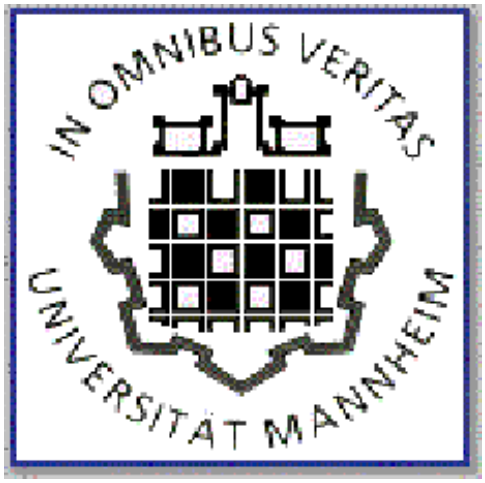
Animation von Flußkontrollmechanismen

Studienarbeit

von Joerg Bernauer



SS 97

Vorgelegt am

Lehrstuhl für Praktische Informatik
Prof. Dr. W. Effelsberg
Fakultät für Informatik
Uni Mannheim
Betreuer: Dipl. Wirtsch. Inf. Christoph Kumünch

Studienarbeit

theoretischer Teil

1.Tele Teaching

Motivation Seite 1
Aufgabe Seite 1
Kurzer Überblick Seite 2

2.Java

Geschichte von Java Seite 3
Wie funktioniert Java?(kurze Beschreibung der Sprache) Seite 4
Java und das Internet Seite 4

3.Flusskontrolle

Stop and Wait Seite 5
Schiebefenster Seite 8
Go back N Seite 11
Empfangsfenster = 1 Seite 12
Empfangsfenster > 1 Seite 15

4.Benutzerführung

Einstellen des Algorithmus Seite 18
Stop and Wait Seite 19
Schiebefenster Seite 19
Go back N Seite 20

5. Interner Programmaufbau

Klassenhierarchie Seite 22
Klassenbeschreibung Seite 22
(Wichtige Methoden und Attribute näher erklären) Seite 24

6.Ausblick Seite 31

1. Tele Teaching

Motivation:

Im Sommersemester 1996 wurde das Projekt Teleteaching zwischen den Universitäten Mannheim und Heidelberg gestartet. Das Projekt sah den gegenseitigen Austausch von Vorlesungen mittels eines digitalen Hochgeschwindigkeitsnetzes vor.

In der Vorlesung Rechnernetze von Professor Effelsberg wurde dann erstmals eine Vorlesung auf diese Weise übertragen. Begleitend zu dieser Vorlesung wurde das Unterichtsmaterial multimedial aufbereitet (Text, Graphiken, Video, Audio, etc.) und dem Studenten per WWW zur Nachbereitung und Prüfungsvorbereitung zur Verfügung gestellt.

Dieses Projekt sollte neben der Verbesserung des Vorlesungsangebot auch zu einem produktiverem Einsatz der Lehrkräfte dienen. Speziell im Falle der Vorlesung Rechnernetze stellte es auch ein praktisches Anwendungsbeispiel der Vorlesung dar.

An dem Projekt waren weiterhin Erziehungswissenschaftler(Mannheim) und Psychologer(Heidelberg) beteiligt. Diese sollten gewährleisten das es bei dem Projekt nicht bei einer technischen Spielerei bleibt sondern die oben genannten Ziele verwirklicht werden.

Zur Ergänzung des Projektes wurden zu einigen Themenbereichen der Vorlesung Studienarbeiten vergeben, die wichtige Kernbereiche durch interaktive Animationen/ Simulantinnen abrunden sollten.

Das Ziel dieser Animationen war es, den Stoff der Vorlesung so aufzubereiten, daß dieser spielerisch erlernt werden kann.

Aufgabe:

Gegenüber klassischen Medien, wie beispielsweise Büchern, bietet der Computer die Möglichkeit, interaktiv zu lernen, statt passiv zu konsumieren. Selbständiges Ausprobieren und Anwenden des Wissens optimiert den Lerneffekt.
Basierend auf diesem „Learning by doing“ Paradigma sollten daher zusätzlich zum reinen Vorlesungsstoff interaktive Animationen/Simulantinnen entwickelt werden, die über WWW abrufbar sind.

Ziel der hier verwirklichten interaktiven Animation war die Visualisierung von Flusskontroll-mechanismen in Rechnernetzen.
Folgende Mechanismen sollten animiert werden:

- Stop and Wait
- Schiebefenterprotokoll (Sliding Window)
- Go back N

Besonderer Wert sollte hierbei auf die objektorientierte Programmierung gelegt werden, daher wurde diese Studienarbeit auch in der neuen an C++ angelegten Programmiersprache Java entwickelt.

Kurzer Überblick:

Nach einer kurzen Abhandlung über die Programmiersprache Java, werden die einzelnen Algorithmen, die in dieser Studienarbeit animiert werden, kurz beschrieben. Danach werden die einzelnen Funktionen des Applets beschrieben. Das nächste Kapitel widmet sich dann der internen Übersicht über die beteiligten Klassen. Zuletzt werden noch weitere Aussichten beschrieben.

2. Java

Geschichte von Java

Eine Gruppe um das Sun Mitglied James Goshling wollte 1991 eine kleine Programmiersprache für Consumer Geräte, wie z.B. Kabelfernsehen- Schaltgeräte, entwickeln. Da solche Geräte weder eine hohe Rechenleistung noch viel Speicherplatz aufweisen, sollte diese Sprache wenig Platz beanspruchen und sehr sparsamen Quelltext erzeugen. Da außerdem verschiedene Hersteller verschiedene CPU´s verwenden, war es wichtig, sich nicht an eine Plattform zu binden. Das Projekt erhielt den Codenamen „Green".

Das Bedürfnis nach sparsamen Quelltext, ließ sie ein Modell aus der Frühzeit des PC wieder aufnehmen: die Programmiersprache UCSD PASCAL. Wie es bei USCD PASCAL der Fall ist entwickelten sie eine Portable Sprache, die Hilfscode erzeugt. Dieser Hilfscode kann nun auf jeder Maschine verwendet werden , die den richtigen Interpreter benutzt.

Da die Sun-Leute aus dem UNIX Hintergrund kamen, entwickelten sie ihre Sprache eher auf der Grundlage von C++ als von PASCAL. Auch wurde die Sprache eher objekt als verfahrensorientiert. Die Sprache war aber während der ganzen Zeit mehr ein Werkzeug als ein Endprodukt.

1992 brachte das Green-Projekt sein erstes Produkt, genannt „*7", auf den Markt. Es handelte sich um eine extrem intelligente Fernbedienung. Es fand sich hierfür jedoch kein Hersteller, und das Green- Team mußte eine andere Möglichkeit der Vermarktung finden. So boten sie ein TV Zusatzgerät an, das für neue Fernsehangebote wie Pay TV geeignet war. Auch dieser Vertrag kam nicht zustande.

Das Green Projekt suchte(unter neuem Namen „First Person Inc.") von Anfang 1993 bis Mitte 1994 nach einem Käufer für ihre Technologie, aber es fand sich keiner. Schließlich wurde First Person Inc. aufgelöst.

Während all dies bei Sun geschah, wurde der World Wide Web Teil des Internets immer größer. Der Schlüssel zum Netz ist ein Browser, der die Hypertext Seiten auf dem Monitor anzeigt. 1994 wurde meist Mosaic benutzt, ein nicht kommerzieller Web- Browser, der 1993 im Rechenzentrum der Universität Illinois entwickelt wurde.

Der eigentliche Browser wurde von Patrick Naughton und Jonathan Payne geschrieben und in den heutigen HotJava Browser integriert. Der HotJava Browser wurde in Java geschrieben, um die Möglichkeiten von Java zu demonstrieren. Aber die Entwickler hatten bereits die Leistungsfähigkeit der heutigen Applets im Kopf; also ermöglichten sie dem Browser Hilfs-

Bytecodes zu interpretieren. Dieser „Technologiebeweis“ wurde auf der SunWorld 95 am 23. Mai 1995 vorgestellt.

Der Durchbruch für die Verbreitung gelang Ende 1995 als Netscape entschied, daß die nächste Version von Netscape(Netscape 2.0) Java unterstützen sollte. Netscape2.0 erschien im Januar 1996 und unterstütze Java1.0.

Wie funktioniert Java?(kurze Beschreibung der Sprache)

Die Syntax von Java ist an die von C bzw, C++ angelegt. Javas Sicherheitseigenschaften werden sowohl Programmierer als auch Anwender der Programme zufriedenstellen. Mit Java ist es auch einfacher, fehlerfreien Quelltext zu produzieren als mit C oder C++.

Im folgenden werden einige Eigenschaften von Java näher beschrieben:

Einfachheit:
Die Syntax entspricht einer vereinfachten Version von C++. Es werden keine Header Dateien, keine Operatorüberladung, keine Zeigerarithmetik usw. mehr verwendet.

Objektorientiertheit:
Die objektorientierten Eigenschaften Javas sind mit denen von C++ vergleichbar, der Hauptunterschied liegt in der Mehrfachvererbung für die Java eine alternative Lösung bietet.

Architekturneutralität:
Die Entwickler von Java schufen einen Bytecode, für den es auf allen heute gebräuchlichen Plattformen Interpreter gibt.

Multithreading:
Die Threads von Java können die Vorteile von Multiprozessor Systemen nutzen, etwa die Basis eines Betriebssystems. Leider differieren Thread Implementierungen stark von Plattform zu Plattform, und Java macht keine Anstrengungen, hier plattformunabhängig zu werden.

Java und das Internet

Durch ihre Laufzeitbibliothek, die ihrer Plattform Unabhängigkeit garantiert (identischer Quelltext kann unter Windows 95, Solaris, UNIX, Macintosh usw. verwendet werden) ist Java prädestiniert für Internet Programmierung.

[JD C]

Nach der Beschreibung des Werkzeugs sollen nun die Algorithmen, die animiert werden sollen beschrieben werden.

Algorithmen zur Flusskontrolle

3.Flusskontrollealgorithmen:

Im folgenden Kapitel werden die Flusskontrollalgorithmen Stop & Wait, Schiebefenster und Go back N beschrieben.
Flusskontrolle ist nötig, damit ein schnellerer Sender einen langsameren Empfänger nicht mit Daten überschwemmt.

1. Stop & Wait:

kurze Erläuterung des Algorithmus:
Der Sender sendet immer ein Packet. Falls dieses ordnungsgemäß bestätigt wurde darf der Sender das nächste Packet senden. Falls er aber ein gewisses Zeitintervall keine Bestätigung erhält, sendet er dasselbe Paket abermals.
kurze Eläuterung der Abbildungen:
-Jedes Kreissekment steht für ein Datenpaket
-Ein dunkles Kreissekment bedeutet, daß ein Datenpaket gesendet werden darf
-Ein hell gefärbtes Kreissekment bedeutet, daß der Sender bzw. Empfänger auf eine Bestätigung des gesendeten Pakets wartet.

Ausgangssituation:

Daten sollen von der Quelle zu der Senke übertragen werden. Dabei kann beim Stop & Wait Protokoll immer nur 1 Paket auf einmal übertragen werden (wie in Abbildung 1 zu sehen ist); es findet keine Pufferung statt.

Abb. 1(Ausgangssituation des bei Stop and Wait)

Die Quelle sendet nun das 1. Paket (Abbildung 2) und verfällt in einen „Ruhestatus“.

Abb.2(Situation nach dem senden des 1. Packets)

I. Paket kommt beim Empfänger an

Der Empfänger nimmt das Paket entgegen und schickt eine Bestätigung des Pakets zurück (Abbildung 3).

Abb.3(Situation nachdem das 1. Packet bestätigt wurde)

Ia.Bestätigung erreicht den Sender unbeschadet

Der Sender weiß nun das 1. Paket ist ordnungsgemäß angekommen (Abbildung 4) und er sendet das nächste; das ganze beginnt von vorne.

Abb.4(Situation nach dem Senden des nächsten Pakets)

Ib.Es kommt keine Bestätigung beim Sender an, da diese unterwegs verloren geht

Der Sender befindet sich nach wie vor im Ruhestatus; er wartet eine gewisse Zeit (Time-Out Intervall), danach sendet er das gleiche Paket abermals (Abbildung 5).

Abb.5(Situation nach dem erneuten Senden des Packets)

II.Paket geht unterwegs verloren

Der Empfänger erhält kein Paket und sendet daher auch keine Rückmeldung.

Schiebefensteralgorithmus:

Beim Stop & Wait Algorithmus wurden Datenrahmen in eine einzige Richtung übertragen. In der Praxis werden jedoch Übertragungsmöglichkeiten in beide Richtungen benötigt. Für Datenübertragungen im Duplexmodus könnte man zwei separate Kommunikationskanäle einrichten und je einen für den Simplexdatenverkehr in je eine Richtung verwenden. In diesem Fall hätte man zwei physisch getrennte Leitungen, je mit einem „Vorwärtskanal“ (für Datenübertragungen) und einem „Rückwärtskanal“ (für Bestätigungen). In beiden Fällen würde man die Bandbreite des „Rückwärtskanals“ fast vollständig ungenutzt lassen. Der Anwender hätte die Gebühren für zwei Leitungen zu zahlen und könnte nur die Kapazität von einem nutzen.
[RN A]

Kurze Erläuterung des Algorithmus in Worten:

Ein wichtiges Element dieses Algorithmus ist die Fenstergröße, die angibt wieviele Pakete, der Sender abschicken darf, bevor er auf die Bestätigungen des Empfängers warten muß.

Im Normal Fall laufen alle Pakete zum Empfänger, dieser bestätigt sie und wenn die Bestätigungen beim Empfänger angekommen sind, darf dieser die nächsten Pakete senden.
Falls nun aber eines der Pakete unterwegs verloren geht, wird dieses natürlich auch nicht vom Empfänger bestätigt, das heißt für dieses Paket läuft keine Bestätigung zum Empfänger zurück.
Dieser Fall tritt auch ein, wenn eine Bestätigung eines Paketes das angekommen ist unterwegs verlorengeht. Der Sender registriert nun, daß das Time-out Intervall abgelaufen ist und sendet das gleich Paket abermals.

Ausgangssituation(Fenstergröße = 4):

Abb.6
Der Sender sendet je nachdem, wie es ihm die Fenstergröße vorgibt, Pakete los. Sobald genauso viele Pakete unterwegs sind, wie Puffer vorhanden waren, darf der Sender nicht mehr weitersenden er verfällt in einen Ruhestatus.

Abb.7

Je nach Bestätigungspolitik bestätigt der Empfänger nun jedes, jedes zweite, jedes dritte etc. Paket, das bei ihm angekommen ist(Abbildung 8).

Abb.8

1.Fall:
(Keine Bestätigung kommt beim Sender an)

Der Sender befindet sich nach wie vor im Ruhestatus. Er verweilt in diesem solange bis eine gewisse Zeit(Time-Out Intervall) verstrichen ist. Jetzt muß er alle Pakete erneut schicken (wie in Abbildung 9 zu sehen ist). Danach verfällt er wieder in den Ruhestatus, das ganze beginnt von vorn.

Abb.9

2.Fall:
(Bestätigungen kommen beim Sender an)

Je nachdem wie viele Bestätigungen beim Sender eingetroffen sind, darf dieser jetzt weitersenden(s. Abbildung 10) .
Er darf allerdings nicht weiter als das letzte bestätigte Paket + Fenstergröße senden.

Abb.10

Go back N:

Probleme bei den anderen Algorithmen:

Bis jetzt ist man einfach davon ausgegangen, daß die Übertragungszeit, die ein Rahmen braucht, bis er beim Empfänger ankommt, und die zum Senden der Bestätigung erforderliche Zeit verschwindend gering ist. Manchmal ist diese Annahme schlichtweg falsch. In solchen Fällen kann eine lange Umlaufzeit negative Folgen in Bezug auf die Auslastung der gesamten Bandbreite haben. Betrachtet man z.B. einen Satelitenkanal mit 50 Kbps und einer Umlauf-Ausbreitungsverzögerung von 500 ms. Angenommen man überträgt einen 1000 Bit- Rahmen per Satellit. Bei t=0 beginnt der Sender mit der Übertragung des ersten Rahmens. Bei t=20 ms ist die Übertragung beendet. Frühestens bei t=270 ms ist der Rahmen vollständig beim Empfänger angekommen und frühestens bei t=520 ms erreicht die Bestätigung den Sender unter optimalen Umständen(d.h. ohne Wartezeit beim Empfänger und kleinen Bestätigungsrahmen). Das bedeutet, daß der Sender 500 bis 520 ms oder 96% der Zeit blockiert war(d.h. nur 4% der verfügbaren Bandbreite wurden genutzt). Dies verdeutlicht, daß die Kombination von langen Transferzeiten, großer Bandbreiten und kleinen Rahmengrößen die Effizienz des Übertragungsweges stark reduziert.

Das oben geschilderte Problem kann als Folge davon angesehen werden, daß der Sender auf eine Bestätigung warten muß, bevor er den nächsten Rahmen abschicken kann. Durch Lockerung dieser Vorgabe kann die Effizienz gesteigert werden. Grundsätzlich kann man das Problem lösen, wenn der Sender bis zu w Rahmen abschicken darf(anstatt nur einen), bevor er blockiert wird. Wird w sinnvoll gewählt, kann der Sender für die Dauer der Umlauf-übertragung ununterbrochen Rahmen abschicken, ohne daß das Fenster gefüllt wird.

Diese Technik nennt man Pipelining.

Die Anwendung von Pipelining zur Rahmenübertragung über einen unzuverlässigen Kommunikationskanal kann folgende Probleme verursachen. Was passiert beispielsweise, wenn ein Rahmen inmitten eines langen Datenflusses zerstört wird oder verlorengeht? Sehr viele nachfolgende Rahmen kommen beim Empfänger an, bevor der Sender herausfindet, daß etwas schiefgegangen ist. Sobald ein zerstörter Rahmen beim Empfänger ankommt, sollte er logischerweise verworfen werden. Doch was soll der Empfänger mit all den darauffolgenden korrekten Rahmen tun?

Zur Handhabung von Fehlern beim Pipelining gibt es zwei grundlegende Ansätze:

Beim sogenannten Go-back-N(mit Empfangsfenster der Größe 1) verwirft der Empfänger alle nachfolgenden Rahmen und schickt keine Bestätigungen.

Beim Go-back-N(mit Empfangsfenster größer als 1) speichert der Empfänger alle korrekten Rahmen, die dem fehlerhaftem folgen.
[CN T]

Go-back-N:
(Empfangsfenster hat die Größe 1)

Der Sender sendet ohne Unterlaß, falls er aber eine bestimmte Zeit (Time-Out Intervall) keine Bestätigung erhält, beginnt er erneut mit dem letzten unbestätigten Paket.

Der Empfänger nimmt die Pakete solange entgegen, bis eines unterwegs verlorengeht, oder fehlerhaft ankommt. Ab diesem Zeitpunkt verwirft der Empfänger alle Pakete, bis er jenes erhält das ursprünglich verlorenging.

Das folgende Beispiel soll den Vorgang verdeutlichen:

Der Sender beginnt Pakete zu senden(Abbildung 11).

Abb.11

Der Empfänger nimmt diese entgegen(Abbildung 12).

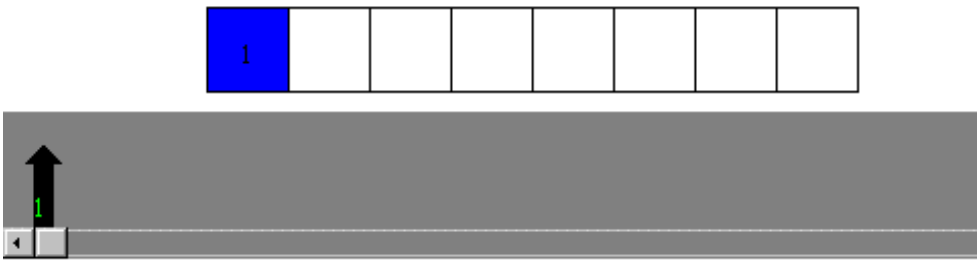


Abb.12

1.Fall
(alle Pakete kommen an):

Der Sender sendet solange bis das Time-Out Intervall abgelaufen ist; falls er bis dahin immer noch keine Bestätigung erhalten hat, sendet er ab dem letzten vom Empfänger bestätigten

Paket weiter. Falls noch keines bestätigt wurde sendet er, wie in Abbildung 13 zu sehen ab, dem ersten.

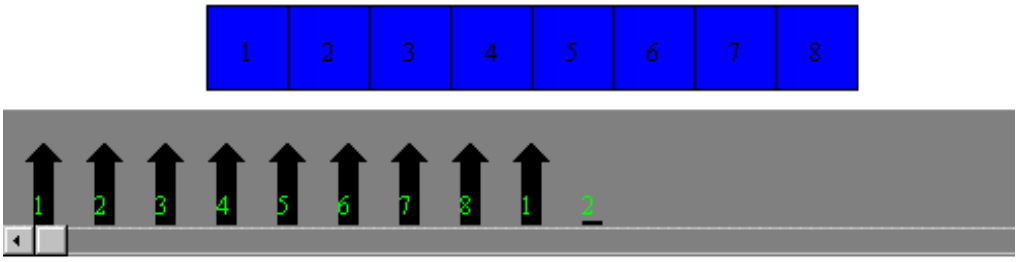


Abb.13

2.Fall
(der Empfänger erhält ein Paket nicht, da dieses unterwegs verlorenging oder fehlerhaft ankam):

Der Sender nimmt dies nicht wahr und sendet bis zum Time-Out Intervall weiter. falls er bis dahin immer noch keine Bestätigung erhalten hat, sendet er ab dem letzten vom Empfänger bestätigten Paket weiter (Abbildung 14). Falls noch keines bestätigt wurde sendet er ab dem ersten. (siehe oben)

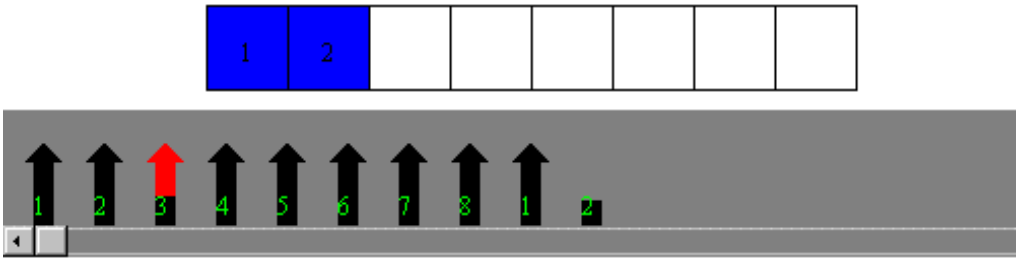


Abb.14

Falls der Sender nun eines oder mehrere Pakete verarbeitet hat, sendet er eine Bestätigung an den Sender zurück(Abbildung 15):

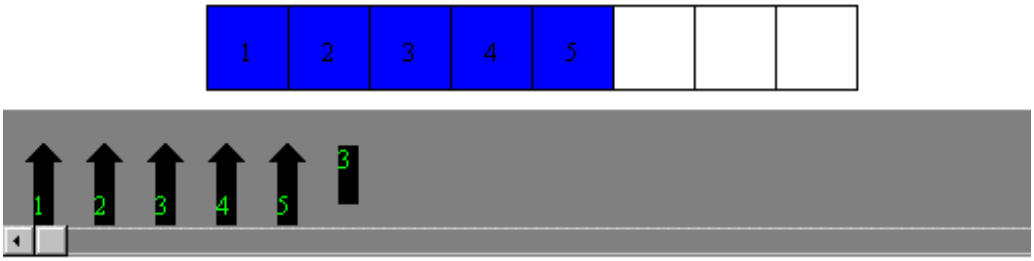


Abb.15

1.Fall
(das Time-Out Intervall ist abgelaufen und das nächste zu sendende Paket ist kleiner als dieses das eben bestätigt wurde):

Der Sender sendet ab dem bestätigtem Paket weiter; das Time-Out Intervall wird zurückgesetzt(Abbildung 16).

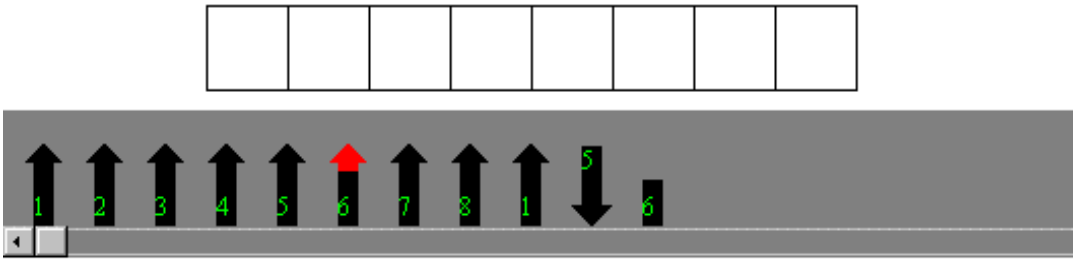


Abb.16

ansonsten 2.Fall
der Sender sendet ganz normal weiter (Abbildung 17), nur das Time-Out Intervall wird zurückgesetzt.

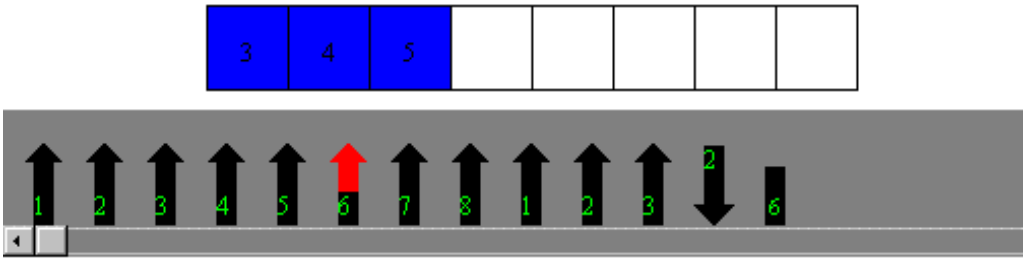


Abb.17

Go-back-N
(Empfangsfenster ist größer als 1)

Der Sender sendet ohne Unterlaß, falls er aber eine bestimmte Zeit (Time-Out Intervall) keine Bestätigung erhält sendet er wieder nach dem letzten bestätigten Paket weiter.

Der Empfänger nimmt alle Pakete entgegen, auch wenn eines unterwegs verloren geht. Er kann ein Paket allerdings nur dann bestätigen, wenn alle vorhergehenden Pakete angekommen sind, denn das Bestätigen eines Pakets schließt das Bestätigen aller anderen Pakete, die zuvor angekommen sind mit ein.

Beispiel:

Der Sender beginnt Pakete zu senden(Abbildung 18).

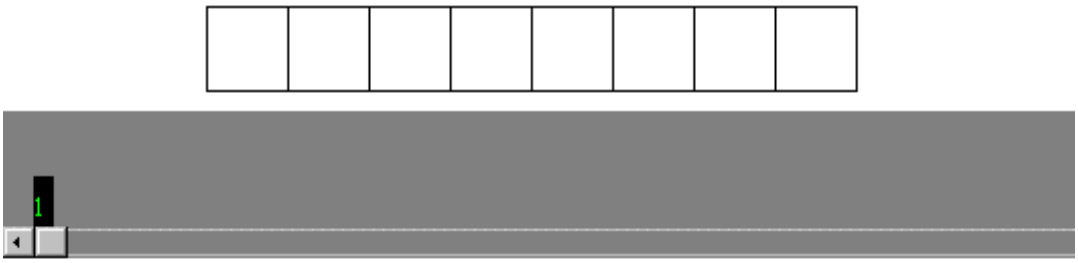


Abb.18

Der Empfänger nimmt diese entgegen(Abbildung 19).

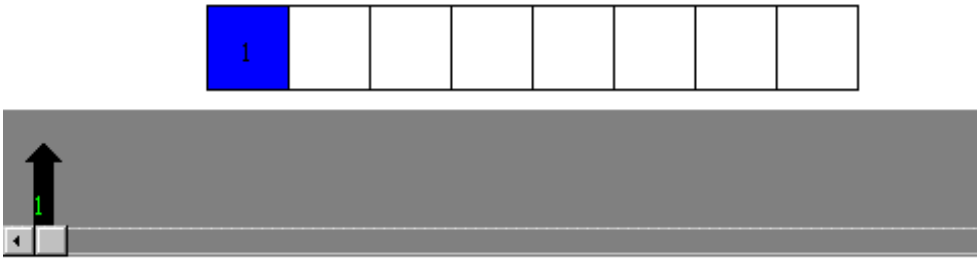


Abb.19

1.Fall
(alle Pakete kommen an):

Der Sender sendet solange bis das Time-Out Intervall abgelaufen ist; falls er bis dahin immer noch keine Bestätigung erhalten hat, sendet er ab dem letzten vom Empfänger bestätigten Paket weiter. Falls noch keines bestätigt wurde sendet er ab dem ersten(Abbildung 20).

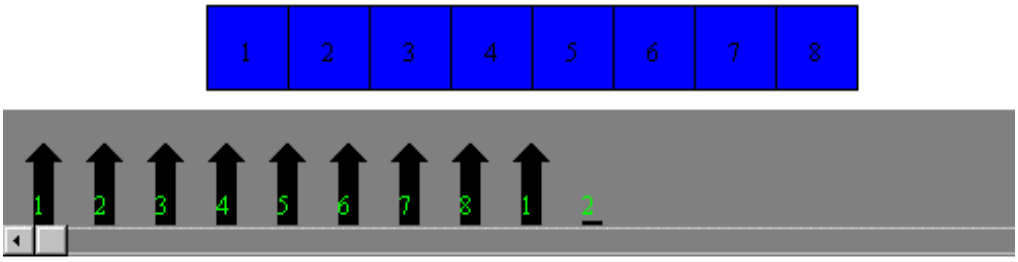


Abb.20

2.Fall
(der Empfänger erhält ein Paket nicht, da dieses unterwegs verlorenging oder fehlerhaft ankam):

Der Sender nimmt dies nicht wahr und sendet bis zum Time-Out Intervall weiter. falls er bis dahin immer noch keine Bestätigung erhalten hat, sendet er ab dem letzten vom Empfänger bestätigten Paket weiter(Abbildung 21). Falls noch keines bestätigt wurde, sendet er ab dem ersten. (siehe oben)

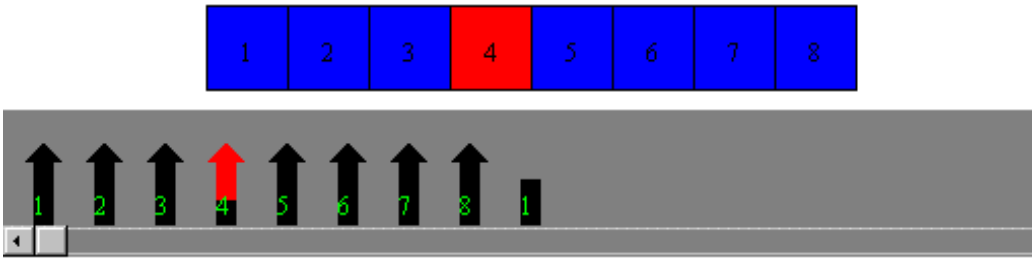


Abb.21

Falls der Sender nun eines oder mehrere Pakete verarbeitet hat sendet er eine Bestätigung an den Sender zurück(Abbildung 22):

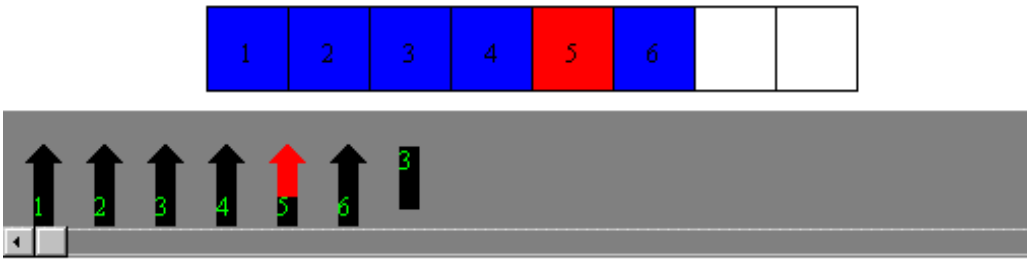


Abb.22

1.Fall
(das Time-Out Intervall ist abgelaufen und das nächste zu sendende Paket ist kleiner als dieses das eben bestätigt wurde):

Der Sender sendet ab dem bestätigtem Paket weiter(Abbildung 23); das Time-Out Intervall wird zurückgesetzt.

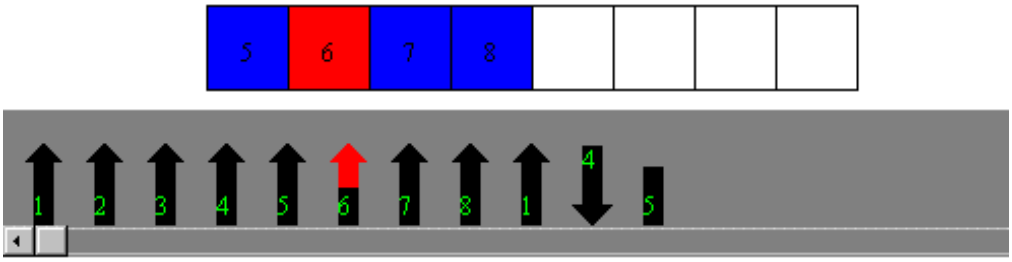


Abb.23

ansonsten 2.Fall

der Sender sendet ganz normal weiter(Abbildung 24), nur das Time-Out Intervall wird zurückgesetzt.

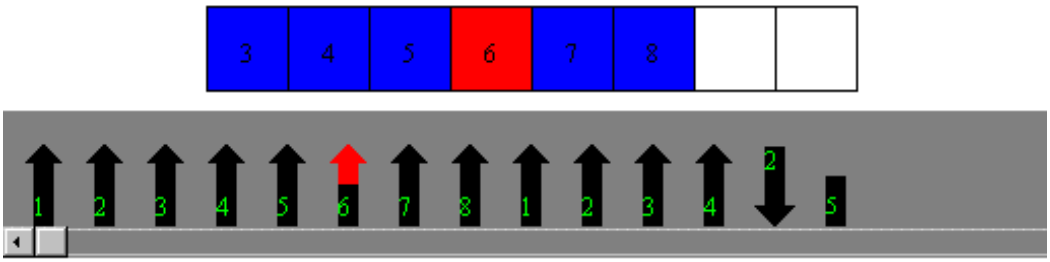


Abb.24

Falls nun ein Paket, das beim 1. Durchlauf nicht angekommen war, bei einem späteren Durchlauf korrekt ankommt, kann auch ein höheres Paket bestätigt werden, das zunächst gepuffert wurde.

Im nächsten Kapitel soll nun erläutert werden wie die konkreten Animationen der Algorithmen bedient werden.

Anhang:

4.Benutzerführung:

Nach dem Laden des Applets ist folgendes Bild zu sehen:

Abb.25

Ganz oben im Bild befindet sich die Menuleiste, direkt darunter das Textfenster. Das Fenster in der Mitte ist das sogenannte Aktionsfenster, worin später der aktuelle Algorithmus zu sehen ist. Ganz unten sind die 3 Steuerungsknöpfe zu sehen(Start, Stop, Halt).

Ab nun werden Anweisungen im Textfenster gegeben, wie das Applet zu bedienen ist.

Sämtliche Voreinstellungen werden mit Hilfe der Menuleiste durchgeführt.

Datei	Bearbeiten	Algorithmen	
Schliessen	*Fenstergroesse*	*Stop & Wait*	
Neu	*Puffergroesse*	*Go back N→*	*Fenstergroesse = 1*
	Geschwindigkeit		*Fenstergroesse > 1*
		Schiebefenster	

Zuerst wird man nun aufgefordert den Algorithmus einzustellen:

Man wählt hierzu den Menupunkt *Algorithmus* aus und klickt den gewünschten Algorithmus an.

I. Stop & Wait:

Nach dem Wählen dieses Algorithmus erscheint rechts und links im Aktionsfenster jeweils ein Pufferungskreis, der angibt, wie viele freie Puffer zur Verfügung stehen. Blau bedeutet der Puffer ist frei, dunkles Grau bedeutet, daß dieses Paket im Moment unterwegs ist. Zwischen den beiden Kreisen befinden sich die beiden Leitungen für Pakete und Bestätigungen.

Vor Starten des Algorithmus kann man noch die Animationsgeschwindigkeit einstellen.
Hierzu wählt man den Menupunkt *Bearbeiten* und klickt hier die Option *Geschwindigkeit* an. Man kann hier zwischen schneller, mittlerer und langsamer Animationsgeschwindigkeit wählen.

Nach Drücken des Start Buttons fängt der Sender nun an zu senden:

Der Algorithmus läuft nun bis ein weiter Aktionsknopf gedrückt wird.
Während der Algorithmus läuft, kann durch Anklicken eines Pakets oder einer Bestätigung ein Übertragungsfehler simuliert werden; das Paket geht verloren.

Mithilfe des Halt Buttons kann der Algorithmus jetzt „Eingefroren“ werden: Er verharrt in der aktuellen Position. Diese kann nun in Ruhe betrachtet werden. Durch das Drücken des Start Buttons läuft der Algorithmus weiter.

Falls man nun den Algorithmus beenden will, ist dies mit Hilfe des Stop Buttons möglich. Der Sender stoppt aber erst dann, wenn das aktuell gesendete Paket vom Empfänger noch bestätigt wurde.

II. Schiebefenster:

Folgende Voreinstellungen können bei diesem Algorithmus vorgenommen werden.

1.	Geschwindigkeit	siehe Stop & Wait
2.	Fenstergröße	Hierzu wählt man den Menupunkt *Bearbeiten* und klickt hier die Option *Fenstergröße* an. Dabei kann festgelegt werden wie viele Pakete der Sender schicken darf bevor er in den „Ruhestatus“ verfällt.

Nach Drücken des Start Buttons fängt der Sender nun an zu senden:

Der Algorithmus läuft nun bis ein weiter Aktionsknopf gedrückt wird.
Während der Algorithmus läuft, kann durch Anklicken eines Pakets oder einer Bestätigung ein Übertragungsfehler simuliert werden; das Paket geht verloren.

Mithilfe des Halt Buttons kann der Algorithmus jetzt „Eingefroren“ werden: Er verharrt in der aktuellen Position. Diese kann nun in Ruhe betrachtet werden. Durch das Drücken des Start Buttons läuft der Algorithmus weiter.

Auch hier betätigt man, falls man den Algorithmus beenden will, den Stop Button. Sobald alle Pakete, die vom Sender noch gesendet wurden, beim Sender wiederum bestätigt wurden endet der Algorithmus.

Go back N:

Der Go back N Algo wurde in den beiden Varianten Fenstergröße=1 und Fenstergröße<>1 programmiert, daher muß man zuerst noch konkret den Algorithmus ausgewählen. Hierzu wählt man den Menupunkt *Algorithmen* nach der Wahl des Go back N Algorithmus stehen folgende beiden Optionen zur Auswahl:

III. Go back N mitFenstergröße 1:

Auch hier kann vor dem Start des Algorithmus eine Voreinstellung vorgenommen werden. Die Puffergröße des Empfängers kann gewählt werden.
Hierzu wählt man den Menupunkt *Bearbeiten* und klickt hier die Option *Puffergröße* an. Dabei kann festgelegt werden wieviele Pakete der Empfänger puffern kann bevor er die Pakete, die er nicht verarbeiten kann, verwerfen muß.

Nach Drücken des Start Buttons fängt der Sender nun an zu senden:

Der Algorithmus läuft nun bis ein weiter Aktionsknopf gedrückt wird.
Während die Pakete zum Empfänger laufen kann man durch Anklicken des unteren Drittels des Aktionsfenster, auch bei diesem Algorithmus einen Übertragungsfehler simulieren. Der Pfeil, der das Paket simulieren soll, färbt sich ab diesem Zeitpunkt rot.

Falls man nun ein oder mehrere Pakete bestätigen will, klickt man das entsprechende Paket im oberen Drittel des Aktionsfenster an. Daraufhin wird dieses und alle Pakete, die zuvor empfangen wurden bestätigt.

Mithilfe des Halt Buttons kann der Algorithmus jetzt „Eingefroren“ werden: Er verharrt in der aktuellen Position. Diese kann nun in Ruhe betrachtet werden. Durch das Drücken des Start Buttons läuft der Algorithmus weiter.

Falls man nun den Algorithmus beenden will, ist die mit Hilfe des Stop Buttons möglich. Der Sender stoppt aber erst dann, wenn das aktuell gesendete Paket vom Empfänger noch bestätigt wurde.

III. Go back N mit Fenstergröße größer als 1:

Auch hier kann vor dem Start des Algorithmus eine Voreinstellung vorgenommen werden. Die Puffergröße des Empfängers kann gewählt werden.

Hierzu wählt man den Menupunkt *Bearbeiten* und klickt hier die Option *Puffergröße* an. Dabei kann festgelegt werden wie viele Pakete der Empfänger puffern kann bevor er die Pakete, die er nicht verarbeiten kann, verwerfen muß.

Nach Drücken des Start Buttons fängt der Sender nun an zu senden:

Der Algorithmus läuft nun bis ein weiter Aktionsknopf gedrückt wird.
Während die Pakete zum Empfänger laufen kann man durch anklicken des unteren Drittels des Aktionsfenster, auch bei diesem Algorithmus einen Übertragungsfehler simulieren. Der Pfeil, der das Paket simulieren soll färbt sich ab diesem Zeitpunkt rot.

Falls man nun ein oder mehrere Pakete bestätigen will, klickt man das entsprechende Paket im oberen Drittel des Aktionsfenster an. Daraufhin wird dieses und alle Pakete, die zuvor empfangen wurden, bestätigt.

Mithilfe des Halt Buttons kann der Algorithmus jetzt „Eingefroren“ werden: Er verharrt in der aktuellen Position. Diese kann nun in Ruhe betrachtet werden. Durch das Drücken des Start Buttons läuft der Algorithmus weiter.

Falls man nun den Algorithmus beenden will, ist die mit Hilfe des Stop Buttons möglich. Der Sender stoppt aber erst dann, wenn das aktuell gesendete Paket vom Empfänger noch bestätigt wurde.

Der Unterschied zum Go back N Algorithmus mit Fenstergröße 1 besteht darin, daß nicht jedes beliebige gepufferte Paket bestätigt werden kann, es können nur Pakete bestätigt werden, vor denen keine Pakete verlorengegangen sind(durch Rotfärbung gekennzeichnet).

Um das Applet neu zu laden kann unter dem Menupukt *Datei* die Option *neu* gewählt werden.

Falls man nun das Applet verlassen will wählt man unter dem Menupunkt *Datei* einfach die Option *schliessen*.

Nachdem das Arbeiten mit dem Applet erklärt wurde, soll nun ein interner Einblick in die Programmstruktur gegeben werden.

5. Interner Programmaufbau:

Hierarchieübersicht über Klassen(Abbildungen 26-29):

= ruft die Klasse auf

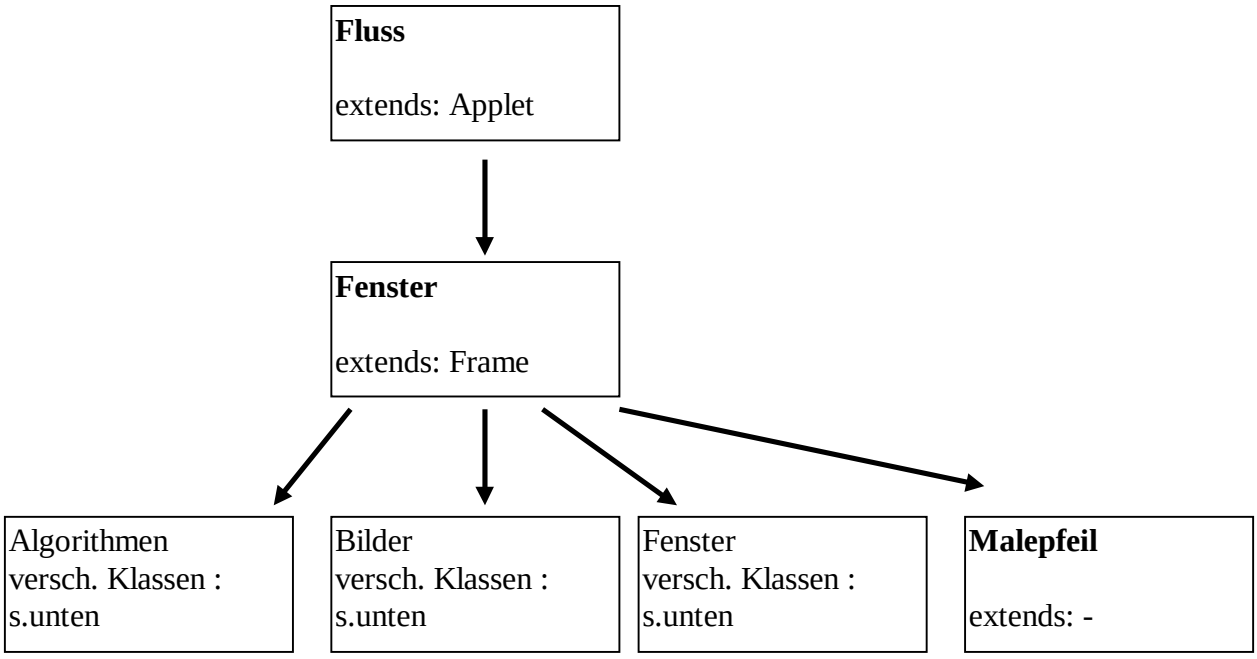


Abb.26

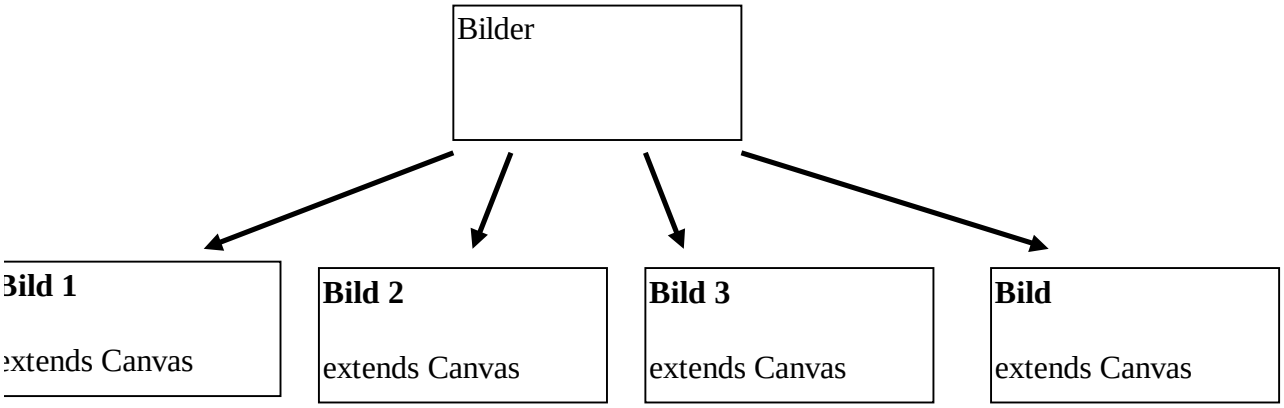
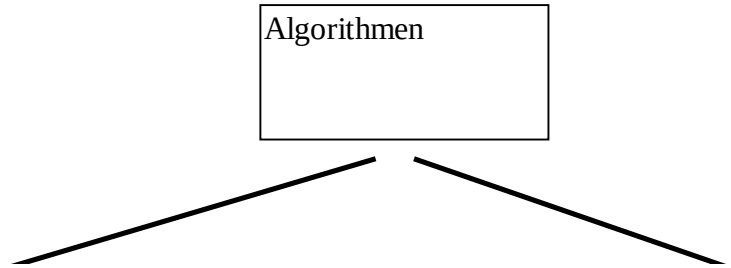


Abb.27



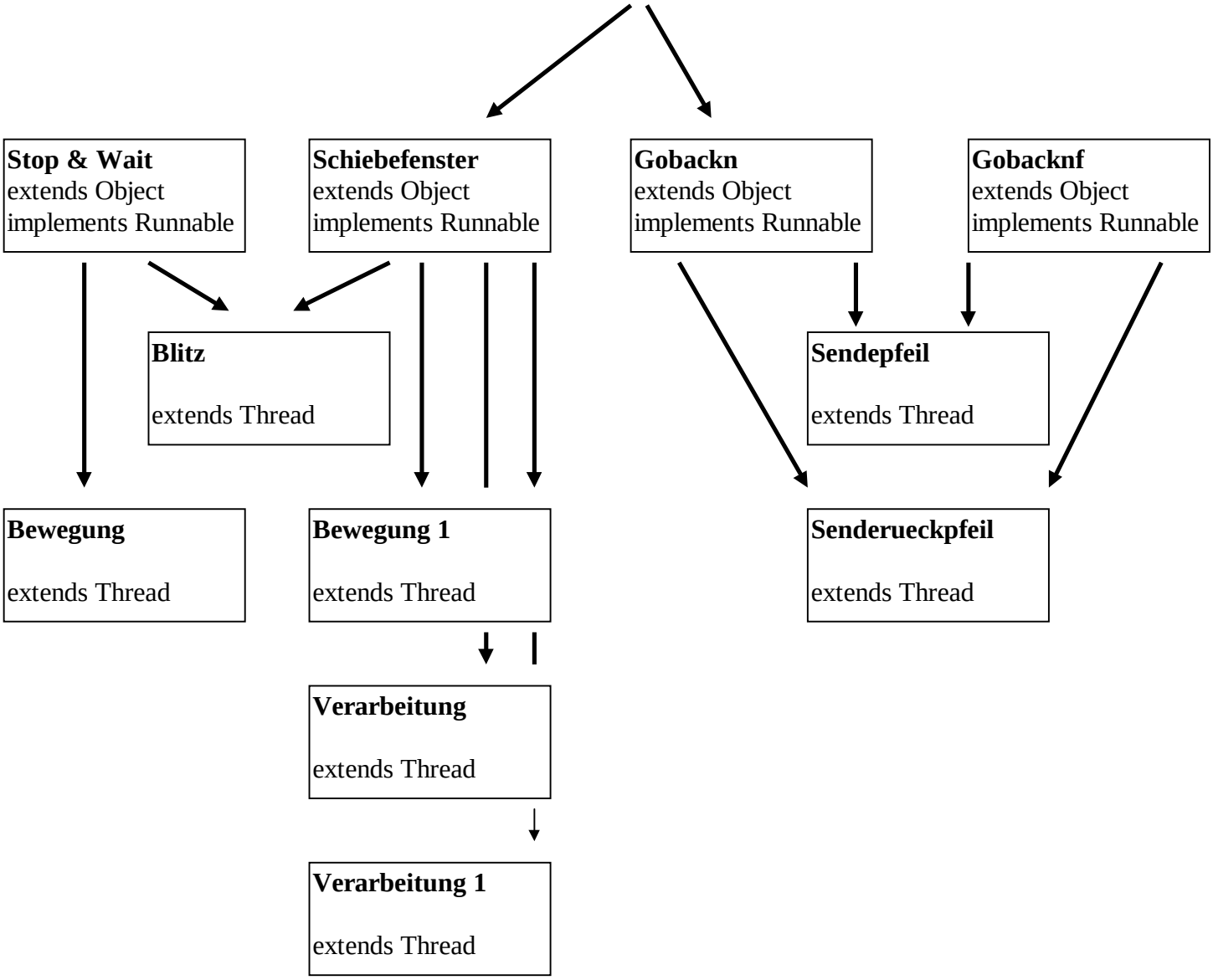
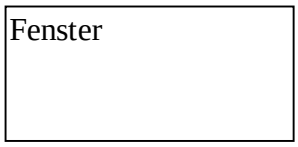

Stop & Wait
extends Object
implements Runnable
Schiebefenster
extends Object
implements Runnable
Gobackn
extends Object
implements Runnable
Gobacknf
extends Object
implements Runnable
Blitz
extends Thread
Sendepfeil
extends Thread
Bewegung
extends Thread
Bewegung 1
extends Thread
Senderueckpfeil
extends Thread
Verarbeitung
extends Thread
Verarbeitung 1
extends Thread

Abb.28


Fenster

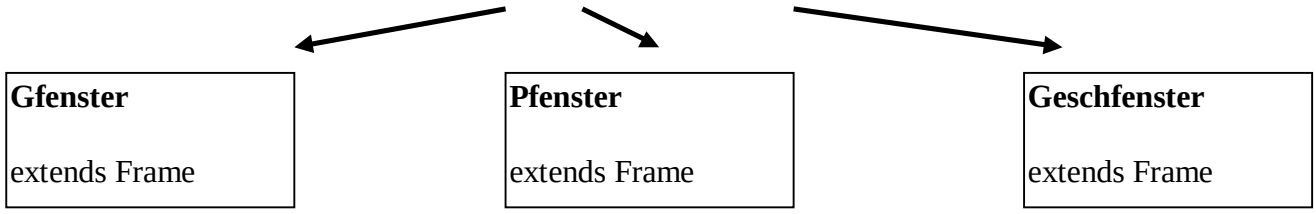


Abb.29

Konkrete Beschreibung der Klassen:

Beim Laden des Applets wird zuerst die **Fluss Klasse** geladen, diese macht jedoch nichts anderes als die ***Fenster Klasse*** aufzurufen und einen Hilfsthread zu definieren.
(Der Hilfsthread dient nur dazu, die unter Netscape auftretenden Security Probleme beim Anhalten eines anderen Threads zu umgehen).

Durch Laden der **Klasse Fenster** wird das Applet erst initialisiert, hier wird alles deklariert, was später auf dem Bildschirm zu sehen ist:

- die Text Area, die im oberen Drittel des Bildschirms zu sehen ist.
- die **Bild Klasse,** in der der Algorithmus später visualisiert wird.
- die **Klassen Bild1, Bild2, Bild3,** die definiert werden, um bei den beiden Go back N Algorithmen das Bild noch mal dreizuteilen.
- die Menubar, mit deren Hilfe die jeweiligen Voreinstellungen vorgenommen werden können.
- Außerdem die 3 Buttons mit denen der jeweilige Algorithmus gestartet, angehalten oder gestoppt werden kann.

All dies wird auf die entsprechende Position auf dem Bildschirm plaziert, anschließend wird er ausgegeben.

Desweiteren werden die 4 Algorithmenklassen **StopAndWait, Schiebefenster, Gobackn, Gobacknf** deklariert.

Zwei weitere Funktionen, die in dieser Klasse definiert werden, sind die handleEvent und action Funktion.

In der handleEvent Funktion wird die Scrollbar, die bei den Go back N Algorithmen benötigt wird, verwaltet. Falls gescrollt wurde, werden hier die Werte ermittelt und an die Translate Funktion weitergegeben, dort werden sie umgesetzt und die Pfeile auf ihren neuen Koordinaten auf dem Bildschirm ausgegeben.

In der action Methode wird die Menubar und die Buttons verwaltet. Falls ein entsprechender Menupunkt aufgerufen wird, wird die entsprechende Funktion ausgeführt oder der

entsprechende Algorithmus gestartet. Auch wenn ein Button gedrückt wird, wird dies hier abgefangen und die entsprechende Funktion ausgeführt, gestoppt oder angehalten.

Die **Bild Klasse (extends Canvas)** ist für die graphische Darstellung des Applets verantwortlich. Alles was in den verschiedenen Algorithmen an Graphik ausgegeben wird wird im Bereich dieser Klasse ausgegeben.

In dieser Klasse sind auch alle Variablen definiert, auf die mehrere Klassen zugreifen müssen, da diese Klasse an alle anderen(wegen der graphischen Umsetzung der Algorithmen) übergeben wird.

Die paint Methode gibt die Grundelemente der beiden Algorithmen Stop and Wait und Schiebefenster, die beiden Kreise und die Leitungen, aus, falls einer der beiden Algorithmen gewählt wurde.

Wie oben in der Bedienungsanleitung geschildert wurde ist es möglich, durch Anklicken der Pakete einen Übertragungsfehler zu simulieren. In der Methode mouseDown wird nun geregelt, ob ein Paket angeklickt wurde und wenn ja welches. Die Informationen werden an die entsprechende Klasse weitergegeben.

Falls in der Menubar unter dem Menupukt *Bearbeiten* der Punkt Fenstergroesse ausgewählt wird wird die **Klasse Gfenster (extends Frame)** aufgerufen.

Wurde dieser Menupukt ausgewählt, wird auf dem Bildschirm ein Fenster ausgegeben. In dieses Fenster wird eine Checkbox Gruppe und ein Button eingebunden. Außerdem wird ein Text ausgegeben.

In der action Methode wird eine Variable gesetzt je nachdem welche Checkbox gerade angeklickt wurde.

Nachdem der Button gedrückt wurde wird das Fenster geschlossen und die Variable an die Bild Klasse weitergegeben.

Falls in der Menubar unter dem Menupukt *Bearbeiten* der Punkt Puffergroesse ausgewählt wird wird die **Klasse Pfenster (extends Frame)** aufgerufen.

Wurde dieser Menupukt ausgewählt wird auch hier auf dem Bildschirm ein Fenster ausgegeben. In dieses Fenster wird eine Checkbox Gruppe und ein Button eingebunden. Außerdem wird ein Text ausgegeben.

In der action Methode wird eine Variable gesetzt je nachdem welche Checkbox gerade angeklickt wurde.

Nachdem der Button gedrückt wurde, wird das Fenster geschlossen und die Variable an die Bild Klasse weitergegeben.

Falls nun in der Menubar unter dem Menupukt *Bearbeiten* der Punkt Geschwindigkeit ausgewählt wird wird die **Klasse Geschfenster (extends Frame)** aufgerufen.

Wurde dieser Menupukt ausgewählt wird auf dem Bildschirm ein Fenster ausgegeben. In dieses Fenster wird eine Checkbox Gruppe und ein Button eingebunden. Außerdem wird ein Text ausgegeben.

In der action Methode wird eine Variable gesetzt je nachdem welche Checkbox gerade angeklickt wurde.

Nachdem der Button gedrückt wurde wird das Fenster geschlossen und die Variable an die Bild Klasse weitergegeben.

Die **Klasse StopandWait (extends Object implements Runnable)** ist für den korrekten Ablauf dieses Algorithmus verantwortlich. In ihr wird zuerst ein Paket in den Kellerspeicher gestellt, der in der Fenster Klasse definiert werden. Danach wird das erste Kreissegment dunkelgrau gefärbt, um zu kennzeichnen, daß dieses Paket unterwegs ist. Schließlich wird die Bewegungsklasse aufgerufen, die dafür verantwortlich ist, daß das Paket über die Leitung läuft. Schließlich wird der Thread angehalten.

Nachdem die Klasse aus der Bewegungsklasse wieder „geweckt“ wurde (dann, wenn ein Paket am Ende der Leitung angekommen ist), läuft der eigentliche Algorithmus ab. Falls es ordnungsgemäß angekommen ist wird das entsprechende Kreissegment gefüllt, und ein neues in den entsprechenden Kellerspeicher gestellt. Falls das Paket nicht angekommen ist, wird das gleiche noch einmal in den Kellerspeicher gestellt. Der Kreis wird nicht verändert. Danach wird wiederum die Bewegungsklasse aufgerufen.

Der eigentliche Algorithmus wird ab dem Moment verlassen, wenn der Stop Button gedrückt wurde. Der Sendevorgang wird aber erst dann gestoppt, wenn das aktuelle Paket noch ordnungsgemäß bestätigt wurde. Falls dies nicht der Fall ist, wird das entsprechende Paket immer wieder in den Kellerspeicher gestellt und die Bewegungsklasse aufgerufen. Falls es dann schließlich angekommen ist, wird eine boolean Variable auf true gesetzt und die Schleife wird verlassen.

Jetzt werden noch die Kreise erneuert und die Variablen zurückgesetzt. Dann wird die Klasse verlassen.

Die **Klasse Schiebefenster (extends Object implements Runnable)** ist für den korrekten Ablauf dieses Algorithmus verantwortlich.

Nach der Initialisierung der notwendigen Variablen werden auch in dieser Klasse - je nachdem wie es die Fenstergröße vorgibt - Variablen in den Kellerspeicher gestellt. Die jeweiligen Kreissegmente, die für die Pakete stehen, werden noch dunkelgrau gefärbt (= Paket ist unterwegs). Danach wird die Bewegung1 Klasse aufgerufen, der Thread wird angehalten.

Sobald die Bewegung1 Klasse, diesen Thread nun aber wieder aufruft (d.h. ein Paket hat das Ende der Leitung erreicht), wird die Verarbeitungsklasse aufgerufen, die entscheidet ob und wenn ja welches Paket in den Kellerspeicher geschoben wird und schließlich wird wieder die Bewegung1 Klasse aufgerufen.

Diese Schleife wird solange durchlaufen bis der Stop Button gedrückt wurde. Dann wird auch in dieser Klasse noch so lange weitergesendet bis das letzte Paket bestätigt wurde. Um dies zu erreichen wird immer wieder die Verarbeitung1 Klasse, die Bewegung1 Klasse aufgerufen und schließlich der Thread angehalten. Dieser wird von der Bewegung1 Klasse immer wieder gestartet. Wenn das letzte Paket ordnungsgemäß angekommen ist wird eine boolean Variable auf true gesetzt.

Schließlich wird die Klasse verlassen.

Die **Klasse Gobackn (extends Object implements Runnable)** ist für den korrekten Ablauf dieses Algorithmus verantwortlich.

Nach der Initialisierung der notwendigen Variablen, werden die Grafiken aus Bild1, 2 und 3 eingelesen. Danach wird der Puffer des Empfängers je nachdem welche Puffergröße eingestellt wurde gezeichnet.

Daraufhin wird der eigentliche Algorithmus gestartet. Die Klasse Sendepfeil wird aufgerufen, die das Senden eines Paketes zum Empfänger visualisiert. Der Thread wird angehalten. Falls das Paket nun den Empfänger erreicht hat wird der Thread aus der Klasse Sendepfeil wieder gestartet. Die x und y Variablen für die Klasse Sendepfeil werden neu gesetzt, und falls der Bereich wo die Pfeile gezeichnet werden bereits voll ist, wird die Methode Translate in der Bild2 Klasse aufgerufen, die dann das Bild nach rechts scrollt. Nachdem geprüft wurde ob das Paket angekommen ist und ob die Packetnummer zurückgesetzt werden muß.

Falls nun ein Paket bestätigt wurde, wird die Klasse Senderueckpfeil aufgerufen und der Thread wird wiederum angehalten. Falls das Paket nun angekommen ist wird die Methode Leerepuffer aus der Klasse Puffer aufgerufen und falls die Packetnummer schon zurückgesetzt wurde wird diese Variable neu gesetzt. Die x und y Variablen für die Klasse Sendepfeil werden neu gesetzt, und falls der Bereich wo die Pfeile gezeichnet werden bereits voll ist, wird die Methode Translate in der Bild2 Klasse aufgerufen, die dann das Bild nach rechts scrollt.

Dies wird solange wiederholt bis der Algorithmus über den Stop Button gestoppt wurde, danach werden die entsprechenden Variablen zurückgesetzt, die Klasse wird verlassen.

Die **Klasse Gobacknf (extends Object implements Runnable)** ist für den korrekten Ablauf dieses Algorithmus verantwortlich.

Diese Klasse funktioniert im Grunde genauso wie die Klasse Gobackn. Der Unterschied besteht nur in der Pufferung der angekommenen Pakete.

Die **Klasse Bewegung (extends Thread)** bewegt die Pakete bzw. Bestätigungen beim Stop & Wait Algorithmus zum Sender bzw. Empfänger.

Nach der Initialisierung der Variablen wird zuerst geprüft in welchem Kellerspeicher(Sender oder Empfänger) sich ein Element befindet. Dieses Element wurde in der StopandWait Klasse in den Kellerspeicher gestellt.

Falls sich im Sender Kellerspeicher ein Element befindet, wird dieser aus dem Kellerspeicher ausgelesen. In dem Element befinden sich unter anderem die x und die y Koordinaten des Pakets und eine boolean Variable, die angibt, ob das Paket „weggeklickt“ wurde oder nicht.

Nun wird das Paket gezeichnet, die Position wird neu berechnet und die neuen Koordinaten werden zurückgeschrieben. Dabei wird das Alte Paket jeweils schwarz übermalt und ein neues davorgezeichnet.

Danach wird geprüft ob das Paket schon am Ende der Leitung angekommen ist. Falls dies der Fall ist wird eine boolean Variable auf true gesetzt, dies bewirkt das am Ende dieses Durchlaufs die Schleife verlassen wird.

Sollte sich kein Element im Sender Kellerspeicher befunden haben, muß sich zwangsläufig ein Element im Empfänger Kellerspeicher befinden (die Klasse wird nur aufgerufen, falls ein Element in einen der beiden Kellerspeicher gesetzt wurde).

In diesem Fall läuft alles genauso ab wie beim Sender.

Danach wird noch geprüft ob eine bestimmte boolean Variable auf true gesetzt wurde. Falls dies der Fall ist, wird die Blitz Klasse aufgerufen.

Am Ende der Schleife wird nun die oben genannte boolean Variable betrachtet. Falls sie auf true gesetzt wurde wird die Schleife verlassen.

Die **Klasse Bewegung1 (extends Thread)** bewegt die Pakete bzw. Bestätigungen beim Schiebefenster Algorithmus zum Sender bzw. Empfänger.

Diese Klasse funktioniert im Grunde genauso wie die Bewegung Klasse.

Der Unterschied besteht darin, daß hier mehrere Elemente sowohl aus dem Sender als auch aus dem Empfänger Kellerspeicher neu gezeichnet werden müssen. Weiterhin werden auch Operationen an den beiden Kreisen durchgeführt.

Am Ende der Schleife wird noch geprüft ob sich im temp Kellerspeicher ein Element befindet. Falls in der Verarbeitung oder Verarbeitung1 Klasse ein Element in diesen gestellt wird dieses auf den Sender oder Empfänger Kellerspeicher verteilt.

Die **Klasse Verarbeitung (extends Thread)** wird aufgerufen(vom der Klasse Schiebefenster), falls ein Paket beim Schiebefenster Algorithmus angekommen ist, sie entscheidet ob und welches Paket bzw. Bestätigung gesendet wird.

Nach der Prüfung ob ein Sender oder Empfänger Paket angekommen ist (und ob es korrekt angekommen ist), werden die entsprechenden Operationen an dem entsprechenden Kreis durchgeführt.

Je nachdem welches Paket angekommen ist, und wie es angekommen ist wird daraufhin ein neues Element in den temp Kellerspeicher gestellt. Dieses wird dann in der Bewegung1 Klasse in den jeweiligen Kellerspeicher gestellt.

Die **Klasse Verarbeitung1 (extends Thread)** funktioniert im Grunde identisch zur Verarbeitung Klasse.

Der einzige Unterschied besteht darin, daß nicht alle Pakete bestätigt werden. Falls eine Bestätigung korrekt den Sender erreicht wird kein neues Element in den temp Kellerspeicher gestellt.

Die **Klasse Blitz (extends Thread)** visualisiert den Paketverlust beim Stop and Wait und beim Schiebefensteralgorithmus.

Es wird hier Stück für Stück ein Blitz gezeichnet, zwischen jedem Zeichenvorgang, tritt der Thread in eine kurze Ruhephase.

Die **Klassen Bild1, 2 und 3 (extend Canvas)** sorgen dafür, daß die Bild Klasse abermals in 3 kleinere Ausschnitte unterteilt wird.

In der Klasse Bild1 ist noch eine mouseDown Methode definiert, die Überprüft in welchem Bereich die Maus Taste gedrückt wurde.

In der Bild2 Klasse sorgt die translate Methode dafür, daß die Pfeile im Scrollbar Bereich gezeichnet werden.

Die mouseDown Methode in der Klasse Bild3 überprüft ob in diesem Bereich die Maustaste gedrückt wurde. Ist dies der Fall wird eine boolean Variable auf true gesetzt(diese sorgt dafür, daß der Pfeil ab jetzt in rot weitergezeichnet wird).

Die **Klassen Sendepfeil und Senderueckpfeil (extend Thread)** zeichnen die Pfeile beim GobackN Algorithmus.

Es wird bei jedem Schleifendurchgang, ein 2 Pixel großes Rechteck am oberen (bzw. unteren) Ende des Pfeils hinzugefügt. Dabei wird jedes Mal die Paketnummer neu geschrieben, damit diese auch ständig zu sehen ist.

Am Ende der Klasse wird die jeweilige Paketnummer, eine Integer Zahl und eine boolean Variable in einen Array geschrieben, damit beim Scrollen auch die genaue Position des Pfeils bekannt sind.

Die **Klasse MalePfeil** sorgt dafür, daß die Pfeile der beiden Gobackn Klassen an die korrekte Stelle gezeichnet werden.

Der Array wird Element für Element eingelesen bis der erste Pfeil kommt der sich im Bildbereich befindet. Danach werden alle Elemente im Bildbereich gezeichnet, zuletzt wird das aktuelle Paket, falls sich gerade eines auf dem Weg befindet gezeichnet.

Letztendlich wird noch versucht einen kurzen Ausblick über die Zukunft des Projekts Teleteaching und mögliche Weiterentwicklungen dieses Applets zu geben.

6. Aussicht

I. Teleteaching Projekt:

Es wäre wünschenswert, wenn alle wichtigen Themen der Rechnernetze Vorlesung, durch Animationen in dieser Art ergänzt werden könnten. Auf diese Weise könnte der Lerneffekt der Vorlesung optimiert werden. Neben dieser Studienarbeit wurden noch weitere in diesem Bereich vergeben. Dies war ein Schritt zur vollständigen Abdeckung der Vorlesung durch Animationen.

II. Probleme mit Java:

Neben der Probleme mit der graphischen Oberfläche, ist das Hauptproblem, daß die Plattformunabhängigkeit von Java noch nicht gegeben ist. Besonders Threads reagieren selbst unter der gleichen Plattform unter Netscape und dem Appletviewer völlig unterschiedlich. Unter verschiedenen Plattformen kann es vorkommen, daß das Applet einmal völlig korrekt abläuft und unter der anderen Plattform erst gar nicht läuft. Mit der neuen Version von Java sollten einige dieser Probleme behoben sein.

III. Verbesserungen dieses Applets:

Folgende Punkte konnten bei diesem Applet noch verbessert werden:

- Paketgröße:

Die Paketgröße könnte variabel gehalten werden, so daß eventuell mehrere Puffer gefüllt werden müßten.

- Verarbeitungszeit beim Empfänger:

Bei der aktuellen Version des Applets verarbeitet der Empfänger immer alle Pakete sofort. In der Realität allerdings braucht er meist länger bis er die Pakete verarbeitet werden. Dies könnte in einer Folgeversion des Applets realisiert werden.

- Bestätigungspolitik:

Beim Schiebefensteralgorithmus könnte noch festgelegt werden, ob der Empfänger jedes, jedes zweite usw. Paket bestätigt wird.

Literatur:

[CN T]	Computernetzwerke	Tannenbaum
[RN A]	Rechnernetze nach ISO	Addison Verlag
[JD C]	Java bis ins Detail	Cornell, Horstman