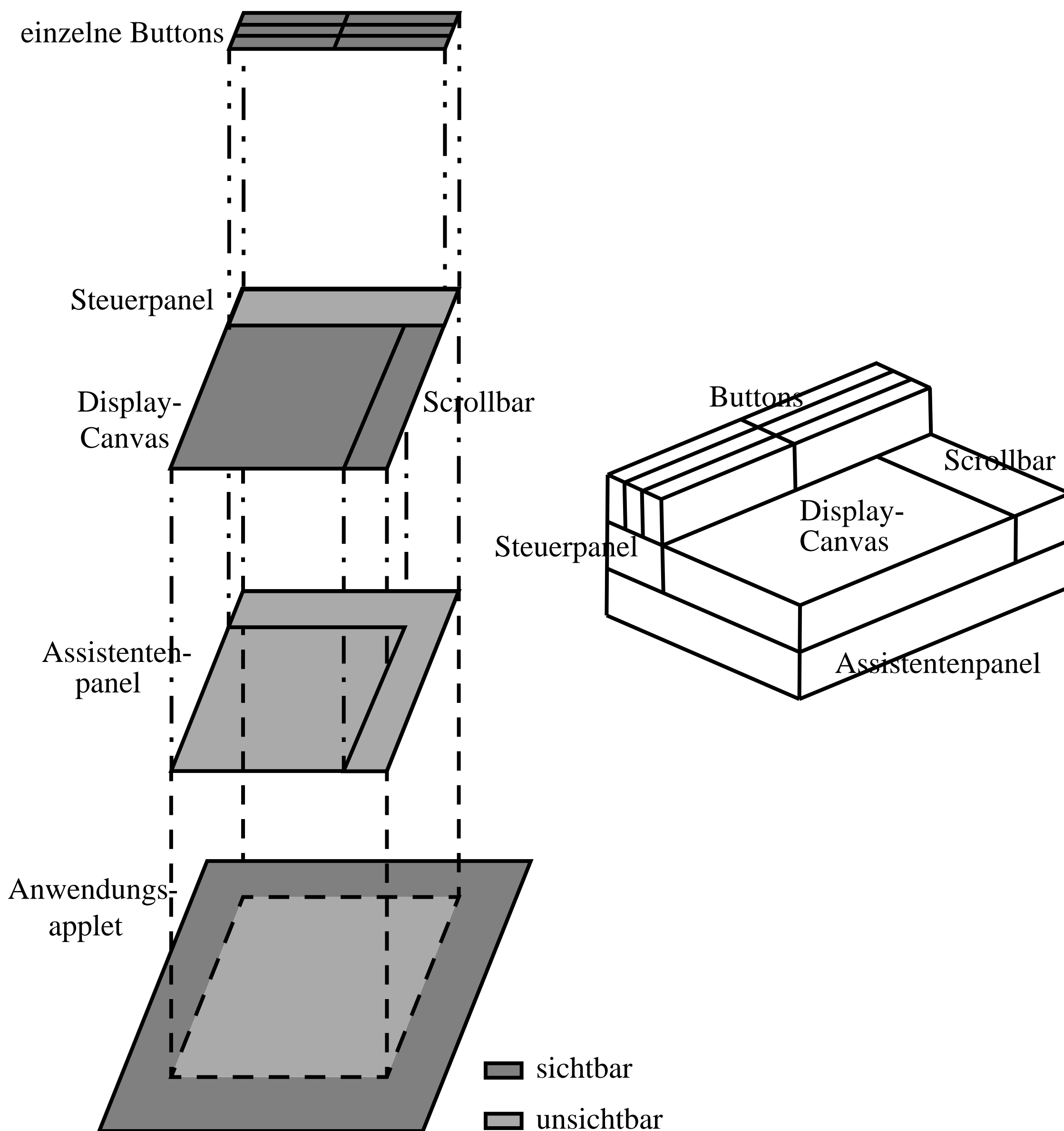

Abbildung 4.4: Schichtstruktur der Graphikobjekte

„nächster Einzelschritt“ und läßt nach anklicken die Animation des nächsten Schritts ablaufen. Während ein solcher Einzelschritt animiert dargestellt wird, ist der linke Button wieder ein normaler „Pause“ bzw. „beende Pause“-Button. Der rechte Button trägt zwar weiter die Beschriftung „nächster Einzelschritt“, ist jedoch deaktiviert und wird deshalb mit grauer Schrift dargestellt. Auf diese Weise kann auch während des Ablaufs eines Single-Step-Animationsschritts eine Pause eingefügt werden und die Animation danach bis zum Ende des Animationsschritts fortgesetzt werden.

Mit Hilfe zweier weiterer Buttons „Aktion zurück“ und „Aktion vorwärts“ können einzelne Schritte unsichtbar und wieder sichtbar gemacht werden. Es wird dabei kein Schritt wirklich rückgängig gemacht, sondern nur dessen Darstellung unterdrückt. Für den Betrachter erscheint es aber so, als ob der jeweils letzte dargestellte Schritt zurückgenommen worden

wäre. Der „Aktion vorwärts“ Button ist nur dann aktiv, wenn vorher Schritte zurückgenommen wurden die noch nicht wieder sichtbar gemacht wurden. Diese zwei Buttons stellen das Interface für das im Entwurf (Abschnitt 3.2) beschriebene Playback-History dar.

4.3 Verwendung von Images

Um eine möglichst schnelle Darstellung auch auf langsamen Rechnern zu erreichen, wird ein Großteil der Darstellung und der Animation über Images gemacht. Die Images werden nach Möglichkeit nicht bei jeder Darstellung eines Bewegungsschritts oder eines vom Fenstersystem ausgelösten Bildschirmrefreshs neu berechnet, sondern nach der Berechnung mehrfach verwendet sofern dies möglich ist.

Nicht möglich ist dies natürlich für die Animation der Nachrichtenübermittlung, da hier die aufeinanderfolgenden Darstellungen unterschiedlich sein müssen, damit der Eindruck einer Bewegung entsteht. Sehr wohl möglich ist eine Wiederverwendung für alle Objekte, die ihre Lage über mehrere Darstellungen hinweg nicht verändern.

Es existieren somit zwei Klassen von Objekten, eine Klasse von Objekten, die über mehrere Darstellungen hinweg unverändert, also statisch sind, und eine Klasse von Objekten, deren Darstellung sich ständig verändert, also dynamisch ist.

Für Objekte, die in die Klasse der dynamischen Objekte fallen, muß die Positionsberechnung und die Darstellung der einzelnen Elemente bei jeder Darstellung neu gemacht werden, was einen nicht vermeidbaren Rechenaufwand bedeutet. Für alle statischen Objekte hingegen kann eine Menge Rechenaufwand eingespart werden, indem die einzelnen statischen Elemente zu einem einzigen Image zusammengefaßt werden, dessen Darstellung auf dem Bildschirm bedeutend weniger aufwendig ist. Dieses Image trägt den Namen statischeObjekteImage und wird in Abschnitt 4.3.1 beschrieben.

Das in Abschnitt 4.3.2 beschriebene antiFlackerImage dient neben der Beruhigung der Darstellung auch dem Zusammensetzen von statischen und dynamischen Elementen zur Gesamtdarstellung.

Abschnitt 4.3.3 erklärt als letztes die Images, die zur Darstellung von Prozessen und Nachrichten verwendet werden.

4.3.1 Das statischeObjekteImage

In diesem Image werden alle statischen Elemente gespeichert und können dann über einen einzelnen drawImage()-Aufruf auf dem Bildschirm (bzw. in das antiFlackerImage von Abschnitt 4.3.2) dargestellt werden.

Das statischeObjekteImage ist somit ein Image, in das die statischen Darstellungselemente nur selten neu hineingezeichnet werden, das jedoch sehr oft in das antiFlackerImage gezeichnet wird.

Um eine komplette Darstellung aller Objekte auf dem Bildschirm zu erreichen, werden die dynamischen Elemente zusammen mit dem statischeObjekteImage dargestellt. Eine genaue Erklärung, welche Images wie miteinander überlagert werden, wird in Abschnitt 4.3.2 gegeben.

Hier wird im folgenden noch erklärt, wann das statischeObjekteImage neu erstellt wird, wie die einzelnen Darstellungselemente in das statischeObjekteImage gelangen und welchen Nachteil die Verwendung des statischeObjekteImage hat.

Das statischeObjekteImage muß in folgenden Fällen neu aufgebaut werden:

Tabelle 4.1: Gründe für den Neuaufbau des statischeObjekteImages

wann	warum
Erzeugen des Darstellungs-Threads	Es gab bisher kein statischeObjekteImage
Veränderung der Größe des Display-Objektes	Es wird ein neues Layout der Prozesse gemacht (Anmerkung: dies geschieht z.B. wenn weiter Graphikobjekte in den Container des Anwendungsapplets eingefügt werden oder wenn das Fenster, in dem das Java Laufzeitsystem läuft verändert wird und dies an das Applet weitergegeben wird, was allerdings nur von manchen Browsern unterstützt wird)
Benutzung Refresh-Button	Benutzer will einen definierten Zustands erzwingen
Erzeugen eines Prozesses	Es muß ein neues Layout der Prozesse gemacht werden
Beenden eines Prozesses	Prozess ist ab sofort komplett statisch
Erzeugen einer Nachricht	Der Weg, den die Nachricht entlang wandert, wird, als statisches Element eingetragen. Eigentlich ist der Weg kein wirklich statisches Element, aber die Berechnung und das Zeichnen der Linien oder Halbkreise ist so rechenintensiv, daß sie nicht dauernd neu gemacht werden soll, sondern als quasi-statisches Element Bestandteil des statischeObjekteImage sein soll
Nachricht hat Empfänger erreicht	Die Nachricht wird zukünftig nicht mehr bewegt

An den entsprechenden Stellen, an denen einer dieser Zustände im Programmcode eintritt, wird eine globale Variable des Assistenten namens „brauchenKomplettaufbau“ auf „true“ gesetzt und der nächste Durchlauf der run()-Methode des Assistenten löscht dann das statischeObjekteImage und läßt alle statischen Elemente der Prozesse und Nachrichten neu zeichnen. Wenn dabei festgestellt wird, daß das statischeObjekteImage nicht groß genug ist, um die komplette virtuelle Darstellungsfläche zu speichern, dann wird ein neues statischeObjekteImage erzeugt und das alte weggeworfen. In diesem Fall wird dann auch der Wertebereich der Scrollbar angepaßt, so daß der Eindruck einer dynamisch wachsenden Darstellungsfläche entsteht.

Dieses dynamische Wachsen der virtuellen Darstellungsfläche hat jedoch auch den Nachteil, daß eine Vergrößerung der virtuellen Darstellungsfläche das Erzeugen eines größeren statischeObjekteImage nach sich zieht. Dies kann bei großen virtuellen Darstellungsflächen oder wenig Hauptspeicher zu Speicherproblemen führen.

4.3.2 Das antiFlackerImage

Neben dem statischeObjekteImage ist das antiFlackerImage das zweite große Image, welches zentraler Bestandteil des Baukastens ist. Im Gegensatz zum statischeObjekteImage ist das antiFlackerImage jedoch konstant in der Größe und entspricht genau der Größe, die das Display-Objekt auf dem Bildschirm einnimmt. Der Hauptgrund für die Verwendung des antiFlakkerImages ist, wie der Name schon sagt, die Vermeidung von Flacker-Effekten bei der Animation. Diese entstehen auf dem Bildschirm, wenn in schneller Folge Zeichen- und Löschoperationen direkt auf dem Bildschirmspeicher ausgeführt werden.

Der normale Ablauf einer Darstellung ist ja ein Löschen der kompletten Darstellungsfläche gefolgt von einzelnen Zeichenoperationen für die einzelnen darzustellenden Elemente. Da diese einzelnen Zeichenoperationen recht langsam sind, vergeht viel Zeit zwischen dem Löschen der Darstellungsfläche und dem Zeitpunkt, zu dem alle Elemente wieder neu dargestellt sind. Die Wahrscheinlichkeit, daß während dieser Zeit die Graphikkarte mehrmals den Bildschirmaufbau an den Monitor überträgt und dieser die nur teilweise gezeichneten Bilder darstellt, ist sehr hoch. Die in diesen Teilbildern fehlenden Elemente werden jedoch bei einem der nächsten Bildschirmaufbauten wieder dargestellt, was dazu führt, daß die Elemente nur kurzzeitig fehlen, was als Flackereffekt wahrgenommen wird.

Um diesen Flackereffekt zu vermeiden, wird das Zusammensetzen der Bildinformationen im Hintergrund gemacht und dann nur das komplett aufgebaute Bild in einer Operation in den Bildschirmspeicher übertragen. Natürlich kann auch während dieser Zeit ein Teilbild an den Monitor übertragen werden, allerdings führt dies nicht zu einem Flackern. Stattdessen wird in diesem Fall einfach im oberen Teil der Darstellungsfläche ein Teil des neuen Bildinhalts und im darunterliegenden Rest noch der alte Bildinhalt zusammen dargestellt. Dieser Effekt wird aber normal nicht als störend empfunden und kann auch nur sehr schwer beseitigt werden.

Im Baukasten verläuft das Zusammensetzen des Bildes aufgrund der in Abschnitt 4.3 beschriebenen Unterscheidung zwischen statischen und dynamischen Elementen wie folgt ab: Als erstes wird das statischeObjekteImage in das antiFlackerImage kopiert. Wenn das statischeObjekteImage dabei größer als das antiFlackerImage ist, so wird nur der durch die Scrollbar festgelegte Ausschnitt der virtuellen Darstellungsfläche aus dem statischeObjekteImage in das antiFlackerImage kopiert. Ein Löschen des antiFlackerImages ist vor dem Kopieren nicht nötig, da das statischeObjekteImage nicht transparent ist und mit diesem der alten Inhalt des antiFlackerImages einfach überschrieben wird.

Als nächstes werden die im statischeObjekteImage über den aktuellen Zeitpunkt hinausgehenden Linien durch ein darüberzeichnen mit der Hintergrundfarbe gelöscht. Dies ist nötig, damit, wie in Abschnitt 4.3.1 beschrieben, die Wege, die die Nachrichten entlangwandern, nicht dauernd dynamisch neu gezeichnet werden müssen. Stattdessen werden die Nachrichtenwege und die Linien, die die Prozesslebenszeit darstellen, als quasi statische Elemente nur bei einem Komplettaufbau in voller Länge eingezeichnet und hier dann der unbenötigte Teil wieder gelöscht. Die Erfahrung bei der Erstellung des Baukastens hat gezeigt, daß dies deutlich schneller geht, obwohl intuitiv ja eigentlich zuviel gemacht wird.

Als letztes werden nun die einzelnen dynamischen Elemente in das antiFlackerImage eingezeichnet. Hierfür wird im Assistenten eine Liste der dynamischen Objekte geführt, so daß nur von diesen eine dynamische Darstellung erbeten wird. Diese Liste ist im Normalfall recht kurz, so daß die Darstellung aus wenigen dynamischen Objekten besteht und somit recht schnell ist. Die Liste enthält neben den Nachrichten, die gerade animiert dargestellt werden,

stets auch alle zum aktuellen Zeitpunkt noch laufenden Prozesse, da deren Bild dynamisch mit der Zeit entlang der Zeitachse nach unten wandern sollen. Abschließend kann das antiFlackerImage auf den Bildschirm kopiert werden.

Abbildung 4.5 zeigt das Zusammensetzen des antiFlackerImages graphisch

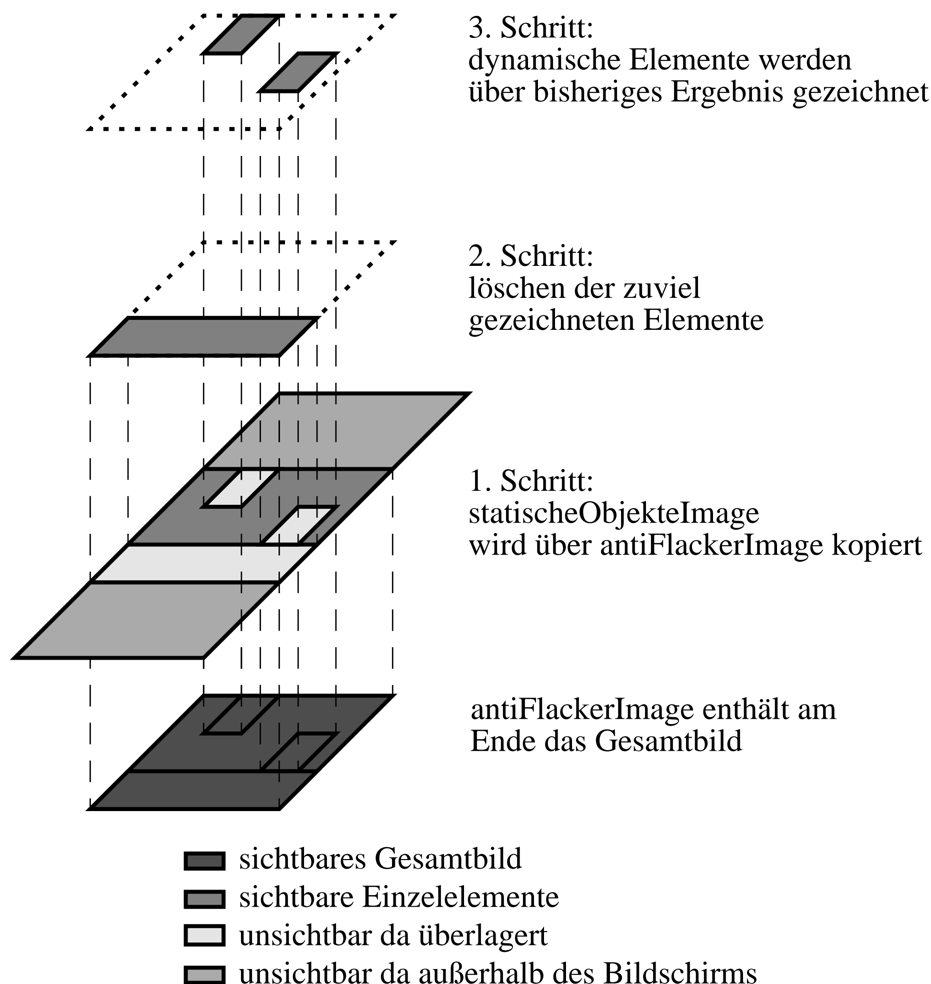

Abbildung 4.5: Zusammensetzung des antiFlackerImages

4.3.3 Sonstige kleinere Images

Neben den in Abschnitt 4.3.1 und 4.3.2 beschriebenen großen Images werden noch weitere zumeist kleinere Images verwendet. Diese enthalten die Bilder, die die Prozesse und Nachrichten auf dem Bildschirm symbolisieren. Diese Bilder sollten vom Lehrenden passend zu dem zu visualisierenden Protokoll zur Verfügung gestellt werden.

Die Bilder können dem Baukasten entweder als Name eines zu ladenden Files relativ zur DocumentBase des Applets oder als Image-Objekt, welches von der Anwendung geladen oder erzeugt wurde, übergeben werden. Wenn keine eigenen Bilder vom Lehrenden verwendet werden sollen, so stehen quasi als Default zwei intern erzeugte Images zur Verfügung.

Das erste default Image ist einfach der Name des Prozesses oder der Nachricht, etwas erhoben dargestellt ähnlich einem Button. Diese Darstellung wird defaultmäßig für die Darstellung der Prozessnamen verwendet. Zu vermerken wäre hier, daß, obwohl nur Strings dargestellt wer-

den, die interne Speicherung in Form von Images geschieht. Dies ermöglicht zum einen eine schnellere Darstellung und zum anderen eine einheitliche Behandlung unabhängig davon, ob die Nachrichten oder Prozesse durch spezielle Bilder oder durch puren Text dargestellt werden sollen.

Das zweite default Image stellt einen Briefumschlag mit Briefmarke und Adressfeld mit einem herausragenden Blatt Papier, auf das der Name der Nachricht geschrieben ist, dar. Dieses Image wird als Defaultsymbol für Nachrichten verwendet. Es ist allerdings praktisch nur mit kurzen Texten sinnvoll einsetzbar, da sonst riesige Briefumschläge große Teile der Darstellungsfläche bedecken, sich mit anderen Elemente überlappen oder diese gar verdecken. Durch Aufrufen von setzeBriefNachricht(false) im Assistenten, kann erreicht werden, daß auch für Nachrichten defaultmäßig das erste Default Image verwendet wird.

Abbildung 4.6 zeigt je zwei der default-Images.

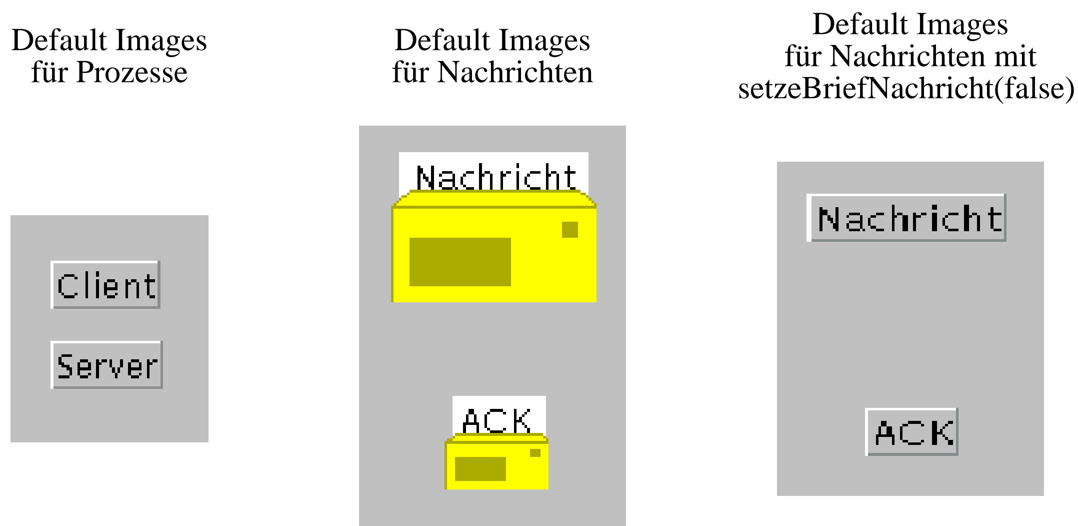

Abbildung 4.6: Verwendete Default-Images

Aufgrund der genannten Unzulänglichkeiten der Defaultbilder empfiehlt sich die Erstellung von speziell zum Protokoll passenden Bildern.

Die Bilder für die Prozesse und Nachrichten werden intern in einem Hash, der als Keys Strings und als Values die Images enthält, gespeichert. Die Verwendung des Hashes wurde gewählt, damit sowohl vom Lehrenden gegebene Bilder als auch die Defaultbilder gleichzeitig zusammen verwendet werden können, ohne daß dies unterschiedlichen Programmcode erfordert.

Der prinzipiell geplante Ablauf ist folgender: Die Anwendung sagt dem Baukasten, welche Images er laden soll oder übergibt ihm selbsterzeugte Images. Zu dem jeweiligen Image wird ein Name, unter dem das Image später referenziert wird, von der Anwendung mitgegeben. Der Baukasten trägt Bild und Name in den ImageHash ein, nachdem er eventuell vorher das Image geladen hat.

Beim Erzeugen eines Prozesses oder einer Nachricht übergibt die Anwendung dann unter anderem den Namen des Bildes, das für den Prozess oder die Nachricht verwendet werden soll. Wenn kein ImageName übergeben wird, dann wird der Name des Prozesses oder der Nachricht als ImageName verwendet. Der Assistent überprüft dann, ob zu dem Namen ein Image existiert, das er verwenden kann. Wenn zu dem Namen noch kein Image existiert, dann wird automatisch ein Defaultbild erzeugt, in den Hash eingetragen und für die Darstellung verwendet.

4.4 Schnittstellen zum Java Laufzeitsystem

Dieser Abschnitt beschreibt die Punkte, an denen der Baukasten mit dem Laufzeitsystem von Java in Berührung kommt und die Auswirkungen, die dies mit sich bringt. Im einzelnen sind dies die verwendeten Threads, die empfangenen Events und die Darstellungsmechanismen von Java.

Abschnitt 4.4.1 erklärt die vom Baukasten verwendete Zeitskala, die Dauer von Aktionen und die Pausen dazwischen. Abschnitt 4.4.1 ist somit Grundlage für das Verständnis der in Abschnitt 4.4.2 erklärten Threads. Abschnitt 4.4.3 erläutert die Events, auf die der Baukasten reagiert und wie er damit umgeht. Abschnitt 4.4.4 erklärt, wie sich der Baukasten in das Komponenten und Container-System von java.awt.* einfügt.

4.4.1 Zeitsystem des Baukastens

Wie im Kapitel 1 und 2 beschrieben wurde, dient der Baukasten zur Darstellung des zeitlichen Ablaufs von Protokollen. Dies setzt voraus, daß der Baukasten ein Zeitsystem bietet, mit dem beschrieben werden kann, wann welche Aktionen vom Protokoll ausgeführt werden. Dabei kommt es meist nicht darauf an, zu welchen absoluten Zeitpunkten eine Aktion stattfindet, sondern es reicht zu wissen, welche Aktionen zeitlich vor oder nach einer anderen Aktion stattfinden. Deshalb verwendet der Baukasten keinen Zeitbegriff, der an die reale Zeit geknüpft ist, sondern einen virtuellen Zeitbegriff, der angibt, wieviel Platz eine Aktion bei der Darstellung auf dem Bildschirm einnimmt.

Dies heißt, daß für die verschiedenen Aktionen festgelegt wird, um wieviel die interne Uhr beim Ausführen der Aktion weitergedreht werden soll. Diese interne Dauer hat keinen Bezug zu der Zeit, die die entsprechende Aktion in Realität dauert, sondern dient nur dazu, eine möglichst ansprechende Darstellung auf dem Bildschirm zu ermöglichen. Neben der Dauer von Aktionen gibt es auch Pausen zwischen den Aktionen, die der Übersichtlichkeit der Darstellung dienen. Ohne diese Pausen würden sich die Bilder der Nachrichten häufig überlappen oder die Wege, die die Nachrichten entlangwandern, so dicht aneinandergrenzen, daß eine Unterscheidung nicht mehr möglich ist.

Sowohl die Dauer der Aktionen als auch die Pausen vor und nach Aktionen können von der Anwendung nach Belieben gesetzt werden. Wenn die Anwendung diese Werte nicht setzt, dann werden automatisch die Werte genommen, die bei der Entwicklung des Baukastens und der Beispiele zu sinnvollen Darstellungen führten. Diese Defaultwerte sind in Anhang A bei der detaillierten Beschreibung der Methoden und Objekte aufgeführt.

4.4.2 Threads und deren Notwendigkeit

Dieser Abschnitt beschreibt sowohl den Thread, der im Baukasten für die Darstellung zuständig ist, als auch den Thread, der in der Anwendung vorhanden sein muß, damit das Laufzeitsystem von Java nicht blockiert wird.

Der erstgenannte Thread wird ohne Zutun der Anwendung vom Baukasten für die Darstellung der Prozesse und Nachrichten im Display-Objekte (beschrieben in Abschnitt 4.2.1) erzeugt und beeinflußt die Anwendung somit nicht. Zweck dieses Threads ist es, die Darstellung unabhängig zu machen um eine möglichst hohe Fehlertoleranz zu erreichen. Der Thread soll sicherstellen, daß eventuell im Baukasten auftretende Fehler nicht zu einem vorzeitigen Ende der Darstellung und des Protokolls führen.

Der Thread besteht abgesehen von einigen Initialisierungen aus einer Endlosschleife, die nur durch Beenden des Threads gestoppt wird. In der Endlosschleife wird bei Bedarf zuerst das in Abschnitt 4.3.1 beschriebene statischeObjekteImage neu erstellt. Anschließend wird das antiFlackerImage wie in Abschnitt 4.3.2 beschrieben erstellt und anschließend im Display-Objekt dargestellt. Damit diese Endlosschleife nicht dauernd durchlaufen wird und dabei Zeit verbraucht auch wenn keine neue Darstellung nötig ist, wird über ein try-wait-catch Statement sichergestellt, daß die Darstellung nur gemacht wird, wenn der Baukasten dies zur Animation benötigt oder Java oder der Benutzer einen Refresh des Bildschirms verlangt hat. Damit wird vermieden, daß die Endlosschleife endlos viel CPU-Zeit verbraucht, wenn eigentlich nichts gemacht werden müßte. Wenn hingegen eine Animation läuft, dann kann diese Endlosschleife die CPU voll auslasten was jedoch nicht als Fehler zu betrachten ist.

Der zweite Thread der oben erwähnt wurde, ist ein Thread, der nicht im Baukasten erzeugt wird, sondern von der Anwendung erzeugt werden muß, damit die Animation nur die Anwendung und nicht einen Großteil des Java-Laufzeitsystems blockiert. Dies mag auf den ersten Blick etwas befremdlich erscheinen und soll deshalb im folgenden erläutert werden.

Alle von der Anwendung aufgerufenen Methoden, die eine Animation auf dem Bildschirm ablaufen lassen, blockieren während dieser Zeit die Anwendung, damit diese nicht weiter Aufrufe an den Baukasten macht, was zu ungewollten Nebeneffekten führen könnte. Durch diese blockierenden Methoden ist außerdem sichergestellt, daß die Anwendung immer mit der Darstellung synchron läuft und dieser nicht, weil sie schneller als die Darstellung ist, davonläuft.

Somit weiß die Anwendung auch immer, daß eine angestartete Animation beendet ist, wenn der Methodenaufruf beendet ist. Dies ist oftmals sinnvoll, wenn die Anwendung nach einer Animation einen Dialog führen will, bei dem diese Animation bereits beendet sein sollte. Ein Beispiel hierfür wäre die Darstellung eines Protokolls, bei dem der Benutzer bestimmen kann wie auf eine Nachricht geantwortet werden soll. Das Fragen, wie die Nachricht beantwortet werden soll, ist natürlich erst sinnvoll, wenn die Nachricht angekommen ist.

Das Blockieren der Anwendung während einer Animation hat jedoch den Nachteil, daß bei Browsern, bei denen das Applet de Vortragenden in dem Thread abläuft, der auch für den Bildschirmrefresh zuständig ist, der Bildschirmrefresh während der Animation nicht funktioniert. Der Appletviewer des Java Development Kits ist ein Beispiel hierfür.

Um diesen unschönen Effekt des fehlenden Bildschirmrefreshs zu vermeiden, müssen die Baukasten-Aufrufe aus einem Thread, der nicht gleichzeitig noch andere wichtige Aufgaben des Java-Laufzeitsystems ausführt, heraus geschehen. Der hierzu benötigte zusätzliche Thread ist der oben beschriebene Thread, der von der Anwendung implementiert werden muß. Die Anwendung sollte deshalb keine Aufrufe an den Baukasten außerhalb dieses Threads machen.

Insbesondere sollten keine Methoden des Baukastens (mit Ausnahme der Methode zur Erzeugung des Baukastens) aus dem Konstruktor des Applets heraus aufgerufen werden. Auch aus Methoden, die vom Laufzeitsystem aufgerufen werden, sollten keine Baukastenaufrufe geschehen. In diese Kategorie fallen alle Methoden, die Eingaben des Benutzers verarbeiten, also handleEvent, action, mouseDown, keyDown, um nur ein paar zu nennen.

4.4.3 Eventhandling

Der Baukasten fängt einige Events, die durch Benutzereingaben ausgelöst werden, ab, um sie selbst zu bearbeiten oder weiterzugeben.

Vollkommen bearbeitet werden Events, die durch das Anklicken von einem der Buttons, durch Anwahl eines Geschwindigkeitswertes aus dem Choice oder durch Verschieben der Scrollbar entstehen. Daß diese Events nicht an die Anwendung weitergeleitet werden, dürfte klar sein, da nur baukasteninterne Funktionen ausgelöst werden.

Ebenso komplett bearbeitet wird die Eingabe einer Ziffer, mit der die Geschwindigkeit alternativ zum Choice gewählt werden kann. Ursprünglich sollte die Geschwindigkeitswahl durch Eingabe einer Ziffer die einzige Methode sein, mit der der Benutzer die Ablaufgeschwindigkeit beeinflussen kann. Leider stellte sich jedoch heraus, daß Tastendruck-Events nicht im Display-Objekt ankommen, wenn die Anwendung ein TextField verwendet (siehe auch Abschnitt 6.3 hierzu).

Im Gegensatz zu den bisher beschriebenen Events werden Events, die durch Drücken der Maustaste im Display-Objekt entstehen, nicht vollständig verarbeitet, sondern nach einer Vorverarbeitung an die Anwendung weitergegeben. Die Vorverarbeitung besteht in der Überprüfung, ob die Maustaste betätigt wurde, während sich die Maus über einer Nachricht befand. Sinn dieser teilweisen Bearbeitung mit anschließender Weiterleitung ist es, der Anwendung die Möglichkeit zu geben, auf das Anklicken einer Nachricht zu reagieren. Damit die Anwendung überprüfen kann, ob eine Nachricht angeklickt wurde, kann sie nach Erhalt des Events den Assistenten fragen, ob und wenn ja welche Nachricht beim letzten Klick getroffen wurde. Um hierüber jedoch Auskunft geben zu können, muß der Event vorher vom Baukasten soweit bearbeitet werden bis feststeht, was angeklickt wurde. Theoretisch könnte auch ohne diese Vorverarbeitung festgestellt werden, was angeklickt wurde, allerdings müßten dann die Koordinaten aus dem Koordinatensystem der Anwendung in das Koordinatensystem des Baukastens bzw. des Display-Objekts zurückgerechnet werden, was nicht ganz einfach sein dürfte.

4.4.4 Die Paint() und Update() Laufzeitmethoden

Neben den in den letzten drei Abschnitten beschriebenen Berührungspunkten des Baukastens mit dem Java-Laufzeitsystem existiert noch ein vierter, den der Anwendungsprogrammierer zwar nicht unbedingt kennen muß, der aber der Vollständigkeit halber hier noch aufgeführt werden soll. Es handelt sich dabei um die Implementierung der paint() und update() Methoden im Assistenten bzw. im Display-Objekt, was ein overloading der entsprechenden Methoden des Laufzeitsystems darstellt.

Das Display-Objekt ruft in seiner paint()-Methode einfach nur die paint() Methode des Assistenten auf. Diese stellt sicher, daß ein Darstellungsthread existiert und sagt diesem dann, daß er im Display-Objekt das aktuelle Bild erneut darstellen soll. Der Thread ist Teil des Assistenten um auf dessen Datenstrukturen zugreifen zu können, deshalb muß diese Weiterleitung vom Display-Objekt zum Assistenten-Objekt gemacht werden.

Das Display-Objekt überbügelt auch die update-Methode seiner Superklasse java.awt.Component, da diese ein Löschen der Darstellungsfläche macht, was zu einem kurzzeitigem Flackern auf dem Bildschirm führen kann. Um dies zu vermeiden, wird ein Aufruf der update() Methode einfach in einen Aufruf der paint()-Methode umgewandelt.

4.5 Die Handhabung von Nachrichten

In diesem Abschnitt sollen die wichtigsten Informationen über die unterstützten Nachrichtentypen, deren Verwendung und die interne Implementierung gegeben werden. Abschnitt 4.5.1 erklärt vorab, warum unterschiedliche Nachrichtentypen existieren und welche Typen vom Baukasten unterstützt werden. Abschnitt 4.5.2 erklärt die Auswirkungen auf die Programmstruktur anhand der drei Schritte, die eine Nachricht durchläuft. Im Abschnitt 4.5.3 wird beschrieben, wie trotz der unterschiedlichen Nachrichtentypen eine Implementierung ohne doppelten Code möglich ist. Abschnitt 4.5.4 beschreibt abschließend, warum einige der Nachrichten-Methoden Werte an die Anwendung zurückgeben, was im Rahmen des Baukastens eine Ausnahme darstellt.

4.5.1 Unterstützte Nachrichtentypen

Der Baukasten unterstützt mehrere Typen von Nachrichten aus zwei verschiedenen Gründen. Der erste Grund ist der, daß in verteilten Systemen unterschiedliche Nachrichtentypen existieren und der Baukasten hierfür eine Unterstützung bieten muß. Natürlich kann nicht jeder in irgendeinem Protokoll verwendete Nachrichtentyp unterstützt werden, weil dies den Rahmen einer Studienarbeit sprengen würde. Auf der anderen Seite ist es auch nicht nötig, daß jeder Unterschied zwischen zwei Nachrichten gleich zu verschiedenen Nachrichtentypen im Baukasten führen muß. Im Gegenteil, es ist meist unerheblich, von welchem Typ eine Nachricht ist. Teilweise wird der Typ auch durch das Protokoll implizit festgelegt. Beispielsweise ist die Unterscheidung zwischen UDP und TCP Paketen meist nicht nötig, da sich dies aus dem Zusammenhang ergibt, wenn der Unterschied überhaupt relevant ist.

Es gibt jedoch auch Nachrichtentypen die eine besondere Behandlung durch den Baukasten benötigen. Es handelt sich hierbei um den Unterschied, an wieviele Empfänger eine Nachricht gehen kann. Neben dem Normalfall, dem sogenannten Unicast, einer Nachricht von einem Sender an einen Empfänger, existieren noch zwei weitere Fälle, die Multicast- und die Broadcastnachricht.

Bei einer Multicast-Nachricht bestimmt der Sender eine Liste von Empfängern, an die die Nachricht durch das Transportsystem geschickt werden soll. Eine Broadcastnachricht hingegen wird an alle Empfänger verschickt, die sich mit dem Sender in einem logischen oder physikalischen Netz befinden. Im Gegensatz zum oben erwähnten UDP/TCP Unterschied benötigt dieser Unterschied eine spezielle Behandlung durch den Baukasten, da die Darstellung auf dem Bildschirm den Unterschied widerspiegeln muß.

Neben dem Grund, daß für Multi- und Broadcastnachrichten eine andere Darstellung angeboten werden muß, gibt es noch einen zweiten Grund, warum im Baukasten Nachrichten unterschiedlich behandelt werden. Es handelt sich hierbei um die in Abschnitt 3.4 beschriebene Unterstützung von unzuverlässigen Transportschichten, die unterschiedliche Nachrichtentypen erfordert. Eine Nachricht, die nie beim Empfänger ankommt, muß natürlich anders dargestellt werden als eine Nachricht, die ankommt oder eine Nachricht, die plötzlich auftaucht, ohne daß ein Sender dazu existiert. Auch die Veränderung des Nachrichteninhalts ist ein Unterschied, der sich in der Darstellung auswirkt und entsprechend einen besonderen Code im Baukasten benötigt.

Zusammenfassend bedeutet dies, daß für unterschiedliche Nachrichtentypen genau dann eine Unterscheidung im Baukasten gemacht werden muß, wenn die Darstellung auf dem Bildschirm eine Unterscheidung benötigt, die über die Verwendung unterschiedlicher Bilder für die Nachrichten hinaus geht.

4.5.2 Schritte einer Nachrichtenübermittlung

Die Übermittlung einer Nachricht besteht intern aus drei Arbeitsschritten. Die Anwendung sieht davon jedoch meist nur einen oder zwei Schritte. Wieviele Schritte die Anwendung machen muß, ist abhängig von der Art der Nachricht. Eine normale Nachricht kann durch einen Methodenaufruf erzeugt, versandt und empfangen werden. Nachrichten, die jedoch zur Darstellung von Protokollen mit möglichen Nachrichtenverfälschungen verwendet werden, müssen eine Nachricht durch den Aufruf von mehreren Methoden erzeugen, versenden und empfangen.

Die Idee hinter dieser mehrstufigen Nachrichtenübermittlung ist die, daß der Benutzer eine Änderung an der Nachricht vornehmen können soll. Deshalb wird im ersten Schritt die Nachricht nur über die Hälfte der Distanz animiert dargestellt. Danach kann der Benutzer eine Veränderung vornehmen, worauf die Nachricht über die zweite Hälfte der Distanz animiert dargestellt wird, sofern die Nachricht nicht verloren gehen soll.

Um die Veränderung durch den Benutzer zwischen dem Absenden und dem Empfangen der Nachricht zu ermöglichen, besteht eine Nachrichtenübermittlung intern aus den folgenden drei Schritten:

* Im ersten Schritt wird ein Nachrichtenobjekt erzeugt und dessen Variablen entsprechend dem aktuellen Zustand gesetzt.
* Im zweiten Schritt wird die Nachricht über die erste Hälfte der Distanz animiert dargestellt. Dies entspricht dem Versenden der Nachricht durch den Sender.
* Der dritte Schritt macht mit der Animation der zweiten Hälfte des Weges die Nachrichtenübermittlung vollständig. Dies entspricht dem Empfang der Nachricht durch den Empfänger.

Wann welche Schritte durch den Aufruf welcher Methoden ausgeführt werden, kann der Tabelle 4.2 entnommen werden

Tabelle 4.2: Funktionen der einzelnen Nachrichtenmethoden

		Schritt			
Fall	Methoden	1	2	3	Bemerkung
1	versendeUnd-Empfange-*Nachricht	ja	ja	ja	Der Normalfall für unveränderte Nachrichten, bei denen alle drei Schritte durch einen Methodenaufruf gemacht werden. * kann dabei für einen leeren String, für „Multicast“, „Broadcast“ oder „BroadcastOhneSender“ stehen.
2	versende-*Nachricht	ja	ja	nein	Wird von der Anwendung aufgerufen, wenn noch nicht bekannt ist, was mit der Nachricht geschehen soll, da dies der Benutzer noch entscheiden muß. * kann die Werte von Fall 1 annehmen
3	empfange-Unveraenderte-Nachricht	nein	nein	ja	Wird von der Anwendung verwendet, um eine Nachricht unverändert ankommen zu lassen
4	empfange-Geaenderte-Nachricht	nein	nein	ja	Führt eine Nachrichtenänderung durch und läßt die Nachricht beim Empfänger ankommen
5	verliere Nachricht	nein	nein	ja	Dient der Anwendung, um einen Verlust der Nachricht darzustellen. Auf der zweiten Weghälfte wird keine Animation mehr gemacht, es wird nur Platz gelassen als ob die Nachricht ankommen würde.
6	verzoegere-Empfang-Nachricht	nein	nein	nein	Ein Spezialfall der benötigt wird, wenn eine Nachricht eine andere überholen soll. Diese Methode führt keinen der drei Schritte aus sondern verändert die Nachricht intern. Durch die Veränderung wandert das die Nachricht darstellende Image ab sofort dem aktuellen Zeitpunkt folgend parallel zur Zeitachse nach unten. Die den Nachrichtenweg darstellende Linie paßt sich dieser Verschiebung an.

Tabelle 4.2: Funktionen der einzelnen Nachrichtenmethoden

		Schritt			
Fall	Methoden	1	2	3	Bemerkung
7	empfange-Geaenderte-Multicast-Nachricht	nein	nein	ja	Entspricht den Fällen 3-6, jedoch für Multicast- und Broadcastnachrichten. Die Zusammenfassung zu einer Methode wurde gewählt, damit an den einzelnen Nachrichten gleichzeitig unterschiedliche Änderungen gemacht werden können. Würde dies in einzelnen Methoden ähnlich denen in 3-6 gemacht werden, so würde die Darstellung und das interne Zeitsystem durch das mehrmalige Ausführen des dritten Schritts beim Nacheinanderaufrufen der Einzelmethoden durcheinandergeraten. Die Art der Veränderung ist in einem Parameter für jede einzelne Nachricht kodiert. Siehe dazu die Beschreibung in Anhang A.
8	empfange-Nirvana-Nachricht	ja	ja	ja	Spezialfall um eine aus dem Nichts auftauchende Nachricht bei einem Empfänger ankommen zu lassen. Auf der ersten Weghälfte wird nichts dargestellt, die Nachricht erscheint nach der Hälfte und wird dann animiert dargestellt, bis sie beim Empfänger ankommt.

4.5.3 Methoden-Wrapping und Overloading

Die in Abschnitt 4.5.2 beschriebenen drei Schritte werden letztlich von nur zwei Methoden, der erzeugeMulticastNachricht und der sendeHalbenWegMulticastNachricht, erledigt. Dies geschieht dabei unabhängig davon, ob es sich um eine Unicast-, eine Multicast- oder eine Broadcastnachricht handelt. Entsprechend führen alle Methoden, die von der Anwendung aufgerufen werden, intern zu Aufrufen der beiden eigentlichen Arbeitsmethoden. Eventuell zusätzlicher Code dient dabei nur der Datenkonvertierung oder dem Setzen einiger Variablen.

In Ermangelung eines passenden deutschen Begriffs sollen die nach außen sichtbaren Methoden als Wrapper um die eigentlichen Arbeitsmethoden bezeichnet werden. Dieses Wrapping geschieht mehrfach und zu unterschiedlichen Zwecken, wodurch sich eine Verschachtelung dieser Wrapper ergibt.

Zum einen sind alle Unicast- und Broadcast-Methoden Wrapper um die entsprechenden Multicast-Methoden, da sie Spezialfälle dieser sind. Zum anderen sind die Multicast-Methoden Wrapper um die erwähnten beiden Arbeitsmethoden.

Abbildung 4.7 zeigt all diese Aufrufverkettungen.

Zusätzlich zu den Wrappern sind einige Methoden zweimal definiert, wobei sich die Methoden nur in ihren Parametern unterscheiden. Diese als Overloading bezeichnete Programmierung wurde bei allen Methoden angewandt, bei denen ein Name für das Bild der Nachricht angegeben werden kann. Wenn kein solcher Name angegeben wird, dann wird der Name, mit

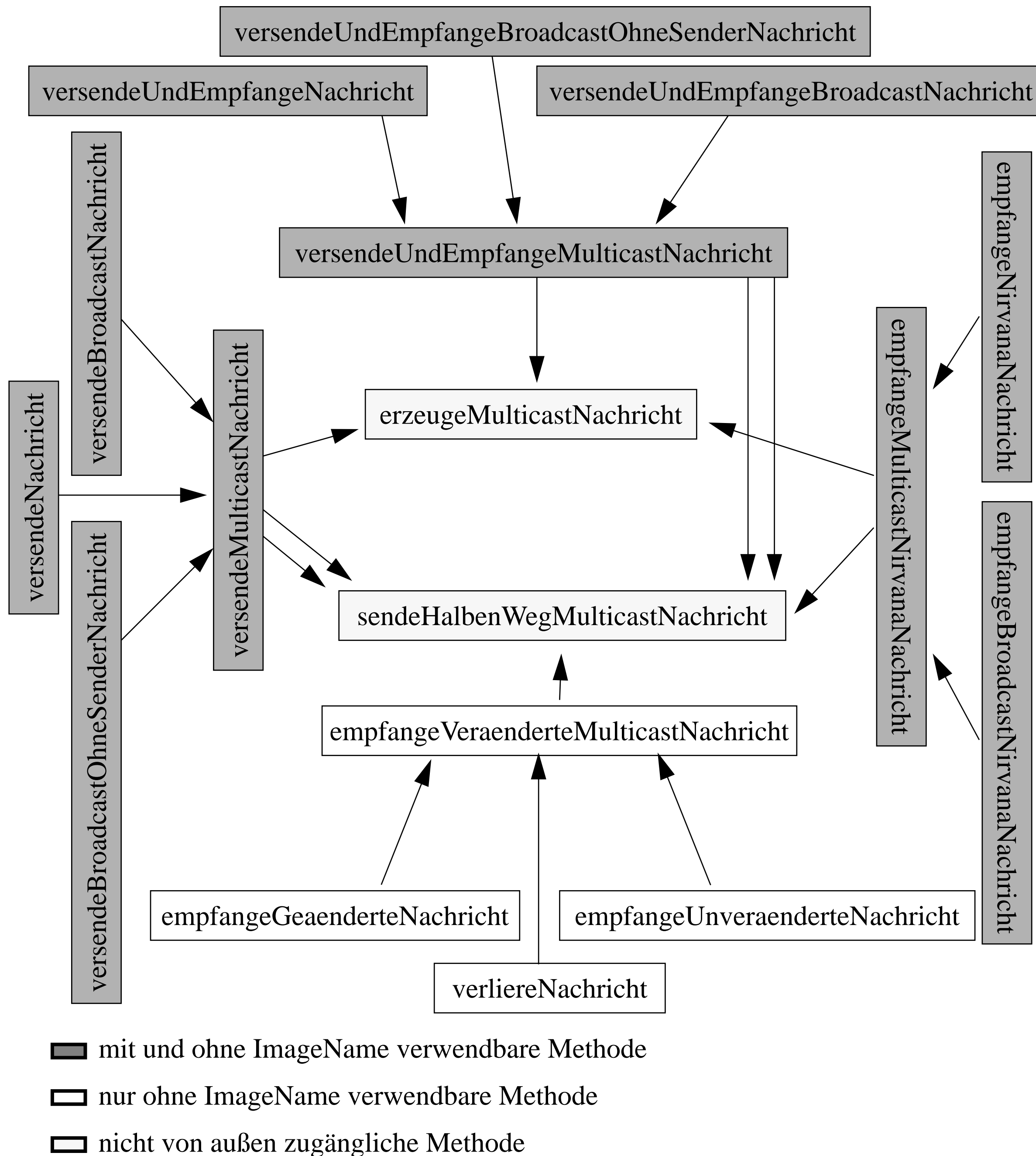

Abbildung 4.7: Aufrufverkettungen

dem die Nachricht identifiziert wird, als Name für das Bild verwendet. Der Anwendungsprogrammierer erspart sich somit oftmals die Angabe eines unnötigen Parameters, was die Anwendung des Baukastens etwas bequemer macht.

Letztlich sind auch die Wrapper nichts anderes als Mittel um die Anwendung des Baukastens für den Anwendungsprogrammierer bequemer zu gestalten. Theoretisch könnten alle Methoden, die Unicast- oder Broadcastnachrichten erzeugen, versenden oder empfangen komplett weggelassen werden, da die Anwendung mit den entsprechenden Multicast-Methoden dasselbe Ergebnis erzielen kann, wenngleich mit etwas höherem Aufwand.

Um es auf einen Nenner zu bringen, kann man sagen, daß das in diesem Abschnitt beschriebene Wrapping und Overloading nur dazu dient, die Schnittstelle des Baukastens benutzerfreundlicher zu gestalten.

4.5.4 Rückgabewerte der Nachrichten-Methoden

Der Baukasten besteht aus drei Gruppen von Methoden. Die erste und größte Gruppe empfängt Werte von der Anwendung über ihre Parameter und liefert keine Werte an die Anwendung zurück. Ein Beispiel hierfür ist die pause-Methode, die die Länge der Pause von der Anwendung als Parameter bekommt und keinen Wert zurückliefert. Dies ist als Normalfall für einen Baukasten wie diesen zu betrachten. Das Besondere an diesem Baukasten ist ja, daß er nicht Bestandteil des darzustellenden Protokolls, sondern quasi nur ein außenstehender Beobachter ist.

Als solcher sollte er natürlich möglichst wenig Punkte haben, an denen er die Anwendung in Form von zurückgelieferten Ergebnissen beeinflußt. Dies ist als Besonderheit der Aufgabenstellung zu betrachten und ist für andere Aufgaben und Baukästen sicher nicht immer gegeben. Entsprechend ist es für andere Baukästen durchaus korrekt, wenn die Mehrheit der Methoden Ergebnisse liefern, für unseren Baukasten wäre dies aber ein Fehler.

Die nächst größere Gruppe von Methoden ist die, die keine Parameter erwarten, dafür aber Informationen über den aktuellen Zustand des Baukastens zurückliefern. Ein Beispiel hierfür ist die Methode aktuelleUhrzeit. Diese nimmt keine Werte von der Anwendung entgegen, liefert aber den Wert der internen Zeit an die Anwendung zurück. Da die gelieferten Informationen meist rein informativen Charakter haben und in den Methoden keine sonstige Arbeit verrichtet wird, entsteht hierdurch auch kein Widerspruch zu dem oben beschriebenen Nichtbeeinflussungs-Kriterium.

Bei der letzten Gruppe hingegen ergibt sich ein solcher Widerspruch, da die Methoden dieser Gruppe sowohl Werte empfangen und damit Arbeit verrichten, als auch Werte zurückliefern. Aufgrund der geforderten Unterstützung allgemeiner Protokolle mit Fehlerfällen wurden diese Methoden mit Ergebnissen leider notwendig.

Die Methoden, die Nachrichten erzeugen und versenden, jedoch nicht gleich auch für das Empfangen der Nachrichten auf Empfängerseite sorgen, müssen Informationen über die erzeugten Nachrichten an die Anwendung zurückgeben, damit diese in späteren Aufrufen an den Baukasten auf die erzeugten Nachrichten Bezug nehmen können. Das heißt, daß die in Fall Zwei in Tabelle 4.1 genannten Methoden Strings oder Felder von Strings an die Anwendung zurückliefern.

Diese zurückgelieferten Strings muß sich die Anwendung merken, wenn sie mit den Nachrichten noch etwas anfangen will, was zu erwarten sein dürfte. Der Inhalt der Strings dient zur eindeutigen Identifizierung der Nachrichten und besteht deshalb aus den folgenden drei Teilen: Der erste Teil ist der beim Aufruf der Methode übergebene BaseName. Da dies für Multicast- oder Broadcastnachrichten nicht eindeutig ist, fügt der zweite Teil die Sender und Empfängernamen hinzu. Da auch dies bei mehreren hintereinander ausgeführten Multicast oder Broadcastnachrichten nicht eindeutig sein muß, wird im dritten Teil die intern verwendete Zeit angehängt. Getrennt werden die drei Teile durch Doppelpunkte, zwischen Sender und Empfänger steht ein Pfeil „->“.

Kapitel 5

Beispielanwendungen

Dieses Kapitel beschreibt die im Rahmen dieser Studienarbeit erstellten Beispielanwendungen des Baukastens. Im einzelnen sind dies die Fakultätsberechnung nach Actors, ein einfacher RPC zwischen zwei Prozessen, ein Trader- und ein Broker-Beispiel, ein Dienstreiseantrag als Beispiel für interagierende Objekte, eine Terminaushandlung und ein Beispiel für Nachrichtenverfälschungen.

Nur in Abschnitt 5.1 muß auch der verwendete Algorithmus genauer beschrieben werden, da es sich dabei nicht nur um eine Hintereinanderreihung von Baukastenaufrufen handelt. Vielmehr ist die Fakultätsberechnung ein voll lauffähiges Programm, das einen existierenden Algorithmus implementiert.

Bei den anderen Beispielen wird hingegen das darzustellende Protokoll nur simuliert und leistet keine reale Arbeit. Die Beispielanwendungen dienen also ausschließlich der Animation der Protokolle.

5.1 Fakultätsberechnung nach Actors

Dieser Beispielanwendung liegt folgender Actors-Code zugrunde (aus [Borghoff]):

```
(define (Fak())
        (Is-Communication
                 (a doit (with customer = m)(with number = n)) do
                 (become Fak)
        (if (=n 0) (then (send m 1))
        (else (let (x=(new FakCust(with customer m)(with number n)))
                 (send Fak(a do(with customer x)(with number n-1))))))))
(define (FakCust(with customer = m) (with number = n))
        (Is-Communication (a number k) do ( send m n*k)))
```

Der Code implementiert die Fakultätsberechnung in Actors.

Aufgabe dieser Beispielanwendung war es nun, den Algorithmus, nach dem dieser Code die Fakultät einer Zahl berechnet, animiert darzustellen. Um nun nicht nur den Ablauf des Algorithmus bei der Berechnung einer ganz bestimmten und vorher festgelegten Zahl darstellen zu können, wurde der Algorithmus in Java implementiert.

Dazu mußte eine Art Übersetzung des Actors-Codes vorgenommen werden. Beachtet werden mußte dabei, daß die wesentlichen Merkmale des Algorithmus bei dieser Übersetzung nicht verloren gehen. Das Hauptmerkmal des Originalcodes ist der, daß die Fakultätsberechnung einer Zahl n durch n+2 Aktoren geschieht.

Der Aktor A0 ist dabei nur der Auftraggeber und Empfänger des Ergebnisses und ist somit nicht an der Berechnung selbst beteiligt.

Aktor A1 hingegen dient als Arbeitsverteiler. Die Verteilung erfolgt dabei nach folgendem Schema:

```
Solange die noch zu berechnende Zahl N größer als Null ist, solange erzeuge
einen neuen Aktor, sage diesem die Zahl N und den Namen des davor zuletzt
erzeugten Aktors oder „A0“, wenn dies der erste Durchlauf ist. Danach sage
dir selbst, daß N nun um eins kleiner ist. Wenn N Null ist, dann sage dem
zuletzt erzeugten Aktor, daß das bisher berechnete Ergebnis 1 ist.
```

Die restlichen Aktoren tun nun nichts anderes, als sich die vom Aktor A1, direkt nach ihrer Erzeugung, übermittelte Zahl zu merken und diese, nachdem ihnen ein anderer Aktor ein bisher berechnetes Ergebnis geliefert hat, mit diesem zu multiplizieren und an den genannten Aktor als neues berechnetes Ergebnis weiterzumelden.

Letztlich ergibt sich das Ergebnis, in dem die Zahlen n bis 1 an n verschiedene Aktoren verteilt werden und diese dann nacheinander die Zahlen in umgekehrter Reihenfolgen, also von 1 bis n, miteinander multiplizieren.

Der als allerletztes erzeugte Aktor kennt somit die Fakultät von 1, der vorletzte Aktor die Fakultät von 2, der davor die Fakultät von 3 und so weiter.

Abbildung 5.1 zeigt die Darstellung des Original-Actors-Beispiels der Fakultät von 2.

Abbildung 5.2 zeigt anschließend die Darstellung der Fakultät von 3 mit dem entwickelten Java-Anwendungsbeispiel und dem Baukasten nach Ablauf der Animation.

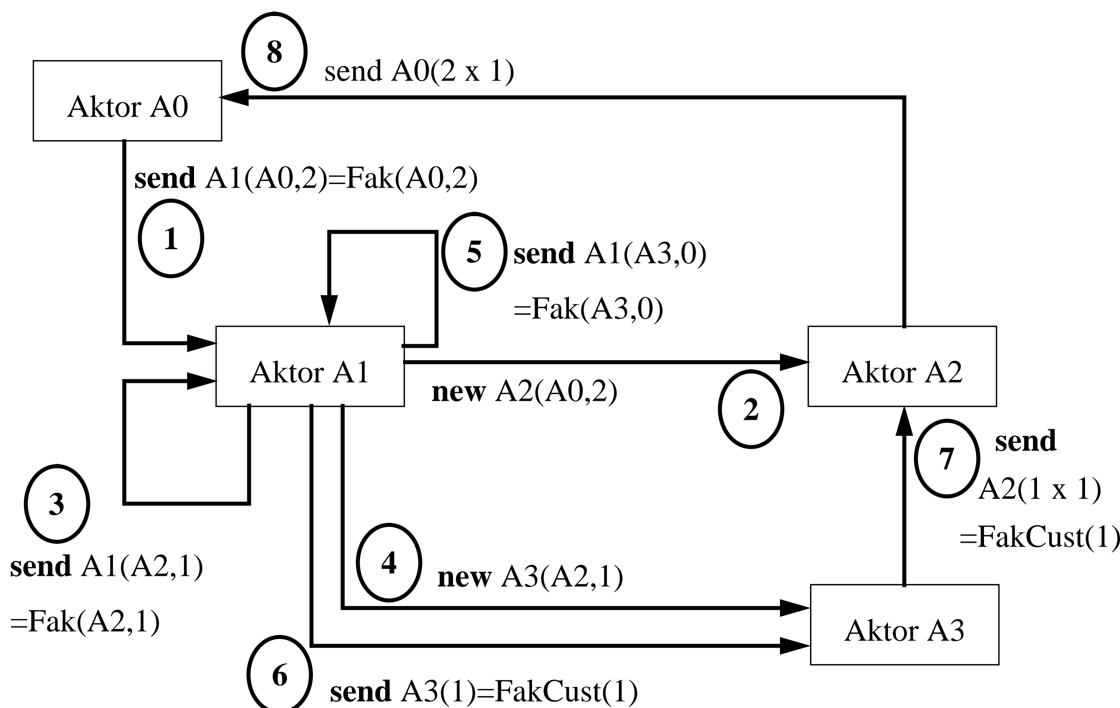

Abbildung 5.1: Berechnung von 2! mit Actors

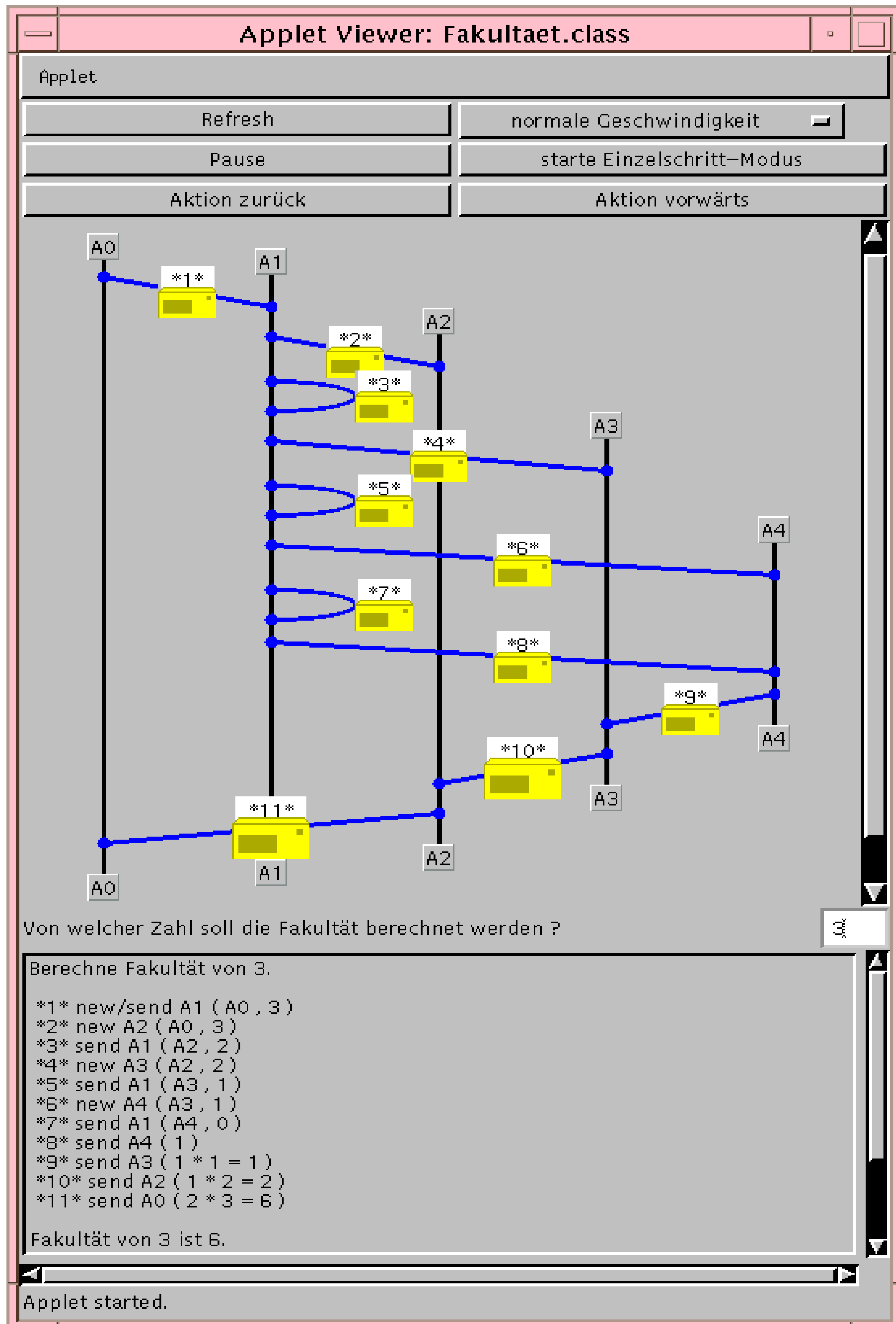

Abbildung 5.2: Berechnung von 3! nach Actors durch den Baukasten dargestellt

5.2 Einfaches RPC-Beispiel

Diese Beispielanwendung stammt aus den Anfängen der Verteilten Systemen. Der Remote Procedure Call (RPC) war einer der ersten Schritte weg vom Einzelsystem und hin zu vernetzten und verteilten Systemen. Daß der RPC-Call trotzdem bis heute einer der am häufigsten verwendeten Mechanismen in Verteilten Systemen ist, liegt höchstwahrscheinlich daran, daß seine Existenz eng an die Existenz von TCP/IP und UNIX gebunden ist. Da sowohl TCP/IP als auch UNIX langezeit die de-facto-Standards im Internet waren und noch sind, gehört auch der rpc zu diesen klassischen Internet Standards.

Diese Beispielanwendung stellt nun einen solchen gewöhnlichen Remote Procedure Call animiert dar. Es handelt sich dabei um das klassische Frage-Antwort-Spiel zwischen einem Client und einem Server. Der Client stellt in der ersten Nachricht eine Anfrage an den Server und erhält von diesem in der zweiten Nachricht eine Antwort zurück.

Das Endergebnis der Darstellung auf dem Bildschirm ist in Abbildung 5.3 wiedergegeben.

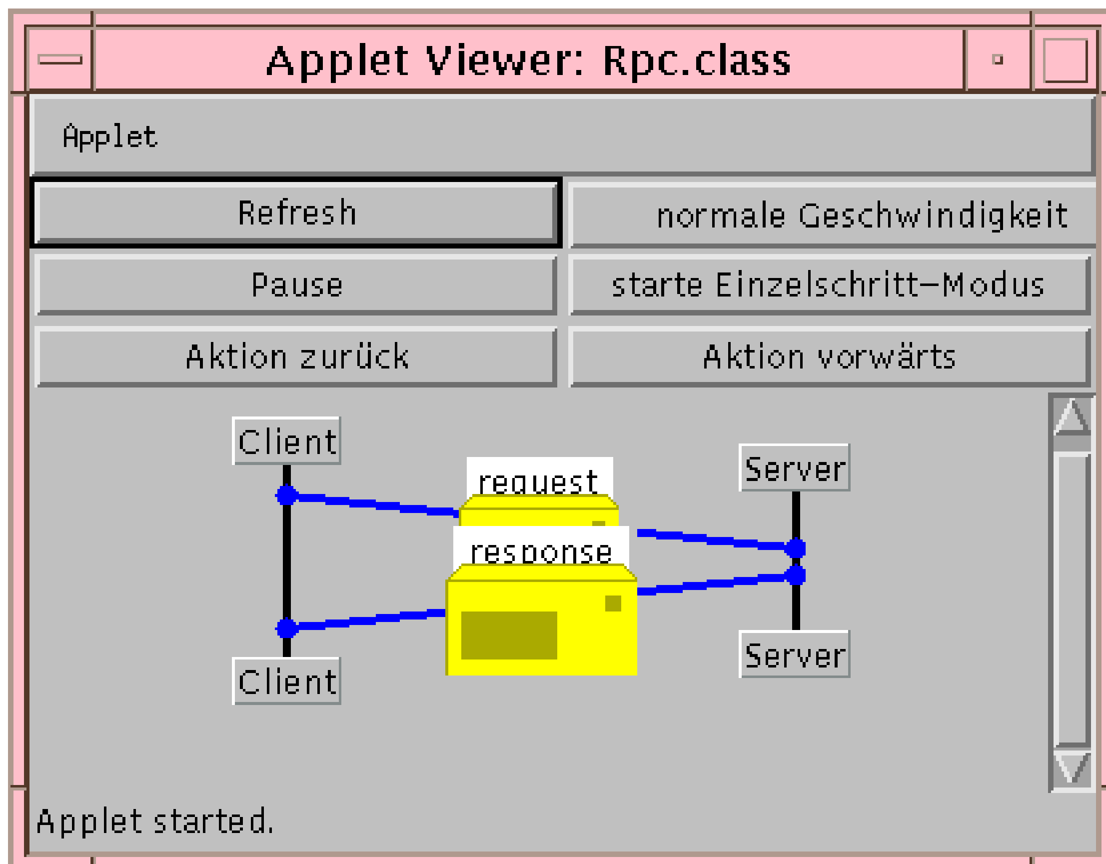

Abbildung 5.3: Einfacher RPC-Call mit dem Baukasten dargestellt

5.3 Trader- und Broker-Beispiele

Der im letzten Abschnitt behandelte klassische RPC setzt voraus, daß der Client genau weiß, an welchen Server er seine Anfrage zu richten hat. In größeren Verteilten Systemen mit vielen Servern stellt dies jedoch ein Problem dar. In solchen Systemen ist es nicht ungewöhnlich, daß mehrere Server einen Dienst anbieten und der Client sich zur Laufzeit entscheiden muß, an welchen dieser Server er seine Anfrage nun stellt. Kompliziert wird dies unter anderem dadurch, daß die Menge der verwendbaren Server, durch das Starten und Beenden von Serverprozessen, dynamisch wachsen und schrumpfen kann.

Zur Lösung dieses Problems werden in Verteilten Systemen oftmals Dienstevermittler, sogenannte Trader oder Broker, eingesetzt. Diese übernehmen die dynamische Zuordnung zwischen den Anfragen des Clients und den Servern, die diese Anfrage beantworten können.

Die Arbeitsweise von Tradern und Brokern ist prinzipiell sehr ähnlich, weist jedoch einen wichtigen Unterschied auf: Im Gegensatz zu einem Trader, der nur Informationen über die ihm bekannten Server an den Client liefert, leitet ein Broker die Anfrage des Clients direkt an einen Server weiter und liefert dessen Ergebnis an den Client zurück. Die Abbildungen 5.4 und 5.5 zeigen diesen Unterschied recht deutlich.

In beiden Fällen müssen sich die Server erst beim Trader oder Broker anmelden, bevor diese deren Dienste vermitteln können. Im Trader-Beispiel erhält der Client, in der Abbildung Importeur genannt, vom Trader nur die Information, welcher Server die Anfrage beantworten kann. Für das Stellen der Anfrage ist der Client dann selber zuständig. Hingegen richtet der Client im Broker-Beispiel seine Anfrage nur an den Broker, der diese mit Hilfe der ihm bekannten Server beantwortet.

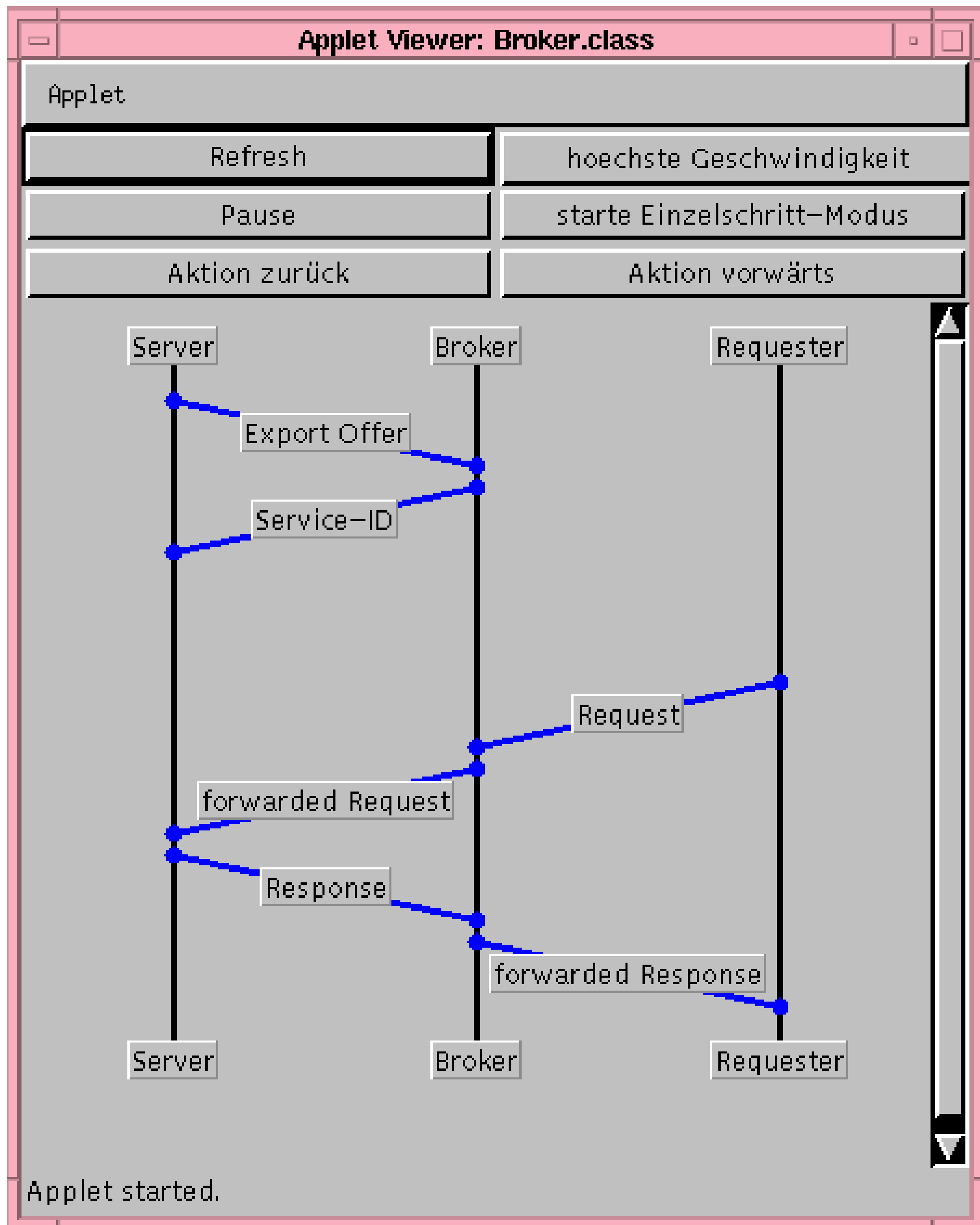

Abbildung 5.4: Beispiel eines Brokers

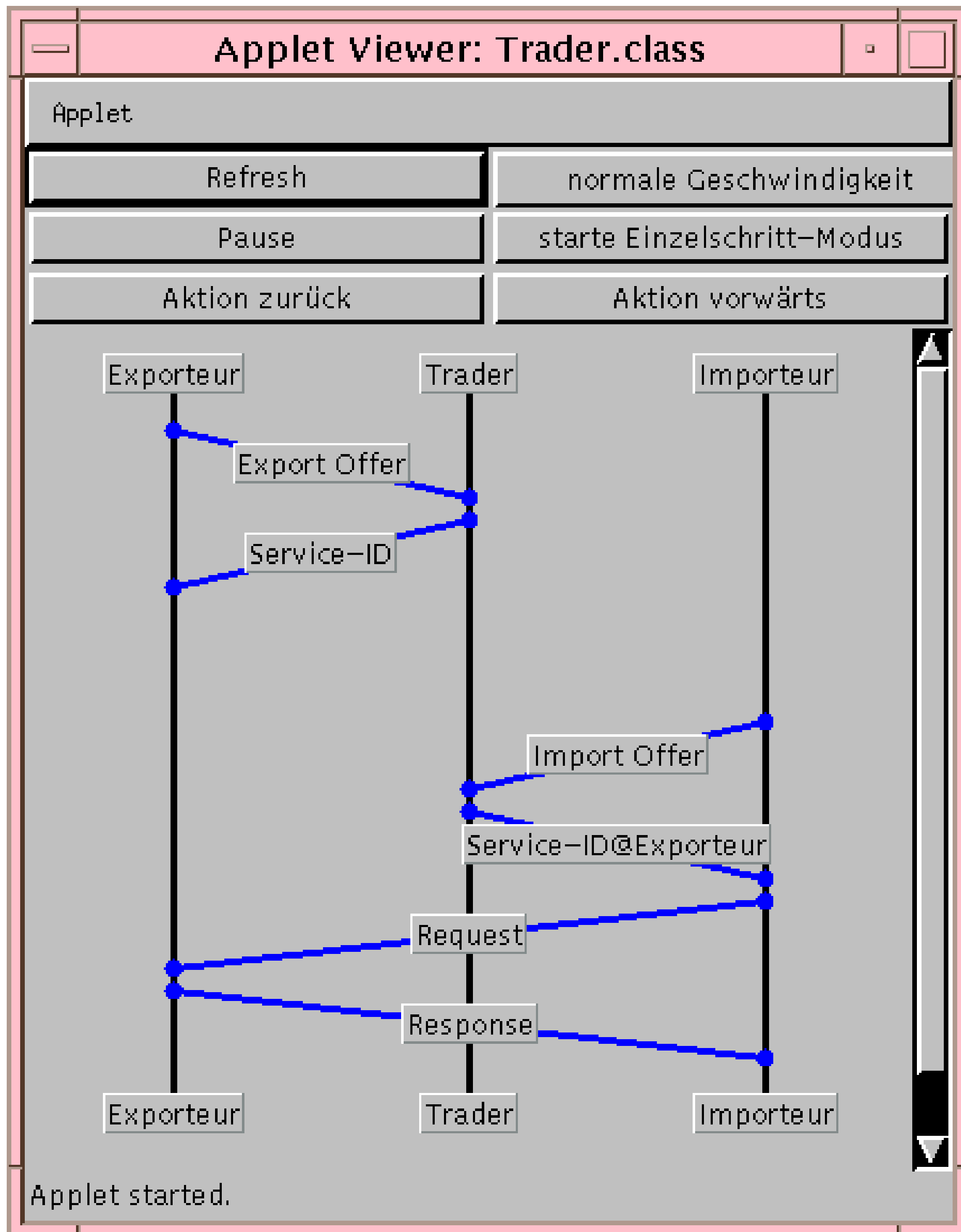

Abbildung 5.5: Beispiel eines Traders

5.4 Beispiel von interagierenden Objekten

Als Beispiel für die in der Vorlesung „Kooperation in Verteilten Systemen" behandelten interagierenden Objekte, wird hier das Modell einer Dienstreise dargestellt.

Die Objekte, die hierbei miteinander interagieren, sind zum einen Personen einer Abteilung und zum anderen Abteilungen als ganzes.

Es wird der Werdegang eines Dienstreiseantrags und der dazugehörenden Reiseabrechnung dargestellt. Der Reiseantrag des Mitarbeiters wird von diesem an die Sekretärin der Abteilung geleitet. Diese legt dem Abteilungsleiter, hier kurz Chef genannt, den Antrag zum Unterschreiben vor und leitet danach den genehmigten Antrag an den Mitarbeiter zurück. Dieser stellt nach seiner Reise einen Antrag auf Reisekostenabrechnung, die wieder über die Sekretärin zur Unterzeichnung durch den Chef gelangt. Mit der unterzeichneten Reisekostenabrechnung kann die Sekretärin bei der Buchhaltung die Auszahlung der beantragten Reisekosten veranlassen. Nach der Auszahlung gehen die gesammelten Unterlagen ins Archiv zur Einlagerung.

Abbildung 5.6 zeigt das hier beschriebene Dienstreiseszenarium durch den Baukasten dargestellt.

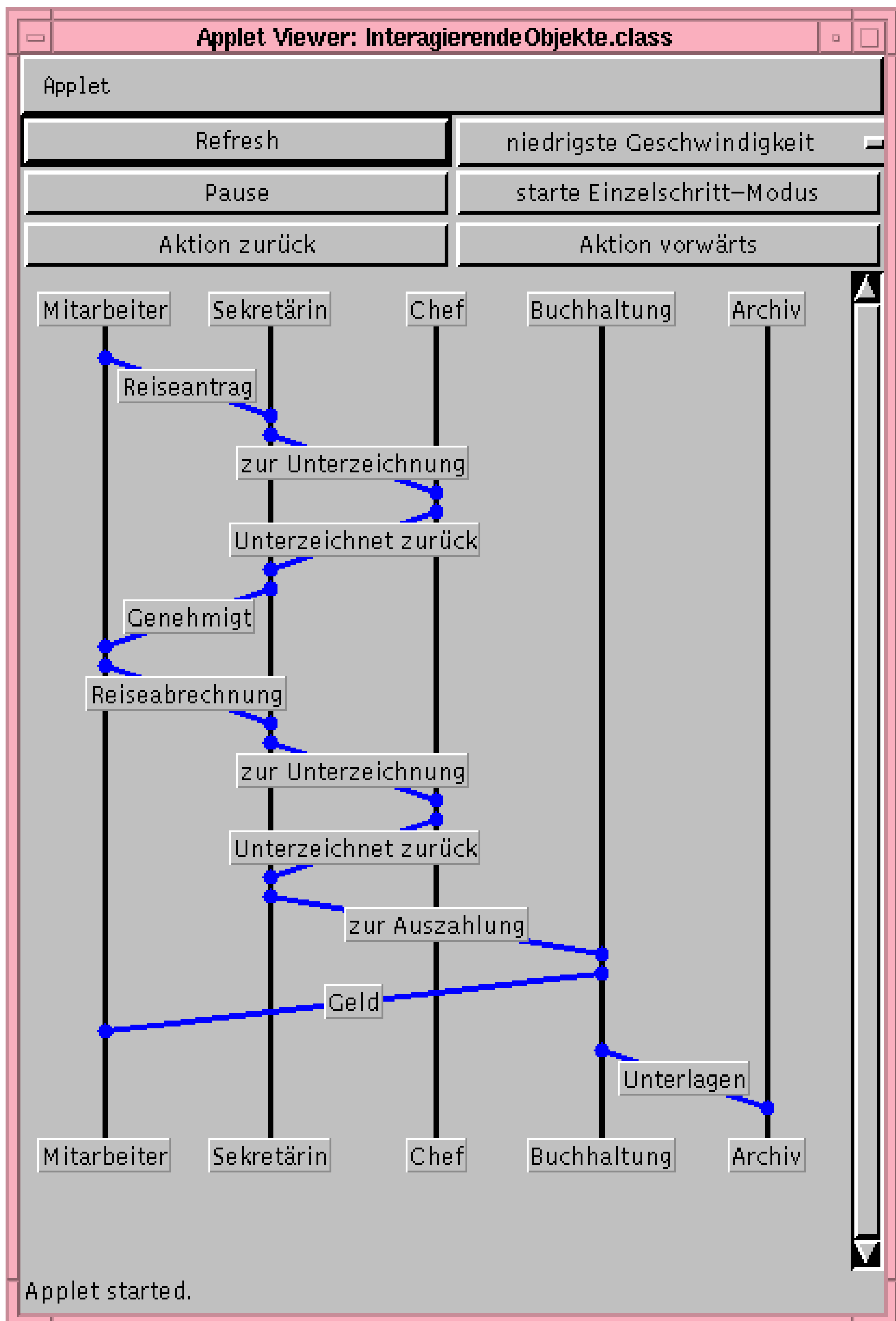

Abbildung 5.6: Verwaltung einer Dienstreise

5.5 Terminaushandlungs-Beispiel

In der Vorlesung „Kooperation in Verteilten Systemen“ wird die Terminplanung mit Hilfe von Agenten als Beispiel für asynchrone kooperative Anwendungen behandelt.

Die Agenten vertreten dabei Personen, die sich zu einem Treffen verabreden möchten.

In der Vorlesung wurde die Aushandlung des Termins durch die Agenten an dem in Abbildung 5.7 gezeigten Zustandsübergangs-Diagramm gezeigt.

Die Abbildung 5.8 zeigt einen möglichen Ablauf dieser Terminaushandlung. Der Programmcode des Applets hierzu ist in Anhang B abgedruckt.

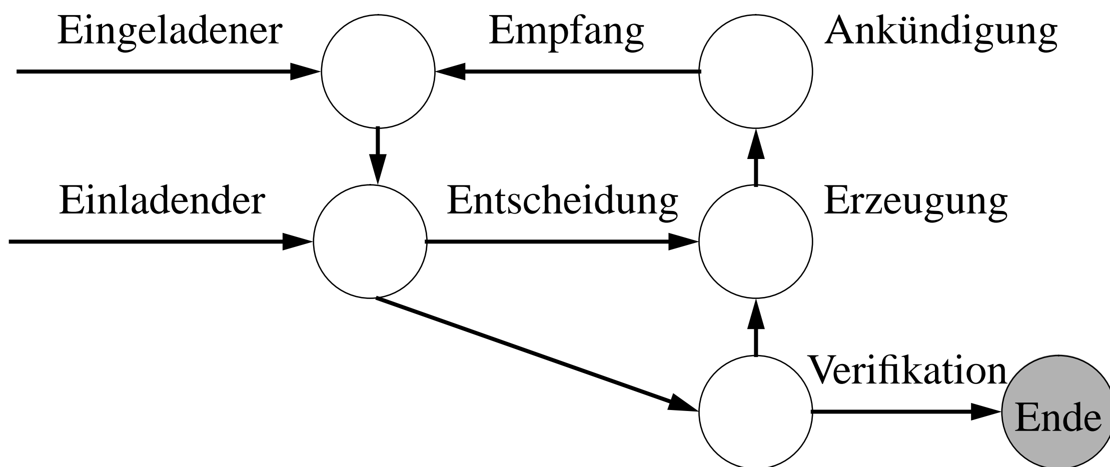

Abbildung 5.7: Terminaushandlungs-Schaubild aus Vorlesung

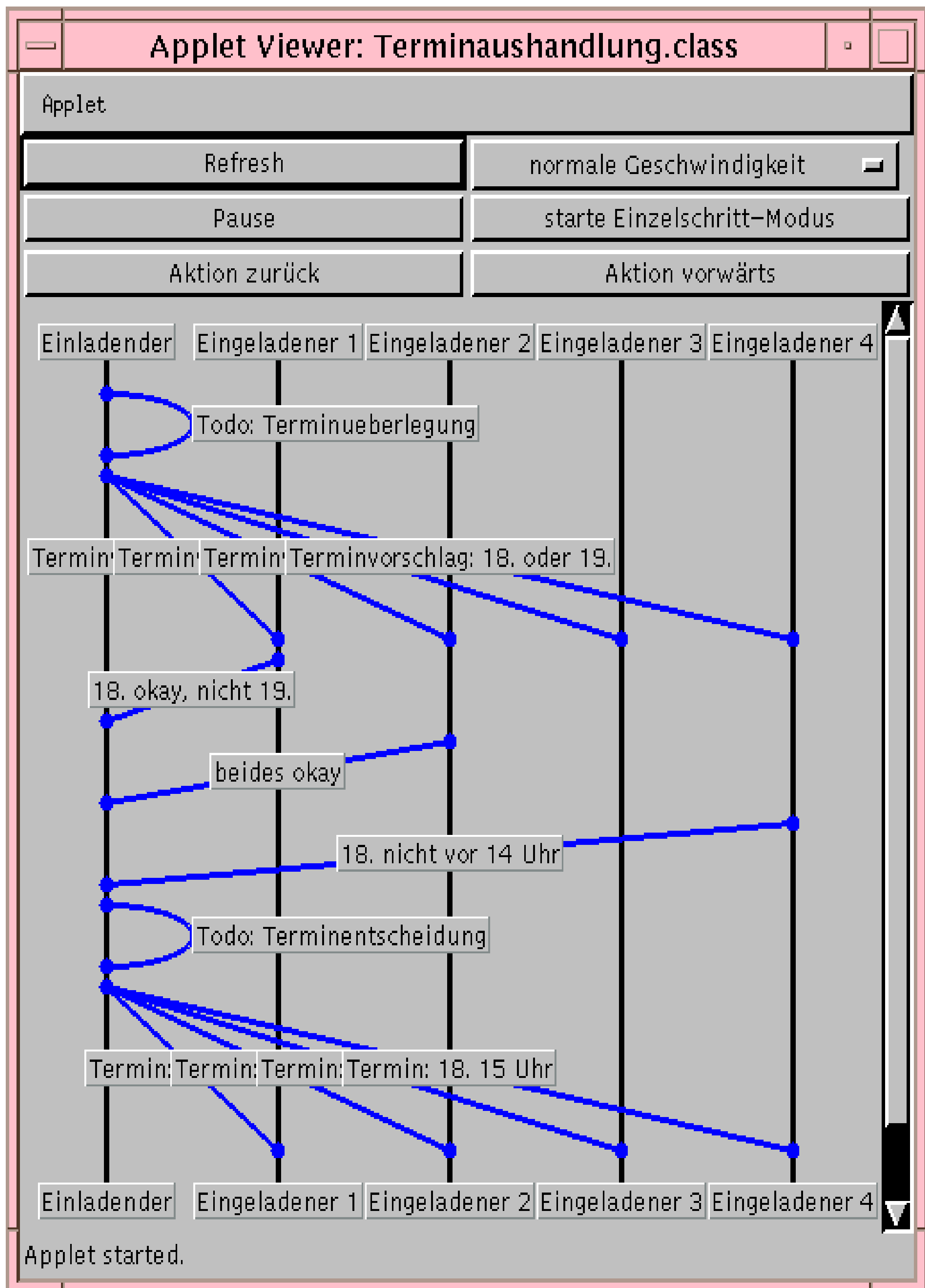

Abbildung 5.8: Terminaushandlung mit dem Baukasten dargestellt

5.6 Beispiel für unzuverlässige Kommunikationskanäle

Dieses Anwendungsbeispiel zeigt die vom Baukasten unterstützten Nachrichtenmanipulationen. Im einzelnen sind dies der Verlust von Nachrichten, das Auftauchen von zusätzlichen Nachrichten, die Veränderung von Nachrichten und die Verzögerung von einzelnen Nachrichten.

Hierzu wird eine fiktive Kommunikation zwischen einem Sender und einem Empfänger in Abbildung 5.9 dargestellt. Der Sender verschickt nacheinander die Nachrichten „Msg 1“, „Msg2“, „Msg 3“ und „Msg 4“. Dieser Nachrichtenfluß wird durch einen unzuverlässigen Kommunikationskanal während der Übertragung auf verschiedene Weisen verfälscht.

So kommt die als erstes abgesendete Nachricht „Msg 1“ als letzte beim Empfänger an und wird somit von den anderen Nachrichten „überholt“. Nachricht „Msg 2“ ist in diesem Beispiel die einzige korrekt übertragene Nachricht. Nachricht „Msg 3“ geht bei der Übertragung verloren und kommt somit nie beim Empfänger an. Dafür erhält dieser eine Nachricht mit der Bezeichnung „Nirvana“. Diese wurde jedoch nie vom Sender verschickt. Nachricht „Msg 4“ wird während der Übertragung inhaltlich verändert und kommt als „Fake Msg 4“ beim Empfänger an.

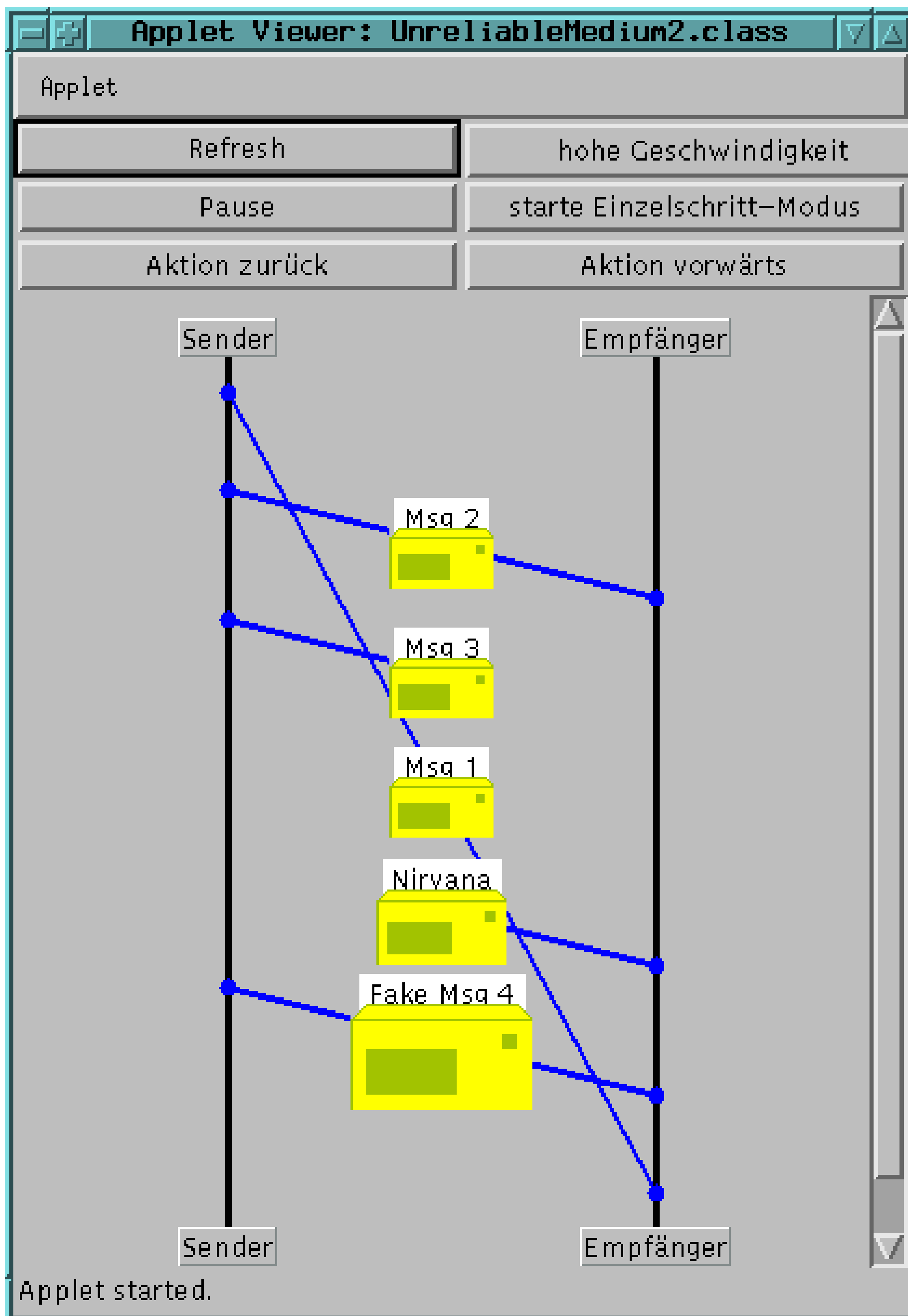

Abbildung 5.9: Beispiele für Nachrichtenverfälschungen

Kapitel 6

Probleme mit der Sprache Java

In diesem Kapitel sollen einige Probleme geschildert werden, die im Verlauf dieser Studienarbeit unangenehm auffielen. Die Abschnitte 6.1 und 6.2 schildern dabei Probleme, die als fehlende Funktionalitäten der java.awt-Klassen bezeichnet werden könnten. Abschnitt 6.3 beschreibt ein Verhalten von java.awt.TextField, das als Bug betrachtet werden kann oder muß. Abschnitt 6.4 spricht den Bereich der Skalierbarkeit von Java anhand zweier Überlastungserscheinungen des Java-Laufzeitsystems an. In Abschnitt 6.5 sei ein kleines Resümee über die Verwendbarkeit von Java aus Sicht des Autors gestattet.

6.1 Transparenzprobleme

Dieser Abschnitt beschreibt ein Problem, das während der Entwicklung des Baukastens gleich an zwei Punkten etwas unangenehm auffiel. Das erste Mal fiel eine fehlende Transparenz ganz am Anfang der Studienarbeit bei der Entwicklung des Prototyps auf. Ein allererster Ansatz war damals das Zusammensetzen der Darstellung durch Überlagerung von Standard-Graphik-Objekten oder Subklassen davon. Ziel war es, wie für einen Prototypen nicht anders zu erwarten ist, auf möglichst einfachem Weg zu der gewünschten Darstellung zu gelangen.

Am einfachsten erschien es, jeden Prozess und jede Nachricht als eigenständiges Graphik-Objekt zu implementieren. Die Objekte hätten dabei jeweils nur die paar Linien oder Bilder dargestellt, aus denen ihre Darstellung auch am Ende der Studienarbeit besteht. Im Gegensatz zur nun vorliegenden Implementierung hätten sie die Linien und Bilder aber nicht in zentral verwaltete Images gezeichnet, sondern jedes Objekt hätte die Darstellung über die von seinen Superklassen ererbten Graphikmethoden direkt auf den Bildschirm gemacht.

Durch Übereinanderlegen dieser Objekte hätte sich dann das Gesamtbild ergeben können. Dazu müßte aber der nicht durch Linien oder Bilder belegte Bereich transparent sein, damit die Linien und Bilder der darunterliegenden Objekte durchscheinen könnten. Leider sind jedoch alle java.awt Objekte vollkommen undurchsichtig, so daß immer nur die zuoberst liegenden Objekte sichtbar sind. Daß bei der Entwicklung von java.awt überhaupt nicht an transparente Objekte gedacht wurde, äußert sich auch darin, daß die Layoutmanager von java.awt keinerlei Überlappung von Objekten ermöglichen. Im Gegenteil, alle Layoutmanager sind bestrebt, ja keine Überlappungen zuzulassen, sondern Objekte immer schön neben- oder übereinander anzuordnen.

Das Verhalten der Layoutmanager ist natürlich absolut korrekt und äußerst konsequent unter der Prämisse, daß es keine transparenten Objekte gibt. Nichtsdestotrotz wären transparente Objekte sicher an manchen Stellen sinnvoll, und es wäre schön, wenn Java dies nicht so unmöglich machen würde wie dies momentan der Fall ist.

Das zweite Mal fiel eine fehlende Transparenz gegen Ende der Studienarbeit, als einige kleinere Verbesserungen der Darstellung gemacht werden sollten, unangenehm auf. Unter anderem wurde dabei daran gedacht, mehrzeilige Texte in Prozessnamen oder Nachrichteninhalten zuzulassen. Damit dabei dann nicht für alle Zeilen der von der längsten Zeile benötigte Platz auf dem Bildschirm in Anspruch genommen wird und somit unnötig viel Fläche verbraucht wird, sollten die unbenötigten Flächen von kurzen Zeilen transparent sein, so daß dort eventuell darunterliegende Informationen durchscheinen könnten.

Wie in Abschnitt 4.3.3 beschrieben, werden diese Texte intern als Images gespeichert. Dies bedeutet, daß für das geplanten Vorhaben transparente Images benötigt worden wären. Das ganze Vorhaben wurde jedoch komplett gestrichen, da die fehlende Transparenz eine Implementierung in der gewünschten Art und Weise nicht zuließ.

Mehrzeilige Texte hätten durch Darstellung mehrerer Images erzeugt werden können, allerdings hätte dies den Aufwand enorm erhöht. Nicht nur, daß für Prozesse oder Nachrichten dann plötzlich mehrere Images mitgeschleppt hätten werden müssen und dies den Code aufgebläht hätte, sondern auch die Tatsache, daß dies die Darstellung deutlich verlangsamt hätte, sprach gegen die Implementierung durch mehrere Images.

Um dieses Problem der fehlenden Transparenz ad absurdum zu treiben, sei hier noch erwähnt, daß die Klassen ColorModel, IndexColorModel und DirectColorModel aus java.awt.Image sehr wohl eine Unterstützung für Transparenz böten, wenn sie nur genutzt würde. Es existieren dort extra Methoden, um den Alpha-Wert einer Farbe, der die Transparenz angibt, zu setzen oder abzufragen.

Die im Verlauf dieser Studienarbeit benutzten Browser Appletviewer und Netscape 3.x verwenden jedoch standardmäßig nur Farbmodelle die keine transparenten Farben enthalten. Leider ist auch ein Ersetzen des Farbmodells nicht möglich, da dieses nur ein einziges Mal gesetzt werden kann.

Eventuell ist es möglich, daß über Umwege wie das Filtern von Bildern mit viel Aufwand eine Transparenz erzeugt werden kann. Allerdings wurde trotz intensiven Studiums der Dokumentation und der Sourcen von java.awt, wie sie mit dem Java Development Kit [Javasoft] mitgeliefert werden, kein Weg gefunden, um dies mit vertretbarem Aufwand zu erreichen.

6.2 Fehlende Popup-Menüs in Applets

Ein Problem, das auch in die Kategorie fehlende Funktionalität eingeordnet werden kann, soll hier beschrieben werden.

Im Verlauf der Implementierung eines Beispiels für die Unterstützung allgemeiner Protokolle mit Fehlerfällen (siehe Abschnitt 3.4) wurde angedacht, daß die Beeinflussung durch den Benutzer auf folgende Art und Weise geschehen soll: Die zu beeinflussende Nachricht wird wie in Abschnitt 4.5.2 beschrieben über den halben Weg animiert dargestellt und bleibt dann auf halbem Weg stehen. In diesem Moment sollte der Benutzer durch Anklicken der Nachricht ein kleines Popup-Menü erhalten, in dem er die gewünschte Beeinflussung auswählen könnte.

Leider erwies sich dieses „kleine Popup-Menü“ als nicht ohne weiteres implementierbar. Es steht mit java.awt.Choice zwar ein Objekt zur Verfügung, das genau solch ein Popup-Menü erzeugt, allerdings existiert dies nur in Verbindung mit dem Button, der auf anklicken das Popup-Menü erzeugt. Dieser Button wiederum funktioniert nur im Zusammenspiel mit den normalen Komponenten und Layoutmanager-Verwaltungsfunktionen, was eine beliebige Positionierung im Display-Objekt sehr erschwert.

Nachdem Versuche mit dem Choice-Objekt zu keinem befriedigenden Ergebnis geführt hatten, wurde untersucht, ob ein solches Popup-Menü durch Erzeugen eines eigenen kleinen Fensters selbst implementiert werden könnte. Dieses Vorhaben wurde jedoch schnell wieder aufgegeben, da ein Applet zwar ein extra Window aufmachen darf, dieses Window dann aber aus Sicherheitsgründen am oberen Rand eine Warnung enthält, daß dieses Window von einem

Applet erzeugt wurde. Abbildung 6.1 zeigt diese Warnung, die in schwarzer Schrift auf gelbem Untergrund groß verkündet, daß ein Applet etwas gemacht hat, was von Standpunkt der Sicherheit der Benutzers als bedenklich eingestuft wird.

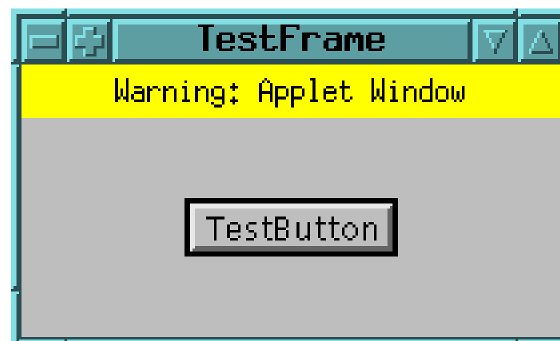

Abbildung 6.1: Screenshot der Appletwarnung

Es sei hier angemerkt, daß diese Warnung absolut berechtigt ist, da ein Applet ansonsten ein Fenster erzeugen könnte, welches absolut identisch aussehen könnte wie ein Fenster eines anderen Programmes, in das man bedenkenlos ein Paßwort oder sonstige private Daten eintippt. Auf diese Weise könnte das Applet über ein solches Trojanisches Pferd an Informationen gelangen, an die es nicht kommen sollte. Für die Anwendung als Popup-Menü ist diese Warnung allerdings nicht akzeptabel da der Benutzer hierdurch unnötig verwirrt würde.

Aufgrund der geschilderten Probleme und dem Fehlen eines Popup-Menüs ohne zugehörigen Button konnte diese Idee der Einflußnahme des Benutzers nicht implementiert werden obwohl dies eine elegante Lösung gewesen wäre.

Traurig daran ist vor allem, daß die Funktionalität eines Popup-Menüs in Java vorhanden ist, sie dem Anwender aber nur über ein Choice oder eine MenuBar (java.awt.MenuBar) eingeschränkt zugänglich gemacht wird.

6.3 Fehlerhafte InputFocusPolicy

In diesem Abschnitt wird ein Problem von Java mit der InputFocusPolicy beschrieben, der im Verlauf der Implementierung des Actors-Fakultätsberechnungsbeispiels festgestellt wurde.

In dem Actors Beispiel (siehe Abschnitt 5.1) kann der Benutzer über ein java.awt.TextField eine Zahl eingeben, von der dann die Fakultät berechnet wird. Zwecks Eingabe der Zahl empfängt das TextField natürlich die Events, die das Java Laufzeitsystem beim Drücken einer Taste auslöst. Daran wäre nichts auszusetzen, wenn es das nur täte, wenn sich der Mauszeiger auch im TextField befindet. Wenn man solch ein TextField jedoch einmal angeklickt hat, um etwas einzugeben, dann gibt das TextField den InputFocus nicht mehr her, auch wenn dem TextField über setEditable(false) explizit gesagt wird, daß der Benutzer gar keine Eingaben mehr machen darf.

Aufgefallen ist dieses Problem, weil nach Eingabe einer Zahl in das TextField die Geschwindigkeit des Assistenten nicht mehr durch Eingabe einer Ziffer gesteuert werden konnte, obwohl der Mauszeiger im Display-Objekt war. Als Ausweg wurde der in Abschnitt 4.2.3 beschriebene Choice in den Baukasten aufgenommen.

6.4 Überlastung des Laufzeitsystems

Dieser Abschnitt beschreibt zwei Überlastungserscheinungen des Java Laufzeitsystems. Die erste Überlastungserscheinung konnte lange Zeit reproduziert werden, tritt aber inzwischen aufgrund von Änderungen am Baukasten praktisch nicht mehr auf. Die zweite Überlastungserscheinung scheint nichtdeterministisch aufzutreten, entsprechend vage ist die Beschreibung.

Das erste Problem scheint darauf zurückzuführen zu sein, daß der Garbage Collection Thread aufgrund einer niedrigen Priorität nicht zum Zuge kommt, wenn andere Threads viel CPU-Zeit benötigen und diese aufgrund ihrer höheren Priorität auch bevorzugt bekommen.

Das Problem trat auf, wenn sehr viele Nachrichten schnell hintereinander dargestellt wurden und deshalb die virtuelle Darstellungsfläche häufig vergrößert werden mußte. Es mußten also zwei Bedingungen gleichzeitig erfüllt sein, um das Problem zu reproduzieren. Zum einen mußten meist mehr als 50 Nachrichten nacheinander dargestellt werden, damit die virtuelle Darstellungsfläche genügend groß wurde. Zum anderen durfte es keine Pausen geben, in denen das Laufzeitsystem die Garbage Collection hätte erledigen könnte.

Alle Bedingungen wurden durch das Actors-Fakultätsbeispiel (siehe Abschnitt 5.1) erfüllt, wenn von Hand auf höchste Geschwindigkeit geschaltet wurde und dann die Fakultät von 20 berechnet und dargestellt wurde. In diesem Fall trat dann etwa ab Nachricht 50 (eventuell früher oder später) folgender Effekt auf: Der Darstellungsbereich scrollte nicht mehr weiter nach oben und die Nachrichten wurden auf den untersten drei Zentimetern der Darstellungsfläche dargestellt, ohne daß die Bilder der einzelnen Animationsschritte korrekt gelöscht würden. Es sah dann so aus, als ob Dutzende von Nachrichtenbildern gleichzeitig und nur leicht versetzt dargestellt würden. Mit etwas Glück füllte sich so der unterste Bereich fast vollständig mit Nachrichtenbildern.

Daß dies kein Fehler des Baukastens ist, läßt sich daran erkennen, daß der Effekt nicht eintritt, wenn mit langsameren Geschwindigkeiten gearbeitet wird. Auch normalisiert sich die Darstellung, wenn der Refresh-Button gedrückt wird, nachdem das Java Laufzeitsystem nach Darstellungsende eine Garbage Collection gemacht hat.

Der zweite Überlastungsfall trat während der Implementierung sporadisch auf, konnte aber durch Änderungen am Ablauf der run() Methode weitestgehend beseitigt werden und sollte jetzt nicht mehr auftreten.

Die Überlastung äußerte sich dadurch, daß ein getGraphics()-Aufruf, der in der Endlosschleife innerhalb der run() Methode gemacht wurde, unter nicht näher geklärten Randbedingungen plötzlich das Null-Objekt lieferte, obwohl davor und auch danach jeweils gültige Graphics-Objekte geliefert wurden.

Ähnliche Effekte traten auch mit anderen Methoden-Aufrufen auf und führten dann oft zum Absterben der Darstellung. Als Gegenmaßnahme wurde der getGraphics()-Aufruf aus der Schleife heraus verlegt und mit extra Überprüfungen auf korrekte Ergebnisse versehen. Auch wurde die Darstellung aufgrund dieser Probleme erst zu einem eigenständigen Thread gemacht und dessen Existenz wird laufend überprüft.

6.5 Einschätzung der Verwendbarkeit von Java

Die Erfahrungen mit Java als Sprache und mit den Standardklassen im Verlauf dieser Studienarbeit lassen sich folgendermaßen zusammenfassen:

1) Der Umgang mit dem Java-API ist gewöhnungsbedürftig, da vieles nicht so funktioniert, wie man es aufgrund von Erfahrungen mit anderen Sprachen erwarten würde. Dies ist besonders verwirrend, da die Sprache an sich aufgrund ihrer Ähnlichkeit zu anderen Sprachen zu Vergleichen einlädt.

2) Probleme mit der Skalierbarkeit wie die oben aufgeführten, stimmen nachdenklich, wenn man an den allgemeinen Einsatz denkt.

3) Mit den Standard-Klassen kann man schnell und einfach viele Standardaufgaben eines GUI implementieren. Wenn man jedoch nur ein kleines bißchen von diesem Standard abweichen will und etwas extravagantere Wünsche hat, dann wird dies von den Standardklassen nicht nur nicht unterstützt, sondern es werden einem auch unnötig Steine in den Weg gelegt, so daß ein Abweichen vom Standard fast unmöglich wird.

Um es kurz und prägnant auszudrücken: Ich kann die Begeisterung, die Java allgemein auslöst wirklich nicht verstehen.

Zusammenfassung

Diese Studienarbeit beschäftigte sich mit der „Visualisierung von Lehrinhalten mit Java Applets“. Es wurde hierzu ein Baukastensystem zur Animation der Kommunikation zwischen Komponenten in Verteilten Systemen erstellt. Das Baukastensystem stellt, die zwischen Prozessen, Agenten oder anderen Komponenten ausgetauschten Nachrichten, in ihrem zeitlichen Ablauf dar. Die Darstellung der Nachrichtenübertragung erfolgt dabei durch eine Folge von Einzelbildern, bei denen die Nachricht jeweils um ein kleines Stück weiter vom Sender zum Empfänger verschoben, dargestellt wird. Auf diese Weise entsteht der Eindruck einer sich bewegenden Nachricht.

Das Baukastensystem unterstützt sowohl Punkt zu Punkt (Unicast) als auch Punkt zu Mehrpunkt (Multi-/Broadcast) Nachrichten. Auch eine Unterstützung von Nachrichtenverfälschungen während der Übertragung ist vorhanden

Die Implementierung erfolgte in der Programmiersprache Java und unterstützt somit eine direkte Einbindung in Java Applets und damit in HTML-Seiten. Die Verwendung des Baukastens aus einem Anwendungsapplet heraus ist dabei denkbar einfach und besteht oft aus nur wenigen Methodenaufrufen.

Neben dem Baukastensystem wurden auch einige Beispielanwendungen erstellt. Diese stellen verschiedene, in der Vorlesung „Kooperation in Verteilten Systemen“ behandelte, Kommunikationsprotokolle, unter Verwendung des Baukastensystems dar. Neben der Funktion, als Beispiel für zukünftige Anwender zu dienen, können die Anwendungen auch direkt in zukünftigen Vorlesungen eingesetzt werden und dienen somit direkt der Verbesserung der Lehre.

Ausblick

Es ist zu erwarten, daß der Einsatz von Rechnern im Bereich der Lehre zukünftig immer mehr an Bedeutung gewinnen wird. Dementsprechend wird auch der Einsatz von Programmen oder Baukästen zur Visualisierung von Lehrinhalten stark zunehmen.

Für den, im Rahmen dieser Studienarbeit entwickelten, Baukasten bedeutet dies, daß bereits mehrere, auf dem Baukasten aufbauende Projekte am Lehrstuhl für Verteilte Systeme existieren. Der Baukasten wird beispielsweise bereits in der Diplomarbeit „Visualisierung von Kommunikationsprotokollen mit Java“ von Edgar Kogel, für zwei weitere Studienarbeiten, im Softwarepraktikum „Visualisierung von Protokollen aus dem elektronischen Markt“ und für die Übungen der Info-AG eingesetzt.

Außerdem werden auf der CD-ROM zum Mitte dieses Jahres erscheinenden Buch „Groupware - Kooperationsunterstützung für verteilte Anwendungen“ von Dr. Cora Burger einige Anwendungen des Baukastens enthalten sein. Auch der Einsatz im Rahmen eines interaktiven Buchs, an dem Prof. Dr. Rothermel mitarbeitet, ist geplant.

Literaturverzeichnis

[Borghoff] Borghoff, Uwe M. & Schlichter, Johann H. (1995)
Rechnergestützte Gruppenarbeit- Eine Einführung in Verteilte Anwedungen
Springer-Verlag Berlin Heidelberg

[Burger] Burger, Cora (WS 1995/1996)
Folien zur Vorlesung „Kooperation in Verteilten Systemen“
Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, Lehrstuhl für Verteilte Systeme

[Flanagan] Flanagan, David (1996)
Java in a Nutshell
O‘Reilly & Associates, Inc.

[Gosling] Gosling, James & McGilton, Henry (May 1996)
The Java Language: A White Paper
http://java.sun.com/doc/language_environment/

[Hohl] Hohl, Fritz (1995)
Konzeption eines einfachen Agentensystems und Implementation eines Prototyps
Diplomarbeit Nr. 1267, Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, Lehrstuhl für Verteilte Systeme

[Hohl96] Hohl, Fritz (1996)
Folien zum Java Crashkurs
Universität Stuttgart, Institut für Parallele und Verteilte Höchstleistungsrechner, Lehrstuhl für Verteilte Systeme

[JavaSoft] JavaSoft - A Sun Microsystems, Inc. Business
The Java Developers Kit Version 1.0.2
http://www.javasoft.com:80/products/jdk/1.0.2/

[Netscape] Netscape Communications Corporation
Netscape Navigator Version 3.01
http://home.netscape.com/comprod/products/navigator/index.html

[Niemeyer] Niemeyer, Patrick & Peck, Joshua (1996)
Exploring Java
O‘Reilly & Associated, Inc.

[Sun] Sun Microsystems, Inc. (1996)
Java-Online-Documentation
http://java.sun.com/doc/

Anhang A

Beschreibung der Schnittstelle des Baukastens

Hier sind alle public-Methoden des Baukastens mit ihren Parametern, eventuellen Rückgabewerten und einer Beschreibung aufgeführt.

A.1 Allgemeine Verwaltungsmethoden

Methode: public Assistent

Parameter 1 - Klasse und Bezeichnung: Applet Anwendung

Parameter 1 - Beschreibung: Das Applet, das diesen Baukasten erzeugt. Wird benötigt, damit der Baukasten einige Methoden, die nur einem Applet zugänglich sind, aufrufen kann.

Beschreibung der Methode: Konstruktor der Klasse Assistent.

Methode: public int aktuelleUhrzeit

Beschreibung der Methode: Liefert die aktuelle Zeit im Baukasten zurück.

Methode: public int getDauerNachrichtenUebermittlung

Beschreibung der Methode: Liefert die momentane Dauer einer Nachrichtenübermittlung zurück.

Methode: public boolean handleEvent

Parameter 1 - Klasse und Bezeichnung: Event e

Parameter 1 - Beschreibung: Der von Java-Laufzeitsystem erzeugte Event

Beschreibung der Methode: Methode um Tastatur- und Maus-Events zu empfangen. Sollte von der Anwendung nicht aufgerufen werden, wird vom Java-Laufzeitsystem verwendet, muß aber laut Interface-Definition public sein.

Methode: public boolean ladeImage

Parameter 1 - Klasse und Bezeichnung: String imgFileName

Parameter 1 - Beschreibung: Name eines zu ladenden Image-Files. Der Name ist relativ zur DocumentBase des Anwendungsapplets.

Parameter 2 - Klasse und Bezeichnung: String imgName

Parameter 2 - Beschreibung: Der Name, unter dem das Image für neue Prozesse oder Nachrichten verwendet werden kann

Beschreibung der Methode: Lädt ein Image und macht es verfügbar.

Methode: public boolean ladeImage

Parameter 1 - Klasse und Bezeichnung: Image img

Parameter 1 - Beschreibung: Ein von der Anwendung erzeugtes oder geladenes Image.

Parameter 2 - Klasse und Bezeichnung: String imgName

Parameter 2 - Beschreibung: Der Name, unter dem das Image für neue Prozesse oder Nachrichten verwendet werden kann

Beschreibung der Methode: Macht das Image img für Prozess- oder Nachrichtendarstellungen verfügbar.

Methode: public String NachrichtAngeklickt

Beschreibung der Methode: Wenn seit dem letzten Aufruf dieser Methode vom Benutzer eine Nachricht mit der Maus angeklickt wurde, dann liefert diese Methode den Namen der Nachricht zurück. Wenn keine Nachricht angeklickt wurde, wird null zurückgeliefert. Kann von der Anwendung, z.B. aus mouseDown oder handleEvent heraus, verwendet werden, um zu testen, ob der Benutzer eine Nachricht angeklickt hat.

Methode: public void pause

Parameter 1 - Klasse und Bezeichnung: int pause

Parameter 1 - Beschreibung: Länge der Pause.

Beschreibung der Methode: Noch eine Methode, um die Raumaufteilung auf dem Bildschirm zu beeinflußen. Diesmal, um für die nächsten pause Zeiteinheiten nur die momentan existierenden Prozesse darzustellen. Hiermit kann beispielsweise die zwischen zwei Nachrichten verstreichende Zeit dargestellt werden.

Methode: public void reset

Beschreibung der Methode: Setzt alle internen Variablen des Assistenten zurück auf den Anfangszustand. Dabei werden alle Prozesse und Nachrichten gelöscht.

Methode: public synchronized void run()

Beschreibung der Methode: run()-Methode des Darstellungs-Threads. Sollte von der Anwendung nicht aufgerufen werden, wird nur intern verwendet, muß aber laut Interface-Definition public sein.

Methode: public void setzeBriefNachricht

Parameter 1 - Klasse und Bezeichnung: boolean yesno

Parameter 1 - Beschreibung: Wenn yesno true ist, dann werden zukünftige Default-Images als Briefumschlag mit Zettel dargestellt, ansonsten als Text in einem 3D-Rahmen

Beschreibung der Methode: Legt fest, welche Art von Default-Images verwendet werden sollen.

Methode: public void setzeDauerNachrichtenUebermittlung

Parameter 1 - Klasse und Bezeichnung: int dauer

Parameter 1 - Beschreibung: Die während einer Nachrichtenübermittlung verstreichende Zeit. Default ist 20 Zeiteinheiten (Entspricht 20 Pixel auf dem Bildschirm)

Beschreibung der Methode: Kleine Werte führen zu einer fast waagrechen Darstellung der Nachrichtenübermittlung während große Werte zu sehr schrägen Darstellungen führen.

Methode: public void setzeGeschwindigkeit

Parameter 1 - Klasse und Bezeichnung: int zahl

Parameter 1 - Beschreibung: Geschwindigkeit der Darstellung, 1 ist langsamste Darstellung, 10 ist schnellste Darstellung, Default ist 5 für mittlere Geschwindigkeit

Beschreibung der Methode:

Methode: public void setzeMinimaleProzessZahl

Parameter 1 - Klasse und Bezeichnung: int zahl

Parameter 1 - Beschreibung: Anzahl der Prozesse die voraussichtlich benötigt werden.

Beschreibung der Methode: Diese Methode beeinflußt die Verteilung der Prozesse auf dem Bildschirm. Wenn ein neuer Prozess hinzukommt, werden normalerweise alle anderen Prozesse etwas verschoben, um für den neuen Prozess Platz zu schaffen. Der durch diese Verschiebung verursachte Sprung in der Darstellung, kann vermieden werden, wenn im Voraus bekannt ist, wieviele Prozesse dargestellt werden sollen.

Methode: public void setzePauseNachBeendeProzess

Parameter 1 - Klasse und Bezeichnung: int pause

Parameter 1 - Beschreibung: Länge der Pause nach dem Beenden eines Prozesses. Der Wert sollte nicht zu klein gewählt werden, weil sonst das Bild, das das Prozessende darstellt, eventuell nur halb auf dem Bildschirm erscheint.
Default ist 5 Zeiteinheiten (Entspricht 5 Pixel auf dem Bildschirm)

Beschreibung der Methode: Mit dieser Methode kann die Raumaufteilung auf dem Bildschirm beeinflusst werden. Kleine Werte führen zu einer zusammengedrängten Darstellung, während große Werte die Darstellung auseinanderziehen.

Methode: public void setzePauseNachEmpfangeNachricht

Parameter 1 - Klasse und Bezeichnung: int pause

Parameter 1 - Beschreibung: Länge der Pause vor dem Empfang einer Nachricht. Auch hier sollte der Wert nicht zu klein sein, sonst werden die Bilder der Nachrichten zeitweise nur halb dargestellt.
Default ist 5 Zeiteinheiten (Entspricht 5 Pixel auf dem Bildschirm)

Beschreibung der Methode: Mit dieser Methode kann die Raumaufteilung auf dem Bildschirm beeinflusst werden. Kleine Werte führen zu einer zusammengedrängten Darstellung, während große Werte die Darstellung auseinanderziehen.

Methode: public void setzePauseNachErzeugeProzess

Parameter 1 - Klasse und Bezeichnung: int pause

Parameter 1 - Beschreibung: Länge der Pause nach dem Erzeugen eines Prozesses. Default ist 5 Zeiteinheiten (Entspricht 5 Pixel auf dem Bildschirm)

Beschreibung der Methode: Mit dieser Methode kann die Raumaufteilung auf dem Bildschirm beeinflusst werden. Kleine Werte führen zu einer zusammengedrängten Darstellung, während große Werte die Darstellung auseinanderziehen.

Methode: public void setzePauseVorBeendeProzess

Parameter 1 - Klasse und Bezeichnung: int pause

Parameter 1 - Beschreibung: Länge der Pause vor dem Beenden eines Prozesses. Default ist 5 Zeiteinheiten (Entspricht 5 Pixel auf dem Bildschirm)

Beschreibung der Methode: Mit dieser Methode kann die Raumaufteilung auf dem Bildschirm beeinflusst werden. Kleine Werte führen zu einer zusammengedrängten Darstellung, während große Werte die Darstellung auseinanderziehen.

Methode: public void setzePauseVorErzeugeNachricht

Parameter 1 - Klasse und Bezeichnung: int pause

Parameter 1 - Beschreibung: Länge der Pause vor dem Erzeugen einerNachricht . Default ist 5 Zeiteinheiten (Entspricht 5 Pixel auf dem Bildschirm)

Beschreibung der Methode: Mit dieser Methode kann die Raumaufteilung auf dem Bildschirm beeinflusst werden. Kleine Werte führen zu einer zusammengedrängten Darstellung, während große Werte die Darstellung auseinanderziehen.

Methode: public void setzePauseVorErzeugeProzess

Parameter 1 - Klasse und Bezeichnung: int pause

Parameter 1 - Beschreibung: Länge der Pause vor dem Erzeugen eines Prozesses. Default ist 5 Zeiteinheiten (Entspricht 5 Pixel auf dem Bildschirm)

Beschreibung der Methode: Mit dieser Methode kann die Raumaufteilung auf dem Bildschirm beeinflusst werden. Kleine Werte führen zu einer zusammengedrängten Darstellung, während große Werte die Darstellung auseinanderziehen.

A.2 Methoden zur Verwaltung von Prozessen

Methode: public void beendeProzess

Parameter 1 - Klasse und Bezeichnung: String Name

Parameter 1 - Beschreibung: Name des Prozesses, der beendet werden soll.

Beschreibung der Methode: Beendet die Darstellung des Prozesses Namen auf dem Bildschirm ab dem aktuellen Zeitpunkt. Der Prozeß existiert intern weiter, wird aber nicht weiter dargestellt.

Methode: public void erzeugeProzess

Parameter 1 - Klasse und Bezeichnung: String Name

Parameter 1 - Beschreibung: Der Name unter dem der Prozeß angesprochen werden soll. Gleichzeitig auch der Name des Bildes, das den Prozeß auf dem Bildschirm repräsentiert.

Beschreibung der Methode: Erzeugt einen neuen Prozess auf dem Bildschirm und macht ihn unter dem angegebenen Namen für weitere Aufrufe verfügbar.

Methode: public void erzeugeProzess

Parameter 1 - Klasse und Bezeichnung: String Name

Parameter 1 - Beschreibung: Der Name, unter dem der Prozeß angesprochen werden soll.

Parameter 2 - Klasse und Bezeichnung: String ImageName

Parameter 2 - Beschreibung: Der Name des Bildes, das den Prozeß auf dem Bildschirm repräsentiert

Beschreibung der Methode: Erzeugt einen neuen Prozess auf dem Bildschirm und macht ihn unter dem angegebenen Namen für weitere Aufrufe verfügbar.

A.3 Methoden für normale Nachrichten

Methode: public void versendeUndEmpfangeBroadcastNachricht
Parameter 1 - Klasse und Bezeichnung: String BaseName
Parameter 1 - Beschreibung: Siehe erste versendeUndEmpfangeNachricht-Methode
Parameter 2 - Klasse und Bezeichnung: String SenderName
Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt.
Beschreibung der Methode: Erzeugt für jeden vorhandenen Prozess eine Nachricht an diesen Prozess und stellt die Übermittlung der Nachrichten animiert dar.

Methode: public void versendeUndEmpfangeBroadcastNachricht
Parameter 1 - Klasse und Bezeichnung: String BaseName
Parameter 1 - Beschreibung: Siehe zweite versendeUndEmpfangeNachricht-Methode
Parameter 2 - Klasse und Bezeichnung: String SenderName
Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt.
Parameter 3 - Klasse und Bezeichnung: String ImageName
Parameter 3 - Beschreibung: Name für das Image, mit dem die Nachricht auf dem Bildschirm dargestellt werden soll.
Beschreibung der Methode: Erzeugt für jeden vorhandenen Prozess eine Nachricht an diesen Prozess und stellt die Übermittlung der Nachrichten animiert dar.

Methode: public void versendeUndEmpfangeBroadcastOhneSenderNachricht
Parameter 1 - Klasse und Bezeichnung: String BaseName
Parameter 1 - Beschreibung: *Siehe erste versendeUndEmpfangeNachricht-Methode*.
Parameter 2 - Klasse und Bezeichnung: String SenderName
Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt.
Beschreibung der Methode: Erzeugt für jeden vorhandenen Prozess eine Nachricht an diesen Prozess und stellt die Übermittlung der Nachrichten animiert dar. Der Senderprozeß bekommt dabei keine Nachricht.

Methode: public void versendeUndEmpfangeBroadcastOhneSenderNachricht
Parameter 1 - Klasse und Bezeichnung: String BaseName
Parameter 1 - Beschreibung: *Siehe zweite versendeUndEmpfangeNachricht-Methode*
Parameter 2 - Klasse und Bezeichnung: String SenderName
Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt.
Parameter 3 - Klasse und Bezeichnung: String ImageName
Parameter 3 - Beschreibung: Name für das Image, mit dem die Nachricht auf dem Bildschirm dargestellt werden soll.
Beschreibung der Methode: Erzeugt für jeden vorhandenen Prozess eine Nachricht an diesen Prozess und stellt die Übermittlung der Nachrichten animiert dar. Der Senderprozeß bekommt dabei keine Nachricht.

Methode: public void versendeUndEmpfangeMulticastNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: *Siehe erste versendeUndEmpfangeNachricht-Methode*

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt.

Parameter 3 - Klasse und Bezeichnung: Vector EmpfaengerNamen

Parameter 3 - Beschreibung: Die Namen der Prozesse, an die die Nachricht verschickt werden soll.

Beschreibung der Methode: Erzeugt für jeden Empfängerprozeß eine Nachricht an diesen Prozess und stellt die Übermittlung der Nachrichten animiert dar.

Methode: public void versendeUndEmpfangeMulticastNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: *Siehe erste versendeUndEmpfangeNachricht-Methode*

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt.

Parameter 3 - Klasse und Bezeichnung: Vector EmpfaengerNamen

Parameter 3 - Beschreibung: Die Namen der Prozesse, an die die Nachricht verschickt werden soll.

Parameter 4 - Klasse und Bezeichnung: String ImageName

Parameter 4 - Beschreibung: Name für das Image, mit dem die Nachricht auf dem Bildschirm dargestellt werden soll.

Beschreibung der Methode: Erzeugt für jeden Empfängerprozeß eine Nachricht an diesen Prozess und stellt die Übermittlung der Nachrichten animiert dar.

Methode: public void versendeUndEmpfangeNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: *Siehe zweite versendeUndEmpfangeNachricht-Methode.* Der BaseName wird gleichzeitig als Name für das Image, mit dem die Nachricht auf dem Bildschirm dargestellt werden soll, verwendet.

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt..

Parameter 3 - Klasse und Bezeichnung: String EmpfaengerName

Parameter 3 - Beschreibung: Name des Prozesses, der die Nachricht empfangen soll.

Beschreibung der Methode: Erzeugt eine neue Nachricht und stellt die Nachrichtenübermittlung animiert dar.

Methode: public void versendeUndEmpfangeNachrichtString

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: Basisname der Nachricht. Dieser wird intern um Sender, Empfänger und die aktuelle Uhrzeit ergänzt um einen eindeutigen Namen zu erzeugen. Der Name wird von NachrichtAngeklickt an die Anwendung zurückgegeben, wenn die Nachricht vom Benutzer angeklickt wurde.

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt.

Parameter 3 - Klasse und Bezeichnung: String EmpfaengerName

Parameter 3 - Beschreibung: Name des Prozesses, der die Nachricht empfangen soll.

Parameter 4 - Klasse und Bezeichnung: String ImageName

Parameter 4 - Beschreibung: Name für das Image, mit dem die Nachricht auf dem Bildschirm dargestellt werden soll.

Beschreibung der Methode: Erzeugt eine neue Nachricht und stellt die Nachrichtenübermittlung animiert dar.

A.4 Methoden für Nachrichtenverfälschungen

Methode: public void empfangeGeaenderteNachricht

Parameter 1 - Klasse und Bezeichnung: String neueNachricht

Parameter 1 - Beschreibung: Ein von versendeNachricht zurückgegebener Name einer Nachricht

Parameter 2 - Klasse und Bezeichnung: String ImageName

Parameter 2 - Beschreibung: Names des Images, das die Veränderung darstellt

Beschreibung der Methode: Setzt das Bild, mit dem die Nachricht dargestellt wird, auf ImageName und stellt die zweite Hälfte der Nachrichtenübertragung animiert dar.

Methode: public void empfangeNirvanaNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: Siehe erste versendeUndEmpfangeNachricht-Methode.

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: Name des Prozesses, von dem die Nachrichtenübermittlung ausgehen würde, wenn es eine normale Nachricht wäre, dient hier nur zur Festlegung, von wo die Nachricht herkommen soll.

Parameter 3 - Klasse und Bezeichnung: String EmpfaengerName

Parameter 3 - Beschreibung: Name des Prozesses, der die Nachricht empfangen soll.

Beschreibung der Methode: Erzeugt eine neue Nachricht, die aus dem Nichts auftaucht, und stellt die Nachrichtenübermittlung animiert dar.

Methode: public void empfangeNirvanaNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: Siehe zweite versendeUndEmpfangeNachricht-Methode.

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: siehe erste empfangeNirvanaNachricht

Parameter 3 - Klasse und Bezeichnung: String EmpfaengerName

Parameter 3 - Beschreibung: Name des Prozesses, der die Nachricht empfangen soll.

Parameter 4 - Klasse und Bezeichnung: String ImageName

Parameter 4 - Beschreibung: Name für das Image, mit dem die Nachricht auf dem Bildschirm dargestellt werden soll.

Beschreibung der Methode: Erzeugt eine neue Nachricht, die aus dem Nichts auftaucht, und stellt die Nachrichtenübermittlung animiert dar.

Methode: public void empfangeBroadcastNirvanaNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: *Siehe erste versendeUndEmpfangeNachricht-Methode*

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: siehe erste empfangeNirvanaNachricht

Beschreibung der Methode: Erzeugt für jeden vorhandenen Prozess eine neue Nachricht, die aus dem Nichts auftaucht und an dem Prozess geschickt wird, und stellt die Nachrichtenübermittlung animiert dar.

Methode: public void empfangeBroadcastNirvanaNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: Siehe zweite versendeUndEmpfangeNachricht-Methode.

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: siehe erste empfangeNirvanaNachricht

Parameter 3 - Klasse und Bezeichnung: String ImageName

Parameter 3 - Beschreibung: Name für das Image, mit dem die Nachricht auf dem Bildschirm dargestellt werden soll.

Beschreibung der Methode: Erzeugt für jeden vorhandenen Prozess eine neue Nachricht, die aus dem Nichts auftaucht und an dem Prozess geschickt wird, und stellt die Nachrichtenübermittlung animiert dar.

Methode: public void empfangeMulticastNirvanaNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: *Siehe erste versendeUndEmpfangeNachricht-Methode*

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: siehe erste empfangeNirvanaNachricht

Parameter 3 - Klasse und Bezeichnung: Vector EmpfaengerNamen

Parameter 3 - Beschreibung: Die Namen der Prozesse, die die Nachricht empfangen sollen.

Beschreibung der Methode: Erzeugt für jeden Empfängerprozess eine neue Nachricht, die aus dem Nichts auftaucht und an dem Prozess geschickt wird, und stellt die Nachrichtenübermittlung animiert dar.

Methode: public void empfangeMulticastNirvanaNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: Siehe zweite versendeUndEmpfangeNachricht-Methode.

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: siehe erste empfangeNirvanaNachricht

Parameter 3 - Klasse und Bezeichnung: Vector EmpfaengerNamen

Parameter 3 - Beschreibung: Die Namen der Prozesse, die die Nachricht empfangen sollen.

Parameter 4 - Klasse und Bezeichnung: String ImageName

Parameter 4 - Beschreibung: Name für das Image, mit dem die Nachricht auf dem Bildschirm dargestellt werden soll.

Beschreibung der Methode: Erzeugt für jeden Empfängerprozess eine neue Nachricht, die aus dem Nichts auftaucht und an dem Prozess geschickt wird, und stellt die Nachrichtenübermittlung animiert dar.

Methode: public void empfangeUnveraenderteNachricht

Parameter 1 - Klasse und Bezeichnung: String neueNachricht

Parameter 1 - Beschreibung: Ein von versendeNachricht zurückgegebener Name einer Nachricht

Beschreibung der Methode: Stellt die zweite Hälfte der Nachrichtenübertragung ohne Veränderung der Nachricht dar.

Methode: public void empfangeVeraenderteMulticastNachricht

Parameter 1 - Klasse und Bezeichnung: Vector neueNachrichten

Parameter 1 - Beschreibung: Eine von versendeMulticastNachricht, versendeBroadcastNachricht oder versendeBroadcastOhneSenderNachricht zurückgegebene Liste von Nachrichten die nun verändert und dann dargestellt werden sollen.

Parameter 2 - Klasse und Bezeichnung: Vector Veraenderungen

Parameter 2 - Beschreibung: Eine Liste, die für jedes Element von Parameter 1, die Art der Veränderung angibt. Für jedes Element kann einer der folgenden Konstanten verwendet werden: NACHRICHT_UNVERAENDERT, NACHRICHT_GEAENDERT, NACHRICHT_VERLIEREN, NACHRICHT_VERZOEGERN

Parameter 3 - Klasse und Bezeichnung: Vector neueImageNamen

Parameter 3 - Beschreibung: Für alle in Parameter 2 als NACHRICHT_GEAENDERT markierten Nachrichten, muß hier der Name des Images, mit dem die Nachricht zukünftig dargestellt werden soll, angegeben werden. Die restlichen Listenwerte müssen null sein.

Beschreibung der Methode: Verändert die angegebenen Nachrichten anhand der verlangten Veränderungen und stellt anschliessend die zweite Weghälfte animiert dar.

Methode: public Vector versendeBroadcastNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: *Siehe erste versendeUndEmpfangeNachricht-Methode*

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt.

Beschreibung des Rückgabewertes: Die aus Parameter 1 erzeugten Namen, unter denen die Nachrichten, für weitere Methodenaufrufe, zur Verfügung stehen. Es handelt sich hierbei um eine Liste von String.

Beschreibung der Methode: Erzeugt für jeden vorhandenen Prozess eine Nachricht an diesen Prozess und stellt die erste Hälfte der Übermittlung der Nachrichten animiert dar.

Methode: public Vector versendeBroadcastNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: Siehe zweite versendeUndEmpfangeNachricht-Methode.

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt.

Parameter 3 - Klasse und Bezeichnung: String ImageName

Parameter 3 - Beschreibung: Name für das Image, mit dem die Nachricht auf dem Bildschirm dargestellt werden soll.

Beschreibung des Rückgabewertes: Die aus Parameter 1 erzeugten Namen, unter denen die Nachrichten, für weitere Methodenaufrufe, zur Verfügung stehen. Es handelt sich hierbei um eine Liste von String.

Beschreibung der Methode: Erzeugt für jeden vorhandenen Prozess eine Nachricht an diesen Prozess und stellt die erste Hälfte der Übermittlung der Nachrichten animiert dar.

Methode: public Vector versendeBroadcastOhneSenderNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: *Siehe erste versendeUndEmpfangeNachricht-Methode*

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt.

Beschreibung des Rückgabewertes: Die aus Parameter 1 erzeugten Namen, unter denen die Nachrichten, für weitere Methodenaufrufe, zur Verfügung stehen. Es handelt sich hierbei um eine Liste von String.

Beschreibung der Methode: Erzeugt für jeden vorhandenen Prozess eine Nachricht an diesen Prozess und stellt die erste Hälfte der Übermittlung der Nachrichten animiert dar. Der Senderprozeß bekommt dabei keine Nachricht.

Methode: public Vector versendeBroadcastOhneSenderNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: Siehe zweite versendeUndEmpfangeNachricht-Methode.

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt.

Parameter 3 - Klasse und Bezeichnung: String ImageName

Parameter 3 - Beschreibung: Name für das Image, mit dem die Nachricht auf dem Bildschirm dargestellt werden soll.

Beschreibung des Rückgabewertes: Die aus Parameter 1 erzeugten Namen, unter denen die Nachrichten, für weitere Methodenaufrufe, zur Verfügung stehen. Es handelt sich hierbei um eine Liste von String.

Beschreibung der Methode: Erzeugt für jeden vorhandenen Prozess eine Nachricht an diesen Prozess und stellt die erste Hälfte der Übermittlung der Nachrichten animiert dar. Der Senderprozeß bekommt dabei keine Nachricht.

Methode: public Vector versendeMulticastNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: *Siehe erste versendeUndEmpfangeNachricht-Methode*

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt.

Parameter 3 - Klasse und Bezeichnung: Vector EmpfaengerNamen

Parameter 3 - Beschreibung: Die Namen der Prozesse, an die die Nachricht verschickt werden soll.

Beschreibung des Rückgabewertes: Die aus Parameter 1 erzeugten Namen, unter denen die Nachrichten, für weitere Methodenaufrufe, zur Verfügung stehen. Es handelt sich hierbei um eine Liste von String.

Beschreibung der Methode: Erzeugt für jeden Empfängerprozess eine Nachricht an diesen Prozess und stellt die erste Hälfte der Übermittlung der Nachrichten animiert dar.

Methode: public Vector versendeMulticastNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: Siehe zweite versendeUndEmpfangeNachricht-Methode.

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt.

Parameter 3 - Klasse und Bezeichnung: Vector EmpfaengerNamen

Parameter 3 - Beschreibung: Die Namen der Prozesse, an die die Nachricht verschickt werden soll.

Parameter 4 - Klasse und Bezeichnung: String ImageName

Parameter 4 - Beschreibung: Name für das Image, mit dem die Nachricht auf dem Bildschirm dargestellt werden soll.

Beschreibung des Rückgabewertes: Die aus Parameter 1 erzeugten Namen, unter denen die Nachrichten, für weitere Methodenaufrufe, zur Verfügung stehen. Es handelt sich hierbei um eine Liste von String.

Beschreibung der Methode: Erzeugt für jeden Empfängerprozess eine Nachricht an diesen Prozess und stellt die erste Hälfte der Übermittlung der Nachrichten animiert dar.

Methode: public String versendeNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: Siehe erste versendeUndEmpfangeNachricht-Methode.

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt..

Parameter 3 - Klasse und Bezeichnung: String EmpfaengerName

Parameter 3 - Beschreibung: Name des Prozesses, der die Nachricht empfangen soll.

Beschreibung des Rückgabewertes: Der aus Parameter 1 erzeugte Name, unter dem die Nachricht, für weitere Methodenaufrufe, zur Verfügung steht.

Beschreibung der Methode: Erzeugt eine neue Nachricht und stellt die erste Hälfte der Nachrichtenübermittlung animiert dar.

Methode: public String versendeNachricht

Parameter 1 - Klasse und Bezeichnung: String BaseName

Parameter 1 - Beschreibung: Siehe zweite versendeUndEmpfangeNachricht-Methode.

Parameter 2 - Klasse und Bezeichnung: String SenderName

Parameter 2 - Beschreibung: Name des Prozesses, der die Nachricht verschickt..

Parameter 3 - Klasse und Bezeichnung: String EmpfaengerName

Parameter 3 - Beschreibung: Name des Prozesses, der die Nachricht empfangen soll.

Parameter 4 - Klasse und Bezeichnung: String ImageName

Parameter 4 - Beschreibung: Name für das Image, mit dem die Nachricht auf dem Bildschirm dargestellt werden soll.

Beschreibung des Rückgabewertes: Der aus Parameter 1 erzeugte Name, unter dem die Nachricht, für weitere Methodenaufrufe, zur Verfügung steht.

Beschreibung der Methode: Erzeugt eine neue Nachricht und stellt die erste Hälfte der Nachrichtenübermittlung animiert dar.

Methode: public void verliereNachricht

Parameter 1 - Klasse und Bezeichnung: String neueNachricht

Parameter 1 - Beschreibung: Ein von versendeNachricht zurückgegebener Name einer Nachricht

Beschreibung der Methode: Stellt auf der zweiten Hälfte des Nachrichtenweges nichts mehr dar und simuliert somit einen Nachrichtenverlust.

Methode: public void verzoegereEmpfangNachricht

Parameter 1 - Klasse und Bezeichnung: String NachrichtenName

Parameter 1 - Beschreibung: Ein von versendeNachricht zurückgegebener Name einer Nachricht die nun verzögert dargestellt werden soll.

Beschreibung der Methode: Die Nachricht folgt ab sofort dem aktuellen Zeitpunkt und kann später empfangen werden. Anwendungen können dies nutzen, um die Reihenfolge, in der Nachrichten ankommen, zu verändern.

Anhang B

Programmcode einer Beispielanwendung

Dies ist das Anwendungsapplet zum Terminaushandlungsbeispiel in Abschnitt 5.5

```
import java.applet.*;
import java.awt.*;
import java.util.*;

public class Terminaushandlung extends Applet implements Runnable
{
  // Thread um Main-Loop des Browsers freizugeben
  Thread broadcastThread = null;

  // Referenz auf den Assistenten des Baukastens
  Assistent Assi = null;

  // Konstruktor damit man diese Objekt ueberhaupt erzeugen kann.
  public Terminaushandlung ()
  {
    // ein recht einfacher LayoutManager reicht uns
    setLayout (new BorderLayout(0,0));

    // Als erstes muessen wir einen Assistenten erzeugen
    // Der Assistent braucht eine Referenz auf uns,
    // deshalb uebergeben wir ihm uns selbst in Form von ‚this‘
    Assi = new Assistent(this);

    // und den Assistent in die Ausgabe einbinden
    add(„Center“,Assi);
  }

  // Thread erzeugen, wenn noch keiner Existiert
  public void start()
  {
    if (broadcastThread == null || broadcastThread.isAlive() == false)
    {
      broadcastThread = new Thread(this);
      broadcastThread.start();
    }
  }

  // Thread auch freundlich stoppen, wenn Browser das moechte
  public void stop()
  {
    if ((broadcastThread != null) && (broadcastThread.isAlive()))
    {
      broadcastThread.stop();
    }
    broadcastThread = null;
  }
```

```
public void run ()
{
  // Netscape startet threads neu wenn man das netscape-window resized,
  // damit dann nicht alles doppelt auf dem Bildschirm erscheint.
  Assi.reset();

  // Damit der Assi von vorneherein weiss, wieviele Prozesse kommen
  // und damit fuer die dritte Broadcast-Nachricht genug platz ist
  Assi.setMinimaleProzessZahl(5);

  // Damit alle Prozesse gleichzeitig beginnen
  Assi.setPauseVorErzeugeProzess(0);
  Assi.setPauseNachErzeugeProzess(0);

  // Prozesse erzeugen
  Assi.erzeugeProzess(„Einladender");
  Assi.erzeugeProzess(„Eingeladener 1");
  Assi.erzeugeProzess(„Eingeladener 2");
  Assi.erzeugeProzess(„Eingeladener 3");
  Assi.erzeugeProzess(„Eingeladener 4");

  // da wir oben setPauseNachErzeugeProzess auf Null gesetzt haben,
  // muessen wir jetzt von Hand Platz schaffen
  Assi.pause(20);

  // wir wollen keine Brief-Symbole sondern nur Text
  Assi.setBriefNachricht(false);

  // etwas laengere Laufzeit, geht ja nicht elektronisch 8-)
  Assi.setDauerNachrichtenUebermittlung(30);

  // Calender-Manager schickt einen Nachricht, dass ein Termin
  // ausgemacht werden sollte.
  Assi.versendeUndEmpfangeNachricht(
    „Todo: Terminueberlegung","Einladender","Einladender");

  // Alle Eingeladenen zusammenfassen, damit sie ueber ein
  // versendeUndEmpfangeMulticastNachricht informiert werden koennen.
  Vector alle = new Vector(4,1);
  alle.addElement(„Eingeladener 1");
  alle.addElement(„Eingeladener 2");
  alle.addElement(„Eingeladener 3");
  alle.addElement(„Eingeladener 4");

  // fuer Multicast's empfiehlt sich eine laenger Uebermittlungdauer
  Assi.setDauerNachrichtenUebermittlung(80);

  // multicast-Nachricht verschicken
  Assi.versendeUndEmpfangeMulticastNachricht(
    „Terminvorschlag: 18. oder 19.","Einladender",alle);
```

```
    // fuer die folgenden Unicast-Nachrichten reicht eine kuerzere Zeit
    Assi.setDauerNachrichtenUebermittlung(30);

    // Antworten der einzelnen Eingeladenen zurueckschicken
    Assi.versendeUndEmpfangeNachricht(
      „18. okay, nicht 19.“,“Eingeladener 1“,“Einladender“);

    Assi.versendeUndEmpfangeNachricht(
      „beides okay“,“Eingeladener 2“,“Einladender“);

    Assi.versendeUndEmpfangeNachricht(
      „18. nicht vor 14 Uhr“,“Eingeladener 4“,“Einladender“);

    Assi.versendeUndEmpfangeNachricht(
      „Todo: Terminentscheidung“,“Einladender“,“Einladender“);

    // fuer Multicast‘s empfiehlt sich eine laenger Uebermittlungdauer
    Assi.setDauerNachrichtenUebermittlung(80);

    // noch eine Nachricht an alle mit dem endgueltigen Termin
    Assi.versendeUndEmpfangeMulticastNachricht(
      „Termin: 18. 15 Uhr“,“Einladender“,alle);

    // fuer die folgenden Unicast-Nachrichten reicht eine kuerzere Zeit
    Assi.setDauerNachrichtenUebermittlung(30);

    // Damit alle Prozesse gleichzeitig enden
    Assi.setPauseVorBeendeProzess(0);
    Assi.setPauseNachBeendeProzess(0);

    // da wir oben setPauseVorBeendeProzess auf Null gesetzt haben,
    // muessen wir jetzt von Hand Platz schaffen
    Assi.pause(20);

    // Dem Assistenten sagen, dass die Prozesse fertig sind
    Assi.beendeProzess(„Einladender“);
    Assi.beendeProzess(„Eingeladener 1“);
    Assi.beendeProzess(„Eingeladener 2“);
    Assi.beendeProzess(„Eingeladener 3“);
    Assi.beendeProzess(„Eingeladener 4“);

    // da wir oben setPauseNachBeendeProzess auf Null gesetzt haben,
    // muessen wir jetzt von Hand Platz schaffen
    Assi.pause(20);
  }
}
```

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfaßt und
nur die angegebenen Quellen benutzt zu haben

(Peter W. Schurr)