

Konzeption und Realisierung eines browserbasierten Editors für Kontrollflussgraphen

Bachelorarbeit

Eingereicht von Alexander Düsel

5. Oktober 2007

Lehrgebiet Software Engineering
Fakultät für Mathematik und Informatik
FernUniversität in Hagen

Betreuer:

Prof. Dr. Hans-Werner Six
Dipl.-Inform. Immo Schulz-Gerlach

Widmung

Für meine liebe Mutter Helene, † 19.09.2007

Überblick

Im Lehrbetrieb des Lehrgebiets Software Engineering können seit acht Jahren Studierende mittels eines Java Applets Kontrollflussgraphen zeichnen und auf den Server des *WebAssign* Systems zur Korrektur übertragen.

Die Realisierung dieses *browserbasierten Editors für Kontrollflussgraphen* ist wegen Inkompatibilitäten mit aktueller Browsersoftware, neuer Techniken im Bereich Webentwicklung und aus Gesichtspunkten der Benutzerführung nicht mehr zeitgemäß. Es wird ein Ersatz gesucht.

Dabei sind spezielle Rahmenbedingungen zu beachten. Zum Beispiel nutzen viele Studierende noch eine langsame Internetverbindung, und das Datenaufkommen bei der Übertragung sollte deshalb nicht zu groß sein. Die Unterstützung einer neuen Lösung durch verschiedene Browsersoftware soll gewährleistet sein. Weiterhin sollte die Anwendung auch unterschiedliche Konfigurationseinstellungen dieser Software berücksichtigen.

Ausgehend von einer Betrachtung verschiedener konzeptioneller Möglichkeiten erfolgt in dieser Arbeit die Realisierung einer neuen Version des Editors. Dabei wird dieser nahtlos in eine Webseite integriert.

Das bedeutet, dass kein kompiliertes Programm übertragen und vom Browser ausgeführt wird, wie bei Java Applets oder Flash Filmen, sondern es wird mittels nativer Frontendtechnologie, wie JavaScript, HTML und CSS, eine interaktive grafische Anwendung realisiert. Änderungen auf der Serverseite werden dabei nicht notwendig, sondern es werden nur die HTML Dokumente angepasst, die der Server ausliefert.

Vom traditionellen Ansatz der Benutzerführung eines grafischen Editors wird ein Stück weit abgerückt: So werden nicht Elemente auf einer Zeichenfläche interaktiv mit der Maus angeordnet, sondern auf Tastatureingaben in Formularfeldern wird ein unmittelbares grafisches Feedback gegeben.

Neben der Editorfunktionalität, gibt es auch eine Möglichkeit zur bloßen Anzeige eines Graphen in einem Pop-up-Fenster. Außerdem wurde eine Version realisiert, bei der man die Darstellung eines Kontrollflussgraphen interaktiv einstellen kann, insbesondere den Knotenradius, den Knotenabstand und den Kurvenverlauf der Kanten.

Abschließend wird kurz angesprochen inwieweit die vorgestellte Technik in weiteren Lehrveranstaltungen eingesetzt werden kann.

Inhaltsverzeichnis

Überblick.....	3
Inhaltsverzeichnis.....	4
1. Einleitung.....	6
2. Zielsetzung.....	7
3. Analyse des Problems.....	8
3.1 Allgemeine Problembeschreibung.....	8
3.2 Funktionalität der bestehenden Lösung.....	10
3.3 Schnittstelle der bestehenden Lösung zur Serverseite.....	13
3.4 Die optimale Benutzerführung.....	13
3.5 Grundlegende Technische Möglichkeiten.....	14
4. Technische Konzeption.....	16
4.1 Realisierungsalternativen.....	16
4.1.1 Realisierung als Webformular mit Serverinteraktion.....	16
4.1.2 Ausführung durch ein Browser-Plug-in.....	19
4.1.3 Realisierung als Programmskript.....	20
4.2 Die Grafikfähigkeiten im Allgemeinen.....	21
4.3 Das Canvas-Element im Speziellen.....	23
4.4 Konzeptionelle Entscheidung.....	25
5. Realisierung.....	27
5.1 Eingeschlagener Realisierungsweg.....	28
5.2 Integration der Zeichenfläche.....	29
5.3 Einbindung von CSS und JavaScript in die HTML Dokumente.....	34
5.4 Alternative ohne JavaScript Unterstützung.....	36
5.6 Realisierung der Interaktiven Grafik.....	41
5.6.1 Zeichnen der Kanten.....	41
5.6.2 Zeichnen der Pfeilspitzen.....	46
5.6.3 Layoutalgorithmus.....	49
5.6.4 Verarbeitung der Benutzereingaben.....	53
5.7 Programmstruktur.....	56
Zusammenfassung und Ausblick.....	57
Ausblick.....	58

Anhang: Browserkompatibilität der Anwendung.....	60
Literaturverzeichnis.....	64
Erklärung.....	68

1. Einleitung

Webapplikationen haben sich wegen der verteilten Struktur des Internets, einer Standardisierung und unterschiedlicher Hardwareanforderungen über die Jahre bezüglich Optik, Benutzerführung und Verhalten in eine andere Richtung entwickelt als Desktopapplikationen. Nicht zu unterschätzen ist auch der Einfluss führender Browserhersteller.

Die Benutzer sind bereit, längere Reaktionszeiten bei Webanwendungen in Kauf zu nehmen, was bei Desktopapplikationen undenkbar wäre. Ebenso sind diese relativ tolerant gegenüber Fehlermeldungen und Datenübertragungsstörungen.

In letzter Zeit vollzieht sich aber wegen höherer Internetübertragungsraten und leistungsfähigerer PC Hardware ein Wandel, und Webanwendungen nähern sich in den oben genannten Punkten stark an Desktopapplikationen an.

Es hat sich ein neuer Begriff *WEB2.0* [WEB05] etabliert. In erster Linie als Marketing-schlagwort gebraucht, bezeichnet er auch relativ unscharf verschiedene neue interaktive Techniken und Dienste des Internets, kurz gesagt, eine neue Generation von Webanwendungen.

Als prominentes Beispiel ist der *Google Maps* Dienst [MAP05] oder die *Live Search* von Microsoft [LSE06] zu nennen. Mit letzterem lässt sich u.a. eine Suchbox in eigene Webseiten integrieren. Bei Eingabe eines Suchbegriffs wird man nicht auf eine Ergebnisseite weitergeleitet, sondern ein Pop-up-Fenster erscheint, welches die Suchergebnisse präsentiert. Dieses Verfahren wird oft kopiert, und man findet z.B. Suchboxen, die sofort auf die Eingabe eines Zeichens reagieren und das eingegebene Wort automatisch vervollständigen [ROJ06]. Die Eingabe eines Wertes in ein herkömmliches Formularfeld auf einer Webseite bewirkt hier unmittelbar eine Änderung des Zustands und eine Ausgabe auf der Webseite, ohne dass der Benutzer den Eindruck hat, dass etwas vom Server nachgeladen werden muss. Feedback ohne Verzögerung trägt hier erheblich zum Bedienkomfort bei.

Motiviert durch diese Beispiele gelingt eine ungewöhnliche, aber in Hinblick auf Benutzerführung und Simplizität, sehr interessante Konzeption und Neuimplementierung eines browserbasierten Editors für Kontrollflussgraphen. Dieser soll im Lehrgebiet Software Engineering ab dem Wintersemester 2007/2008 eingesetzt werden.

2. Zielsetzung

In dieser Arbeit gilt es, in einem bestehenden Softwaresystem veraltete und teilweise fehlerhafte Komponenten durch eine verbesserte und fehlerfreie Lösung zu ersetzen. Dabei handelt es sich um einen *browserbasierten grafischen Editor für Kontrollflussgraphen*, der in das *WebAssign* System integriert ist. Auf eine bislang zur Option stehende, alternative Eingabemöglichkeit in Form eines Webformulars soll möglichst verzichtet werden. Neben der Editorfunktionalität ist auch eine reine Anzeigemöglichkeit erforderlich.

Ausgehend von diesem Primärziel lassen sich drei Unterziele identifizieren:

- A) Auswahl und Einsatz der für den Anwendungsbereich optimalen Technik;
- B) Minimierung der notwendigen Änderungen am Restsystem;
- C) Verbesserung der Benutzerführung des Editors; Ansprechende und platzsparende Darstellung eines Kontrollflussgraphen;

Wie man im Verlauf der Arbeit noch sehen wird, impliziert Ziel A) das Ziel B) und vorteilhaft ist, dass es keine Konkurrenz zwischen den Zielsetzungen gibt. Die Verfolgung von Ziel A) wird sich am aktuellen Stand der Technik orientieren, während Ziel B) eine hinreichende Beschäftigung mit der bestehenden Lösung nach sich zieht. Die Erreichung von Ziel C) begründet sich u.a. auf einer Untersuchung der zuletzt eingesetzten Anwendung aus Benutzersicht.

Von Seiten der Betreiber gibt es weiterhin Anforderungen:

1. Bei der Verfolgung von Ziel A) sollen verschiedene Realisierungsmöglichkeiten voneinander abgegrenzt werden.
2. Die neue Lösung soll ausnahmslos von allen Anwendern mit minimalem Aufwand benutzbar und erreichbar sein. Insbesondere soll es keinen Installations- und Einarbeitungsaufwand geben.

Den zweiten Punkt sehen die Betreiber nicht nur als Qualitätsmerkmal, sondern sind insbesondere wegen des kritischen Einsatzbereichs der Software interessiert, dass diese Anforderung unbedingt erfüllt wird. Der browserbasierte Editor soll von Studierenden im Übungsbetrieb der Kurse *1613 Einführung in die Imperative Programmierung* und *1612 Konzepte Imperativer Programmierung* verwendet werden. Beide Lehrveranstaltungen sind Pflichtteil des Studiums und die erfolgreiche Bearbeitung der Einsendeaufgaben zu den Übungen ist Voraussetzung für eine Klausurzulassung.

Der analytische Teil der vorliegenden Arbeit beginnt mit einer allgemeinen Problembeschreibung. Danach wird die Funktionalität der Clientseite der bestehenden Lösung betrachtet. Darauf folgt eine kurze Untersuchung der Schnittstelle zur Serverseite. Eine Besprechung der Benutzerführung sowie der grundlegenden technischen Möglichkeiten bildet den Abschluss der Analyse des Problems.

3. Analyse des Problems

3.1 Allgemeine Problembeschreibung

Wie in der Zielsetzung besprochen, wird ein browserbasierter grafischer Editor für Kontrollflussgraphen realisiert. Mit diesem sollen Teilnehmer der Kurse 1612 und 1613 eine Teilaufgabe zu einer Einsendeaufgabe bearbeiten. Korrektoren und Betreuern dieser Kurse soll ebenfalls eine browserbasierte Möglichkeit gegeben werden, mit der sie die erstellten Graphen betrachten können.

Im Verlauf dieser Arbeit wird zwischen diesen beiden Varianten der Anwendung unterschieden. Es wird von zwei Modi gesprochen:

- Editormodus (für die Kursteilnehmer)
- Anzeigemodus (für die Korrektoren / Betreuer)

Zu Beginn der Problemanalyse sollen Fragestellungen allgemeiner Art geklärt werden:

1. Wie sieht ein Kontrollflussgraph aus? Ist die geforderte grafische Editorfunktionalität notwendig? Kann man den Graphen auch textuell beschreiben?
2. Wieso soll der Editor bzw. das Anzeigeprogramm für die Korrektoren im Internetbrowser ausgeführt werden? Kann die Applikation nicht auch als Desktopanwendung realisiert werden?

Zur Beantwortung der Fragen unter Punkt 2 lässt sich feststellen, dass eine browserbasierte Lösung offensichtlich zu einer örtlichen Unabhängigkeit der Benutzer beiträgt. Die Applikation ist von einem bestimmten Rechnersystem entkoppelt und sollte ohne Aufwand auf jedem beliebigen Computer genutzt werden können, sofern eine Internetanbindung vorhanden ist, und eine Browsersoftware zur Verfügung steht.

Die Anwender sollten auch unabhängig von einer spezifischen Rechner- und Softwareausstattung mit der Applikation arbeiten können.

Prinzipiell abstrahiert zwar eine Browsersoftware, die sich an Webstandards hält, von einer spezifischen Plattform, in der Praxis zeigen sich aber oft technische Probleme (vgl. 3.2)

Als weiterer Vorteil lässt sich festhalten, dass bei einer browserbasierten Lösung kein expliziter Installations- und Downloadaufwand für den Benutzer anfällt, wie das bei einer Desktopsoftware üblich ist.

Es liegt nahe, dass gerade Studierende der *FernUniversität in Hagen* das Internet für ihre Studientätigkeit an verschiedenen Orten nutzen, zum Beispiel zu Hause, in der Bibliothek, auf einer Präsenzveranstaltung in Hagen, am Arbeitsplatz, etc. und von der größeren Flexibilität einer browserbasierten Lösung profitieren würden.

Zu den Fragen unter Punkt 1 betrachten wir den folgenden Screenshot. Hier ist beispielhaft ein Graph dargestellt. Dieser wurde mit dem *browserbasierten grafischen Editor für Kontrollflussgraphen* gezeichnet, so wie er bis zum Wintersemester 2005/2006 eingesetzt wurde.

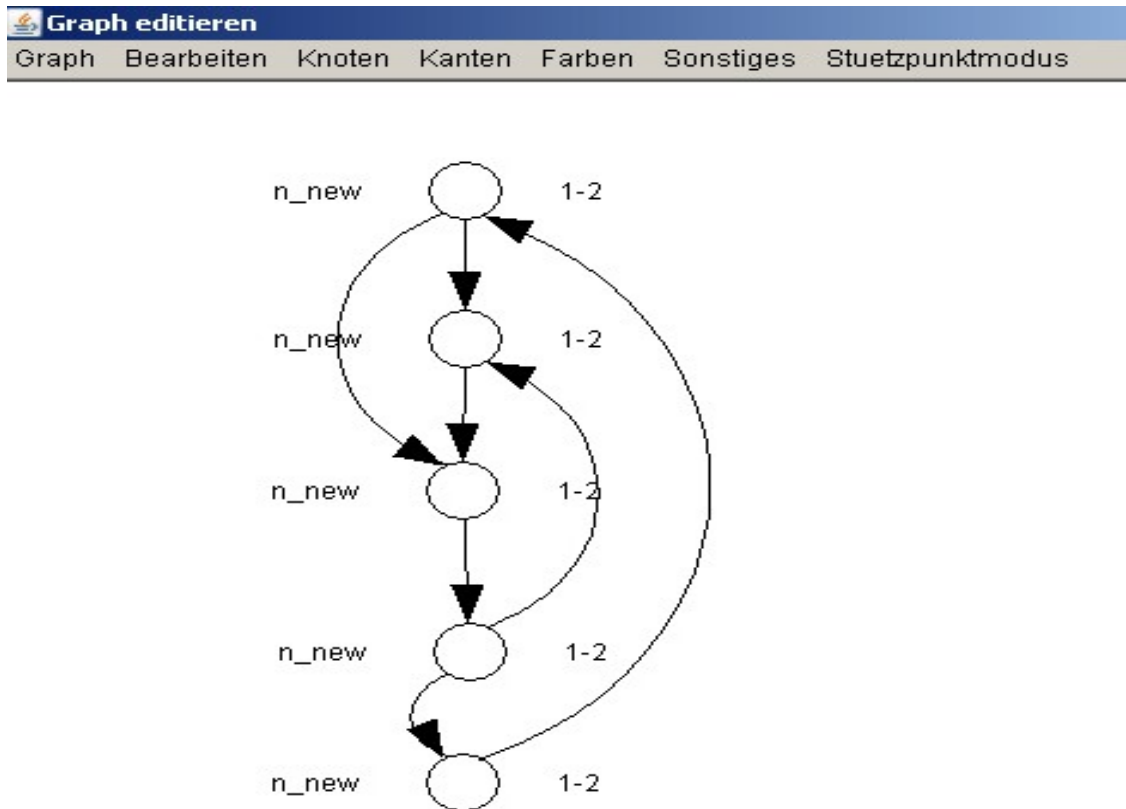


Abbildung 1: Screenshot der bestehenden Lösung (Appletversion).

Man sieht in Abb. 1, dass alle Knoten des Graphen auf einer Vertikalen angeordnet sind, der Knotenabstand jeweils gleich ist, und nach unten führende Kanten links von den Knoten verlaufen, bzw. bei benachbarten Knoten mit den Knoten auf einer Linie liegen. Nach oben gerichtete Kanten sind rechts von den Knoten gezeichnet.

Eine textuelle Repräsentation eines einfachen Kontrollflussgraphen würde sich - mit dieser Darstellung im Sinn - auf den folgenden Überlegungen begründen:

Da es keine Verzweigungen in den Kanten gibt, kann eine Kante als ein Tupel zweier Knotennummern aufgefasst werden. Der Graph in der Abbildung kann beispielsweise mit einer Menge KG von Tupeln beschrieben werden (Wenn man sich den Graphen, von oben nach unten durchnummeriert, beginnend mit der Nummer 1, vorstellt):

$$KG = \{ (1,2), (2,3), (3,4), (4,5), (1,3), (4,2), (5,1) \}$$

Die erste Komponente eines Tupels gibt den Startknoten der Kante an, während die zweite Tupelkomponente den Endknoten der Kante bezeichnet. Mit der Angabe dieser Menge ist ein Graph vollständig und widerspruchsfrei spezifizierbar. Auf die Angabe eines Beweises wird an dieser Stelle verzichtet.

Die Frage nach der textuellen Beschreibung ist damit geklärt. Aber ohne weitere Hilfsmittel ist diese Art der Darstellung eines Graphen schwierig zu erzeugen. Man könnte zwar als Gedächtnisstütze eine Zeichnung auf Papier anfertigen und sich anschließend die Tupelmenge überlegen, dieses Vorgehen erscheint aber wenig komfortabel und eine rechnergestützte Hilfe in Form eines grafischen Editors wäre sinnvoll.

Mit dieser Motivation und dem Grundwissen bezüglich der Darstellung eines einfachen Kontrollflussgraphen, sowohl grafisch, als auch in Form einer textuellen Beschreibung, sowie der Identifizierung der Vorteile einer browserbasierten Lösung, wird im nächsten Abschnitt die bestehende Lösung näher betrachtet.

3.2 Funktionalität der bestehenden Lösung

Im Folgenden wird die Lösung betrachtet, so wie sie bis zum Wintersemester 2005/2006 am Lehrgebiet Software Engineering eingesetzt wurde.

Man kann zwei Gruppen von Anwendern unterscheiden (vgl. 3.1):

- Zum einen gibt es die Kursbeleger, die die Repräsentation eines einfachen Kontrollflussgraphen über das WWW zur Korrektur einsenden.
- Zum anderen sehen sich Korrektoren und Betreuer diese Repräsentation des Graphen an und bewerten diese.

Die erste Gruppe kann zwischen zwei Eingabealternativen wählen: Einerseits kann mittels eines Java Applets der Kontrollflussgraph gezeichnet werden (Appletversion), andererseits gibt es eine Alternativlösung in Form eines herkömmlichen Webformulars (Formularversion). Zu jedem Knoten des Graphen gibt es bei dieser Variante ein Eingabefeld in welches die Nummern der Nachfolgeknoten eingetragen werden.

Die Gruppe der Korrektoren und Betreuer hat diese Option nicht und muss eine Einsendung, die mit der Appletversion erstellt wurde, auch mit dieser betrachten. Im anderen Fall kann ein Graph, der mit der Formularversion eingesendet wurde, auch nur textuell in Form einer Tabelle angezeigt werden.

Wie man anhand des Screenshots in Abbildung 1 sehen kann, verfügt die Appletversion über eine Menüleiste mit verschiedenen Menüpunkten. Es stellt sich auf den ersten Blick die Frage, ob die Funktionalität für diesen Anwendungsfall nicht zu umfangreich ist.

Die Benutzerführung der Appletversion ist folgendermaßen realisiert: Mittels eines Klicks mit der linken Maustaste auf die Zeichenfläche wird ein neuer Knoten des Graphen an der Mausposition gezeichnet.

Mit gedrückter Umschalttaste und durch Halten der linken Maustaste kann man Kanten anlegen. Dabei muss man zunächst in der Menüleiste die gewünschte Art der Kante, ob gerade Kante, runde Kante oder eckige Kante, auswählen. Die Knoten können dann frei auf der Zeichenfläche positioniert werden, und die Kanten bewegen sich dabei mit. Das Löschen eines Knotens ist umständlich. Zunächst muss der zu löschende Knoten durch einen Linksklick selektiert werden und kann dann durch eine Menüauswahl gelöscht werden.

Insgesamt gesehen macht das Applet zwar einen guten Eindruck, aber für diesen Einsatzbereich erscheint die Bedienung zu kompliziert. Eine längere Einarbeitungszeit ist nicht wünschenswert, da ein Student diesen Editor im Übungsbetrieb der Lehrveranstaltung im günstigsten Fall nur einmal benutzen wird, nämlich zur Bearbeitung einer Teillaufgabe einer Einsendeaufgabe.

Die relativ umständliche Benutzerführung ist vermutlich ein Grund, warum sich die meisten Studierenden für die Formularversion entscheiden. Diese wird nach Erfahrungen der Betreiber gegenüber der Appletversion bevorzugt. So wurden im Wintersemester 2004/2005 nur 26 von 244 Einsendungen mit der Appletversion durchgeführt.

Die schlechte Akzeptanz der Appletversion begründet sich nicht nur auf der relativ komplizierten Bedienung, sondern liegt vermutlich auch an technischen Problemen mit der Java Applet Technologie. Es ist auch denkbar, dass die Studierenden den Link, der auf die Webseite der optionalen Appletversion verweist, übersehen und deshalb die Formularversion benutzen.

Java Applets sind kompilierte Programme, die von der sogenannten *Java Virtual Machine* (JVM) ausgeführt werden, welche es als Plug-in für Browsersoftware gibt. Alle populären Internetbrowser werden derzeit standardmäßig mit einem solchen vorinstallierten Plug-in ausgeliefert. Mit dem Begriff *populäre Browser* wird im weiteren Verlauf dieser Arbeit die Software Internet Explorer, Mozilla/Firefox, sowie Opera auf dem Betriebssystem Microsoft Windows zusammengefasst.

Diese JVM sowie die Programmiersprache Java unterliegen einer Evolution. Es ist nicht gewährleistet, dass Applet Programme, die mit einer früheren Version des Java Compilers übersetzt wurden, auch mit einer aktuellen Version der JVM problemlos funktionieren. Die Appletversion des Online Editors für Kontrollflussgraphen wurde im Jahr 2000 entwickelt und kompiliert. Seit einigen Semestern kommt es zu Programmfehlern und Inkompatibilitäten mit verschiedener Browsersoftware.

Eine Neu-Übersetzung und gegebenenfalls eine Anpassung des Quelltextes des Programms wird von den Betreibern abgelehnt, denn es kann nicht eine Version für jede erdenkliche Browser/JVM/Betriebssystem-Kombination geben.

Besonders problematisch ist dieser Umstand für die Korrektoren, die eine Einsendung, die mit der Appletversion erstellt wurde, auch mit dieser betrachten müssen. Dieser Punkt war für den Kursbetreuer ausschlaggebend, die Appletversion schließlich zum Wintersemester 2006/2007 nicht mehr anzubieten, sondern nur noch die Formularversion.

Wenn man sich den folgenden Screenshot der Formularversion ansieht, kann man sich denken, dass eine Verbesserung bezüglich der Gestaltung motivierend auf die Anwender wirken würde, und es sich bei dem Webformular eher um eine Notlösung handelt:

Knotenname	Knotennummer	Nummern der Nachfolgeknoten	Programzeilen von	bis
<input type="text" value="n_start"/>	1	<input type="text" value="2"/>	<input type="text"/>	<input type="text"/>
<input type="text" value="n_if"/>	2	<input type="text" value="3,4"/>	<input type="text" value="2"/>	<input type="text" value="3"/>
<input type="text" value="n_then"/>	3	<input type="text" value="7"/>	<input type="text" value="4"/>	<input type="text" value="7"/>
<input type="text" value="n_else"/>	4	<input type="text" value="5"/>	<input type="text" value="8"/>	<input type="text" value="11"/>
<input type="text" value="n_while"/>	5	<input type="text" value="6,7"/>	<input type="text" value="12"/>	<input type="text" value="12"/>
<input type="text" value="n_do"/>	6	<input type="text" value="5"/>	<input type="text" value="13"/>	<input type="text" value="18"/>
<input type="text" value="n_final"/>	7	<input type="text"/>	<input type="text"/>	<input type="text"/>

Abbildung 2: Screenshot der Formularversion

Die Eingabe der Kanten des Graphen erfolgt über die Formularfelder in der Spalte `Nummern der Nachfolgeknoten`. Offensichtlich ist der große Nachteil bei dieser Lösung der Umstand, dass diese Kanten nicht dargestellt werden. Deshalb wird den Anwendern geraten, zunächst den Graphen auf Papier zu zeichnen und daraufhin das Formular lediglich zur Übermittlung der Lösung zu benutzen.

Eine Einsendung, die mit der Formularversion erstellt wurde, wird den Korrektoren als Tabelle analog zu Abbildung 2 angezeigt. Mit einer grafischen Anzeige könnte vermutlich die Korrekturleistung und die Zufriedenheit der Korrektoren gesteigert werden. Wie die Datenübertragung bei der bestehenden Lösung und somit die Übermittlung der Einsendung zum Server erfolgt, wird anhand der Schnittstelle zur Serverseite im nächsten Abschnitt angesprochen.

3.3 Schnittstelle der bestehenden Lösung zur Serverseite

Neben der Betrachtung der Funktionalität der bestehenden Lösung im vorangegangenen Abschnitt, kann auch die Untersuchung der Schnittstelle zwischen Client- und Serverseite ohne größeren Aufwand, wie z.B. eine umfangreiche Quelltextrecherche, durchgeführt werden.

Wie sich im weiteren Verlauf dieser Arbeit zeigen wird, reicht insbesondere die Analyse der Schnittstelle der Formularversion aus, um die Ziele dieser Arbeit zu erreichen. Nachdem die Betreiber mit der Appletversion nicht zufrieden waren, wird diese auch nicht weiter betrachtet.

Da es sich bei der Formularversion um ein einfaches Webformular handelt, ist die Schnittstelle trivial. Eine Untersuchung des Quelltextes eines relevanten HTML Dokuments ergibt, dass allein über die Methode *POST* des *HTTP Protokolls* die Eingabedaten zum Server übermittelt werden.

3.4 Die optimale Benutzerführung

Nach den Erfahrungen aus dem Lehrbetrieb mit Appletversion und Formularversion (vgl. 3.2) sehen die Betreiber eine Kombination der Vorteile beider Lösungen als optimal an. Es gilt deshalb die Einfachheit der Formularversion bezüglich des Eingabeverhaltens und das unmittelbare grafische Feedback der Appletversion zu kombinieren. Wichtig ist, dass die Benutzer die Anwendung intuitiv bedienen können.

Bisher wurde den Anwendern überlassen, ob sie mit der Appletversion oder mit der Formularversion arbeiten wollen. Diese Auswahl und die Bereitstellung einer Alternativlösung wollen die Betreiber zukünftig vermeiden, sind aber bereit, weiterhin eine Formularversion für wenige Ausnahmefälle anzubieten.

Nach diesen Aussagen der Verantwortlichen liegt es nahe, die Tabellenform der Formularversion für eine Neuimplementierung beizubehalten, vgl. Abbildung 2. Statt der Spalte `Knotennummer` soll eine Zeichenfläche für die Grafikausgabe in die Webseite integriert werden.

An dieser Stelle sei mit dem folgenden Bildschirmausschnitt auf einen ersten Prototypen der neuen Lösung vorgegriffen.

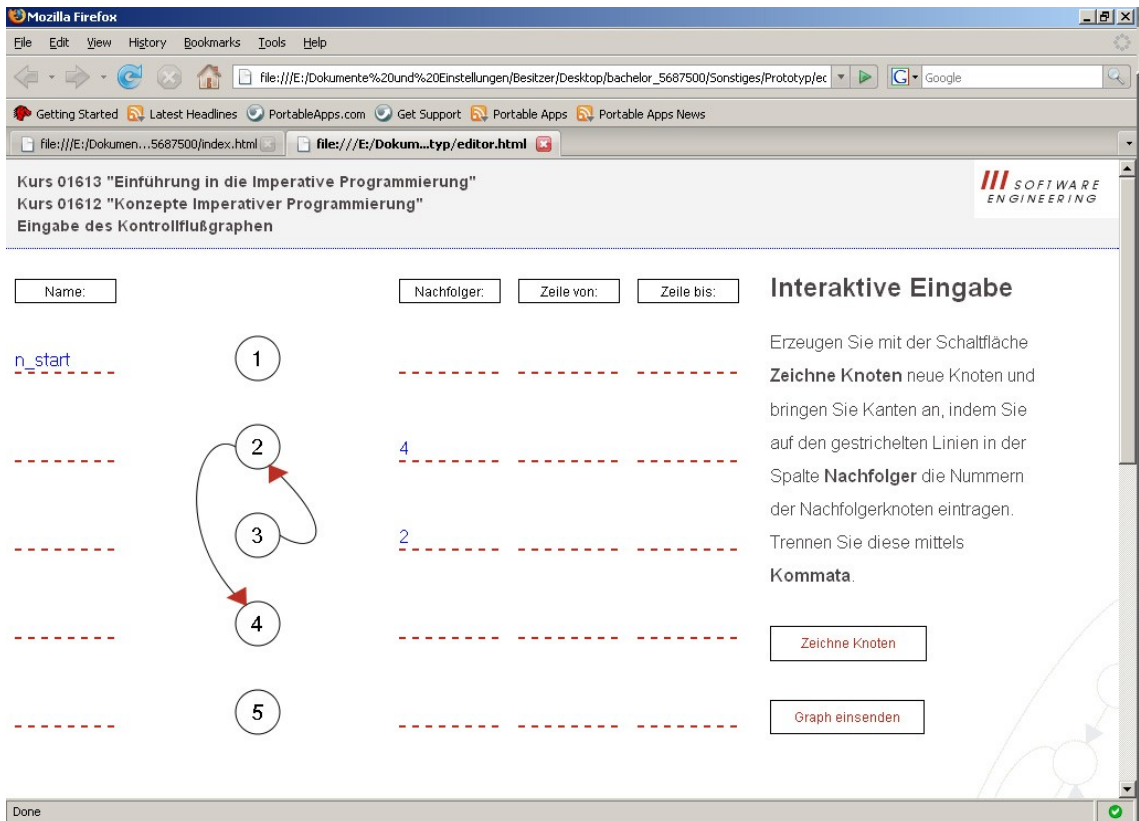


Abbildung 3: Screenshot eines Prototypen der neuen Lösung

3.5 Grundlegende Technische Möglichkeiten

Die technischen Möglichkeiten werden durch die Verwendung eines Webbrowsers von Seiten der Benutzer limitiert. Diese Software kann, je nach Version und Hersteller, Dokumente anzeigen, kompilierte Programme durch ein Plug-In ausführen und Programmskripts interpretieren.

Weiterhin kann die Kommunikation zwischen der Browsersoftware und dem Webserver zum einen synchron und zum anderen asynchron erfolgen. In der folgenden Matrix sind die Möglichkeiten gegenübergestellt. Die Appletversion der bestehenden Lösung ist dabei mit einem „A“ und die Formularversion mit einem „F“ markiert.

	Synchrone Kommunikation	Asynchrone Kommunikation
Dokumentanzeige	F	-
Plug-In Programm	A	-
Programmskript	-	-

Anmerkung: *Ajax*

Eine asynchrone Kommunikation zwischen Client (Browser) und Server ist in letzter Zeit im Zuge des *Web2.0* (vgl. Einleitung) populär geworden.

Es hat sich der Begriff *Ajax* (Asynchronous JavaScript and XML) etabliert. Dieser bezeichnet den Zusammenschluss verschiedener Techniken im Bereich clientseitiger Webentwicklung mit denen es z.B. möglich ist, HTML-Fragmente oder auch nur reine Nutzdaten bei Bedarf vom Server nachzuladen. Diese können dynamisch in die Darstellung der Webseite integriert werden, ohne dass die Webseite im Browser komplett neu geladen werden muss.

Auch das Senden von Benutzereingaben (z.B. in Formularfeldern) zum Server kann mittels *Ajax* asynchron erfolgen. Offensichtlich verbessert der Einsatz von *Ajax* die Reaktionszeiten einer Webanwendung und damit die Usability. Weitere Informationen über *Ajax* findet der interessierte Leser in einem Artikel von Jesse James Garret, der diesen Begriff geprägt hat [JJG07]. Auf *Ajax* wird kurz im folgenden konzeptionellen Teil dieser Arbeit zurückgekommen.

Im nächsten Kapitel werden konkrete Realisierungsmöglichkeiten voneinander abgegrenzt. Schließlich wird eine Entscheidung zu Gunsten einer dieser Alternativen getroffen.

4. Technische Konzeption

4.1 Realisierungsalternativen

Eine Kombination der grundlegenden technischen Möglichkeiten, unter Einbeziehung der im Abschnitt 3.4 getroffenen Aussagen des Betreibers bezüglich der Benutzerführung einer neuen Version, liefert die folgenden Alternativen für eine neue Konzeption der Anwendung.

Auf technische Details wird an dieser Stelle noch nicht eingegangen. Es wird anhand der zwei elementaren Funktionalitäten der Anwendung unterschieden: Eingabeverhalten und Feedback.

Dabei ist zu beachten, dass die Anwendung umso benutzerfreundlicher wird, je mehr Feedback auf das Eingabeverhalten gegeben wird.

4.1.1 Realisierung als Webformular mit Serverinteraktion

- **Eingabe:** erfolgt über herkömmliches Webformular.
- **Feedback:** vom Server generierte Bilder des Graphen.

Die erste Realisierungsmöglichkeit erweitert die in Abschnitt 3.2 vorgestellte Formularversion um eine Vorschaufunktion, d.h. es wird z.B. eine weitere Schaltfläche zum Webformular hinzugefügt, die mit der Bezeichnung `Vorschau` versehen ist.

Klickt der Benutzer auf diese Schaltfläche, dann werden die in den Formularfeldern vorhandenen Daten zum Server übermittelt, und dieser generiert ein Bild des Graphen, welches er zurück an den Client schickt. Die Darstellung des Graphen kann dann entweder in einem neuen Browserfenster, in einem Frame oder in einem Pop-up-Fenster erfolgen. Es ist auch denkbar, dass die Darstellung des Graphen direkt in Webseite des Webformulars integriert ist, so wie das in Abschnitt 3.4 vorgeschlagen wird.

Die zuletzt genannte Möglichkeit hat den Nachteil, dass die Webseite des Formulars erneut im Browser geladen werden muss und während der Datenübertragung für den Benutzer nicht zur Verfügung steht. Diesen Nachteil kann man durch den Einsatz von *Ajax* (vgl. 3.5) beheben. Vorteilhaft ist bei dieser Lösung, dass kein Fokuswechsel zwischen verschiedenen Browserfenstern nötig wird. Diesen Vorteil hat man auch bei der Realisierung mittels eines HTML Frames.

Es ist auch denkbar, dass es keine Schaltfläche `Vorschau` gibt, sondern die Webseite auf Benutzereingaben in den Formularfeldern automatisch reagiert und für jede Benutzereingabe der Server zur Grafikerzeugung konsultiert wird. Je nach Geschwindigkeit der Internetanbindung erscheint dieses Vorgehen mehr oder weniger praktikabel.

Außerdem kann man sich noch eine Variante vorstellen: Statt der mehrmaligen Übertragung des Bildes des kompletten Graphen können auch nur Bilder eines Teilgraphen bzw. Bilder der Kanten übertragen und an vordefinierter Stelle in der Browserdarstellung eingefügt werden.

Diese Methode hat Vorteile bezüglich der übertragenen Datenmenge und daraus resultiert eine kürzere Reaktionszeit der Anwendung. Die Realisierung ist aber nicht unproblematisch, da man sich überlegen muss, wie die vordefinierten Stellen für die einzelnen Kanten in die Webseite integriert und referenziert werden können.

Offensichtlich kann auch nicht für jede beliebige Kante - bei bspw. zwölf Knoten und der Möglichkeit von reflexiven Kanten wären das $12 * 12 = 144$ Kanten - eine Position in der Darstellung der Webseite vorab festgelegt werden.

Kreuzungen der einzelnen Kanten mit anderen Kanten sollten auch sauber gezeichnet werden können, was die Darstellung zusätzlich erschweren würde. Bei dieser Vorgehensweise wird vermutlich eine Limitierung der Eingabemöglichkeiten notwendig. Aus diesem Grund wird diese Idee nicht weiter verfolgt.

Diskussion

Während das oben Genannte unmittelbar einzusehen ist, bedarf es bei den folgenden Aspekten dieser Realisierungsmöglichkeit einer weiterführenden Erläuterung:

- Serverseitige Generierung der Grafik
- Zentrale Programmausführung
- Asynchrone Kommunikation

Serverseitige Generierung der Grafik

Auf den ersten Blick erscheint die Realisierung der oben beschriebenen Vorschaufunktionalität wie ein so genannter „Workaround“, weil die Eingabedaten schrittweise, mehrmals an den Server geschickt werden, und diese nur in einer anderen Form, nämlich als Grafik, wieder zurückkommen.

Offenbar wäre in dieser Hinsicht die Generierung der Grafik auf der Clientseite vorzuziehen. Eine Erzeugung der Grafik auf der Serverseite findet aber in verschiedenen Szenarien Anwendung.

Zum einen, wenn die schrittweise erzeugten Grafiken auch anderen Clients zur Verfügung gestellt werden, z.B. bei einer kollaborativen Anwendung, wie im Fall des gemeinschaftlichen Erstellens eines Graphen von mehreren Benutzern.

Zum anderen kann man die Erzeugung der Grafik auf der Serverseite in Betracht ziehen, wenn der Client aus technischen Gründen nicht in der Lage ist, die Grafik zu generieren, z.B. wegen unzureichender Rechenleistung, aufgrund der Softwareausstattung oder der Notwendigkeit eines Datenbankzugriffs.

Ein weiterer Vorteil wäre der Schutz des Programms, weil dieses auf einem zentralen Server vor der Verbreitung durch Dritte besser geschützt ist.

Schließlich erscheint eine Archivierung des Graphen in einem herkömmlichen Bildformat auf der Serverseite sinnvoll, weil dieser dann ohne spezielle Aufbereitung betrachtet und ggf. einfach in andere Anwendungen integriert werden kann. Ein Bild kann auch ohne Weiteres ausgedruckt werden.

Zentrale Programmausführung

Einen Vorteil stellt bei dieser Alternative die zentrale Programmausführung auf der Serverseite dar. Allgemein betrachtet bedeutet dies, dass die Software nicht auf vielen heterogenen Systemen zum Einsatz kommt, bei denen jeweils Konfliktpotential bestehen könnte, sondern nur auf einem einzigen System.

Im konkreten Fall würde aber theoretisch die Verwendung einer Browsersoftware vom darunter liegenden heterogenen System abstrahieren, so dass eine dezentrale Programmausführung keinen großen Nachteil darstellen sollte.

Leider können die Browserhersteller die strengen Vorgaben des W3 Konsortiums bezüglich einer Standardisierung der Browserfunktionalität nicht immer einhalten, wie der Autor aus Erfahrungen gelernt hat. Gerade bei der Ausführung von JavaScript kommt es oft zu einem unterschiedlichen Verhalten. Auch bei der Ausführung eines Programms durch ein browser- und plattformübergreifendes Plug-in, kann es zu Inkompatibilitäten auf verschiedenen Rechnern mit unterschiedlicher Softwareausstattung kommen (vgl. 3.2).

Insofern würde eine zentrale Programmausführung die Fehleranfälligkeit der Anwendung reduzieren, weil man dadurch eine Abhängigkeit von Dritten minimiert.

Allerdings kann eine zentrale Programmausführung auch einen Nachteil bezüglich der Netzwerklast haben, nämlich genau dann, wenn die mehrmalige Übertragung der Nutzdaten zwischen Client und Server teurer ist als die einmalige Übertragung eines Clientprogramms, sofern dieses eine Realisierungsmöglichkeit darstellt.

Auch das Risiko wird bei einer zentralen Programmausführung zentralisiert: Bei einem Programmausfall auf der Serverseite kann kein Client mehr die Anwendung nutzen. Bei einer dezentralen Programmausführung sind dagegen nur die Clients von der Nutzung der Anwendung ausgeschlossen, bei denen das Programm z.B. aufgrund der Softwareausstattung nicht funktioniert.

Aufgrund aktueller Web- und Anwendungsservertechnik kann die Synchronisierungsproblematik bzgl. der Clients, die bei einer zentralen Programmausführung von Natur aus gegeben ist, vernachlässigt werden, weil sich Web- oder Applikationsserver darum kümmern.

Asynchrone Kommunikation

Zwischen den Tätigkeiten *Eingabe* von Seiten des Benutzers und *Feedback* von Seiten des Servers, die bei dieser Anwendung aufeinander folgen sollten, gibt es im Anwendungsfall keine weiteren Tätigkeiten. So gesehen kann man sagen, dass eine asynchrone Kommunikation zwischen Client und Server keinen direkten Nutzen bringt.

Das Feedback, in Form von Bilddaten, eingebettet in ein HTML Dokument oder anderweitig formatiert, ist unteilbar und muss vollständig übertragen werden. Diese Übertragungszeit bildet die minimale Reaktionszeit der Anwendung auf Benutzereingaben. Hinzu kommt noch die Zeit, die der Browser benötigt um den Request zu senden und das HTML Dokument erneut in der Browserdarstellung aufzubauen.

Um den Benutzern das Gefühl zu geben, nicht auf das Feedback warten zu müssen, kann die Darstellung des Graphen in einem separaten Browserfenster oder Frame geladen werden. Dieses Laden erfolgt implizit asynchron, und somit würde das Webformular für neue Eingaben oder Korrekturen weiterhin zur Verfügung stehen. Erfolgt eine Eingabe während Feedback vom Server geladen wird, wird dieser Vorgang vom Browser abgebrochen und ein aktuelles Bild des Graphen vom Server angefordert.

Mittels *Ajax* (vgl. 5.3) kann das gleiche Verhalten realisiert werden. Zudem lässt sich damit die Darstellung des Graphen nahtlos in das HTML Dokument integrieren. Die Kosten für eine Datenübertragung sind dann etwas geringer, weil die Nutzdaten beispielsweise mittels XML oder JSON [JSON07] formatiert werden können und nicht in ein HTML Dokument eingebunden werden müssen. Jedoch ist diese Methode komplexer zu realisieren, weil auch die Serverseite an dieses Kommunikationsmodell explizit angepasst werden muss.

4.1.2 Ausführung durch ein Browser-Plug-in

- **Eingabe** wird durch ein kompiliertes Programm im Browser verarbeitet
- **Feedback** wird durch das gleiche Programm erzeugt

Bei dieser Alternative wird ein kompiliertes Programm beim ersten Aufruf der Webseite zum Browser des Benutzers übertragen und dann von einem Browser-Plug-In ausgeführt. Es sind auch ähnliche Lösungen vorstellbar, die nicht direkt vom Browser ausgeführt werden, wie z.B. bei Anwendung der *Java Web Start* Technologie [JWS07].

Mit der existierenden Appletversion wird in Abschnitt 3.2 schon ein Anschauungsbeispiel vorgestellt. Die beschriebenen Probleme lassen die Nachteile dieser Lösung erahnen.

Allerdings liegt hier eine konkrete Ausprägung dieser Realisierungsmöglichkeit vor. Es stellt sich die Frage, ob es bessere Vertreter dieser Methode gibt. Zum Beispiel die Technik *Flash* von der Firma Adobe/Macromedia sollte in Erwägung gezogen werden, bevor man sich ein Urteil bildet.

Theoretisch sollte diese Alternative in dem Maß browserunabhängig sein, wie es der Plug-in-Hersteller in den Produktinformationen angibt. Leider gibt es keine Garantie, dass dieses Ideal eingehalten wird.

Als Nachteil kann man also eine hohe Herstellerabhängigkeit notieren. Insbesondere kann man bei der Realisierung dieser Alternative abhängig von einer proprietären Programmiersprache oder eines Autorensystems sein.

Allerdings bleibt die Programmierung weitgehend auf die Clientseite beschränkt. Inwiefern die Serverseite angepasst werden muss, kann erst bei eingehender Betrachtung der Schnittstellen erörtert werden. Die einfache Schnittstelle des HTTP Protokolls (vgl. 3.3) kann jedenfalls nicht ohne weiteres genutzt werden, weil eine weitere Schnittstelle zwischen Plug-in und Browser zu beachten ist.

Nachdem das Programm kompiliert werden muss, entsteht dadurch ein höherer Wartungsaufwand. Bei einem Update der Plug-in-Software muss ggf. auch das Programm neu kompiliert werden. Es gibt nicht nur einen Quelltext, der für die Serverseite von einem Kundigen gepflegt und archiviert werden muss, sondern auch für die Clientseite existieren umfangreiche Entwicklungsdokumente.

Die Benutzer müssen falls das Plug-in nicht im Browser vorinstalliert ist, die Plug-in-Software von den Seiten des Herstellers herunterladen und installieren. Das ist ein zusätzlicher Aufwand für die Anwender, den es zu vermeiden gilt.

Sollte das zu realisierende Programm sehr groß werden, dann müssen die Anwender eine längere Downloadzeit in Kauf nehmen, und es kommt zu einem hohen Transfer volumen. Hier zeigen sich die Nachteile einer dezentralen Programmausführung, die bereits im Abschnitt 4.1.1 erläutert wurden.

Ist das zu übertragende Programm dagegen sehr klein, dann zeigen sich Vorteile, weil während der Programmausführung keine Daten zum Server übermittelt werden müssen. Erst nach Abschluss der Programmausführung werden die Nutzdaten an den Server gesendet. Aus diesem Grund ist auch eine Diskussion, ob eine asynchrone Kommunikation mit dem Server erfolgen sollte (vgl. 4.1.1) nicht notwendig.

4.1.3 Realisierung als Programmskript

- **Eingabe:** Ausführung eines Programmskripts durch den Browser
- **Feedback:** Ausführung eines Programmskripts durch den Browser

Bei dieser Realisierungsmöglichkeit wird sowohl die Eingabeverarbeitung, als auch die grafische Darstellung mittels eines JavaScript Programms auf der Clientseite realisiert. An dieser Stelle sei angemerkt, dass es auch noch die Programmiersprache VBScript gibt. Diese kann aber nur vom Internet Explorer interpretiert werden und wird daher nicht weiter betrachtet.

Zur Zeit der Programmausführung findet bei dieser Realisierungsalternative keine Datenübertragung statt, was sich positiv auf das Transfervolumen auswirkt. Man kann vermuten, dass im Gegensatz zu der vorangegangenen Alternative, die einfache Schnittstelle des HTTP Protokolls (vgl. 3.3) mittels JavaScript genutzt werden kann. Somit würde kein Programmieraufwand auf der Serverseite anfallen.

Anwendererfahrungen zeigen, dass JavaScript bzgl. einer Browserunabhängigkeit problematisch ist, und eine angepasste Programmierung für Browsersoftware unterschiedlicher Hersteller und Versionen notwendig wird. Weiterhin ist die Skriptprogrammierung fehleranfälliger im Vergleich zur Programmierung mit einer typisierten Programmiersprache.

Die Betrachtung einer asynchronen Kommunikation ist hier nicht notwendig, weil eine Beteiligung des Servers nur zur Abnahme der Nutzdaten nach vollständiger Eingabe und damit nach Abschluss der Programmausführung erfolgt.

Wie schon bei der Alternative im vorangegangenen Abschnitt muss hier evtl. eine neue Technik erlernt und angewendet werden. Bei der Realisierung als *Webformular mit Serverinteraktion* kann dagegen weitgehend die Technik genutzt werden, die bereits im Lehrbetrieb des Lehrgebiets vermittelt wird.

Bei der Reflexion der besprochenen Realisierungsalternativen kommt der Autor zu dem Schluss, dass erst nach einer genaueren Untersuchung von technischen Details eine Entscheidung für die eine oder andere Alternative gefällt werden kann.

Nachdem eine grafische Ausgabe im Feedbackverhalten der Anwendung gefordert ist, wird im nächsten Abschnitt auf die Grafikfähigkeiten der einzelnen Realisierungsmöglichkeiten eingegangen.

4.2 Die Grafikfähigkeiten im Allgemeinen

In diesem Abschnitt werden einige Realisierungsansätze bezüglich der Grafik besprochen, beginnend mit der einfachsten Art der Anzeige einer Grafik im Webbrowser, nämlich der Einbindung einer Bilddatei über das `` Tag in ein HTML Dokument.

Diese Bilder sind in einem herkömmlichen Grafikformat, wie zum Beispiel GIF [GIF90], JPEG [JPG92] oder PNG [PNG04] in einem Verzeichnis auf dem Webserver oder in einer Datenbank abgelegt.

Sie können aber auch dynamisch von einem Serverprozess auf eine Anfrage des Clients generiert werden. Daher würde diese Art der grafischen Anzeige bei der Realisierung als *Webformular mit Serverinteraktion* (vgl. 4.1.1) eingesetzt werden.

Die serverseitige Erzeugung der Bilddaten kann zum Beispiel bei einer Java Implementierung der Serverseite mit der *Java Media API* [JAV50] realisiert werden.

Grafiken können auch mittels der Formate SVG [SVG98] und VML [VML98] auf einer Webseite dargestellt werden. Dabei werden Vektorgraphiken in XML beschrieben. Zur Anzeige im Browser ist ein Browser-Plug-in erforderlich.

Während das `SVG` Format ein Standard nach dem W3 Konsortium ist, ist `VML` kein Standard. Grafiken in diesem Format sind aber in aktuellen Versionen des Internet Explorers ohne weitere Installation darstellbar. Im nächsten Abschnitt wird noch einmal auf das `VML` Format zurückgekommen, weil es für die Implementierung des *Editors für Kontrollflussgraphen* technisch eine tragende Rolle spielt. Für eine dynamische Generierung von Grafiken im `SVG` Format auf der Serverseite gibt es zum Beispiel die Bibliothek *Batik* der Apache Software Foundation [BAT07].

Die Grafikfähigkeiten von Plug-in-Programmen (vgl. 4.1.2) beruhen auf der verwendeten Plug-in-Technologie. Bei Java Applets wäre zum Beispiel mit der *Java AWT Bibliothek* eine Basis für clientseitige Grafikausgabe im Webbrowser verfügbar.

Ebenfalls durch ein Browser-Plug-in werden Flash Filme ausgeführt und können als gute Realisierungsmöglichkeit für eine grafische Ausgabe betrachtet werden. Das Flash-Plug-in ist im Gegensatz zu einem Plug-in für Java Applets derzeit häufiger in den Browsern der Internetnutzer (vor-) installiert - mit einer Verbreitung von über 98%, nach Angaben des Herstellers Macromedia/Adobe [ADO07]. Weiterhin wird auch keine Neukompilierung bei Updates der Plug-in-Software notwendig, weil Flash Programme weitgehend rückwärtskompatibel sind [FLA07].

Flash Filme sind inzwischen als Standard für komplexe interaktive grafische Online-Anwendungen, wie Animationen, Spiele, Werbebanner, etc. anzusehen. Nach Anwendererfahrungen des Autors mit mehreren Flash Filmen, ist dieser von der Eignung der Grafikfähigkeiten von Flash für diese Applikation überzeugt.

JavaScript Unterstützung ist in allen modernen Browsern unterschiedlicher Hersteller verfügbar. Bislang war browserübergreifend mit JavaScript jedoch nativ keine Grafikausgabe auf einer Webseite vorgesehen und Grafikfähigkeiten werden nur über Workarounds implementiert. So nutzt zum Beispiel Walter Zorn pixelgroße `DIV`-Layer zum Zeichnen von Vektorgrafiken und stellt mit der JavaScript Bibliothek *High Performance JavaScript Vector Graphics Library* eine populäre Möglichkeit für clientseitige Grafikausgabe mittels JavaScript zur Verfügung [ZOR07].

Andererseits gibt es seit einiger Zeit ein neues HTML Element, nämlich das Canvas-Element. Damit lassen sich mit Hilfe von JavaScript Grafiken auf einer Webseite zeichnen. Dieser Ansatz klingt aufgrund seiner Aktualität sehr interessant.

Die beiden zuletzt angesprochenen Möglichkeiten sind auf die Realisierung als Programmskript (vgl. 4.1.3) anwendbar und eine vom Autor entwickelte Testwebseite, in der sowohl die JavaScript Bibliothek von Walter Zorn als auch das Canvas-Element Verwendung findet, führt zu der Überzeugung, dass beide Methoden zur Realisierung der Anwendung geeignet sind. In der folgenden Abbildung ist ein Screenshot von der Grafik auf dieser Testwebseite dargestellt (Diese Darstellung wird nur mit Hilfe von JavaScript erzeugt)

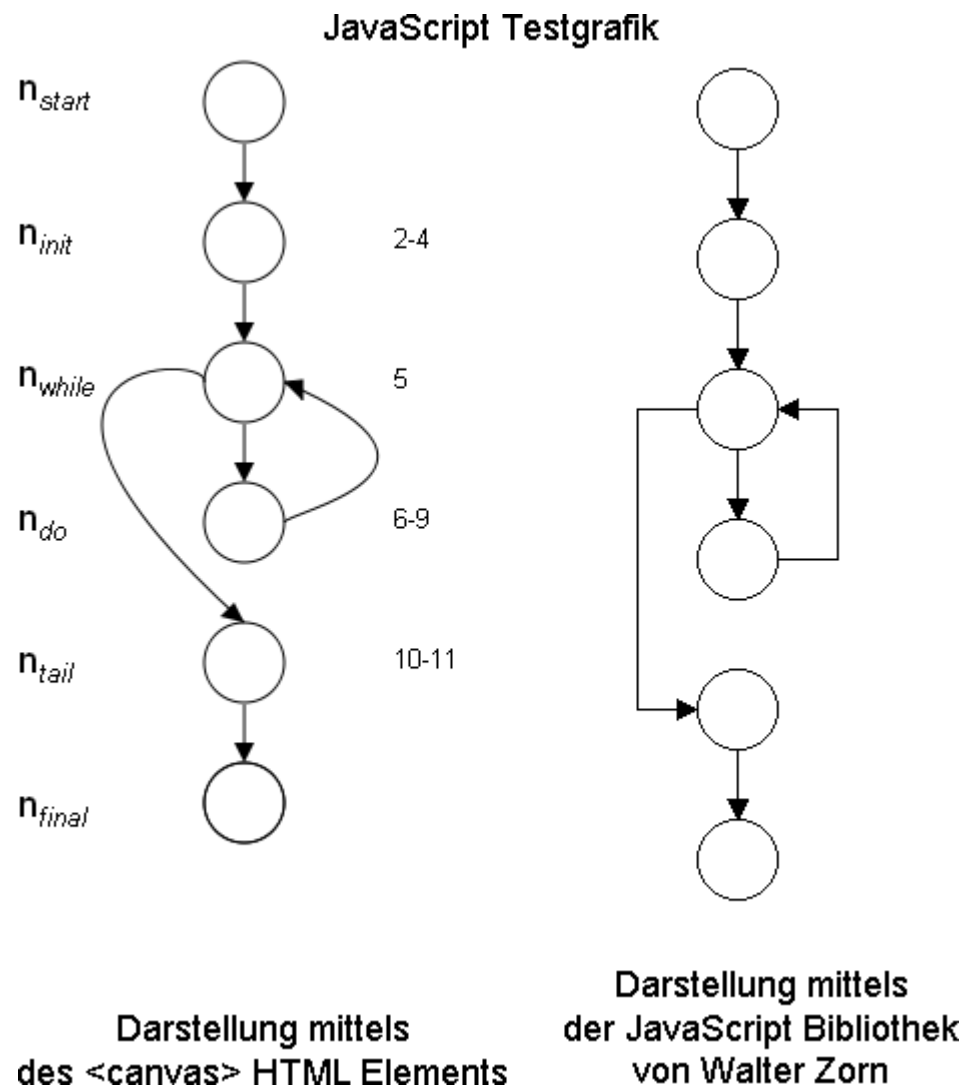


Abbildung 4: Gegenüberstellung von JavaScript Grafik

Weil man mit der JavaScript Bibliothek von Walter Zorn noch keine Kurven zeichnen kann, scheint die Verwendung des Canvas-Elements geeigneter. Für Walter Zorns Bibliothek spricht dagegen, dass diese auch mit sehr alten Browserversionen kompatibel ist. Im folgenden Abschnitt wird das Canvas-Element näher vorgestellt, welches die Grundlage für die nachfolgende Realisierung darstellt.

4.3 Das Canvas-Element im Speziellen

Das Canvas-Element ist ein neues Element der HTML, welches zuerst von der Firma Apple für die Anwendung *Mac OS Dashboard* eingeführt und später im Browser Safari implementiert wurde [WIK07]. Browser, die die *Gecko Engine* verwenden, wie zum Beispiel Firefox, unterstützen mittlerweile dieses Element auch [MDC07].

Ein Blick in die Dokumentation von Apple zu diesem HTML-Element zeigt die umfangreichen Grafikfähigkeiten [APP07]. Leider ist dieses HTML-Element noch kein Standard, der sich auch beim Browser Internet Explorer durchgesetzt hat.

Die Spezifikation der Auszeichnungssprache HTML wird von einer Arbeitsgruppe des *World Wide Web Konsortiums* (W3C) standardisiert und weiterentwickelt [W3C07]. Es gibt Repräsentanten von allen führenden Browserherstellern in dieser Gruppe.

Eine andere - weniger bekannte - Gruppe, die sich im Jahr 2004 nach einem W3C Workshop formierte, ist die *Web Hypertext Application Technology Working Group* (WHATWG). Diese Formation hat mit der Weiterentwicklung von HTML das gleiche Ziel, legt den Schwerpunkt aber auf Webapplikationen und will HTML in Zusammenarbeit mit JavaScript und CSS speziell für die Anwendungsentwicklung vorantreiben.

Das Dokument *HTML5 Working Draft* [HTM05] zeigt den aktuellen Entwicklungsstand. Unter anderem wird in diesem Text auch das Canvas-Element spezifiziert. Repräsentanten der Firma Microsoft gehören bislang dem Kern der WHATWG, der im Juli 2007 aus acht Mitgliedern bestand, offiziell nicht an.

So ist es nicht verwunderlich, dass das Canvas-Element vom Internet Explorer noch nicht unterstützt wird. Daher würde sich eigentlich diese Realisierungsmöglichkeit für die grafische Darstellung disqualifizieren, weil man Benutzer des Internet Explorers nicht ausgrenzen will.

Es gibt jedoch einen Workaround von Mitarbeitern der Firma Google, welcher Abhilfe schafft. Mit dem JavaScript Programm *Explorer Canvas* [EXC07], das unter einer Open Source Lizenz zur Verfügung steht, existiert ein Wrapper der die Canvas-Methodenaufrufe auf eine Realisierung mittels des `VML` Formats abbildet (vgl. 4.2). Mit Hilfe dieses Skripts ist auch eine Grafikausgabe für die Software Internet Explorer möglich.

Das Skript befindet sich zwar noch im Betastadium, die grundlegenden Grafikmethoden sind aber verfügbar, so dass die einfache Darstellung des Graphen problemlos gezeichnet werden kann. Die Dateigröße von 21 KB des unkomprimierten *Explorer Canvas* Skripts, welches beim Internet Explorer zusätzlich übertragen werden muss, fällt dabei kaum ins Gewicht.

Das Canvas-Element definiert eine Zeichenfläche, deren Größe mit den Attributen `width` und `height` festgelegt wird. Auf dieser Zeichenfläche kann man mittels JavaScript zeichnen. Damit die Zeichenfläche aus einem JavaScript Programm heraus referenziert werden kann, ist die Vergabe des Attributs `id` notwendig.

Es wird ein „Turtle Grafik Interface“ genutzt, welches den Grafikfähigkeiten der Programmiersprache Logo [LOG60] ähnelt. Das bedeutet, dass man einen Cursor auf einer virtuellen Zeichenfläche bewegt und Zeicheninstruktionen zuweist. In der folgenden Abbildung ist eine JavaScript Funktion als Beispiel angegeben, die eine einfache Linie zwischen zwei Punkten mit den Koordinaten (x_1, y_1) und (x_2, y_2) zeichnet:

```
function drawLine(x1,y1,x2,y2) {
    var canvas = document.getElementById("canvas");
    if (!canvas.getContext) return;
    var ctx = canvas.getContext("2d");
    ctx.beginPath();
    ctx.moveTo(x1,y1);
    ctx.lineTo(x2,y2);
    ctx.stroke();
}
```

Abbildung 5: JavaScript Anweisungen zum Zeichnen einer Linie

Um eine Zeichenfläche mit dem Canvas-Element in ein HTML Dokument einzubinden ist die folgende HTML Anweisung notwendig:

```
<canvas height="1010" width="240" id="canvas">
    Canvas wird nicht unterstützt;tz
</canvas>
```

Abbildung 6: HTML Tag zur Einbindung der Canvas-Zeichenfläche

Text zwischen dem Anfangs- und Endtag ist optional und wird nur vom Browser dargestellt, wenn dieser das Canvas-Element nicht unterstützen sollte. Die Attribute `height` und `width` können dynamisch verändert werden. So kann die Dimension der Zeichenfläche zur Laufzeit angepasst werden. Dieses Verhalten wird leider nicht von allen Browsern unterstützt. Aus diesem Grund konnte leider die `Zeichne Knoten Schaltfläche` (vgl. Abb. 3) des Prototyps in der endgültigen Version nicht umgesetzt werden.

Mit diesem technischen Know-how fällt eine Entscheidung bezüglich einer Realisierungsmöglichkeit leichter und wird im nächsten Abschnitt begründet.

4.4 Konzeptionelle Entscheidung

Die Abwägung der Vor- und Nachteile der Realisierungsmöglichkeiten aus Abschnitt 4.1 unter Berücksichtigung der Grafikfähigkeiten in Abschnitt 4.2 und der erfolgreiche Test des Canvas-Elements (vgl. 4.3) führen zu der konzeptionellen Entscheidung, dass eine Realisierung der Anwendung als Programmskript (vgl. 4.1.3) die beste Wahl ist.

Die Ausführung durch ein Browser-Plug-In (vgl. 4.1.2) macht zwar unter Verwendung der Flash Technik einen guten Eindruck, aber leider verfügt der Autor weder über die nötigen Programmierkenntnisse in der Programmiersprache ActionScript, noch steht ihm ein Autorenwerkzeug von der Firma Macromedia/Adobe zur Verfügung.

Die Flash-Technologie wird auch nicht am Lehrgebiet Software Engineering eingesetzt, so dass man von den Lehrgebietsmitarbeitern nicht erwarten kann, eine mit Flash realisierte Anwendung zu warten. Java Applets werden aufgrund der schlechten Erfahrung mit der existierenden Applet-Lösung von Seiten der Betreiber (vgl. 3.2) abgelehnt.

Die Realisierung als *Webformular mit Serverinteraktion* ist wegen der Beteiligung des Servers und einer Rücksichtnahme auf die bestehende serverseitige Implementierung vermutlich komplizierter zu implementieren.

Ein Test mit einer Testgrafik (siehe 4.3) mit verschiedener Browsersoftware unterschiedlicher Hersteller und Version, bekräftigt diese Entscheidung und zeigt das die Grafik mit dem Canvas-Element weitgehend browserkompatibel ist.

Für ältere Browser kann leicht eine Version ohne Grafik zur Verfügung gestellt werden, die automatisch von der Browsersoftware beim Laden der Webseite ausgeführt wird, falls das Canvas-Element nicht zur Verfügung steht. Diese Funktionalität ist mit Grundfunktionen von JavaScript bzw. HTML realisierbar.

Der auf Basis dieser konzeptionellen Entscheidung eingeschlagene Realisierungsweg wird im folgenden Kapitel dargestellt. Weiterhin werden wichtige Realisierungsdetails erläutert.

5. Realisierung

Es gibt wegen der Forderung nach absoluter Browserunabhängigkeit zwei Varianten der Anwendung. Zum einen eine interaktive Variante für moderne Browsersoftware, die eine Unterstützung von JavaScript durch den Browser verlangt.

Anwender, die JavaScript aus verschiedenen Gründen in ihrer Browsersoftware deaktiviert haben, wird eine einfache Formularversion (vgl. 3.2) ersatzweise präsentiert - jedoch mit einem modernen CSS-basierten Design. Eine Notiz macht hier den Benutzer darauf aufmerksam, dass bei Aktivierung von JavaScript eine Version mit einer interaktiven grafischen Darstellung zur Verfügung steht.

Bemerkenswert ist, dass die Auswahl der Variante automatisch durch den Browser erfolgt. Diese Funktionalität wird mit Hilfe von JavaScript und CSS auf die folgende Weise realisiert: Mittels einer CSS Eigenschaft können Teile eines HTML Dokuments ein- und ausgeblendet werden. Initial wird beim Laden des HTML Dokuments im Browser der interaktive Teil ausgeblendet. Erst wenn JavaScript Unterstützung vorhanden ist, wird dieser und ein beschreibender Text eingeblendet.

Man kann sagen, dass die interaktive Grafik nur als Zusatz in die existierende Formularversion integriert ist, was insbesondere bedeutet, dass selbst wenn kein Graph dargestellt werden kann, immer noch die Formularfassung der Anwendung angeboten wird. Die Daten des Graphen werden bei beiden Varianten mit dem gleichen Webformular zum Server übertragen.

Die Webseite der Anwendung wird mittels CSS ansprechend gestaltet. Für den Fall, dass ein Benutzer auf die Anwendung mit einem sehr alten Browser - der CSS noch nicht unterstützt - zugreift, wird ein Link auf die Vorgängerversion der Anwendung bereitgestellt. Dieser Link wird initial durch CSS verborgen und wird nur sichtbar, wenn CSS-Unterstützung nicht zur Verfügung steht. Obwohl selbst die Browsersoftware Netscape 4.7 mit der neuen Version zurechtkommt, wird auf diese Sicherheitsmaßnahme nicht verzichtet.

Der Anzeigemodus (vgl. 3.1) wird mit einem Pop-up-Browserfenster realisiert, auf das verlinkt wird und in dem eine grafische Darstellung des Kontrollflussgraphen abgebildet ist. Falls JavaScript nicht zur Verfügung steht, funktioniert der Link nicht und der Anwender muss sich mit der textuellen Repräsentation des Graphen zufrieden geben, welche im Elternfenster angezeigt wird.

Bevor auf wichtige Realisierungsdetails eingegangen wird, soll zunächst einleitend die Idee beschrieben werden, die hinter der Realisierung steckt, und wie diese Idee umgesetzt wird.

5.1 Eingeschlagener Realisierungsweg

Die Idee, die hinter der neuen Realisierung des Editors für Kontrollflussgraphen steckt, ist angelehnt an die in der Einleitung angesprochene *Live Search* Funktionalität. Die Eingabe eines Zeichens in ein *Live Search* Formularfeld bewirkt unmittelbar eine Änderung des Zustands und eine Ausgabe auf der Webseite. Feedback ohne Verzögerung trägt hier erheblich zum Bedienkomfort bei.

Wendet man dieses Prinzip auf den zu realisierenden Editor an, dann könnte man zu jedem Knoten des Graphen die Nummern von Nachfolgeknoten jeweils in ein einfaches Formularfeld eines HTML Formulars eingeben. Eine Anzeige des Graphen würde unmittelbar auf eine Eingabe hin aktualisiert.

Bezüglich des einzugebenden Graphen kann man feststellen, dass dieser aufgrund des Anwendungsbereichs nicht besonders komplex sein wird.

Die Knoten des relativ einfachen Graphen können in einer Vertikalen angeordnet werden. Auf diese Weise lassen sich die einzelnen Knoten gut beschriften und Daten wie Knotenname, zugeordnete Programmzeilen, etc. finden horizontal neben den Knoten Platz. Es wird angenommen, dass die Anzeige von maximal zwölf Knoten, so wie in der zuletzt eingesetzten Formularversion, ausreicht.

Das Layout der Formularversion (vgl. 3.2) wird auch für die interaktive Editorversion beibehalten. Das bedeutet, dass es die Spalten `Knotenname`, `Nachfolger`, `Zeile von` und `Zeile bis` gibt.

Relevant für die Realisierung der interaktiven Grafik ist nur die Spalte `Nachfolger`, in welcher zwölf Formularfelder zur Eingabe der Nummern der Nachfolgeknoten angeordnet werden. Durch die Formularfelder, die im HTML Dokument mittels des HTML Tags `<input>` hartcodiert sind, ist auch die Knotenzahl in der grafischen Darstellung beschränkt.

Um die interaktive Grafik zu realisieren, muss an geeigneter Stelle eine Zeichenfläche für die Darstellung des Graphen integriert werden. Nach der Realisierungsalternative in Abschnitt 4.1.3, die hier umgesetzt wird, handelt es sich bei dieser Zeichenfläche um eine rein clientseitige Anwendung, die mittels JavaScript realisiert und von der Browsersoftware ausgeführt wird.

Alle Daten, die zur Anzeige des Graphen notwendig sind, sind schon auf der Clientseite in den Formularfeldern vorhanden. Es besteht also kein Anlass den Server bei der Programmausführung einzubeziehen.

Das abschließende Übermitteln der Daten zum Server kann weiterhin über das HTML Formular erfolgen. Diese Überlegungen bilden die Grundlage für die im Folgenden beschriebenen Realisierungsdetails.

5.2 Integration der Zeichenfläche

Es werden zwei verschiedene Möglichkeiten betrachtet, wie die Zeichenfläche für den Graphen in die Webseite bzw. in das Webformular integriert werden kann. Nachdem eine Tabelle mit den Spalten `Name`, `Nachfolger`, `Zeile von` und `Zeile bis` vorgegeben ist (vgl. Abb. 3) liegt es nahe, die Zeichenfläche sowie die Formularfelder zur Eingabe der einzelnen Attribute des Graphen in einer HTML Tabelle anzuordnen.

Diese Vorgehensweise wird nachfolgend betrachtet: Im Anschluss sieht man, dass die Zeichenfläche auch mittels CSS Regeln (Cascading Style Sheets) integriert werden kann.

Die folgende Abbildung zeigt beispielhaft den Quelltext einer HTML Tabelle und dessen Ausgabe im Browser. Aus Platzgründen ist nur der Code dargestellt, der die erste Zeile der Tabelle implementiert. Es ist auch zu beachten, dass weitere Attribute der einzelnen Zellen gesetzt werden müssen, um eine korrekte Darstellung mit unterschiedlicher Browsersoftware zu erreichen, insbesondere die Attribute `width` und `height`.

```
<table width="385" border="1" cellspacing="0" cellpadding="5">
  <tr>
    <td>Name:</td>
    <td rowspan="13">CANVAS</td>
    <td>Nachfolger</td>
    <td>Zeile von:</td>
    <td>Zeile bis:</td>
  </tr>

  [...]

</table>
```

Abbildung 7: Beispielhafter Quellcode zur HTML Tabelle

Name:		Nachfolger:	Zeile von:	Zeile bis:
	CANVAS			

Abbildung 8: Ausgabe der Tabelle aus Abb.7 im Browser

Wie in Abbildung 7 und 8 ersichtlich, werden mittels des `rowspan` Attributs die Zeilen in der zweiten Spalte zu einer Zeile verschmolzen, so dass Platz für die Zeichenfläche `CANVAS` geschaffen wird. Formularfelder zur Eingabe der Attribute des Graphen finden jeweils in den anderen Zellen der Tabelle Platz. Obwohl das offensichtlich eine saubere Lösung ist, fiel zunächst die Entscheidung zu Gunsten einer Positionierung mittels CSS-Regeln, denn die Codierung des umfangreichen Quelltextes einer komplexen HTML Tabelle per Hand ist fehleranfällig. Es galt auch einen Bug in einer externen JavaScript Ressource zu beherrschen, der ein Zeichnen der Knoten erst ab einer bestimmten vertikalen Position auf der Zeichenfläche zuließ, so dass nach oben ein freier Platz beachtet werden musste. Der Bug wurde inzwischen von den Autoren des *Explorer Canvas* Skripts behoben.

Die Positionierung der Formularfelder und der `CANVAS` Zeichenfläche mittels CSS fiel dagegen wesentlich leichter. Zudem ist bei dieser Variante der HTML Quellcode viel kürzer und leichter lesbar, was zu einer besseren Wartbarkeit führt. Aus diesem Grund wurde bei der Implementierung eines ersten Prototyps der Anwendung diese Vorgehensweise angewandt. Beispielhaft wird an dieser Stelle mit der folgenden Abbildung gezeigt, wie sich Content-Blöcke mittels CSS anordnen lassen. Auf diese Art und Weise lassen sich auch Spalten einer Tabelle formatieren.

```
<html>
  <head>
    <style type="text/css">
      <!-- .spalte {
        float: left;
        margin: 20px;
        padding: 10px;
        background: #f4f4f4;
      } -->
    </style>
  </head>
  <body>
    <div class="spalte">
      Spalte 1
    </div>
    <div class="spalte">
      Spalte 2
    </div>
    <div class="spalte">
      Spalte 3
    </div>
    <div class="spalte">
      Spalte 4
    </div>
    <div class="spalte">
      Spalte 5
    </div>
  </body>
</html>
```

Abbildung 9: Quellcode zur CSS Positionierung

Im HTML Kommentar des Quelltextes in Abbildung 9 ist die notwendige CSS-Regel für die Formatierung einer Spalte angegeben.

Der HTML Code oben produziert die folgende Ausgabe in einem CSS kompatiblen Browser:



Abbildung 10: Anzeige zur CSS Positionierung

Bei diesem Vorgehen wird die `float` Eigenschaft, die eigentlich zur Darstellung eines Textumfluß dient, ausgenutzt. Zusammen mit den CSS-Eigenschaften `width` und `margin` lassen sich so Content-Blöcke nebeneinander anordnen. Diese Technik findet man in vielen modernen CSS-Layouts, die mehrere Spalten darstellen. Der interessierte Leser möge sich z.B. auf der Webseite *A list apart* [ALA07], die im Bereich Webdesign sehr populär ist, weitere Anregungen holen.

In der zweiten Spalte in Abb. 10 könnte man die Zeichenfläche positionieren und in den anderen Spalten könnte man die Formularfelder untereinander, abhängig von der Formularfeldhöhe und einem Abstand, der zum Beispiel durch das CSS-Attribut `margin` angegeben werden kann, anordnen. Zu beachten ist hierbei, dass sich die Formularfelder neben den entsprechenden Knoten auf der Zeichenfläche befinden sollen, so wie in Abb. 3.

Wie sich später gezeigt hat, führt dieses Vorgehen zu einem Problem. Die Formularfeldhöhe wird eigentlich durch die Schriftgröße bzw. Zeilenhöhe festgelegt. Leider weicht die Formularfeldhöhe bei gleicher Schriftgröße insbesondere beim Browser Camino auf dem Betriebssystem Mac/OS so stark ab, dass sich die Formularfelder optisch nicht mehr sauber den Knoten auf der Zeichenfläche zuordnen lassen.

Die Problematik Formularfelder und andere Eingabelemente browserübergreifend gleich aussehen zu lassen, wird auf einschlägigen Internetseiten bzgl. Webdesign diskutiert und Roger Johansson zeigt Screenshots verschiedener Eingabelemente, dargestellt mit unterschiedlicher Browsersoftware, und kommt zu dem Schluss, dass dieses Problem nicht lösbar ist [JOH07].

Aus diesem Grund wurde der CSS Ansatz wieder verworfen und doch eine HTML Tabelle angefertigt. Dabei gibt es das oben angesprochene Problem nicht, denn die Formularfelder können sich in einer Tabellenzelle in einem gewissen Maß vertikal ausdehnen, ohne dass die feste Höhe der Zelle verändert wird.

Insgesamt gesehen ist die Integration der Zeichenfläche nicht unproblematisch und Probleme zeigen sich im Detail. An dieser Stelle soll beispielhaft ein solches Detail angesprochen werden.

Der geneigte Leser mag sich bei der Betrachtung von Abb.8 die Frage stellen, warum die `CANVAS` Zelle bis in die Kopfzeile der Tabelle ausgedehnt ist und nicht erst in der zweiten Tabellenzeile ein entsprechender „Rowspan“ über die restlichen Zeilen eingefügt wird.

Wegen des oben angesprochenen Bugs im *Explorer Canvas* Skript konnten zunächst die Knoten auf der Zeichenfläche erst ein Stückchen weiter unten gezeichnet werden. Von oben musste es einen Abstand geben und dieser wurde gut mit der Überschrift in der Kopfzeile „ausgefüllt“. Inzwischen wurde, wie gesagt, dieser Bug behoben.

Die Modellierung wird aber trotzdem so beibehalten, weil sich sonst die Nummern der Knoten und der Eingabetext in den entsprechenden Formularfeldern nicht ohne weiteres auf einer Linie anordnen lassen. Die folgende Abbildung illustriert dieses Problem:



Abbildung 11: Die Positionierung der Formularfelder im Detail

Betrachten wir die erste Tabellenzeile in Abb. 11, und nehmen wir dabei an, dass die `CANVAS` Zelle nicht bis in die Kopfzeile reicht. Die beiden Spalten ganz rechts in Abb. 8 werden in Abb. 11 aus Platzgründen nicht dargestellt.

Anschaulich gesprochen kann dann der Graph nicht weiter nach oben geschoben werden, weil man schon am oberen Ende der Zeichenfläche angelangt ist. Reicht dagegen die Zeichenfläche bis in die Kopfzeile der Tabelle (vgl. Abb. 8) dann hat man auf der Zeichenfläche nach oben noch Platz, so dass Knotennummer und Formularfeld optimal nebeneinander platziert werden können.

Alternativ können auch die Formularfelder in den jeweiligen Tabellenzellen weiter nach unten geschoben werden, bspw. mittels eines transparenten Abstandhalterbildes oder über eine CSS Regel. Beides erscheint komplizierter und findet daher keine Anwendung.

Nachdem die Schriftgröße der Überschriften in der Kopfzeile mit einer Pixelangabe absolut festgelegt ist, und auch ein Zeilenumbruch durch ` `-Zeichen (Non Breaking Space) in den Überschriften ausgeschlossen werden kann, sollte ein „Zoomen“ der Webseite, bspw. im Browser Opera, keine Probleme bereiten. D.h. der Text der Kopfzeile bleibt im Verhältnis zu den Knoten und Formularfeldern gleich groß. Die Formularfelder werden nicht unverhältnismäßig nach unten geschoben.

Um die interaktive Zeichenfläche zu betreiben ist die Ausführung von JavaScript durch den Browser notwendig. Jedoch sollte es auch eine Version ohne JavaScript Unterstützung geben, da viele Internetnutzer in ihrem Browser JavaScript deaktiviert haben. Im nächsten Abschnitt wird zunächst gezeigt, wie CSS und JavaScript in die HTML Dokumente eingebunden werden. Anschließend wird die Alternative ohne JavaScript vorgestellt.

5.3 Einbindung von CSS und JavaScript in die HTML Dokumente

Eine Trennung von Inhalt und Layout mittels CSS Regeln führt zu einer wesentlich besseren Wartbarkeit der Webseiten. Um das Design zu verändern, genügt es eine globale CSS Ressource anzupassen oder auszutauschen. Auch bei dieser Anwendung sind Formatierungsregeln für Schrift, Farben und Rahmen in ein separates CSS File ausgelagert.

Ebenfalls separiert ist größtenteils der JavaScript Programmcode. Dieser könnte zwar über das `<script>` Tag direkt in die HTML Dokumente eingebunden werden. Wegen einer besseren Wartbarkeit der Anwendung wird aber im HTML Code nur eine Referenz auf verschiedene JavaScript Ressourcen angegeben. Nur ein sehr kleiner Teil der JavaScript Anweisungen wird direkt in die HTML Dokumente aufgenommen.

So gibt es also neben der HTML Dokumente weitere Dateien: Dateien mit der Dateinamenserweiterung `.css` beinhalten die oben angesprochenen CSS Formatierungsregeln, während Ressourcen mit der Endung `.js` den Code für das JavaScript Programm enthalten.

Weiterhin gibt es die Datei `excanvas.js`. Diese enthält das sogenannte *Explorer Canvas* Skript, welches einen Workaround für den Internet Explorer implementiert. Dieser kann den Graphen nur mit Hilfe dieser Ressource darstellen. Dieses File stammt von einer externen Quelle, vgl. 4.3.

Schließlich bilden drei HTML Dateien, diese seien beispielhaft mit `editor.html`, `display.html` und `devel.html` benannt, die Eintrittspunkte der Applikation für den Webbrowser. Die zuerst genannte Ressource ermöglicht den Editor- und die zweite den Anzeigemodus. Als dritte Variation gibt es noch einen speziellen Entwicklermodus, der notwendig wurde um eine schöne Darstellung der Kanten des Graphen zu finden. Dieser Modus startet bei Anzeige der Datei `devel.html`.

Bei Einsatz im *WebAssign* wird die Editier- und die Anzeigemöglichkeit in verschiedene HTML Dokumente integriert, die vom System mit einem vorgegebenen Namen referenziert werden. Die oben vorgestellten Ressourcendateien (JavaScript und CSS) werden dabei in einer Datenbank abgelegt.

Wie nun das CSS File und das JavaScript File in die HTML Dokumente eingebunden werden, sieht man in der nächsten Abbildung. Hier ist der Kopfteil der Datei `editor.html` dargestellt:

```
<head>
  <meta http-equiv="Content-Type" content="text/html" />
  <style type="text/css" media="all">
    <!-- @import url(css/editor.css); -->
  </style>
  <!--[if gte IE 6]>
    <script type="text/javascript"
      src="script/excanvas.js">
    </script>
  <![endif]-->
  <script src="script/Point.js" type="text/javascript"></script>
  <script src="script/Line.js" type="text/javascript"></script>
  <script src="script/Polygon.js" type="text/javascript"></script>
  <script src="script/BezierCurve.js" type="text/javascript"></script>
  <script src="script/ArrowHead.js" type="text/javascript"></script>
  <script src="script/Node.js" type="text/javascript"></script>
  <script src="script/Graph.js" type="text/javascript"></script>
  <script src="script/FormField.js" type="text/javascript"></script>
  <script src="script/Editor.js" type="text/javascript"></script>
  <script type="text/javascript">

  <!--

  var editor;

  function draw() {
    var canvas = document.getElementById("canvas");
    if (!canvas.getContext) return;
    var ctx = canvas.getContext('2d');
    editor = new Editor(ctx);
    editor.update();
  }

  function redraw() {
    if(!editor) return;
    editor.update();
  }

  -->
</script>
</head>
```

Abbildung 12: Kopfteil des HTML Dokuments editor.html

Das `<style>` Tag bindet die Formatierungsregeln ein, die im CSS File `editor.css` enthalten sind. Hierbei ist die Syntax mit dem `@import` Statement bewusst so gewählt, weil dadurch verhindert wird, dass ein älterer Browser, wie z.B. Netscape 4.7, das CSS File einbindet und evtl. fehlerhaft interpretiert. Eine Integration der Ressource über das `<link>` Tag hat sich diesbezgl. als problematisch herausgestellt.

Die `<script>` Tags integrieren das JavaScript Programm zur Realisierung des Editors. Die Implementierung folgt einem objektorientierten Entwurf und einzelne Klassen sind in separate Ressourcen-Dateien ausgelagert.

Interessant ist die Kondition `[if gte IE 6]`. Diese bedeutet, dass das oben angesprochene *Explorer Canvas* Skript nur in das Dokument eingebunden wird, wenn der Benutzer mit einer Version des Internet Explorers arbeitet, die jünger ist, als Version 6 oder dieser entspricht. Im anderen Fall wird der interaktive Modus nicht unterstützt und die Formularversion präsentiert.

Das dabei verwendete Konstrukt nennt sich *Conditional Comment* und ermöglicht eine für den Internet Explorer spezifische Technik, um die Version des Browsers zu erkennen. [MIC07]

Die zwei JavaScript Funktionen, die im Dokument direkt angegeben werden, sind mit `draw()` und `redraw()` bezeichnet. Die erste Funktion wird beim Laden des Dokuments aufgerufen und erzeugt die notwendigen Programm-Objekte. Die zuletzt genannte Methode wird aufgerufen, wenn eine Eingabe in ein Formularfeld der Spalte *Nachfolger* getätigt wird. Die Zeichenfläche wird daraufhin aktualisiert. Wie dabei die Ereignisbehandlung im Detail erfolgt, wird in Abschnitt 5.6.5 erläutert.

Es folgt nun die Beschreibung einer Option für Studierende, die mit einem Browser ohne JavaScript Unterstützung arbeiten. Dabei wird im Prinzip die Formularversion (vgl. 3.2) automatisch bereitgestellt.

5.4 Alternative ohne JavaScript Unterstützung

Viele Internetsurfer haben aus sicherheitstechnischen Gründen JavaScript in ihrem Browser deaktiviert. Das trifft wohl vor allem auf Benutzer des Internet Explorer zu, da gerade für diesen Browser besonders viele Sicherheitslücken - JavaScript betreffend - bekannt geworden sind. Es gibt eine Kritik dazu auf Wikipedia.de [CRI07].

Einerseits will man den Benutzer nicht zwingen, JavaScript für diese Anwendung zu aktivieren, andererseits könnte ein Benutzer auch nicht die notwendigen Rechte für diese Aktion auf Betriebssystemebene haben, weil er beispielsweise an einem Rechner in der Bibliothek arbeitet.

Aus diesen Gründen ist es angebracht, neben der interaktiven JavaScript-Version der Anwendung auch eine statische Version zur Verfügung zu stellen. Weiterhin soll diese statische Version automatisch vom Browser ausgeführt werden, wenn JavaScript nicht zur Verfügung steht. Eine Notiz kann den Anwender darauf aufmerksam machen, dass bei Aktivierung von JavaScript eine interaktive Version ausgeführt wird.

Die statische Version entspricht im Wesentlichen der existierenden Formularversion der Anwendung, vgl. 3.2.

Zur Orientierung wird neben jedem Eingabefeld für die Nachfolgeknoten ein Knotensymbol mit Nummer angezeigt. Die folgende Abbildung zeigt einen Screenshot:

Eingabeformular

Zeichnen Sie den Kontrollflußgraphen am besten erst auf Papier. Numerieren Sie anschließend die Knoten in der Reihenfolge durch, die durch die Zuordnung zum Programmtext gegeben ist. Tragen Sie in den Formularfeldern zu jedem Knoten den Knotenbezeichner (z.B. `n_while`), die Liste der direkten Nachfolgeknoten (z.B. 3, 4) in aufsteigend sortierter Reihenfolge sowie die zum Knoten gehörenden Programmtextzeilen ein! Sie müssen nicht alle 12 Zeilen benutzen. (Die voreingestellten Werte sind falsch und dienen nur als Beispiel.)

Aktivieren Sie JavaScript und/oder arbeiten Sie mit einem der folgenden Browser IE 6.0+, FF 1.5+, Safari 2.0+, Opera 9.0+ für eine interaktive Eingabemöglichkeit...

	Name:	Knoten:	Nachfolger:	Zeile von:	Zeile bis:
1.	<input type="text"/>	1	2	<input type="text"/>	<input type="text"/>
2.	<input type="text"/>	2	<input type="text"/>	<input type="text"/>	<input type="text"/>
3.	n_while	3	3,4	<input type="text"/>	<input type="text"/>
4.	<input type="text"/>	4	<input type="text"/>	<input type="text"/>	<input type="text"/>

Abbildung 13: Screenshot von der neuen Formularversion

Bis auf die fehlende Darstellung von Kanten sieht die statische Version genauso aus wie die interaktive Version. Es wird bei beiden Alternativen das gleiche HTML File und CSS File verwendet.

Die Umschaltung von der statischen Anzeige des Eingabeformulars in einen interaktiven Editor bei JavaScript Unterstützung wird mittels CSS und der `display` Eigenschaft erreicht.

Die Elemente für die statische und die interaktive Version stehen dabei in einem HTML Dokument und werden - je nachdem ob JavaScript vorhanden ist - mit der `display` Eigenschaft ein- oder ausgeblendet.

Das heißt, dass es zwei Sektionen im HTML Dokument gibt, von denen immer nur eine zu einem Zeitpunkt im Browser dargestellt wird. Diese Sektionen sind über die Identifikatoren `interactive` und `static` referenzierbar - für die interaktive Version und statische Version der Anwendung respektive.

```
<div id="interactive">
  <h2>Interaktive Eingabe</h2>
  Erzeugen Sie mit der Schaltfläche <strong>Zeichne Knoten</strong>
  neue Knoten und bringen Sie Kanten an, indem Sie in die
  Eingabefelder in der Spalte Nachfolgeknoten die Nummern der Nach-
  folgeknoten angeben...

  [...]

</div>
<div id="static">
  <h2>Eingabeformular</h2>
  Zeichnen Sie den Kontrollflussgraphen am besten erst auf Papier.
  Numerieren Sie anschließend die Knoten in der Reihenfolge durch...

  [...]

</div>
```

Abbildung 14: HTML Code bzgl. statischer und interaktiver Version

Abb. 14 zeigt beispielhaft einen Ausschnitt aus dem Quelltext zum Dokument `editor.html` mit den Sektionen `interactive` und `static`. Zunächst sind nur die Elemente der statischen Version sichtbar. Zusätzliche Texte und Schaltflächen der interaktiven Version erhalten in ihrer Stil-Definition im CSS File die Eigenschaft `display:none` und werden dadurch initial nicht angezeigt.

```
#interactive {
  display: none;
}
```

Abbildung 15: CSS Formatierungsregel bezüglich der interaktiven Version

Abb. 15 zeigt eine CSS-Klasse, die für die initiale Ausblendung der Sektion `interactive` beim Laden des HTML Dokuments im Browser sorgt.

Erst bei Verfügung von JavaScript und damit der Möglichkeit die interaktive Version auszuführen wird mittels JavaScript die Eigenschaft `display:none` in die Eigenschaft `display:block` überführt, so dass die Sektion `interactive` im HTML Dokument eingeblendet wird.

Ermöglicht wird diese Vorgehensweise durch das DOM (Document Object Model). Das DOM ist ein Standard des W3 Konsortium [DOM07] und regelt wie die einzelnen HTML Elemente einer Webseite, das sind zum Beispiel Tabellen, Texte, Bilder, Absätze, etc. in Beziehung zueinander stehen. Aktuelle Browser implementieren diesen Standard.

Es wird dabei eine baumartige Objekthierarchie definiert, mit einem Objekt `document` als Wurzel. Über dieses Wurzelement können alle Elemente der Webseite angesprochen und mittels JavaScript verändert werden. Es ist sogar möglich die einzelnen Elemente in diesem Baum umzuhängen, zu löschen, ein neues Element dynamisch zu erzeugen und an einer Stelle im Baum anzubringen.

Im Laufe der Zeit hat sich dieser Standard weiterentwickelt, und man unterscheidet *DOM Level 0*, *DOM Level 1* und *DOM Level 2*. In der letzten Version des Standards sind alle oben genannten Aktionen möglich. Jedes Element einer Webseite wird dabei als eigenständiges Objekt betrachtet.

In der folgenden Abbildung ist das HTML Dokument `editor.html` als *DOM Level 2* Baum vereinfacht dargestellt:

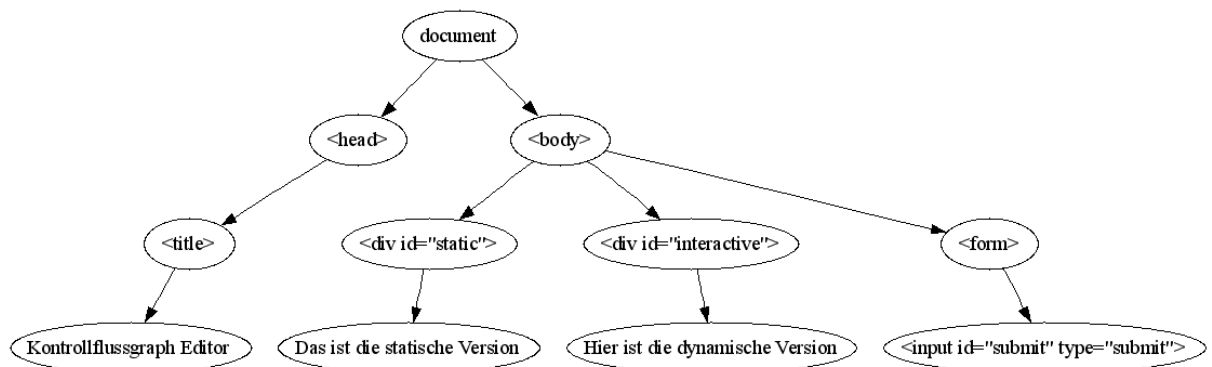


Abbildung 16: DOM Baum des Dokuments `editor.html` (vereinfacht)

Z.B. kann mittels JavaScript über die `getElementById` Methode des `document` Objekts das `div` Element mit dem Identifikator `static` in Erfahrung gebracht und manipuliert werden.

Beispielhaft ist in der folgenden Abbildung ein Codestück aus dem JavaScript Programm dargestellt, das auf diese Funktionalität zurückgreift und mit Hilfe des DOM das Ein- und Ausblenden der Elemente für die interaktive bzw. statische Version implementiert.

```
function renderInteractiveText() {  
    // blendet den HTML Text fuer die statische Version aus  
    // und den Text fuer die interaktive Version ein  
    document.getElementById("static").style.display="none"  
    document.getElementById("interactive").style.display="block"  
}
```

Abbildung 17: JavaScript zum Einblenden der interaktiven Elemente

Realisierungstechnisch ist das Vorteilhaft bei dieser Lösung, dass sich sämtlicher HTML Code der beiden Sektionen `static` und `interactive` in einer `.html` Resource befindet, was die Wartung der Applikation erleichtert.

Alternativ zum Ein- und Ausblenden der HTML Elemente kann auch direkt einem HTML Tag ein neuer Inhalt zugewiesen werden. Dazu kann bspw. die JavaScript Eigenschaft `innerHTML` verwendet werden.

Im folgenden Beispiel wird gezeigt, wie der Sektion im HTML Dokument mit dem Identifikator `images` ein neuer Inhalt zugewiesen wird. Diese Sektion stellt ein Bild der Knoten für die statische Version dar. Zunächst findet sich der relevante Abschnitt in der folgenden Weise im Quelltext von `editor.html`:

```
[...]  
<div id="images">  
    <h3>Knoten:</h3>  
      
</div>  
[...]
```

Abbildung 18: HTML Quelltext der ein statisches Bild der Knoten einbindet

Mit der JavaScript Anweisung in Abb. 19 kann der HTML Code innerhalb des `<div>` Tags aus Abb. 18 gelöscht werden. Dazu wird dem Tag der leere String als Inhalt zugewiesen.

```
document.getElementById("images").innerHTML = "";
```

Abbildung 19: Löschen des Inhalts eines HTML Tags.

Die `document.getElementById` Methode findet auch Anwendung, wenn es darum geht die Eingaben in den Formularfeldern auszulesen (vgl. 5.6.4).

5.6 Realisierung der Interaktiven Grafik

Der Algorithmus bezüglich der interaktiven Grafik gliedert sich in drei Teile, die hintereinander ausgeführt werden:

1. **Eingabe:** Auslesen der Eingabewerte aus den Formularfeldern in der Spalte `Nachfolger` unmittelbar nach einem Tastaturanschlag. Details findet man in Abschnitt 5.6.4.
2. **Ordnen:** Jeder Eingabewert repräsentiert eine Kante des Graphen. Um eine ansprechende Darstellung zu erhalten, wird jeder Kante ein fester Abstand zu den Knoten zugeordnet. Dies geschieht durch einen Layoutalgorithmus, der in Abschnitt 5.6.3 erläutert wird.
3. **Zeichnen:** Die Knoten liegen äquidistant auf einer Vertikalen und werden als einfacher Kreis mit einer Nummer gezeichnet (Diese Zeichnung ist trivial und wird im Folgenden nicht weiter beschrieben). Eine Kante wird als Bezierkurve zwischen zwei Knotensymbolen gezeichnet. Weiterhin muss beachtet werden, dass sich Pfeilspitzen an den Kurvenverlauf der Kante anpassen.

Um den Algorithmus im Detail zu erklären, soll mit Punkt 3 begonnen werden.

5.6.1 Zeichnen der Kanten

Eine Kante des Kontrollflussgraphen wird in dieser Arbeit als Bezierkurve gezeichnet, insbesondere wird eine *kubische Bezierkurve* verwendet. Diese ist charakterisiert durch einen Start- und Endpunkt sowie zwei Kontrollpunkte. Diese vier Punkte spannen das so genannte Kontrollpolygon auf. Die Kurve verläuft nun innerhalb der konvexen Hülle des Kontrollpolygons. Eine Erklärung zum Thema Bezierkurven findet der interessierte Leser z.B. auf Wikipedia [BEZ07]. Die folgende Abbildung stammt von dieser Quelle, wurde leicht angepasst und veranschaulicht die Konstruktion einer *kubischen Bezierkurve*. Die Punkte `START`, `CONTROL_1`, `CONTROL_2` und `END` bilden dabei das Kontrollpolygon. Die anderen Punkte sowie die Konstante t brauchen in dieser Arbeit nicht berücksichtigt werden, weil sich die Implementierung des Canvas-Elements (vgl. 4.3) um diese Details kümmert.

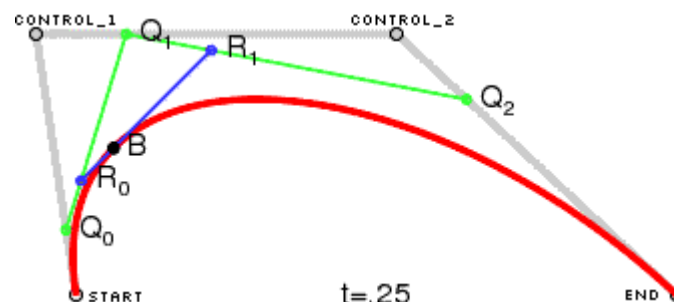


Abbildung 20: Konstruktion einer *kubischen Bezierkurve*

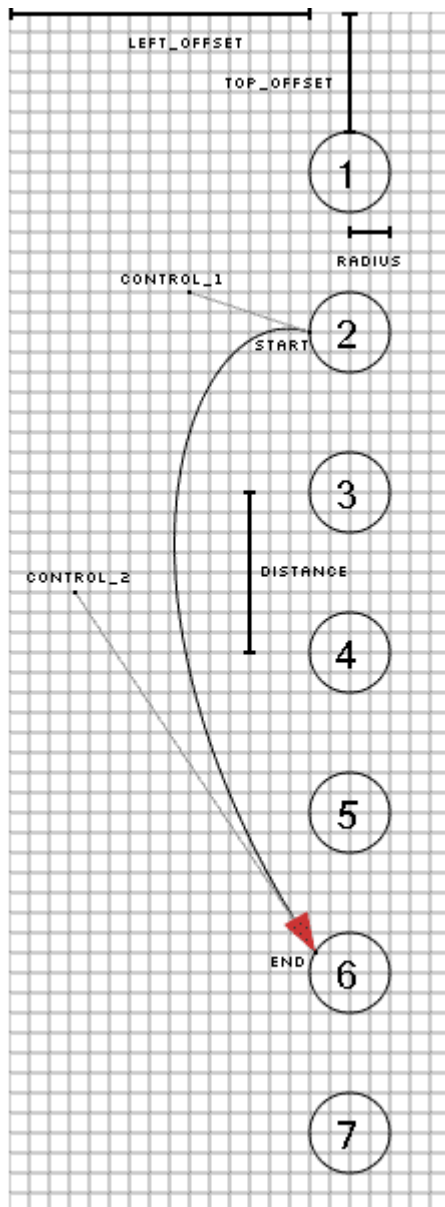


Abbildung 21: Eine Kante als Bezierkurve im Kontrollflussgraphen

Abb. 21 zeigt die Anwendung der Bezierkurve im Kontrollflussgraphen. Wie in Abb. 20 bestimmen auch hier die Punkte `START`, `CONTROL_1`, `CONTROL_2` und `END` den Kurvenverlauf.

Mit Hilfe von vier Konstanten lässt sich die Position der oben genannten Punkte berechnen:

1. `LEFT_OFFSET` beschreibt den Abstand des Graphen vom linken Rand der Zeichenfläche, gemessen am linken Rand des ersten Knotens.
2. `TOP_OFFSET` beschreibt den Abstand des Graphen vom oberen Rand der Zeichenfläche, gemessen an der Oberkante des ersten Knotens.
3. `RADIUS` beschreibt den Radius eines Knotensymbols.
4. `DISTANCE` beschreibt den Abstand zweier benachbarter Knotensymbole.

Weiterhin ist die Nummer des Startknotens und die Nummer des Endknotens der Kante für die Berechnung der oben genannten Punkte notwendig. Der Ursprung eines Koordinatensystems sei in der linken oberen Ecke der Abbildung vorhanden mit den Koordinaten $(0, 0)$.

Im folgenden ist beispielhaft JavaScript Quellcode für die Berechnung der Kante in Abb. 21 angegeben. Dieser Code ist im Vergleich zur tatsächlichen Implementierung vereinfacht dargestellt.

Die Methode `clone()` erzeugt dabei eine Kopie eines Punktes, während die Methode `translate(x, y)` einen Punkt um x Pixel horizontal und um y Pixel vertikal verschiebt.

Die Variable `level` bezeichnet einen zusätzlichen Faktor, der - anschaulich gesprochen - bestimmt, wie weit die Kante von den Knoten entfernt liegt. Auf diesen Faktor wird noch einmal zurückgekommen, wenn der Layoutalgorithmus besprochen wird.

```
// JavaScript Anweisungen zum Zeichnen der Kante aus Abb. 21
var i = 1; // Nummer des Startknotens - 1
var j = 5; // Nummer des Endknotens - 1
var r = RADIUS; // Radius eines Knotensymbols

var level = 5; // beispielhaft festgelegt

// bestimme die Punkte START und END aus Abb. 21
// Der Konstruktor der Klasse Point hat die X-Koordinate und
// Y-Koordinate des zu erzeugenden Punktes als Parameter.
var start = new Point(LEFT_OFFSET, i*DISTANCE+TOP_OFFSET+r);
var end = new Point(LEFT_OFFSET+r-r/2*1.7, j*DISTANCE+TOP_OFFSET+r/2);

// bestimme den Punkt CONTROL_2 aus Abb. 21
var control2 = end.clone();
var control2.translate(1.2*-r*level, 1.8*-r*level);

// bestimme den Punkt CONTROL_1 aus Abb. 21
var control1 = start.clone();
var control1.translate(-0.6*r*level, -0.2*r*level);

// zeichne Bezierkurve mittels des Canvas-Elements
// Das Objekt ctx repraesentiert die Zeichenflaeche und
// wird von der Browsersoftware bzw. vom 'Explorer Canvas'
// Skript im Fall des IE implementiert, vgl. Abschnitt 4.3

ctx.bezierCurveTo( control1.x,
                  control1.y,
                  control2.x,
                  control2.y,
                  end.x,
                  end.y      );
```

Abbildung 22: JavaScript Anweisungen zum Zeichnen der Kante aus Abb. 21

Entscheidend für den Kurvenverlauf sind die unterstrichenen Faktoren. Diese werden im Folgenden mit *Kantenparameter* bezeichnet. Diese Bezeichnung ist willkürlich gewählt.

Die *Kantenparameter* wurden mittels eines speziellen Entwicklermodus der Anwendung gefunden. Dieser Modus ist eine interaktive Anwendung, die in ein gesondertes HTML Dokument (`devel.html`) eingebettet ist. Die JavaScript Ressourcen des *Editors für Kontrollflussgraphen* werden dabei wiederverwendet. Im Folgenden ist ein Bildschirmausschnitt dieser Variante der Applikation dargestellt:

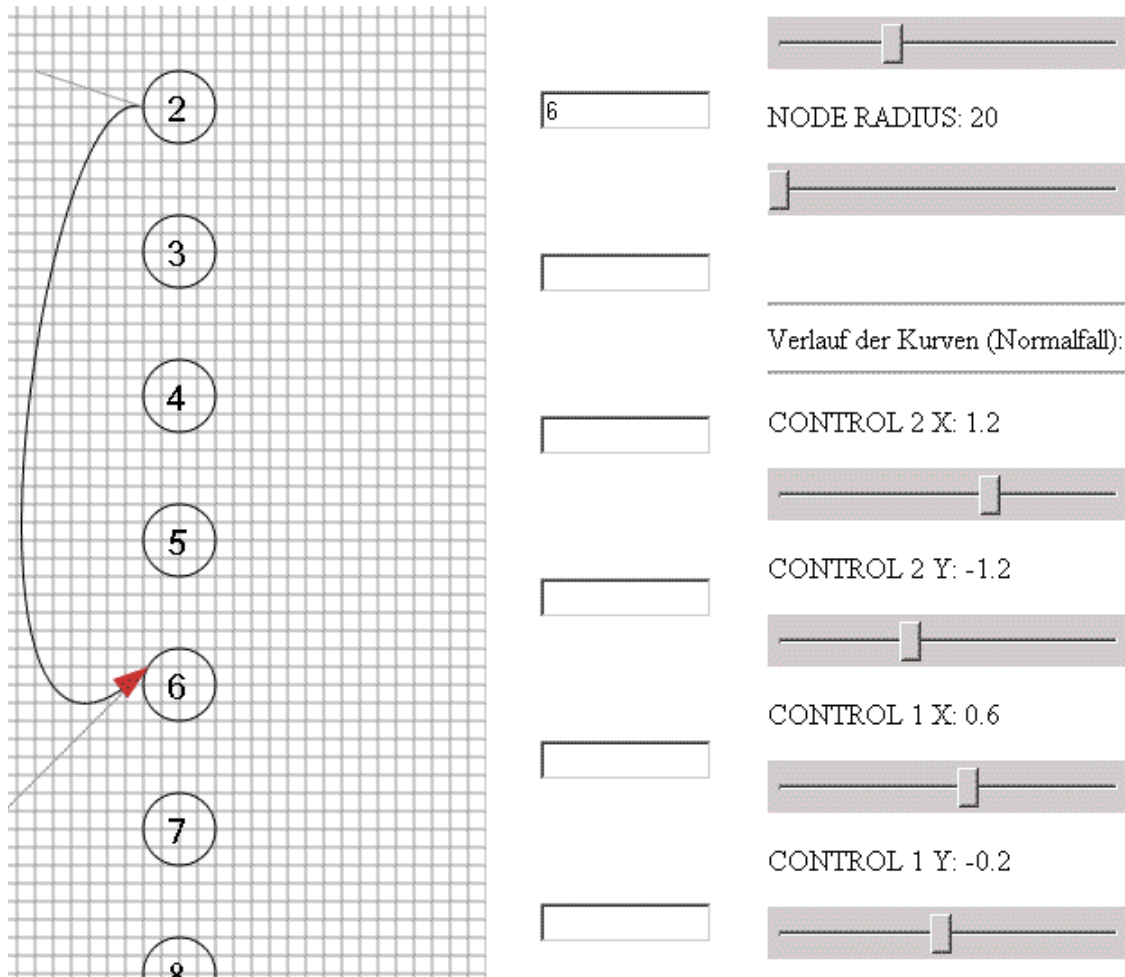


Abbildung 23: Screenshot des Entwicklermodus der Applikation

Wie man in Abb. 23 erkennen kann, lassen sich die *Kantenparameter* mittels Schieberegler einstellen. Der Kurvenverlauf verändert sich unverzüglich bei Ziehen eines Schiebereglers mit der Maus.

In Abb. 23 ist fast die gleiche Kurve wie in Abb. 21 dargestellt, lediglich ein *Kantenparameter* ist im Vergleich zu Abb. 22 verändert. Statt dem Faktor 1.8 findet man hier bei der Berechnung der Y-Koordinate des Punktes `CONTROL_2` den Faktor -1.2 , vgl. Wert des 4. Schiebereglers (von oben). Dieses Beispiel soll an dieser Stelle nur die Bedeutung der *Kantenparameter* für den Verlauf der Kurve veranschaulichen.

Neben nach unten gerichteten Kanten zwischen zwei Knoten, die keine Nachbarn sind (Fall a), werden noch die folgenden Fälle beim Zeichnen der Kanten berücksichtigt:

- b) Nach oben gerichtete Kanten zwischen zwei Knoten, die keine Nachbarn sind;
- c) Nach oben gerichtete Kanten zwischen zwei Knoten, die Nachbarn sind;
- d) Nach unten gerichtete Kanten zwischen zwei Knoten, die Nachbarn sind;
- e) Reflexive Kanten;

Im Graphen der folgenden Abbildung kommen alle oben genannten Fälle vor:

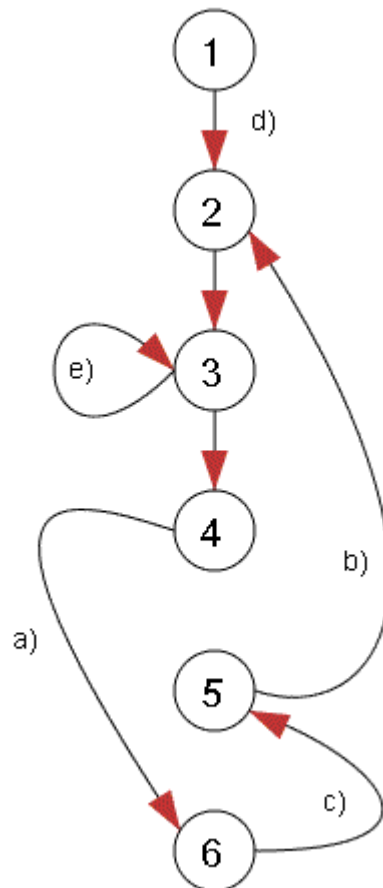


Abbildung 24: Graph mit allen möglichen Kantentypen

Die Fälle b) und c) können wie Fall a) behandelt werden. Lediglich die geänderte Richtung der Kante ist zu beachten. Bei Fall c) werden zusätzlich noch *Kantenparameter* mit geringfügig anderen Werten eingesetzt, um einen ansprechenden Kurvenverlauf zu erzielen.

Wie man auch in Abb. 24 sieht, wird im Fall a), b) und c) festgelegt, dass nach unten gerichtete Kanten auf der linken Seite des Graphen und Kanten, die nach oben verlaufen, auf der rechten Seite angebracht sind.

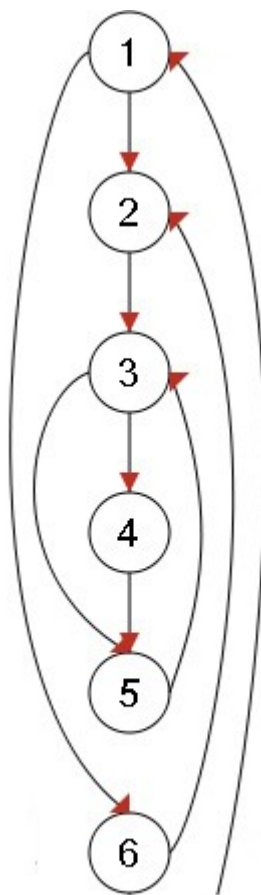
Die Realisierung von Fall d) ist trivial, weil nur eine Gerade zwischen zwei Knotensymbolen und eine einfache Pfeilspitze gezeichnet werden muss.

Im Fall der reflexive Kante (Fall e), entspricht der Punkt `END` dem Punkt `START`. Außerdem wird eine reflexive Kante auf der linken Seite des Graphen angebracht, um optisch nicht mit Fall c) zu kollidieren. Angepasste Werte der *Kantenparameter* werden auch in diesem Fall verwendet.

Die Pfeilspitzen müssen sich an den Verlauf der Kurve anpassen. Wie dieses Verhalten realisiert wird, ist Thema des nächsten Abschnitts.

5.6.2 Zeichnen der Pfeilspitzen

Eine Kante, die weiter von den Knoten entfernt liegt, geht steiler ab und kommt steiler an, als eine Kante, die näher bei den Knoten liegt. Diese Aussage lässt sich anhand der nächsten Abbildung einsehen. Hierbei sind die Pfeilspitzen als diagonale Unterteilung eines Maßstabskästchens modelliert, was eine sehr einfache Realisierungsmöglichkeit darstellt.



[...]

Wie man auf der linken Seite in Abbildung 25 sehen kann, sieht die Pfeilspitze der Kante, die bei Knoten 1 ankommt, noch relativ schön aus.

Die Kante, die bei Knoten 2 ankommt, verläuft ein wenig flacher und trifft deshalb die Grundlinie des Dreiecks, welches die Pfeilspitze symbolisiert, nicht mehr genau in der Mitte.

Nicht mehr akzeptabel ist die Darstellung der Pfeilspitze der Kante, die bei Knoten 3 ankommt.

Um dieses Darstellungsproblem zu lösen, müssen sich die Pfeilspitzen an den Kurvenverlauf anpassen.

Anhand der folgenden Abbildung soll gezeigt werden, wie dieses Verhalten im Kontrollflussgrapheditor erreicht wird.

Abbildung 25: Einfache Modellierung der Pfeilspitzen.

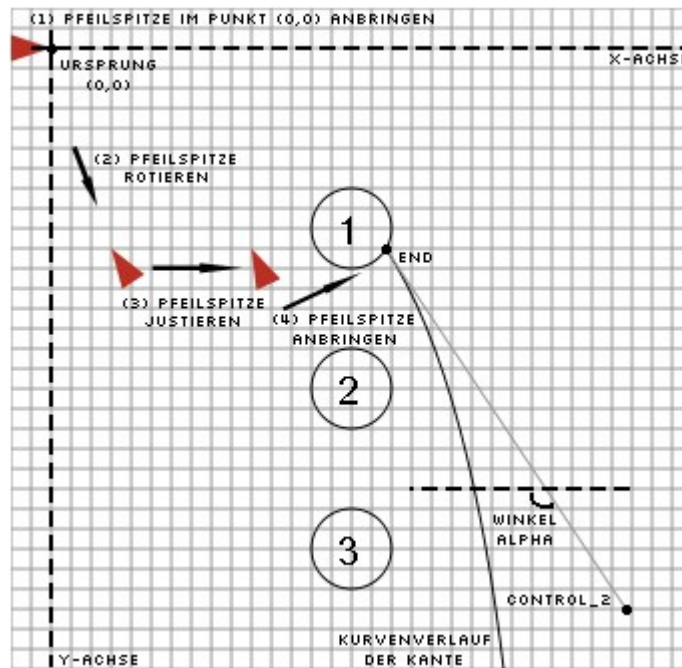


Abbildung 26: Konstruktion der Pfeilspitzen

Ein Dreieck wird zunächst am Ursprung angebracht, vgl. Schritt (1) in Abb. 26. Die Spitze zeigt dabei auf den Punkt $(0, 0)$.

Als nächstes wird in Schritt (2) das Dreieck um dessen Spitze rotiert. Und zwar so, dass es als Pfeilspitze auf die Linie passen würde, die durch die Punkte `CONTROL_2` und `END` gegeben ist. Um den Winkel der Rotation zu bestimmen, wird der Winkel `ALPHA` mit Hilfe der Koordinaten der Punkte `CONTROL2` und `END` berechnet. Hierzu wird die Funktion `Math.atan2` (Arcustangens) der JavaScript Standardbibliothek verwendet. Details dazu findet der interessierte Leser im Quelltext der Applikation.

Die tatsächliche Position der Spitze des Dreiecks hat sich bisher nicht verändert. Dieser Sachverhalt kann nur in Abb. 26 nicht gut dargestellt werden, deshalb befindet sich in der Abbildung das Dreieck bei Schritt (2) und (3) an einer anderen Stelle.

Die Pfeilspitze wird nach der Rotation in Schritt (2) nicht optimal zum Kurvenverlauf der Kante passen, wie man sich vorstellen kann und auch in der Abbildung sieht. Deshalb ist es notwendig eine kleine Justierung „per Hand“ vorzunehmen. Auch hierbei wird wieder mit Hilfe des Entwicklermodus der Anwendung ein *Kantenparameter* bestimmt, welcher auf die Y-Koordinate eines Duplikates des Punktes `CONTROL_2` - abhängig vom Abstand von Start- und Zielknoten der Kante und abhängig vom Abstand der Kante von den Knoten - als Faktor einwirkt.

Den JavaScript Quelltext für dieses Vorgehen findet man in der nächsten Abbildung. Die Variable `level` bezeichnet dabei einen zusätzlichen Faktor, der - bildlich gesprochen - bestimmt, wie weit die Kante von den Knoten entfernt liegt.

Auf diesen Faktor wird im nächsten Abschnitt zurückgekommen, wenn der Layoutalgorithmus besprochen wird.

```
// Quellcode zum Justieren der Pfeilspitzen
var point = control2.clone(); // Eine Kopie von CONTROL_2 anlegen
var r = RADIUS; // Abk. Radius eines Knotensymbols

// 'dir' unterscheidet zwischen nach unten und nach oben gerichteter
// Kante; i ist die Nummer des Startknotens, j die des Zielknotens
var dir = j<=i ? 1 : -1;

// dx ist die Anzahl der Knoten, die zwischen i und j liegen
var dx = Math.abs(i-j);

// die Y-Koordinate des Punktes CONTROL_2 wird in der Kopie
// abhängig von dx, level und dem Kantenparameter 2.7 versetzt
point.translate(0, dx*(r/4-level/2)*dir*2.7);
// eine „Kontrolllinie“ wird erzeugt. end entspricht dem Endpunkt der
// Bezierkurve
var arrow_line = new Line(point, end);

// die Pfeilspitze wird an der unsichtbaren 'Kontrolllinie'
// angebracht; das entspricht Schritt (4) in Abb. 26
arrowHead.putOn(arrow_line);
```

Abbildung 27: JavaScript Code zum Justieren der Pfeilspitzen

In Abb. 27 ist der *Kantenparameter* unterstrichen. An dieser Stelle möchte der Autor anmerken, dass die *Kantenparameter* nicht „hartcodiert“ im Quelltext der Implementierung vorliegen, sondern an zentraler Stelle verändert werden können.

In der folgenden Abbildung ist ein Bildschirmausschnitt des relevanten Teils zum Justieren der Pfeilspitzen im Entwicklermodus dargestellt, d.h. der Schieberegler mit dem sich der *Kantenparameter* einstellen lässt. Der Leser sei an dieser Stelle ange-regt, die umfangreichen Einstellungsmöglichkeiten des Entwicklermodus selbst ausprobieren.

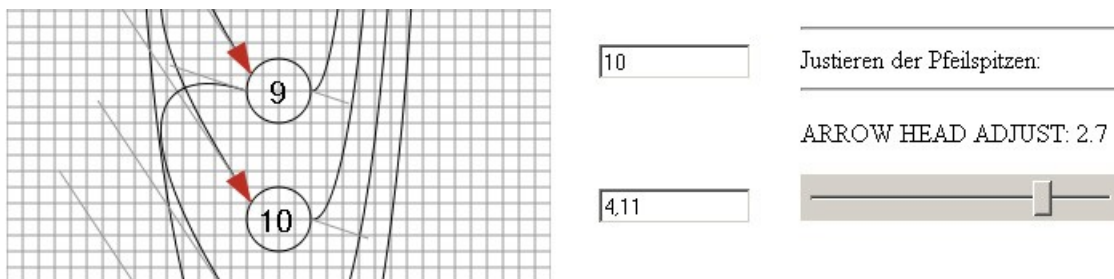


Abbildung 28: Justieren der Pfeilspitzen im Entwicklermodus

Die Kurvenverläufe der Kanten lassen sich also mit Hilfe des Entwicklermodus interaktiv gestalten. Hat man eine schöne Darstellung der Kanten gefunden, dann können die ermittelten Werte an zentraler Stelle als Konstanten in den Sourcecode der Anwendung (Editor- und Anzeigemodus) eingetragen werden.

Weiterhin gilt es sicherzustellen, dass sich umfassende Kanten an möglichst keiner Stelle berühren. Wie dieses Verhalten realisiert wird, wird im nächsten Abschnitt beschrieben.

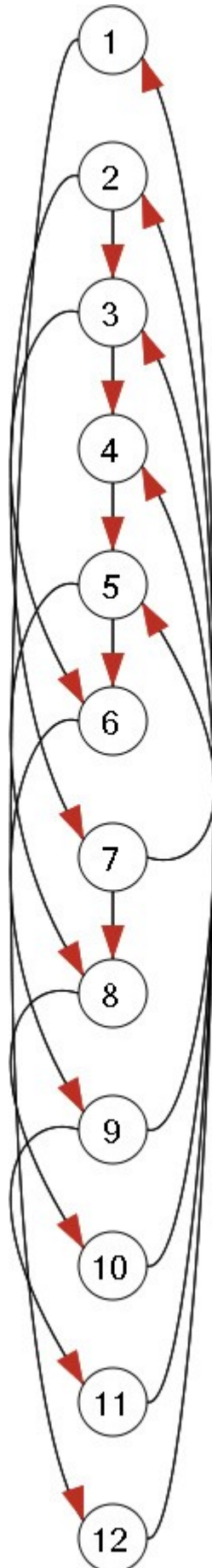
5.6.3 Layoutalgorithmus

Sich umfassende Kanten sollten sich aus Darstellungsgründen an keiner Stelle berühren. Offensichtlich wäre es praktisch, wenn man zu einer gegebenen Kante einen festen Abstand zu den Knoten festlegen könnte. Das wird durch die Einführung eines Faktors `level` erreicht. Dieser Faktor wird schon in Abb. 22 bei der Berechnung der Koordinaten der Kontrollpunkte verwendet.

In Abb. 22 ist `level` fest auf den Wert 5 gesetzt. Je größer dieser Faktor gewählt wird, umso weiter liegen die Kontrollpunkte von den Knoten entfernt, nämlich um den Faktor `level`, wie man bei der Betrachtung von Abb. 22 nachvollziehen kann.

Und je weiter die Kontrollpunkte entfernt liegen, umso „weiter“ ist auch die *kubische Bezierkurve* von den Knoten entfernt, die durch diese Kontrollpunkte definiert ist.

In der folgenden Abbildung ist als Beispiel ein komplexer Graph dargestellt, bei dem die Positionen aller Kanten mit einem festen Wert für `level` berechnet wurden.

Abbildung 29: Graph mit festem Abstand `level`

Bei der Erstellung des Graphen in Abb. 29 wurde `level` mit dem Wert 3 festgesetzt.

Wie man sieht, haben alle Kanten des Graphen den gleichen maximalen Abstand zu der Vertikalen, auf der die Knoten liegen. Kanten, die Nachbarknoten verbinden, seien bei dieser Betrachtung einmal außer Acht gelassen.

Dieses Verhalten liegt in der folgenden Eigenschaft *kubischer Bezierkurven* begründet: Wenn die X-Koordinaten paarweise entsprechender Kontrollpunkte `CONTROL_1` und `CONTROL_2` (vgl. Abb. 21) verschiedener Kurven jeweils denselben Wert haben, dann haben diese Kurven auch den gleichen „Abstand“ von den Knoten (anschaulich gesprochen).

Diese Besonderheit lässt sich für die im folgenden beschriebenen Vorgehensweise ausnutzen.

Das Ziel dabei ist es, dass sich umfassende Kanten nicht berühren und alle Kanten möglichst gleich weit voneinander entfernt liegen.

Zunächst soll eine erste Idee zur Lösung dieses Problems kurz besprochen werden. Dazu wird der Faktor `level` abhängig von der Anzahl der Knoten, die sich zwischen Start- und Zielknoten der Kante befinden, gewählt.

Bei einem dazwischen liegenden Knoten wäre `level = 1`, bei Startknoten 3 und Endknoten 6 wäre `level = 2`, usw.

Ein ähnliches Verfahren wird in der existierenden Appletversion der Anwendung offensichtlich umgesetzt. Hier wird ein Mittelpunkt zwischen zwei zu verbindenden Knoten bestimmt. Ein Halbkreis über diesem Mittelpunkt macht eine Kante aus.

Die folgende Abbildung soll dieses einfache Prinzip illustrieren. Es ist tatsächlich ein Graph dargestellt, der mit der Appletversion erstellt wurde:

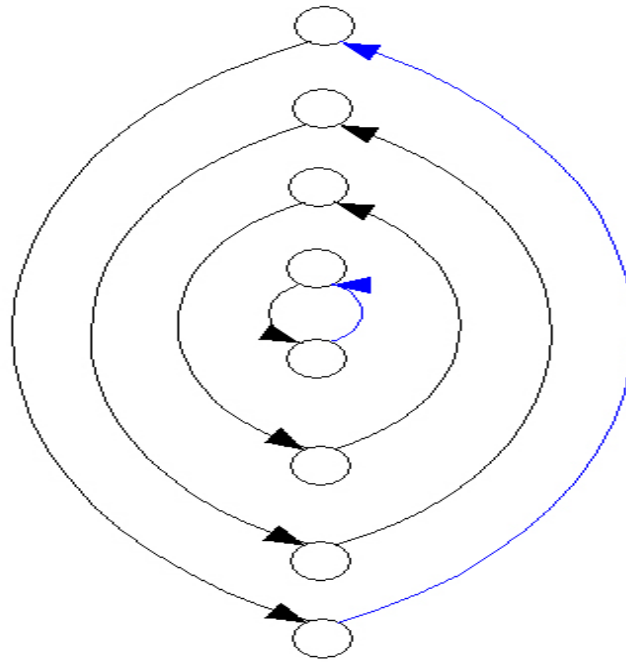


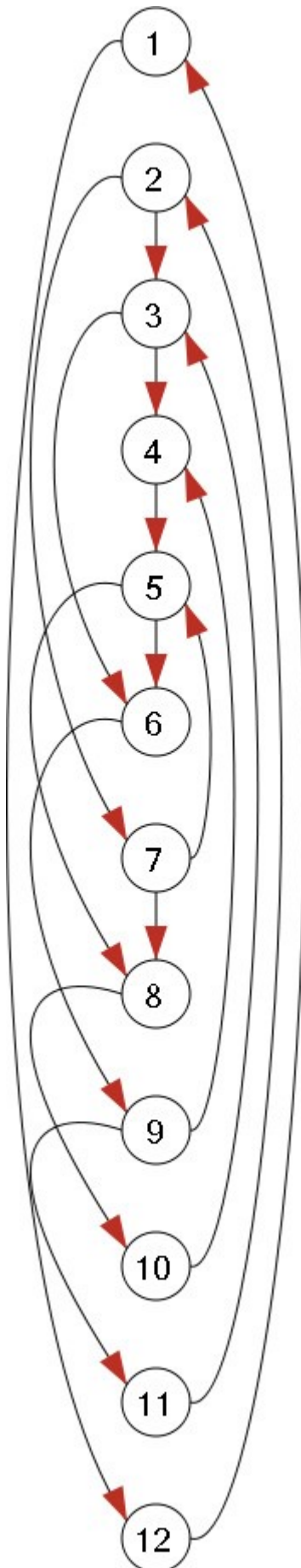
Abbildung 30: Einfache Positionierung von sich umfassenden Kanten

Selbst wenn kein Halbkreis, sondern eine weniger gebogene Kurve zur Darstellung einer Kante gezeichnet werden würde, würde die Abbildung möglicherweise sehr breit werden. Dieses Verhalten ist bei der Integration der Zeichenfläche (vgl. 5.2) nicht wünschenswert und eine schmalere Darstellung des Graphen sollte erreicht werden. Weiterhin fällt in Abb. 30 auf, dass z.B. eine Kante mit Startknoten 1 und Endknoten 8 wesentlich weiter von den Knoten entfernt liegt, als einer Kante mit Startknoten 4 und Endknoten 5. Das Ziel, dass alle Kanten denselben Abstand voneinander haben, wird durch diese Methode nicht erreicht.

Zur Lösung des Problems wird eine komplexere Vorgehensweise angewandt: Für jede Kante wird der Wert von `level` in Abhängigkeit von der Anzahl der Kanten, die diese Kante umfassen, separat festgesetzt. Auf der nächsten Seite findet man zur Illustration eine Abbildung des gleichen Graphen wie in Abb. 29, allerdings wurde hier der im Folgenden beschriebene Layoutalgorithmus angewendet:

Die Zuordnung von Werten für `level` zu verschiedenen Kanten erfolgt im *Editor für Kontrollflussgraphen* durch einen Algorithmus. Dieser greift auf die nachfolgend definierten Konstanten zurück: Die Konstante `NUM_NODES` gibt die Zahl der Knoten des Graphen an. Außerdem wird die Konstante `MAX_LEVEL` verwendet. Diese errechnet sich auf die folgende Weise: $MAX_LEVEL = NUM_NODES / 2 + 1$

Diese Formel kann man auch bei der Betrachtung von Abb. 30 nachvollziehen. Der Summand 1 hat seine Berechtigung, weil auch reflexive Kanten modelliert werden sollen. Nun werden die einzelnen Schritte des Algorithmus beschrieben:

Abbildung 31: Graph mit variablem Abstand `level`

1. Initialisiere eine leere Liste `list`. Durchlaufe die Knoten von unten nach oben, angefangen mit dem Knoten mit Nummer `NUM_NODES` bis zum Knoten mit Nummer 2 und überprüfe, ob der aktuell betrachtete Knoten, die Nummer dieses Knotens werde in der Variablen `x` gespeichert, einen Nachfolger hat. Falls ein Nachfolger existiert, so werde die Nummer des Nachfolgers in der Variablen `y` gespeichert und mit Schritt 2. fortgeföhren. Sonst wird der Durchlauf fortgeföhrt.
2. Gilt $y < x$, dann werde eine Variable `l` mit dem Wert `MAX_LEVEL` initialisiert. Betrachte nun nacheinander alle Knoten von Nummer `x` bis Nummer `NUM_NODES`. Jedes Mal, wenn man dabei auf einen Knoten trifft, der einen Nachfolger mit einer Nummer hat, die kleiner als `y` ist, so werde die Variable `l` dekrementiert.
3. Das Tripel (x, y, l) werde in `list` gespeichert. Ist der in Schritt 1 beschriebene Durchlauf an dieser Stelle noch nicht abgeschlossen, so werde dieser fortgeföhrt.
4. Zeichne jedes Element (x, y, l) in `list` mit der in Abschnitt 5.6.1 beschriebenen Methode. Dabei gibt das Tupel (x, y) eine Kante vom Knoten mit der Nummer `x` zum Knoten mit der Nummer `y` an und die Zahl `l` gibt die Zahl `level` an, bzw. den relativen Abstand der Kante zu den Knoten.
5. Stelle in Gedanken den Graphen auf den Kopf und wiederhole die Schritte 1 bis 4 um die Kanten auf der anderen Seite des Graphen abzuarbeiten.

Anschaulich gesprochen werden beide Seiten des Graphen hintereinander betrachtet. Zuerst wird die Seite rechts von den Knoten behandelt. Dabei wird der Graph von unten nach oben durchlaufen (Schritt 1) und für jede abgehende Kante eines Knotens wird gezählt, wie viele Kanten es gibt, die diese Kante umfassen (Schritt 2).

Sei die Zahl der umfassenden Kanten k , dann wird in Schritt 3, die Kante und die Zahl $l = \text{MAX_LEVEL} - k$ in einer Liste gespeichert. In Schritt 4 wird diese Liste der Reihe nach durchlaufen und jede Kante gezeichnet.

Mit dieser Vorgehensweise wird eine optisch ansprechende Darstellung eines komplexen Graphen erreicht, wie man z.B. in Abbildung 31 sehen kann.

5.6.4 Verarbeitung der Benutzereingaben

Im ersten Teil dieses Abschnitts wird die Verarbeitung der Benutzereingaben im Editormodus beschrieben.

Es gibt auch noch eine reine Anzeigmöglichkeit für die Anwendergruppe der Betreuer und Korrektoren, (vgl. 3.1). Wie bei dieser Variante die Eingabedaten aus einer Tabelle gelesen und dem Layoutalgorithmus zugeführt werden, wird im zweiten Teil angesprochen. Betrachten wir im folgenden Bildschirmausschnitt noch einmal den Prototypen der Anwendung:

The screenshot shows a web browser window with the following elements:

- Browser Title:** Mozilla Firefox
- Address Bar:** file:///E:/Dokumente%20und%20Einstellungen/Besitzer/Desktop/bachelor_5687500/Sonstiges/Prototyp/ec
- Page Header:**
 - Kurs 01613 "Einführung in die Imperative Programmierung"
 - Kurs 01612 "Konzepte Imperativer Programmierung"
 - Eingabe des Kontrollflußgraphen
 - SOFTWARE ENGINEERING logo
- Form Fields:**
 - Name:
 - Nachfolger:
 - Zeile von:
 - Zeile bis:
- Graph Diagram:** A control flow graph with five nodes (1-5) and edges. Node 1 is the start node. Edges connect 1 to 2, 2 to 3, 3 to 4, and 4 to 2.
- Interactive Input Section:**
 - Erzeugen Sie mit der Schaltfläche **Zeichne Knoten** neue Knoten und bringen Sie Kanten an, indem Sie auf den gestrichelten Linien in der Spalte **Nachfolger** die Nummern der Nachfolgerknoten eintragen.
 - Trennen Sie diese mittels **Kommata**.
 - Buttons: and
- Status Bar:** Done

Die Eingaben des Benutzers lassen sich in verschiedene Kategorien unterteilen:

- Eingaben, die keine unmittelbare Reaktion der Anwendung bewirken. Das sind Eingaben in die Formularfelder der Spalten `Name`, `Zeile von` und `Zeile bis`.
- Eingaben, die eine unmittelbare Reaktion bewirken:
 - Klicken der Schaltfläche `Zeichne Knoten`. Dadurch wird die Zahl der dargestellten Knoten um einen Knoten erhöht. (Diese Schaltfläche gibt es aus Kompatibilitätsgründen in der endgültigen Fassung der Implementierung nicht mehr. Dieser Punkt wird deshalb nicht weiter ausgeführt)
 - Eingaben in die Formularfelder der Spalte `Nachfolger`. Jede Eingabe entspricht dem Eintragen einer neuen Kante in den Graphen. Diese wird unmittelbar nach der Eingabe auf die Zeichenfläche gezeichnet.
 - Klicken der `Graph einsenden` Schaltfläche. Damit wird die Eingabe abgeschlossen. Über die Methode `POST` des `HTTP Protokolls` werden danach alle Eingabedaten des Webformulars zum Server übermittelt.

Auf die zuerst und die zuletzt genannten Eingaben wird in diesem Abschnitt nicht näher eingegangen, weil erstere nur auf der Serverseite weiterverarbeitet werden und letztere bekanntermaßen über das `HTTP Protokoll` abgewickelt werden.

Um Eingaben in der Spalte `Nachfolger` zu registrieren, wird jeweils ein Eventhandler an den Formularfeldern in der Spalte `Nachfolger` angebracht. Dieser reagiert auf Tastaturanschläge. Dazu genügt die Angabe von Tags im HTML Dokument, die so aussehen:

```
<input class="editor"
      type="text"
      name="FeldA12"
      size="6"
      maxlength="6"
      id="node2"
      onkeyup="redraw();" >
```

Das `<input>` Tag stellt das Formularfeld dar. Das zuletzt aufgeführte Attribut `onkeyup` registriert den angesprochenen Eventhandler. Bei einem Tastaturanschlag - um genau zu sein beim Loslassen der Taste - wird die JavaScript Funktion `redraw()` aufgerufen und der Graph wird neu gezeichnet. Diese Funktion ist im Kopfteil des HTML Dokuments definiert ist (vgl. 5.3).

Nach Aufruf von `redraw` werden zunächst alle Formulareingaben in der Spalte `Nachfolger` durchlaufen und für jede Eingabe einer Knotennummer wird der Layoutalgorithmus aufgerufen (vgl. 5.6.3). Dieser zeichnet die Kante mit Hilfe der Vorgehensweise aus Abschnitt 5.6.1. Fehlerhafte Eingaben quittiert das Programm mit einer Fehlermeldung. Diese bleibt solange unter dem betreffenden Eingabefeld eingeblendet, bis die fehlerhafte Eingabe korrigiert wird.

In puncto Eingabeverarbeitung unterscheidet sich realisierungstechnisch der Editormodus vom Anzeigemodus. Bei letzterem sind die Daten bezüglich der Nachfolgeknoten in einer HTML Tabelle hinterlegt. Anstatt Formularfelder auszulesen werden die Zellen dieser Tabelle ausgelesen. Die entnommenen Daten werden wie oben beschrieben weiterverarbeitet.

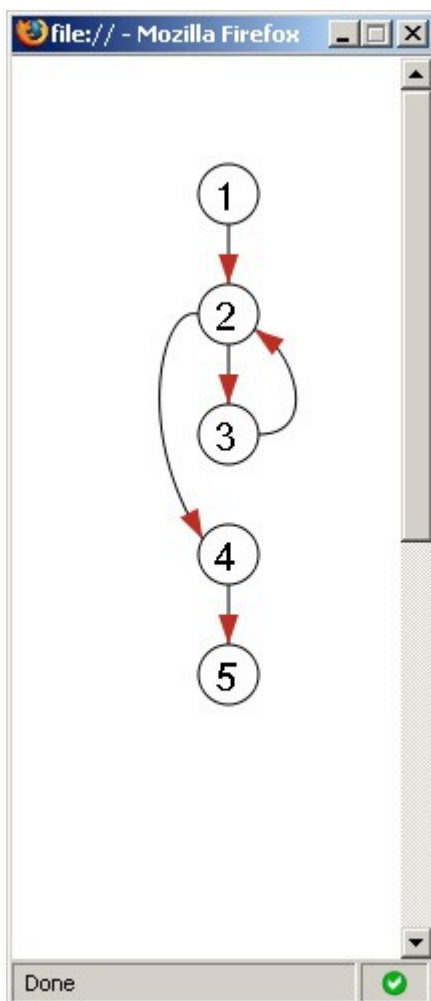


Abbildung 32: Graph im Pop-up-Fenster

Ein Pop-up-Fenster stellt bei dieser Variante den Graphen dar, vgl. Abb. 32.

Das Pop-up wird bei Anklicken eines Links mit dem Text „Grafische Darstellung“ geöffnet.

Über das `window.opener` Objekt der JavaScript Standardbibliothek erhält das JavaScript Programm, das im Pop-up-Fenster ausgeführt wird, eine Referenz auf das Elternfenster in dem die Daten des Graphen angezeigt werden.

So können die Daten des Graphen im Elternfenster gelesen werden, ohne das Programmlogik im HTML Dokument des Elternfensters implementiert werden müsste.

Diese Eigenschaft ist vorteilhaft bei der Integration der Anwendung in die bestehenden dynamischen Webseiten, weil als einzige Änderung in den betreffenden HTML Schablonen nur ein einfacher Link angebracht werden muss.

Das Dokument des Pop-up-Fensters wird in der statischen Ressource `graph.html` hinterlegt.

Nachdem der erste Prototyp noch im Sinn einer Imperativen Programmierung erstellt wurde, folgt die endgültige Version einem objektorientierten Entwurf.

Das wirkt sich positiv auf die Wartbarkeit der Anwendung aus. Im folgenden Abschnitt wird mittels eines Klassendiagramms die Programmstruktur kurz vorgestellt.

5.7 Programmstruktur

Die folgende Abbildung zeigt die Programmstruktur. Die Klasse *GraphOwner* kommt in der tatsächlichen Implementierung nicht vor, weil es in JavaScript keine abstrakten Klassen gibt. Diese Klasse wurde nur aus Darstellungsgründen eingezeichnet und soll verdeutlichen, dass ihre Attribute und Methoden jeweils auch in den Klassen *Display*, *Devel* und *Editor* implementiert werden. Die *draw* Methoden der Klassen *Polygon*, *Point*, *Node* und *BezierCurve* verkapseln die Zeichenroutinen des Canvas-Elements. Alle dargestellten Klassen wurden speziell für diese Anwendung implementiert.

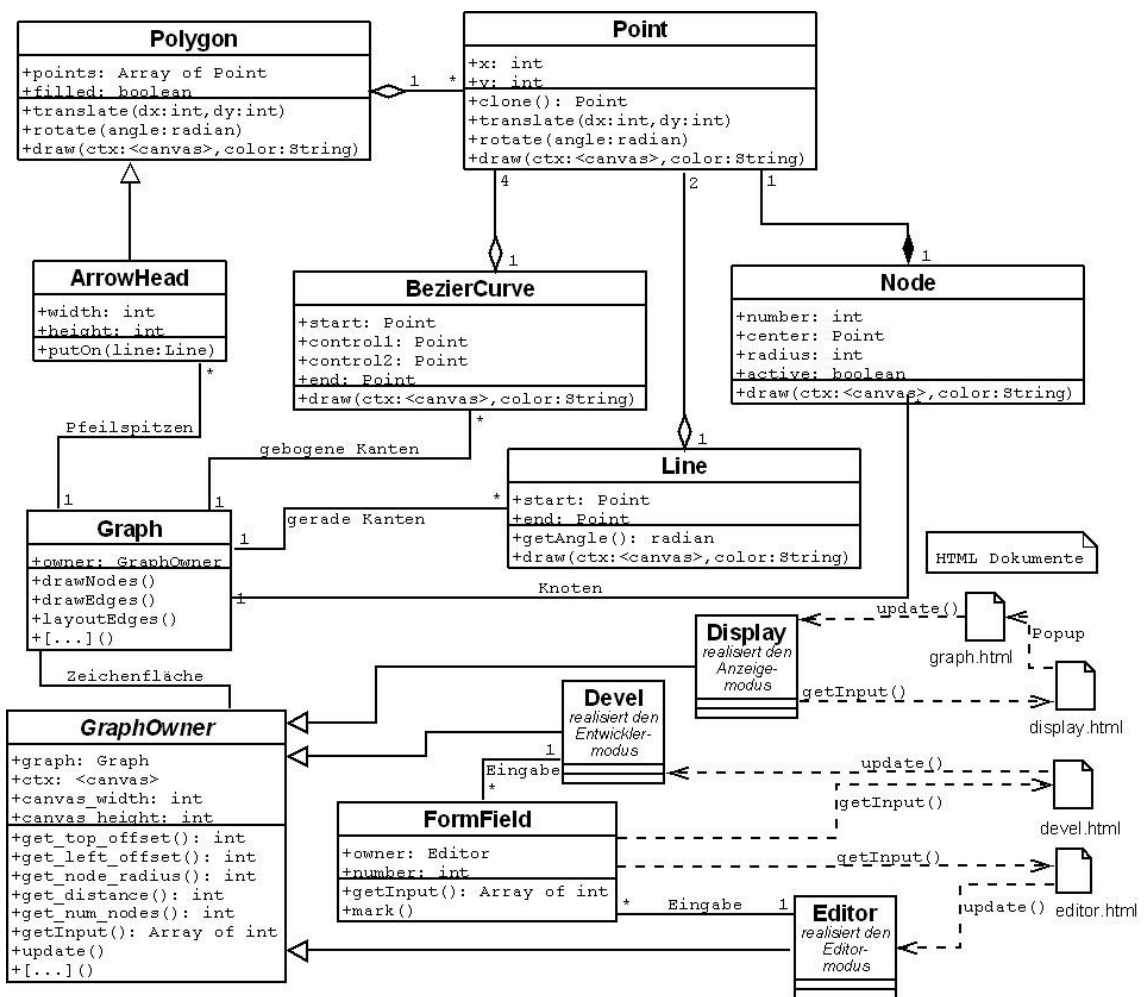


Abbildung 33: Klassendiagramm

In Abb. 33 repräsentieren die Dokumente *devel.html*, *display.html* und *editor.html* die verschiedenen Modi der Applikation, nämlich Entwickler-, Anzeige- und Editormodus. Nach dem Laden der Dokumente im Browser wird jeweils ein *Devel*-, *Display*- bzw. *Editor*-Objekt erzeugt. vgl. Abb. 12. Das Dokument *display.html* verfügt über einen Link auf ein Pop-up-Fenster, welches das Dokument *graph.html* anzeigt, vgl. Abschnitt 5.6.4.

Zusammenfassung und Ausblick

In diesem Kapitel sollen wichtige Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick gegeben werden.

Die Betrachtung der Realisierung gliedert sich in die Teile Benutzerführung und Technik. Im Anschluss werden einige interessante Aspekte angesprochen, die nicht unmittelbar in der Zielsetzung gefordert werden und auf dem Weg zur Lösung des Problems aufgefallen sind.

Ein kurzer Ausblick bezüglich einer weiteren Verwendbarkeit des neuartigen Canvas-Elements bildet den Abschluss der vorliegenden Arbeit.

Bezüglich der Benutzerführung der neuen Lösung lässt sich festhalten, dass von der herkömmlichen Bedienung eines graphischen Editors ein Stück weit abgerückt wird und anstatt die Knoten und Kanten des Graphen frei mit der Maus auf einer Zeichenfläche anzuordnen, werden Nummern von Nachfolgerknoten in Formularfelder eingegeben - mit einem unmittelbaren grafischen Feedback. Die Eingabe des Graphen kann so wesentlich schneller und unkomplizierter erfolgen, wie Benutzererfahrungen zeigen.

Weiterhin muss im Vergleich zur alten Lösung der Anwender nicht mehr zwischen zwei verschiedenen Varianten der Anwendung wählen, sondern ihm wird je nach Browsersoftware eine passende Version automatisch präsentiert.

Bemerkenswert ist, dass die Applikation nahtlos in die Webseite integriert werden kann. Eine optisch ansprechende Gestaltung trägt zur Motivation von Studierenden, Korrektoren und Betreuern bei.

Bezüglich der Technik der neuen Lösung fällt die Verwendung des relativ neuen Canvas-Elements zur Realisierung der grafischen Darstellung auf. Während bislang die Flash Technologie als defakto Standard für eine clientseitige interaktive Grafik gehandelt wird, wird mit dieser Arbeit eine Alternative dazu vorgestellt, die zwar weniger performant ist, aber ohne weitere technische Hilfsmittel, wie kommerzielle Autorensysteme, etc. auskommt.

Außerdem wird eine schon seit längerem bekannte Technik, die im Zuge des *WEB2.0* wieder populär geworden ist, verwendet. Es handelt sich dabei um DHTML (Dynamisches HTML). Mittels JavaScript erfolgt hierbei eine Manipulation des DOM (Document Object Model) was zu einer Veränderung des HTML Dokuments zur Laufzeit führt. Ohne dieses Verhalten, wäre diese Lösung so nicht möglich gewesen, wie man sich vorstellen kann.

In diesem Zusammenhang wird auch die CSS Eigenschaft `display:none` ausgenutzt. Damit werden je nach Browsersoftware relevante Abschnitte des HTML Dokuments aus- und eingeblendet. Der HTML Quellcode befindet sich so geschlossen in einer Datei und Teile müssen nicht mit in den JavaScript Quelltext der Anwendung aufgenommen werden. Die Trennung von HTML Quellcode, CSS Formatierungsregeln und JavaScript Programmcode erhöht die Wartbarkeit der Applikation.

Im Zuge der Evaluierung von alternativen Realisierungsmöglichkeiten hat sich der Autor über verschiedene Aspekte Gedanken gemacht, die zwar nicht unmittelbar mit der Aufgabenstellung dieser Arbeit zu tun haben, die aber an dieser Stelle trotzdem kurz erwähnt werden sollen.

So findet man in Abschnitt 4.1.1 eine kurze Diskussion darüber, in welchen Szenarien eine zentrale Programmausführung, eine serverseitige Verarbeitung von Bilddaten und eine asynchrone Kommunikation sinnvoll ist.

Ebenfalls von Interesse ist die Existenz der *Web Hypertext Application Working Group* (WHATWG), die HTML speziell für Webanwendungen weiterentwickelt und sich parallel zu den Bemühungen der bekannten Organisation W3C um eine Standardisierung der Formate im Internet kümmert. Sie wird in Abschnitt 4.3 kurz vorgestellt.

Abschließend wird ein Ausblick bezüglich weiterer Einsatzgebiete des Canvas-Elements gegeben

Ausblick

Kern bezüglich der technischen Realisierung der vorgestellten Applikation ist die Verwendung des Canvas-Elements zur Darstellung der Grafik. Nach einer Betrachtung verschiedener Realisierungsansätze, ist die Wahl auf diese Technik gefallen und wegen der Neuartigkeit soll ein kurzer Ausblick auf eine weitere Verwendbarkeit gegeben werden.

Leider ist das Zeichnen auf der Canvas-Zeichenfläche noch wenig performant, und großflächige Animationen oder Spiele sind derzeit damit noch nicht implementierbar. Es gibt ein kleines Benchmarkprogramm, das die Leistungsfähigkeit des Canvas Elements eindrucksvoll demonstriert und ein 3D-Spiel realisiert: *Canvascape* [CAN07]. Die Darstellung des Spiels ist aus Performanzgründen bisher nur auf einer kleinen Fläche möglich.

Vermutlich bleiben Flash Filme die erste Wahl für Animationen. Ein denkbarer Anwendungsbereich im Lehrbetrieb wie die Visualisierung von bestimmten Lehrinhalten als großflächige Animation, kann mit dem Canvas-Element bislang noch nicht entwickelt werden.

Für kleine grafische Anwendungen hat sich das Canvas-Element mit der Applikation im *browserbasierten Editor für Kontrollflussgraphen* schon bewährt.

Weitere Anwendungsfälle könnte man z.B. im Rahmen des Kurses *1661 Datenstrukturen* finden. Der Autor erinnert sich, dass in der Online Version dieses Kurses auf Java Applets verlinkt wird, die einfache Sortieralgorithmen visualisieren.

Hier könnte es mittlerweile ähnliche Probleme mit verschiedener Browsersoftware geben wie bei der existierenden Appletversion. Ein Ersatz kann nach Einschätzung des Autors mit dem Canvas-Element und JavaScript implementiert werden.

Mit dieser Empfehlung sind die Ausführungen zu der vorliegenden Arbeit beendet. Es folgt ein Anhang mit Informationen zur Browserkompatibilität der Anwendung.

Anhang: Browserkompatibilität der Anwendung

In diesem Kapitel wird aufgelistet mit welcher Browsersoftware die Anwendung getestet wurde und welche Probleme es bei einigen Browsern gibt.

Zu beachten ist, dass es im JavaScript Programm keine explizite browserspezifische Anpassung in Form einer Browserweiche gibt.

Lediglich wegen der Browsersoftware Opera wurde eine Ausnahme gemacht, weil Opera in der Version 9.21 die Pfeilspitzen des Graphen fehlerhaft anzeigt, wenn diese gefüllt sind.

Deswegen wurde mittels JavaScript geregelt, dass nicht-gefüllte Pfeilspitzen dargestellt werden, wenn der Graph im Browser Opera dargestellt wird.

Internet Explorer

Die **Version 5.01** des Browsers Internet Explorer wurde im Dezember 1999 veröffentlicht. Bei dieser Version des IE wird die Formularversion automatisch angezeigt, weil dieser Browser das Canvas-Element noch nicht unterstützt.

Bemerkenswert ist, dass diese Version noch auf Windows Version 3.x läuft und sogar eine Benutzergruppe mit diesem Betriebssystem die Anwendung nutzen könnte.

Ebenfalls getestet wurde die **Version 5.5**, die im Juli 2000 auf den Markt kam. Auch hier wird die Formularversion der Anwendung fehlerfrei und automatisch angezeigt.

Ab der **Version 6.0** des Internet Explorers, die es seit Oktober 2001 gibt, wird die interaktive Fassung dargestellt. Die Anwendung funktioniert ebenfalls sehr gut mit der aktuellen **Version 7.0** des Internet Explorers.

Somit kann man sagen, dass der Internet Explorer zuverlässig bis zurück zum Jahr 1999 unterstützt wird. Auf anderen Betriebssystemen als Windows wurde die Software Internet Explorer zum Test nicht betrieben.

Mozilla / Firefox / Seamonkey / Iceweasel

Weiterhin sehr verbreitet sind Browser, die die Gecko Layoutengine verwenden. Zu dieser Gruppe gehört die Software-Familie Mozilla/Firefox [USW07]. Hier wurde die Applikation mit der jeweils aktuellen Browserversion sowohl auf dem Betriebssystem Microsoft Windows XP, als auch auf Linux und Mac OS getestet. Die Varianten Seamonkey und Iceweasel (Debian/Linux) stellen die Anwendung ebenfalls fehlerfrei dar.

Opera

Das Problem bzgl. der fehlerhaften Darstellung der Pfeilspitzen wurde oben schon angesprochen. Ansonsten lässt sich die Software fehlerfrei auf Windows XP und Mac/OS mit dem Browser Opera betreiben.

Mac/OS

Auf Mac/OS wurde die Anwendung fehlerfrei mit den Browsern Safari, Camino und Firefox betrieben.

Netscape Navigator 6 + 4.7

Abschließend wurde die Software noch mit einer Browsersoftware auf Windows getestet, die in den Jahren 1995 bis 1999 sehr verbreitet war und im Jahr 1996 - nach Angaben von Wikipedia - sogar einen Marktanteil von knapp 80% erreicht hat [NET96].

Die jüngere **Version 6** des Browsers Netscape Navigator kommt zwar mit CSS zurecht und kann das Formular sauber darstellen, unterstützt jedoch das Canvas-Element nicht, so dass automatisch die Formularversion der Anwendung dargestellt wird.

Die ältere **Version 4.7** kam im Oktober 1998 auf den Markt, ist also noch älter als die Version 5.01 des Browsers Internet Explorer, der im Dezember 1999 veröffentlicht wurde. Es verwundert daher nicht, dass diese Browsersoftware mit der Formatierung des HTML Dokuments mittels CSS nicht zurecht kommt. (CSS Level 2 wurde im Juli 1998 eingeführt [CSS07]). Das HTML Dokument der Anwendung wird deshalb unformatiert aber korrekt dargestellt.

Abbildungsverzeichnis

Abbildung 1: Screenshot der bestehenden Lösung (Appletversion).	9
Abbildung 2: Screenshot der Formularversion.....	12
Abbildung 3: Screenshot eines Prototypen der neuen Lösung.....	14
Abbildung 4: Gegenüberstellung von JavaScript Grafik.....	23
Abbildung 5: JavaScript Anweisungen zum Zeichnen einer Linie.....	25
Abbildung 6: HTML Tag zur Einbindung der Canvas-Zeichenfläche.....	25
Abbildung 7: Beispielhafter Quellcode zur HTML Tabelle.....	29
Abbildung 8: Ausgabe der Tabelle aus Abb.7 im Browser.....	30
Abbildung 9: Quellcode zur CSS Positionierung.....	31
Abbildung 10: Anzeige zur CSS Positionierung.....	31
Abbildung 11: Die Positionierung der Formularfelder im Detail.....	33
Abbildung 12: Kopfteil des HTML Dokuments <code>editor.html</code>	35
Abbildung 13: Screenshot von der neuen Formularversion.....	37
Abbildung 14: HTML Code bzgl. statischer und interaktiver Version.....	38
Abbildung 15: CSS Formatierungsregel bezüglich der interaktiven Version.....	38
Abbildung 16: DOM Baum des Dokuments <code>editor.html</code> (vereinfacht).....	39
Abbildung 17: JavaScript zum Einblenden der interaktiven Elemente.....	40
Abbildung 18: HTML Quelltext der ein statisches Bild der Knoten einbindet.....	40
Abbildung 19: Löschen des Inhalts eines HTML Tags.....	40
Abbildung 20: Konstruktion einer kubischen Bezierkurve.....	41
Abbildung 21: Eine Kante als Bezierkurve im Kontrollflussgraphen.....	42
Abbildung 22: JavaScript Anweisungen zum Zeichnen der Kante aus Abb. 21	43
Abbildung 23: Screenshot des Entwicklersmodus der Applikation.....	44
Abbildung 24: Graph mit allen möglichen Kantentypen.....	45
Abbildung 25: Einfache Modellierung der Pfeilspitzen.....	46
Abbildung 26: Konstruktion der Pfeilspitzen.....	47
Abbildung 27: JavaScript Code zum Justieren der Pfeilspitzen.....	48
Abbildung 28: Justieren der Pfeilspitzen im Entwicklersmodus.....	48

Abbildung 29: Graph mit festem Abstand <code>level</code>	50
Abbildung 30: Einfache Positionierung von sich umfassenden Kanten.....	51
Abbildung 31: Graph mit variablem Abstand <code>level</code>	52
Abbildung 32: Graph im Pop-up-Fenster.....	55
Abbildung 33: Klassendiagramm.....	56

Literaturverzeichnis

- ADO07** (2007): Adobe, Flash Player Penetration,
http://www.adobe.com/products/player_census/flashplayer/
(Datum des Zugriffs: 26. Juli 2007).
- ALA07** (2007): A List Apart, Webseite über Webdesign von Jeffrey Zeldman et al.,
Artikel über die Formatierung von "Multi-Column Layouts" mittels CSS,
<http://www.alistapart.com/articles/multicolumnlayouts>
(Datum des Zugriffs: 27. Juli 2007).
- APP07** (2007): Apple Developer Connection, Safari JavaScript Reference,
Dokumentation zum Canvas-Element,
<http://tinyurl.com/zx7so>
(Datum des Zugriffs: 29. Juli 2007).
- BAT07** (2007): Apache Software Foundation, Batik SVG Toolkit
<http://xmlgraphics.apache.org/batik/>
(Datum des Zugriffs: 26. Juli 2007).
- CAN07** (2007): Abraham Joffe, Canvascape „3D Walker“, Canvas Demo Programm
<http://www.abrahamjoffe.com.au/ben/canvascape/>
(Datum des Zugriffs: 26. Juli 2007).
- CRI07** (2007), (kein Autor, Beiträge aus Nutzergemeinde),
Wikipedia Eintrag zu Criticism of Internet Explorer
http://en.wikipedia.org/wiki/Criticisms_of_Internet_Explorer
(Datum des Zugriffs: 24. Juli 2007).
- CSS07** (2007), (kein Autor, Beiträge aus Nutzergemeinde),
dt. Wikipedia Eintrag zu Cascading Style Sheets
http://de.wikipedia.org/wiki/Cascading_Style_Sheets
(Datum des Zugriffs: 10. Juni 2007).
- DOM05** (2005), W3 Konsortium, Document Object Model (DOM)
<http://www.w3.org/DOM/#what>
(Datum des Zugriffs: 24. Juli 2007).
- DRAW2D** (2007), Open Jacob Projekt, Draw2D JavaScript Bibliothek,
<http://www.openjacob.org/draw2d.html>
(Datum des Zugriffs: 26. Juli 2007).
- EXC07** (2007): Google Inc, Explorer Canvas Script
<http://code.google.com/p/explorercanvas/>
(Datum des Zugriffs: 26. Juli 2007).

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum

Unterschrift