*Vi*sualisierung *v*on *a*usgewählten *L*ehrinhalten *d*er *I*nformatik - Beispielanimationen aus dem Projekt *Vivaldi*

Rainer Oechsle, Dieter Jahn

Fachhochschule Trier, Fachbereich Angewandte Informatik

Kurzfassung

Im Rahmen des Projekts ***Vivaldi*** wurden ausgewählte Lehrinhalte der Informatik visualisiert. Die Visualisierungen werden unterstützend zur traditionellen Vorlesung verwendet. Außerdem wird den Studierenden eine Plattform zur Verfügung gestellt, mit der sie eigene Programme mit kontinuierlichen Bewegungen visualisieren können. Grundlage der Visualisierungen ist der ***Vivaldi***-Kern, der eine Reihe von Klassen umfasst, mit deren Hilfe grafische Objekte auf dem Bildschirm dargestellt werden können. Ferner ist es möglich, für die grafischen Objekte Vorschriften anzugeben, wie sich ihr Aussehen oder ihre Position im Laufe der Zeit verändert. Damit können dann kontinuierliche Bewegungen von Objekten spezifiziert werden.

1. Einleitung

Im Rahmen des verhältnismäßig kleinen Projekts ***Vivaldi*** (**Vi**sualisierung **v**on **a**usgewählten **L**ehrinhalten **d**er **I**nformatik) im Fachbereich Angewandte Informatik der FH Trier war es nicht möglich, eine Lernsoftware zu entwickeln, die einen gewissen Lehrbereich komplett abdeckt. Die Ziele dieses Projekts, das vom 1. April 1998 bis zum 31. Dezember 1999 an der FH Trier mit zwei halben Stellen durchgeführt wurde, waren statt dessen wesentlich bescheidener:

- Es sollten Lehrinhalte der Informatik punktuell ausgewählt werden, die sich besonders gut für eine Visualisierung eignen.
- Die Visualisierungen sollten real ablaufende Algorithmen illustrieren, statt einen solchen Ablauf nur vorzutäuschen. Das heißt zum Beispiel, dass ein Sortieralgorithmus tatsächlich abläuft und dabei über eine Visualisierung anzeigt, was gerade getan wird. Damit kann durch Veränderung von Parametern die Anwendung und damit auch die Visualisierung unterschiedlich ablaufen und interaktiv gesteuert werden.
- Die entwickelten Visualisierungen haben nicht den Anspruch, dass sie einen zuvor gänzlich unbekannten Sachverhalt erklären können, sondern die Visualisierungen sollen unterstützend zur traditionellen Vorlesung verwendet werden. Den Studierenden sollen die Visualisierungen im Anschluss an die Vorlesung zugänglich sein, so dass sie diese mit eventuell anderen Parametern nochmals selber durchspielen können, um ihr Verständnis dadurch zu vertiefen.

- Im Projekt sollte eine Beobachtung berücksichtigt werden, die wir im Vorfeld des Projekts gemacht haben: Häufig haben die Entwickler einer Visualisierung mehr Nutzen davon als die späteren Betrachter. Aus diesem Grund sollte das Projekt ***Vivaldi*** eine Plattform zur Verfügung stellen, mit der die Studierenden eigene Programme visualisieren können, wobei in diesem Fall die Visualisierung nicht primär für andere gedacht ist. Stehen den Studierenden z.B. entsprechende Visualisierungsoperationen zum Vergleich zweier Elemente und zum Vertauschen zweier Elemente zur Verfügung, so können sie selber unterschiedliche Sortieralgorithmen programmieren und an entsprechender Stelle die zur Verfügung gestellten Visualisierungsoperationen in ihr Programm einbauen (d.h. sie können ihr Sortierprogramm zum Zwecke der Visualisierung instrumentieren). In diesem Fall kann die Visualisierung zum schnellen Erkennen von Programmierfehlern dienen. Es sollte nicht unterschätzt werden, dass eine derartige Nutzung von Visualisierungen den Spaß am Programmieren erhöht und dadurch die Studierenden (und auch die Autoren dieses Beitrags) in besonderer Weise motiviert.
- In den Visualiserungen sollten „sanfte Übergänge“ (kontinuierliche Bewegungen) statt lediglich eine Folge von Schnappschüssen vorkommen. Dadurch werden Veränderungen auf dem Bildschirm besser erkennbar. Die Autoren haben selber die Beobachtung gemacht, dass viele Visualisierungen aufgrund abrupter Übergänge schwer zu verfolgen sind.

Das Projekt ***Vivaldi*** lief in zwei Phasen ab: In der ersten Phase wurde eine Klassenbibliothek, der ***Vivaldi***-Kern, entwickelt. Der ***Vivaldi***-Kern besteht aus einer Reihe von Klassen, mit deren Hilfe grafische Objekte wie Rechtecke und Kreise auf dem Bildschirm dargestellt werden können. Ferner ist es möglich, für diese grafischen Objekte Vorschriften anzugeben, wie sich ihr Aussehen oder ihre Position im Laufe der Zeit verändert. Damit können dann u.a. kontinuierliche Bewegungen von Objekten spezifiziert werden. In der zweiten Phase wurden Visualisierungen aus unterschiedlichen Bereichen der Informatik unter Nutzung des ***Vivaldi***-Kerns entwickelt. Zum größten Teil handelt es sich dabei um Algorithmenanimationen; nur eine Visualisierung ist dem Gebiet der Programmanimationen zuzurechnen (zur Klassifikation der Software-Visualisierung siehe z.B. [Price, Baecker, Small 93]).
Dieser Beitrag ist wie folgt gegliedert: Im nächsten Abschnitt gehen wir kurz auf den ***Vivaldi***-Kern ein, der bereits an anderer Stelle ausführlicher beschrieben wurde [Oechsle 99; Oechsle, Becker 99]. Im dritten Abschnitt werden einige ausgewählte Animationen vorgestellt, die mit Hilfe des ***Vivaldi***-Kerns entwickelt wurden. Der vierte Abschnitt enthält Hinweise auf verwandte Arbeiten. Abschließend schildern wir im fünften Abschnitt die Erkenntnisse, die wir gewonnen und deren Konsequenzen wir bei der Entwicklung der Animationen berücksichtigt haben, und geben einen Ausblick auf zukünftige Aktivitäten.

2. Der *Vivaldi*-Kern

Die im folgenden Abschnitt beschriebenen Visualisierungen besitzen in der Regel folgende Struktur: Ein zu visualisierender Algorithmus (die eigentliche Anwendung

wie z.B. ein Sortierverfahren) wird mit Methodenaufrufen zur visuellen Darstellung instrumentiert (z.B. Methoden, die das Vergleichen oder Vertauschen zweier Elemente grafisch anzeigen). Die Klassen, welche diese visuelle Darstellung realisieren, bezeichnen wir als "Script". Der Name "Script" soll dabei an ein Drehbuch für einen Film erinnern; ein Visualisierungsscript hat nichts mit einem Script im Sinne eines interpretierten Programms wie z.B. einem Shell-Script in UNIX zu tun. Bei der Entwicklung der anwendungsspezifischen visuellen Darstellung als Script wird auf die Klassenbibliothek ***Vivaldi***-Kern zurückgegriffen, die es erlaubt, grafische Objekte darzustellen, die sich nach anzugebenden Vorschriften bewegen und ihr Aussehen verändern. Abbildung 1 fasst den soeben geschilderten Sachverhalt zusammen.

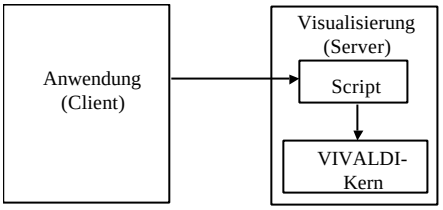

Abbildung 1: Trennung von Anwendung und visueller Darstellung

Der ***Vivaldi***-Kern basiert auf dem AWT (Abstract Window Toolkit) von Java JDK 1.1. Er besteht aus Klassen zur Repräsentation grafischer Objekte auf dem Bildschirm und zur Festlegung, wie sich das Aussehen und die Position grafischer Objekte im Laufe der Zeit verändern sollen.

Grafische Objekte: Alle Klassen zur Repräsentation grafischer Objekte werden von der Basisklasse `GraphicElement` abgeleitet. Die Klassen bilden eine echt objektorientierte Verkleidung der in Java grafisch darstellbaren Objekte. So ist z.B. die Farbe ein Attribut der grafischen Objekte, während in Java die Farbe als eine Art von globaler Variable im Grafikkontext vor dem Zeichnen eines grafischen Objekts gesetzt werden muss. Ferner gibt es einige Klassen, deren Funktionalität nicht direkt in Java verfügbar ist. Die Klasse `ArrowElement` z.B. erweitert die Klasse `LineElement` um das Konzept von Pfeilen; ein Pfeil ist eine Linie mit einer Pfeilspitze. Die Klasse `PhaseImageElement` ist eine Folge von Pixelbildern im GIF- oder JPEG-Format. Diese werden im Konstruktor durch ein Feld von Dateinamen spezifiziert. Die Bilder sind entsprechend dieses Dateinamenfelds von 0 an durchnummeriert. Das `int`-Attribut `phase` beschreibt, welches der Bilder aktuell angezeigt wird. Durch eine rasche Veränderung dieses Attributwerts entsteht bei entsprechend geeigneten Bildern der Eindruck eines Films. Mit Hilfe der Klasse `GraphicElementGroup` können mehrere `GraphicElement`-Objekte in einem Objekt zu einer Gruppe zusammengefasst werden. Da `GraphicElementGroup` selbst aus `GraphicElement` abgeleitet ist, kann eine Gruppe u.a. wieder eine Gruppe enthalten. Damit kann ein Baum von `GraphicElement`-Objekten generiert werden. Der Entwurf der graphischen Elemente ist gemäß dem Entwurfsmuster "Composite" [Gamma et al. 95] gestaltet. Die Auswahl der hier beschriebenen Klassen von grafischen Elementen sind in Abbildung 2 mit ihren Vererbungsbeziehungen und einigen ausgewählten Attributen zusammenfassend dargestellt.

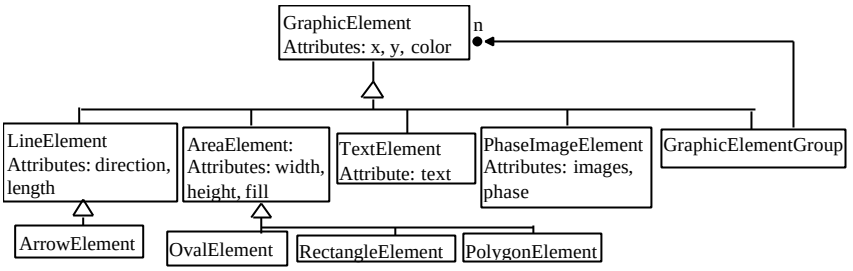

Abbildung 2: Teil des Klassendiagramms der grafischen Elemente

Jedes grafische Objekt kann auf Mausereignisse (Klicken, Bewegen usw.) reagieren, indem - wie in Java entsprechend dem Observer-Observable-Muster [Gamma et al. 95] allgemein üblich - ein sogenannter Listener an das Objekt gebunden wird. Dieser Listener muss eine Reihe von Methoden implementieren, die bei Eintreffen der entsprechenden Mausereignisse aufgerufen werden. Die Pfeile in Abbildung 1 können daher durch Doppelpfeile ersetzt werden, da Rückmeldungen vom ***Vivaldi***-Kern über das Script an die Anwendung möglich sind.

Attributtransitionen: Die Grundidee für die Spezifikation von kontinuierlichen Veränderungen der grafischen Objekte des ***Vivaldi***-Kerns ist, dass Vorschriften für die zeitliche Veränderung der Attribute von grafischen Objekten angegeben werden. Diese Vorschriften werden Attributtransitionen genannt. Eine Attributtransition bezieht sich dabei immer auf *genau ein* Attribut von *genau einem* grafischen Objekt. Will man also z.B. eine Bewegung eines einzigen Objekts beschreiben, so benötigt man zwei Attributtransitionen, eine für die x- und eine für die y-Koordinate dieses Objekts. Eine Transition besitzt im wesentlichen zwei Referenzen: eine auf das Objekt, auf das sie sich bezieht, und eine auf eine Funktion, welche die Zeit auf einen Attributwert abbildet. Auf welches Attribut des grafischen Objekts sich eine Transition bezieht, wird durch die Klasse des Transitionsobjekts festgelegt. So gibt es für jedes Attribut der grafischen Objekte eine entsprechende Transitionsklasse.
Abbildung 3 stellt die wichtigsten Objekte und ihre Beziehungen in einem Beispiel dar. Im oberen Teil der Abbildung sieht man die Enthaltenseinshierarchie der grafischen Objekte. Der untere Teil zeigt die Menge von Transitionen, wobei jede Transition eine Referenz auf ein grafisches Objekt und eine Zeitfunktion besitzt. Zum leichteren Umgang des Programmierers mit Transitionen gibt es wie bei den Grafikobjekten auch Transitionsgruppen. Damit können mehrere Transitionen auf einfache Art zusammengefasst werden. Durch das Konzept der Transitionsgruppe erhält man für die Transitionen ebenfalls einen Baum von Transitionen wie für die Grafikobjekte.

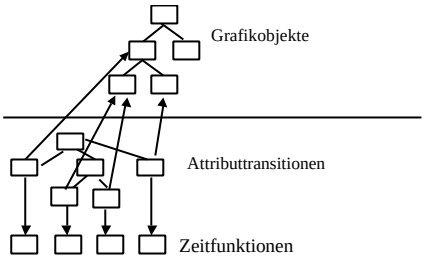

Abbildung 3: Beispiel für Beziehungen zwischen Baum von Grafikobjekten, Baum von Attributtransitionen und Zeitfunktionen

Aus der Abbildung können folgende Sachverhalte gefolgert werden:

- Die Transitionen können sich auf primitive Grafikobjekte oder auf Gruppen von Objekten beziehen. Damit kann z.B. eine Gruppe von Objekten bewegt werden.
- Es kann mehrere Transitionen geben, die sich auf dasselbe grafische Objekt beziehen. Dies macht allerdings nur dann Sinn, falls sich die Transition auf unterschiedliche Attribute dieses Objekts bezieht (z.B. x- und y-Koordinate).
- Nur die primitiven Transitionen (also die Nicht-Gruppentransitionen) besitzen Referenzen auf grafische Objekte und Zeitfunktionen.

Weitere Eigenschaften der Attributtransitionen sind:

- Die Transitionen können in die Liste aller Transitionen dynamisch eingefügt und daraus gelöscht werden. Damit können z.B. Bewegungen gestartet und beendet werden.
- Die Transitionen können endlich oder unendlich sein.
- Eine endliche Transition entfernt sich aus ihrer Gruppe selbst, sobald sie beendet ist. Dieses Prinzip wird rekursiv angewendet: Eine Transitionsgruppe ist dann beendet, falls alle ihre Mitglieder beendet sind. Dies ist genau dann der Fall, falls die Gruppe keine Mitglieder mehr hat.
- Jede Transition hat eine `synch`-Methode, mit der man sich auf das Ende einer Transition synchronisieren kann. Das heißt, ein Thread, welcher die `synch`-Methode auf eine Transition anwendet, wird solange blockiert, bis die entsprechende Transition beendet ist. Die `synch`-Methode kann auch auf eine Transitionsgruppe angewendet werden.

3. Visualisierungsbeispiele

Im Folgenden werden einige Beispiele für Animationen aus unterschiedlichen Bereichen der Informatik vorgestellt, die im Rahmen des ***Vivaldi***-Projekts entstanden sind.

Bereich Software-Entwicklung

Mit Hilfe eines Editors können Methodenaufrufe innerhalb eines beliebigen Java-Programms markiert und mit einem Kommentar versehen werden. Wird aus diesem

Editor heraus das Programm übersetzt, so entsteht beim Ablauf dieses Programms eine Animation, welche die Methodenaufrufe im Stil eines Sequenzdiagramms nach der Unified Modeling Language (UML) dynamisch zusammen mit den Kommentaren darstellt. Das heißt, dass für ein beliebiges Java-Programm automatisch eine Animation generiert werden kann, welche die Interaktionen der Objekte des gerade ablaufenden Programms darstellt. Das System kann sogar rekursive Aufrufe korrekt und übersichtlich darstellen – eine Fähigkeit, die vielen kommerziell verfügbaren CASE-Tools immer noch fehlt. Das entwickelte System eignet sich sowohl zur Unterstützung der Lehre als auch als Hilfe bei der Software-Entwicklung.
Eine automatische Visualisierung des gesamten Programms ist in der Regel nicht ratsam, da man dann beim Ablauf des Programms "den Wald vor lauter Bäumen nicht mehr sieht". Deshalb haben wir uns dafür entschieden, dass der Entwickler spezifizieren muss, was er beim Ablauf sehen möchte.

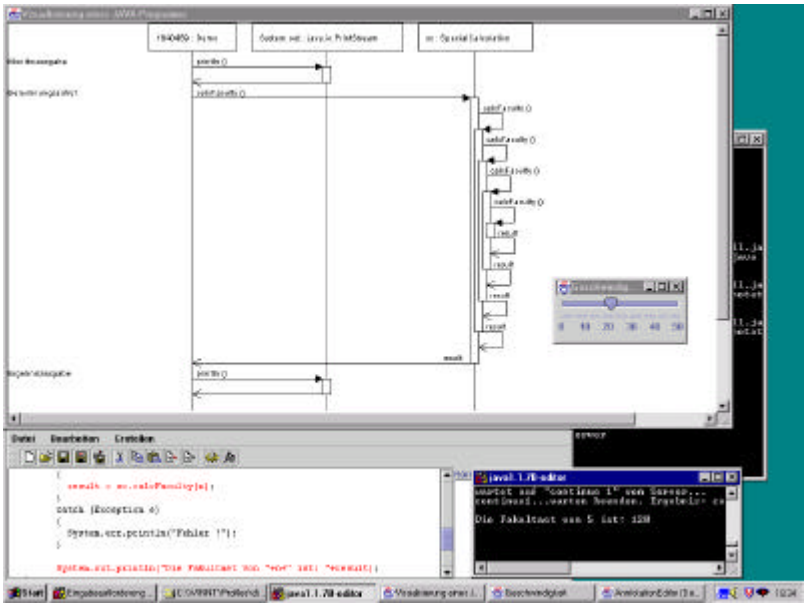

Abbildung 4: Dynamische Sequenzdiagramm-Visualisierung

In Abbildung 4 ist ein Schnappschuss der ablaufenden Visualisierung zu sehen. Das untere linke Fenster enthält den Editor, in den das zu visualisierende Programm geladen wurde. Die Methodenaufrufe, die visualisiert werden sollen, sind entsprechend farbig markiert worden. Das kleine Fenster rechts unten enthält die Ausgabe des rein zeilenorientierten Programms, nämlich das Ergebnis der Fakultätsberechnung. Das obere große Fenster beinhaltet die Visualisierung des gerade ablaufenden Programms. Die dargestellten Pfeile und Balken werden nicht auf einen Schlag, sondern mit kontinuierlichen Bewegungen allmählich auf den Bildschirm gebracht. Durch die Visualisierung verzögert sich der Ablauf des Programms entsprechend, da diese Visualisierung synchron zum ablaufenden Programm durchgeführt wird. Als Beispiel wurde die Berechnung der Fakultätsfunktion in rekursiver Weise gewählt,

um zu zeigen, wie rekursive Aufrufe visualisiert werden. Diese Visualisierung ist die einzige Programmanimation, die im Rahmen des ***Vivaldi***-Projekts entwickelt wurde; alle anderen Beispiele stellen Algorithmenanimationen dar.

Bereich Grundlagen der Informatik

Sortieralgorithmen: Sortieralgorithmen wie Selection-, Insertion-, Bubble-, Quick- und Shell-Sort sind klassische Beispiele für Algorithmenanimationen. Die Animationen dienten als erster Test des ***Vivaldi***-Kerns und wurden als Java-Applets implementiert. Jedes Applet stellt dem Benutzer eine Auswahlmöglichkeit bezüglich der Menge der zu sortierenden Elemente zur Verfügung. Der Benutzer kann jeweils zwischen einer Menge mit zufällig erzeugten Elementen oder zwei unterschiedlichen, fest vorgegebenen Beispielmengen wählen. Ebenso kann die Visualisierungsgeschwindigkeit bei jedem Applet vom Benutzer selbst eingestellt werden.
Damit die erstellten Visualisierungen der Sortieralgorithmen im Grundstudium der Angewandte Informatik in den Übungsstunden eingesetzt werden können, wurde zu jedem Sortieralgorithmus eine ausgesuchte Menge von Visualisierungsmethoden bereitgestellt. Die Studierenden erhalten zu jeder Methode eine Erklärung über deren Funktion. Dann sollen sie die Sortieralgorithmen programmieren und den Aufruf der Visualisierungsmethoden an der richtigen Stelle im Sortieralgorithmus einfügen. Die Visualisierungen auf den Internetseiten sollen hierbei dazu beitragen, dass die Funktionsweise der einzelnen Algorithmen von den Studierenden besser verstanden werden können.

Synchronisations- und Kommunikationsverfahren: Ebenfalls als Java-Applets wurden Visualisierungen verschiedener Synchronisations- und Kommunikationsverfahren mit Hilfe von Semaphorgruppen, Dateisperren, Pipes und Message Queues realisiert. Die Semantik dieser grundlegenden Konzepte entspricht derjenigen des Betriebssystems UNIX.
Die Realisierung dieser Visualisierungen erfolgte, indem die aus UNIX bekannten Konzepte in Java nachimplementiert wurden. Dabei wurden die Aufrufe für die Visualisierung lediglich auf die Methoden der Synchronisations- und Kommunikationsobjekte beschränkt. Dennoch werden die beteiligten aktiven Komponenten, die Threads, ebenfalls dargestellt. Dies geschieht dadurch, dass bei einem Aufruf einer Methode zuerst geprüft wird, ob der aufrufende Thread schon dargestellt ist oder nicht. Ist er noch nicht zu sehen, so erfolgt seine Darstellung zu diesem Zeitpunkt.
Der Vorteil dieses Konzepts ist, dass Studierende unter Verwendung der Synchronisations- und Kommunikationsobjekte Anwendungen programmieren können und beim Ablauf dieser Anwendungen eine Animation ihres Programms sehen, ohne dass sie sich selbst um die Animation kümmern müssen.
Die auf den Internet-Seiten zu sehenden Animationen sind zum einen solche, die vom Benutzer manuell gesteuert werden müssen: Es werden mehrere Threads gestartet, wobei für jeden Thread ein Dialogfenster auf dem Bildschirm erscheint, so dass der Benutzer spezifizieren kann, was dieser Thread als nächstes tun soll (z.B. welche Methode [Senden oder Empfangen] soll ausgeführt werden). Zum anderen wurden bekannte Synchronisationsprobleme wie das Leser-Schreiber-Problem oder das

Problem der fünf essenden Philosophen (Dinning Philosophers) mit Hilfe der Semaphorgruppenobjekte visualisiert.
Die Visualisierungen der Sortieralgorithmen und der Synchronisations- und Kommunikationsverfahren sind öffentlich zugänglich. Als Einstiegspunkt empfehlen wir die Adresse http://www.informatik.fh-trier.de/projekte/vivaldi/vivaldi.html, die ***Vivaldi***-Projektseite.

Bereich Rechnernetze

Visualisierung von Routing-Algorithmen: In einer weiteren Arbeit wurde das im Internet wichtige RIP-Verfahren (Routing Information Protocol) visualisiert. RIP ist ein sogenanntes Routing-Verfahren, mit dem die Router im Internet untereinander Informationen austauschen, um daraus ihre Wegewahltabellen zu berechnen. Damit werden die Wegewahltabellen ständig auf dem aktuellen Stand gehalten.
Um den RIP-Algorithmus visualisieren zu können, ist es zunächst nötig, ein Netzwerk zu erstellen. Die Anwendung stellt daher im Hauptfenster eine Visualisierungsfläche zur Verfügung. In dieser Fläche können Router frei platziert und anschließend durch Ziehen einer Linie verbunden werden. Auf diese Weise kann der Benutzer ein Netzwerk erstellen, welches genau auf seine Bedürfnisse zugeschnitten ist. Beispielsweise kann er ein Netz aus nur zwei oder drei Routern entwerfen, um die Komplexität gering zu halten und so ein grundsätzliches Verständnis der Abläufe zu erlangen. Andererseits kann aber auch ein Netz entworfen werden, um eine bestimmte Situation, wie z.B. das „Zählen bis unendlich“, zu demonstrieren. Ist ein solches Netzwerk einmal erstellt, kann es in einer Datei abgespeichert werden.
Um die Simulation des RIP-Algorithmus zu starten, reicht es, einen Router doppelt anzuklicken. Daraufhin sendet dieser seinen Distanzvektor an alle seine Nachbarn, wobei jeder Distanzvektor als Rechteck auf einer Leitung dargestellt wird. Die Änderungen in der Wegewahltabelle eines Routers können nun in einem Dialogfenster, welches für jeden Router geöffnet werden kann, beobachtet werden. In diesem Fenster ist die neu berechnete Wegewahltabelle, der empfangene Distanzvektor, der die Neuberechnung verursacht hat, mit Absender und die Ausgangswegewahltabelle zu sehen. Durch diese Informationen ist es möglich, die Berechnung der Wegewahltabelle aus dem empfangenen Distanzvektor und der bisherigen Tabelle nachzuvollziehen. Durch wiederholtes Doppelklicken auf weitere Router kann nun Schritt für Schritt die Berechnung der Wegewahltabellen im ganzen Netz erfolgen. Diese sogenannte „manuelle“ Vorgehensweise ist besonders dann günstig, wenn bestimmte „was-wäre-wenn"-Situationen erzeugt und beobachtet werden sollen. Es ist aber auch möglich, die Anwendung selber entscheiden zu lassen, welcher Router an der Reihe ist, seinen Distanzvektor zu versenden. In jedem Fall muss aber die Animation manuell gesteuert werden, denn ein automatischer Ablauf ist auch bei geringer Geschwindigkeit wegen der Komplexität der Anwendung vom Betrachter nicht nachvollziehbar. Während der Simulation können jederzeit Verbindungen als defekt gekennzeichnet oder Router gelöscht werden.
In ähnlicher Weise wurde das OSPF-Verfahren (OSPF: Open Shortest Path First) visualisiert.

Visualisierung des TCP-Protokolls: Die TCP-Visualisierung gliedert sich in zwei Bereiche, bei denen es sich zum einen um das TCP-Verbindungsmanagement und zum anderen um den Datentransfer handelt. Beim Verbindungsmanagement werden die Zustandsänderungen beider Partner mit endlichen Automaten dargestellt. Hier hat der Benutzer die Möglichkeit, den Verbindungsaufbau und –abbau selbst zu beeinflussen, z.B. gleichzeitiger Verbindungsaufbau von beiden Partnern. Beim Datentransfer wird weniger auf den zeitlichen Ablauf des datenstromorientierten TCPs Wert gelegt, sondern auf die Sicherstellung der Zuverlässigkeit und der Flusskontrolle. Durch die manuellen Eingaben können die verschiedensten Szenarien durchgespielt werden, z.B. Paketverzögerung und –verlust im Netz, Timeoutbehandlung und Überflutung des Empfängers. Dabei wird der Benutzer nach jeder Eingabe über die Änderung der Sequenznummer, Quittungsnummer und Fenstergröße informiert. In Abbildung 5 ist die TCP-Visualisierung dargestellt.

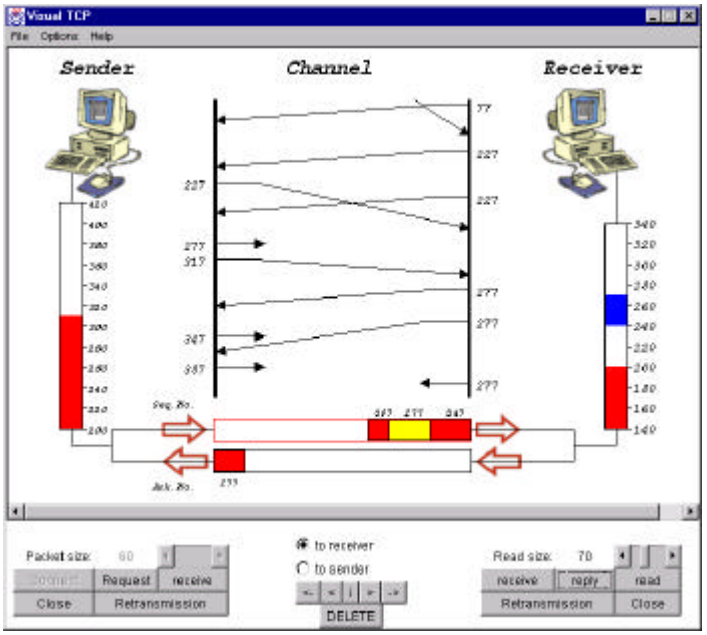

Abbildung 5: Programmfenster der TCP-Visualisierung

4. Verwandte Arbeiten

In vielen Veröffentlichungen werden immer wieder das Balsa-System [Brown 84] und dessen Nachfolger Balsa-II [Brown 88a; Brown 88b] und Zeus [Brown 91] als die ersten bedeutenden Arbeiten auf dem Gebiet der Algorithmenanimation angeführt. Diese und viele andere Artikel konzentrieren sich sehr stark auf die jeweiligen Anwendungen, so dass man nicht erkennen kann, inwieweit diesen Anwendungen ein Basis system zugrunde liegt, das mit dem ***Vivaldi***-Kern verglichen werden kann.

Der einzige uns bekannte Artikel, der ein Basissystem zur Programmierung von Animationen etwas detaillierter darstellt, ist ein von J. Stasko verfasstes Papier über das von ihm entwickelte Tango-System [Stasko 90], das auch Thema seiner Dissertation war [Stasko 89]. Der Artikel [Stasko 90] war eine wesentliche Inspiration für das ***Vivaldi***-System. Stasko beschreibt sehr deutlich die Trennung von Anwendung (Algorithmuskomponente) und Visualisierung (Animationskomponente). Die Animationskomponente beinhaltet das Pfad-Transitions-Paradigma (dies entspricht in ***Vivaldi*** dem Kern), das wie der ***Vivaldi***-Kern u.a. "sanfte Übergänge" (kontinuierliche Bewegungen) unterstützt. Der Ansatz unterscheidet sich dennoch stark vom ***Vivaldi***-Kern. Eine Schwäche des Pfad-Transitions-Paradigmas liegt unserer Meinung nach in der fehlenden Objektorientierung, was u.a. bewirkt, dass das System schwerer verständlich und programmierbar ist. Auch sind einige Funktionalitäten des ***Vivaldi***-Kerns wie unendliche Transitionen und Synchronisationsoperationen nicht vorhanden. Dennoch ist das Tango-System ein beeindruckendes System, anhand dessen wesentliche Basiskonzepte von Animationssystemen umfassend und systematisch dargelegt wurden, und für das einige interessante Erweiterungen wie z.B. die Integration mit einem Debugger [Mukherjea 94] entwickelt wurden.
In Übersichtsartikeln wird u.a. über weitere Aktivitäten im Bereich der Visualisierung von Informatiklehrinhalten berichtet [Price, Baecker, Small 93; Roman 93].

5. Gewonnenen Erkenntnisse, Ausblick auf zukünftige Aktivitäten

Bei der Betrachtung vieler über das Internet verfügbarer Animationen bekamen wir den Eindruck, dass für einen Teil dieser Animationen gilt:

- Einige Animationen stellen triviale Sachverhalte dar (Beispiel: Das Wandern von Daten durch die Schichten des OSI-Modells von oben nach unten und dann anschließend von unten nach oben). Solche Animationen bringen den Studierenden keine Erkenntnisse, die diesen nicht auch auf herkömmliche Art und Weise vermittelt werden könnten. Hier werden Animationen unserer Meinung nach nur der Effekthascherei wegen eingesetzt.
- Ein anderer Teil von Animationen zeigen nach dem Start eine verwirrende Fülle von Vorgängen, die von den betrachtenden Personen nicht verstanden werden können. Die durchgeführten Änderungen sind abrupt oder zu schnell. Oft erfolgen auch mehrere Veränderungen an unterschiedlichen Stellen gleichzeitig. Der Ablauf kann von den betrachtenden Personen nicht nachvollzogen werden. Dies sind sicherlich einige der Gründe, weshalb die Erfahrungen über den Einsatz von Animationen in der Lehre teilweise enttäuschend sind [Stasko 93].

Folgende Konsequenzen aus diesen Erkenntnissen versuchten wir, in den entwickelten Animationen umzusetzen:

- Man sollte sorgfältig die Sachverhalte auswählen, die durch Animationen visualisiert werden. Besonders gut geeignet sind Sachverhalte, die relativ komplex sind und die sich anhand von Beispielen gut erklären lassen:
  Auf der einen Seite lassen sich viele Sachverhalte ohne Verluste auch auf herkömmlichen Weg erklären. Wenn für jede Kleinigkeit eine Animation gezeigt

wird, dann stumpfen die Lernenden mit der Zeit ab; die Animationen haben ihren Reiz verloren, da die Studierenden sich daran gewöhnt haben. Auf der anderen Seite können Animationen unserer Meinung nach nicht die Beherrschung theoretischer und formaler Verfahren und Methoden "auf dem Königsweg" bieten. Die gründliche Auseinandersetzung der Studierenden mit dem Lernstoff, das eigenständige Denken und Grübeln kann den Studierenden nicht erspart bleiben.

- Animationen müssen sehr stark von den benutzenden Personen gesteuert werden. Eine Animation soll nicht wie ein Film nur betrachtet werden:
  Die manuelle Steuerung hat zwei Vorteile: Zum einen kann damit die Ablaufgeschwindigkeit reguliert werden. Es muss nach einem gesehenen Effekt die Animation beliebig lang gestoppt werden können, um über den gesehenen Effekt nachdenken zu können bzw. in einer Vorlesung den Effekt mit den Studierenden zu diskutieren. Zum anderen erzwingt eine explizite Angabe des nächsten Schrittes die betrachtenden Personen, sich aktiv mit der Animation auseinanderzusetzen. In vielen Diskussionen mussten die Entwickler der Animationen überzeugt werden, manche Automatismen, die technisch einfach machbar gewesen wären, nicht in die Animationen einzubauen, sondern diese Steuerungsaufgabe den späteren Nutzern zu überlassen. Hier zeigt sich die alte Erkenntnis, dass nicht alles, was technisch machbar ist, in einer konkreten Situation auch nützlich sein muss.
- Es sollen in der Lehre nicht nur fertige Animationen "konsumiert" werden, sondern die Studierenden sollen mit verhältnismäßig geringem Aufwand eigene Animationen "produzieren":
  Den Studierenden wird mit dem ***Vivaldi***-Kern eine Plattform zur Verfügung gestellt, mit der sie eigene Programme visualisieren können, wobei in diesem Fall die Visualisierung nicht für andere gedacht ist, sondern primär für sie selbst. Stehen den Studierenden z.B. entsprechende Visualisierungsoperationen zum Vergleich zweier Elemente und zum Vertauschen zweier Elemente zur Verfügung, so können sie selber unterschiedliche Sortieralgorithmen programmieren und an entsprechender Stelle die zur Verfügung gestellten Visualisierungsoperationen in ihr Programm einbauen (d.h. sie können ihr Sortierprogramm zum Zwecke der Visualisierung instrumentieren). In diesem Fall kann die Visualisierung zum schnellen Erkennen von Programmierfehlern dienen.

In Zukunft sind folgende weitere Aktivitäten geplant:

- Die Animationen sollen in unterschiedlichen Veranstaltungen an der FH Trier eingesetzt werden (z.B. Vorlesungen des Studiengangs "Angewandte Informatik", Präsenzphase des Fernstudiums "Allgemeine Informatik"). Dabei sollen zum einen vorbereitete Animationen während der Vorlesung präsentiert werden. Zum anderen sollen die Studierenden vorgegebene Visualisierungsoperationen in ihre selbst entwickelten Programme einbauen, um damit besser erkennen zu können, ob ihre Programme auch das ausführen, was sie sollen.
- In Software-Engineering-Projekten, Projekt- und Diplomarbeiten sollen weitere Animationen erstellt und bestehende verbessert werden.

Danksagung: Wir bedanken uns bei allen Studierenden, die Beiträge zum ***Vivaldi***-Projekt geliefert haben, vor allem bei Bernd Greve, Torsten Martin, Christoph Mettler, Thomas Schmitt und Markus Sell. Außerdem bedanken wir uns bei den Förderern des ***Vivaldi***-Projekts, der Nikolaus-Koch-Stiftung in Trier und dem Ministerium für Bildung, Wissenschaft und Weiterbildung Rheinland-Pfalz in Mainz.

6. Literatur

[Brown 84] M.H. Brown, R. Sedgewick: A System for Algorithm Animation, Computer Graphics, July 1984, pp. 177-186.

[Brown 88a] M.H. Brown: Algorithm Animation, MIT Press, Cambridge, Mass., 1988.

[Brown 88b] M.H. Brown: Exploring Algorithms Using Balsa-II, Computer, Vol. 21, No. 5, May 1988, pp. 14-36.

[Brown 91] M.H. Brown: Zeus: A System for Algorithm Animation and MultiView Editing, Proceedings IEEE Workshop Visual Languages, IEEE CS Press, Los Alamitos, California, 1991, pp. 4-9.

[Gamma et al. 95] E. Gamma, E. Helm, R. Johnson, J. Vlissides: Design Patterns - Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[Mukherjea 94] S. Mukherjea, J.T. Stasko: Toward Visual Debugging: Integrating Algorithm Animation Capabilities within a Source-Level Debugger, ACM Transactions on Computer-Human Interaction, Vol. 1, No. 3, September 1994, pp. 215-244.

[Oechsle 99] R. Oechsle: Basiskonzepte des Visualisierungsprojekts ***Vivaldi***, Interner Bericht 99/01, Fachbereich Angewandte Informatik, FH Trier, Januar 1999.

[Oechsle, Becker 99] R. Oechsle, R. Becker: Das Projekt ***Vivaldi*** (*Vi*sualisierung *v*on *a*usgewählten *L*ehrinhalten *d*er *I*nformatik), 15. GI/ITG-Fachtagung "Architektur von Rechnensystemen" (ARCS '99) und "Arbeitsplatz-rechensysteme" (APS '99), Friedrich-Schiller-Universität Jena, 4. – 7. Oktober 1999, pp. 263-270.

[Price, Baecker, Small 93] B.A. Price, R.M. Baecker, I.S. Small: A Principled Taxonomy of Software Visualization, Journal of Visual Languages and Computing 3, 1993, pp. 211-266.

[Roman 93] G.-C. Roman, K.C. Cox: A Taxonomy of Program Visualization Systems, IEEE Computer, Vol. 26, No. 12, December 1993, pp. 11-24.

[Stasko 89] J.T. Stasko: Tango: A Framework and System for Algorithm Animation, PhD Thesis, Brown University, Providence, RI, May 1989.

[Stasko 90] J.T. Stasko: Tango: A Framework and System for Algorithm Animation, IEEE Computer, Vol. 23, No. 9, September 1990, pp. 27-39.

[Stasko 93] J.T. Stasko, A. Badre, C. Lewis: Do Algorithm Animation Assist Learning? An Empirical Study and Analysis, Proceedings of the INTERCHI '93 Conference on Human Factors in Computing Systems, Amsterdam, Netherlands, April 1993.