Generierung interaktiver Animationen von Berechnungsmodellen

Stephan Diehl, Andreas Kerren

Universität des Saarlandes, FB 14 Informatik
Postfach 15 11 50, D-66041 Saarbrücken
{diehl,kerren}@cs.uni-sb.de

Zusammenfassung In diesem Artikel wird ein generativer Ansatz für animierte Berechnungsmodelle vorgestellt. Zuerst werden die technischen und pädagogischen Ziele diskutiert, dann wird der Ansatz im Kontext einer Lernsoftware für den Compilerbau angewandt und die Implementierung eines ersten Prototypen beschrieben. Basierend auf der Erfahrung mit diesem Prototypen wurde das GANIMAL Framework entworfen. Es handelt sich dabei um ein generisches Algorithmenanimationssystem, das eine für ein solches System einzigartige Fülle an Möglichkeiten durch sein graphisches Basispaket, das nebenläufige Laufzeitsystem mit graphischer Benutzeroberfläche und die Visualisierungskontrollsprache GANILA bietet.

1 Einleitung

Heutige *Lernsoftware* deckt größtenteils Themen außerhalb der Informatik ab und zielt meist auf die Vermittlung von Faktenwissen und weniger auf das Verständnis komplexer Abläufe. Hierzu zählen elektronische Bücher, Lexika und Wörterbücher. Schaut man sich z.B. die von Schulmeister beschriebene Lernsoftware [13] (u.a. 16 interaktive, hypermediale Lernprogramme und 23 tutorielle Programme) genauer an, so stellt man fest, daß nur eine geringe Zahl von diesen komplexe Vorgänge darstellen kann. Dies sind vor allem Systeme im Bereich der Physik. Im Gebiet der Informatik überwiegen Kurse für Programmiersprachen. Nur wenige Systeme behandeln Themen der theoretischen Informatik [1].

Häufig tritt im Informatikunterricht das Problem auf, daß der Lehrer ein dynamisches System nur unzureichend auf Papier, Folien oder an der Tafel darstellen kann. Er muß die Dimension Zeit in der Dimension Raum abwickeln, d.h. Auswischen, Überschreiben, Übereinanderlegen von Folien oder die Verwendung aufeinanderfolgender Diagramme, Tabellen oder Abbildungen. Für diesen Zweck ist die Computeranimation, insbesondere Algorithmenanimation, das Mittel der Wahl. Der größte Teil der Arbeiten zur *Algorithmenanimation* lassen sich grob in folgende Kategorien einteilen:

1. Darstellung einfacher Graphenalgorithmen [12].
2. Geometrische Algorithmen werden mit komplexen Basisprozeduren beschrieben, die sich direkt auf graphische Operationen abgebilden lassen (siehe z.B. GASP [17]).

3. Andere entwickeln meist einfache Animationssprachen, mit denen Algorithmen annotiert werden können (siehe z.B. Zeus, Samba, Polka, XTango [14, 15]). Zum Beispiel würde die Zuweisung `x:=y` annotiert mit `moveto(obj(y), obj(x))`. So kann z.B. der Entwickler einer Animation in Zeus [2,3] graphische Sichten (views) auf Daten definieren und im Algorithmus geeigneten Stellen mit sogenannten 'interesting events' annotieren. Während der Ausführung eines annotierten Programmpunktes werden diese Ereignisse an alle Sichten geschickt, die dann darauf reagieren können.

Im Folgenden betrachten wir Algorithmenanimationen in Zusammenhang mit Lernsoftware für den Compilerbau. Dort spielt die Visualisierung von Generatoren eine wesentliche Rolle. Wir stellen diesbezüglich in Abschnitt 3 zwei Methoden vor. Dann betrachten wir eine Implementierung, die auf der ersten Methode beruht, analysieren diese und entwerfen ein Framework, das als Grundlage für die zweite Methode verwendet werden kann.

1.1 Technische Ziele

Im Software-Engineering gewinnen generative und generische Techniken immer mehr an Bedeutung. Im Projekt GANIMAL [1] entwickeln wir Generatoren, die interaktive Visualisierungen und Animationen verschiedener Compilerphasen, aber auch anderer Algorithmen erzeugen. Diese sollen Teil einer web-basierten Lernsoftware für den Compilerbau werden. Die Ergebnisse, d.h. insbesondere die Beschreibungssprache und das Basispaket, könnten in späteren Projekten auch zur Entwicklung von Generatoren für andere prozeßorientierte Lerninhalte, wie z.B. Algorithmen, Rechnerarchitektur, chemische Reaktionen oder gar der Modellierung ökonomischer Prozesse eingesetzt werden.

1.2 Pädagogische Ziele

Der generative Ansatz erlaubt neue Übungsformen. Als Teil einer Übungsaufgabe schreibt der Lerner Spezifikationen von Vorgängen oder Berechnungsmodellen. In konventioneller Lernsoftware werden solche Antworten oft auf Korrektheit geprüft. Fehler werden dem Lerner angezeigt und er kann seine Antwort überarbeiten. Als Folge des Halteproblems ist jedoch die automatische Überprüfung der Korrektheit für viele Aufgaben, die sich auf Berechnungsmodelle beziehen, nicht möglich. Daher werden in unserem Ansatz interaktive Animationen aus der eingegebenen Spezifikation des Lerners erzeugt. Dann kann er diese mit Hilfe eigener oder vorgegebener Beispiele testen. Auf diese Weise kann er eigenständig Fehler entdecken. Die Software tritt in diesem Fall nicht als nüchterne, allwissende Autorität auf, die ihm seine Fehler zeigt.

Viele Autoren sind der Meinung, daß Lernsoftware, in der der Computer als Korrektor auftritt, entmutigend wirkt und daher wenig erfolgreich ist. Die

[1] Das Projekt wird von der DFG unter Aktenzeichen WI 576/8-1 und WI 576/8-3 gefördert.

existierenden Untersuchungen hierzu sind jedoch teilweise widersprüchlich, siehe [13]. Unser Ansatz ermöglicht exploratives, selbst gesteuertes Lernen. Der Lerner kann bestimmte Aspekte in den erzeugten, interaktiven Animationen fokussieren und herausfinden, welche Auswirkungen kleine Änderungen in der Spezifikation haben. Aufgrund solcher Beobachtungen kann er Hypothesen formulieren und diese empirisch überprüfen. Dabei soll jedoch nicht das mathematische Beweisen von Hypothesen ersetzt, sondern ein tieferes Verständnis der Berechnungsmodelle gefördert und damit eine spätere Beweisführung erleichtert werden [7].

Die Akzeptanz und Effektivität einer solchen explorativen Lernsoftware kann nur in der Praxis, d.h. im Unterricht, belegt werden. In Zusammenarbeit mit kognitiven Psychologen haben wir einige Lernversuche durchgeführt und entwickeln derzeit neue.

2 Lernsoftware für den Compilerbau

In der Informatik und insbesondere im Compilerbau sind die Theorie und Algorithmen sehr abstrakt und meist auch komplex. Daher ist ihre visuelle Darstellung ein wichtige Hilfe im Informatikunterricht [10, 11]. Obgleich der Compilerbau meist als praktisches Gebiet innerhalb der Informatik gesehen wird, beruhen seine Techniken auf Ergebnissen der theoretischen Informatik wie z.B. formalen Sprachen, Automatentheorie und formaler Semantik.

Im Compilerbau werden Techniken entwickelt, mit denen Programme höherer Programmiersprachen in effiziente und korrekte Programme einer realen oder abstrakten Maschinen übersetzt werden können [18]. Die Übersetzung eines Programms kann in verschiedene Phasen unterteilt werden. Im GANIMAL Projekt werden Generatoren entwickelt, die Visualisierungen und interaktive Animationen aus Spezifikationen dieser Compilerphasen erzeugen. In der folgenden Tabelle werden Compilerphasen, ihre Spezifikationssprachen und zugrundeliegende Berechnungsmodelle aufgeführt:

Compilerphase	**Spezifikationssprache**	**Berechnungsmodell**
Lexikalische Analyse	Reguläre Ausdrücke	Endliche Automaten
Syntaktische Analyse	Kontextfreie Grammatiken	Item-Kellerautomaten
Semantische Analyse	Attributgrammatiken	Attributauswerter
Codeerzeugung	Baumgrammatiken	Baumautomaten
Optimierung	Gleichungssysteme	Fixpunktlöser
Laufzeitsystem	**Spezifikationssprache**	**Berechnungsmodell**
Abstrakte Maschine	Kontrollflußsprache	Maschine mit Kellern, Halde und Registern

Häufig werden abstrakte Maschinen als plattformunabhängige Zwischenarchitekturen bei der Übersetzung höherer Programmiersprachen verwendet. Die Anweisungen einer abstrakten Maschine sind für die Übersetzung einer bestimmten Quellsprache oder für Quellsprachen des selben Sprachenparadigmas (imperativ, funktional, logisch, objektorientiert) maßgeschneidert. In Abbildung 3 ist eine interaktive Animation einer abstrakten Maschine zu sehen.

Es gibt nur wenige Arbeiten zur Interaktion von Programmen und multimedialen Dokumenten, d.h. wie Programme zur Betrachtungszeit eines Dokuments Teile des Dokuments modifizieren oder gar ganz neu erzeugen können. Das DOM (Document Object Model) ist ein erster Versuch zur Standardisierung einer eingeschränkten Programmierschnittstelle für diesen Zweck. Der Bedarf ist groß, da Technologien wie Java-Applets, Dynamic HTML, XML oder Schnittstellen wie das EAI (External Authoring Interface) von VRML, Netscape's LiveConnect, derzeit völlig unterschiedliche Ansätze zur Kommunikation mit dem Rest eines Dokuments verfolgen.

3 Die generativen Methoden von GANIMAL

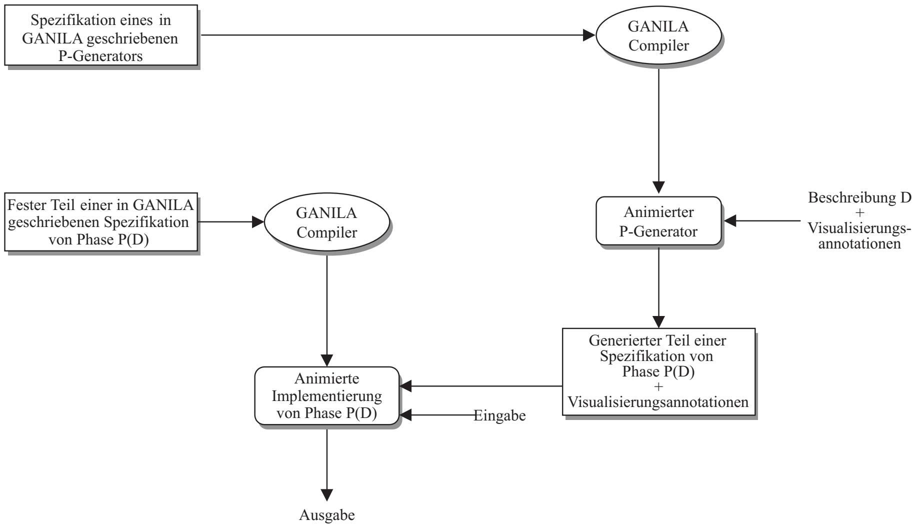
3.1 Methode I: Erweitere existierende Generatoren

Eine etablierte Spezifikationssprache für eine Compilerphase, wie z.B. reguläre Ausdrücke für die lexikalische Analyse, wird um Animationsannotationen erweitert. Zusätzlich werden Komponenten des Berechnungszustands mit graphischen Strukturen (Fenster, Graphen, Textfelder, ...) assoziiert. Dann erzeugt ein erweiterter Generator eine interaktive Animation der Compilerphase aus einer annotierten Spezifikation. Abschnitt 4 stellt eine Implementierung vor, die auf dieser Methode beruht.

3.2 Methode II: Implementiere Generatoren in einer Visualisierungssprache

Die erste Methode erlaubt es nicht, den Generator selbst zu visualisieren. In der zweiten Methode nutzen wir eine Eigenschaft aus, die viele Generatoren im Compilerbau haben. Diese Generatoren erzeugen Tabellen, die zusammen mit einem festen Treiber die Implementierung einer Compilerphase ergeben. Zuerst wurde eine Visualisierungskontrollsprache GANILA (siehe Abschnitt 6) entwickelt, in der die Generatoren und Treiber spezifiziert werden. Der GANILA Compiler erzeugt daraus eine Implementierung des Generators und des Treibers.

Aus der erweiterten Spezifikation erzeugt der GANILA Compiler eine Animation des Generators und der generierten Compilerphase, siehe Abbildung 1. Zusätzlich zur Annotierung der Spezifikation der Compilerphase, wie in der ersten Methode beschrieben, annotieren wir die Spezifikation des Generators und des Treibers durch Markieren von Programmpunkten mit sogenannten 'interesting events' und Definition von Sichten auf deren Datenstrukturen, d.h. u.a. die generierte Tabelle. Für jede Sicht muß festgelegt werden, wie sie auf jeden Event reagiert. Auf dieser Methode beruhen die interaktiven, animierten Visualisierungen im elektronischen Textbuch zur Theorie der Generierung endlicher Automaten [9].



Als Beispiel betrachten wir einen lexikalischen Analysatorgenerator. P ist die lexikalische Analyse und D eine regulärer Ausdruck. Der feste Teil ist der Treiber für die lexikalische Analyse, der generierte Teil ist eine Lookup-Tabelle und die Implementierung von P(D) ist ein lexikalischer Analysator, auch Scanner genannt.

Abbildung1. Generierung animierter Generatoren und Compilerphasen

4 Ein erster Prototyp: GANIMAM

Als Beispiel der ersten Methode betrachten wir die Entwicklung von GANIMAM, unseres web-basierten Generators für interaktive Animationen abstrakter Maschinen [5, 6]. Abbildung 3 zeigt eine Momentaufnahme einer solchen Animation. Im Folgenden beschreiben wir, wie GANIMAM verwendet werden kann und was das System generiert. Abschließend diskutieren wir dann die Vorteile von GANIMAM und der erzeugten interaktiven Animationen sowohl als Entwicklungswerkzeug, als auch als Teil einer Lernsoftware.

4.1 Technischer Überblick

GANIMAM kann über die Webseite des GANIMAL-Projekts [8] aufgerufen werden. Der Benutzer gibt die Spezifikation einer abstrakten Maschine ein, die dann zum Server geschickt wird. Ein CGI-Skript auf dem Server erzeugt Java-Quellcode, der durch einen Java-Compiler in Klassendateien übersetzt wird. In Kombination mit den Klassen des Basispakets von GANIMAM ergeben die Klassendateien ein interaktives Java-Applet, siehe Abbildung 2.

Das Applet kann über das Internet geladen werden und der Benutzer kann dann Maschinenprogramme eingeben, das Layout der verschiedenen Teile der

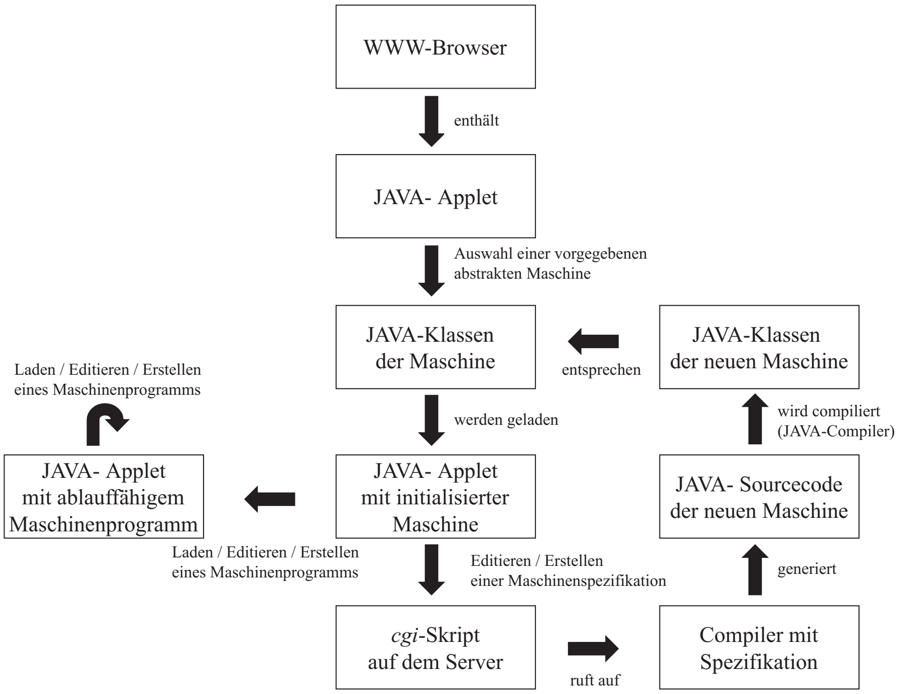


Abbildung2. Interaktion der Systemkomponenten

visualisierten abstrakten Maschine ändern und die animierte Ausführung seines Maschinenprogramms steuern.

Abstrakte Maschinen können eine unterschiedliche Anzahl von Registern, Kellern, Halden haben, daher wird für jede generierte Maschine ein automatisches Layout benötigt. Das automatische Layout gruppiert die verschiedenen Speicherarten um den Akkumulator, eine konzeptionelle Recheneinheit, herum (der Chip in der Mitte von Abbildung 3). Programmcode und Keller werden links, die Halde rechts, lokale Variablen über und Register unter dem Akkumulator plaziert. Mit dem Akkumulator ist ein Akkumulatorfenster verbunden, das den Ausdruck anzeigt der gerade im Akkumulator berechnet wird, sowie die Definition der Instruktion bzw. Funktion, die gerade ausgeführt wird. Durch einen zweifachen Mausklick mit der rechten Maustaste auf eine Instruktion im Programmcode wird die Definition der Instruktion in das Akkumulatorfenster geladen. Ein zweifacher Mausklick mit der linken Maustaste auf eine Instruktion im Programmcode setzt den Wert des Programmzählers auf die Adresse der Instruktion, d.h. die Ausführung des Programms wird an dieser Stelle fortgesetzt. Ein Klick auf eine Zelle des Kellers, der Halde oder auf ein Register öffnet ein Fenster. In diesem Fenster kann der Benutzer den Wert und den Typ, bei Registern nur den Wert ändern.

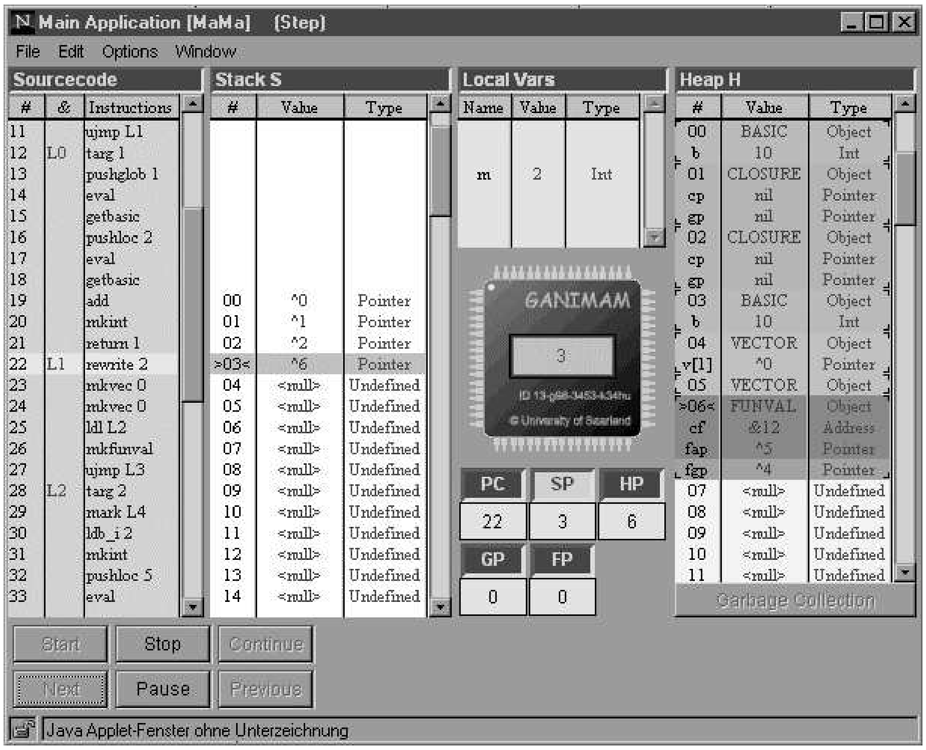


Abbildung3. Momentaufnahme einer animierten abstrakten Maschine

4.2 Vorteile interaktiver Animationen

GANIMAM bietet verschiedene Arten der Interaktion. Erstens kann der Benutzer Spezifikationen einer abstrakten Maschine eingeben oder ändern [4]. Beim Visualisieren einer abstrakten Maschine ist es wichtig, daß man nicht nur Daten und Vorgänge auf dem geringen Abstraktionsniveau und der Granularität der Spezifikationssprache darstellt, sondern auch Abstraktionen auf höherem Abstraktionsniveau. Um dies zu ermöglichen, wurden Visualisierungsannotationen zur Spezifikationssprache hinzugefügt. Diese Annotationen ähneln den 'interesting events' einschlägiger Algorithmenanimationssysteme, siehe Abschnitt 1.

Nach der Erzeugung der Implementierung der abstrakten Maschine kann der Benutzer Programme in der Sprache der abstrakten Maschine eingeben, diese Schritt für Schritt ausführen und den Inhalt jedes Registers und jeder Speicherzelle inspizieren. Während der Ausführung einer Instruktion zeigt eine Animation den Fluß der Information von Registern oder Speicherzellen in den Akkumulator und vom Akkumulator zurück zu Register oder Speicherzellen. Die Berechnung, die im Akkumulator durchgeführt wird, wird in einem separaten Fenster angezeigt.

Annotationen helfen nur solche Prinzipien sichtbar zu machen, die wir schon vorher kennen. GANIMAM kann auch dazu verwendet werden, neue Prinzipien durch Experimentieren mit Maschinenprogrammen oder Spezifikationen von abstrakten Maschinen zu entdecken. Dieses experimentelle Vorgehen kann u.a. für folgende Zwecke eingesetzt werden:

– Als Teil einer explorativen Lernsoftware ermöglicht es Studenten, neue Hypothesen zu formulieren und diese durch Änderung der Spezifikation oder des Maschinenprogramms zu überprüfen. Zusätzliche Hinweise in textueller Form sollte dabei sicherstellen, daß der Lerner auch die wesentlichen Punkte untersucht. Solche könnten z.B. der Vergleich zwischen caller-save-registers und callee-save-registers oder zwischen lazy und eager Auswertung sein oder das Finden des Kellerrahmens des statischen Vorgängers [18].
– Als Entwicklungswerkzeug kann es helfen, Fehler oder Optimierungsmöglichkeiten zu entdecken. Eine solche Optimierung könnte z.B. die Endrekursion betreffen: Durch das Verfolgen der Ausführung von Beispielprogrammen könnte der Benutzer feststellen, daß bestimmte Informationen in einem Kellerrahmen nach rekursiven Aufrufen nicht mehr benötigt werden.

GANIMAM kann auch von Forschern eingesetzt werden, um ihre neusten Implementierungstechniken vorzuführen oder einfach für Rapid Prototyping.

5 Das GANIMAL Framework

Basierend auf den mit GANIMAM und verwandten Arbeiten zur Softwarevisualisierung [16] gesammelten Erfahrungen haben wir das GANIMAL Framework für die Erstellung generativer Lernsoftware entwickelt, das in Abbildung 4 schematisch dargestellt ist. In diesem Framework wird aus einer in GANILA geschriebenen Spezifikation ein Algorithmenmodul automatisch erzeugt. Während der Ausführung des Algorithmus sendet dieses Modul ein sogenanntes "interesting event" `IE`, den Programmpunkt `pp` an dem das Event aufgetreten ist, sowie den aktuellen Animationsmodus (`RECORD`, `PLAY`, `REPLAY`) zum Kontrollobjekt, welches in seinen Einstellungen überprüft, ob für diesen speziellen Programmpunkt das interesting event zu allen Views weitergeleitet werden muß. In diesem Fall wird ein Broadcast des Events `IE` zu allen Views initiiert. Jede View hat ihre eigenen Einstellungen und überprüft nun wiederum selbst, ob sie den Eventhandler für `IE` aufrufen soll oder nicht. Abhängig vom aktuellen Modus produziert dieser Eventhandler graphische Ausgaben oder ändert lediglich den internen Zustand. Die graphische Benutzerschnittstelle GUI kann nun verwendet werden, um die Einstellungen für jeden Programmpunkt zu ändern, die Ausführung des Algorithmus zu kontrollieren und die Variablenwerte während der Ausführung zu ändern. Weiterhin können Programmpunkte, die über GANILA für parallele Ausführung markiert wurden, in der GUI so markiert werden, daß sie sequentiell ausgeführt werden.

Sämtliche Views sollten das objektorientierte Basispaket verwenden, das grafische Basisfunktionen enthält. Das umschließende Dokument ist ein Hypermediadokument, welches die verschiedenen Compilerphasen beschreibt. Es enthält Übungen, die teilweise mit Hilfe der erzeugten Generatoren lösbar sind.

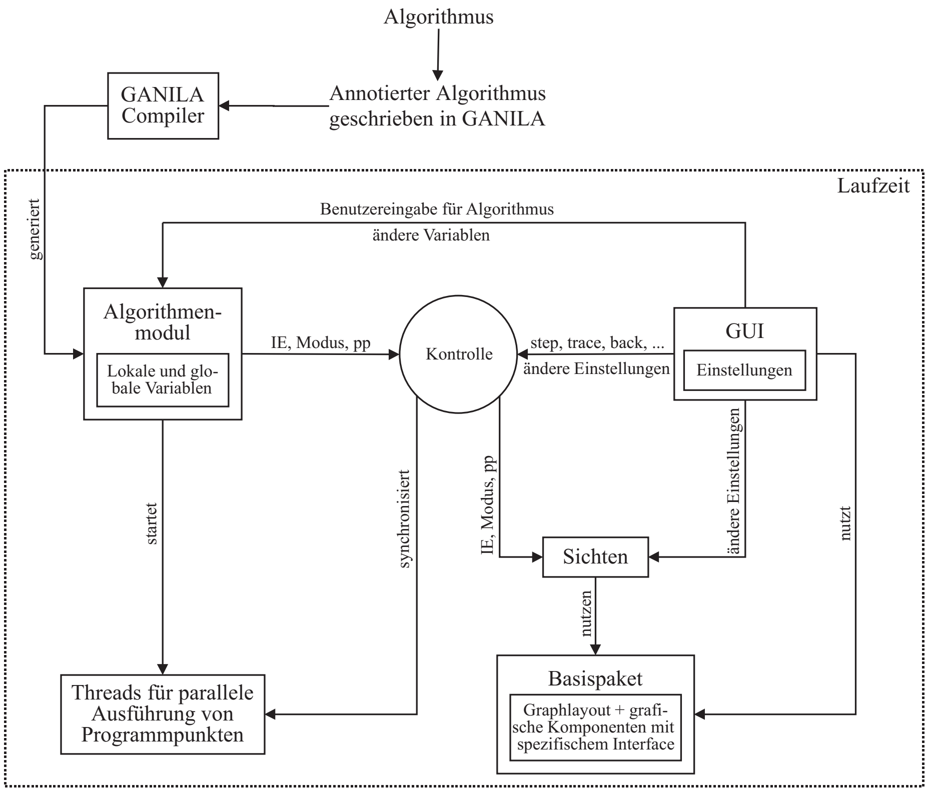


Abbildung4. Das GANIMAL Framework

Das Basispaket besteht aus einer Menge von Java Klassen, die primitive Methoden zur Kommunikation, zum Zeichnen und für Animationen beinhalten. Dies fördert zudem ein konsistentes Erscheinungsbild (look-and-feel) der verschiedenen Abschnitte der Lernsoftware. Somit erlaubt das GANIMAL Framework auch klassische Algorithmenanimationen in einfacher Weise zu implementieren, etwa die Animation des Heapsortalgorithmus, siehe Abbildung 5.

Die Plattformunabhängigkeit der Programmiersprache Java ermöglicht es, unabhängig vom lokal verfügbaren Computersystem, die Lernsoftware in allen Gebieten der Lehre zu verwenden. Die mächtige Java API (Klassenbibliothek) erleichtert die Benutzerinteraktion ebenso, wie die Entwicklung des graphischen Frontends.

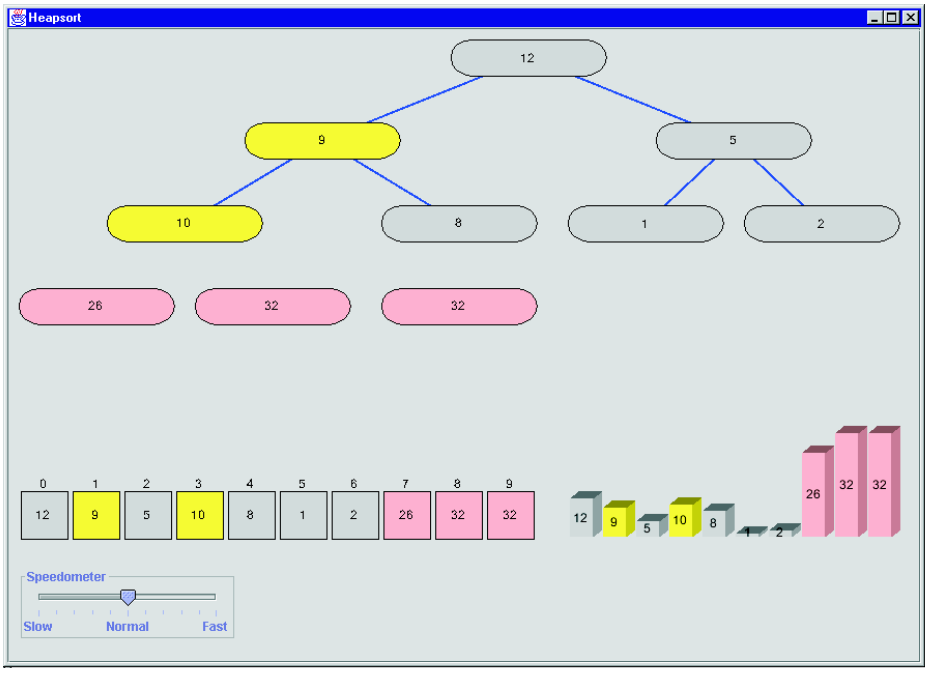


Abbildung5. Algorithmanimation von Heapsort

5.1 Vordefinierte Sichten

GANILA und das Basispaket stellen eine Menge von vordefinierten Sichten zur Verfügung, die durch ein GANILA-Sprachkonstrukt einfach in das Programm eingebunden werden. Dabei sind die Methoden der Sichten die Eventhandler der Events, die sie selbst verstehen. Andere Events werden einfach ignoriert.

HTMLView: In GANILA gibt es die Möglichkeit, Programmpunkte mit HTML-Dokumenten zu verbinden. Einerseits können HTML-Seiten über eine URL an diesem Programmpunkt durch einen kleinen HTML-Browser (IceBrowser-Java-Bean) angezeigt werden. Andererseits können Informationen, die zur Laufzeit an diesen Programmpunkten zur Verfügung stehen, durch Parameterübergabe an CGI-Skripten zu einem Server transferiert werden.

GraphView: Die GraphView bietet mehrere Graphlayoutalgorithmen, siehe auch Abschnitt 6.3. Knoten und Kanten können beliebig hinzugefügt bzw. gelöscht werden. Hierbei können die Knoten (fast) beliebige AWT-Komponenten sein.

CodeView: Die CodeView ermöglicht eine textuelle Sicht auf das auszuführende Programm und zeigt den aktuell ausgeführten Programmpunkt an.

Aura: Erhält über einen Event die URL einer Sounddatei und spielt diese ab. Start, Stop, Schleifen, Lautstärkeregelung, etc. werden ebenfalls über das Senden von Interesting Events gesteuert.

6 Design der visualisierenden Programmiersprache GANILA

Für die Spezifikation und Animation der Generatoren wurde die Sprache GANILA entworfen. GANILA erweitert Java um Interesting Events, parallele Ausführung von Programmpunkten, Annotationen für vorausschauendes Graphenlayout, Einbinden von Views und um Break- und Backtrackpunkte. GANILA wird durch einen Compiler (GAJA) nach Java übersetzt. Für jeden annotierten Programmpunkt kann diese Annotation zur Laufzeit der Animation über eine Benutzerschnittstelle ein- und ausgeschaltet werden. Die resultierenden Einstellungen können sowohl für den ganzen Algorithmus, als auch für jede View einzeln definiert werden, falls keine Parallelausführung eingestellt ist.

6.1 Interesting Events

Programm 1 zeigt ein GANILA Programm für eine einfache Operation: das Vertauschen der Inhalte zweier Feldelemente `a[i]` und `a[j]`. Interesting Events haben den Präfix `*IE_` und übertragen lokale Informationen über ihre Argumente zu den einzelnen Views, die diese Events bearbeiten. Ein solcher Event wird zuerst zu einer zentralen Kontrolle gesendet. Jede View registriert sich bei dieser Kontrolle und die Kontrolle leitet alle Interesting Events zu den registrierten Views weiter. Ein Eventhandler einer View kann wiederum neue Views kreieren und muß diese bei der Kontrolle registrieren.

Programm 1 GANILA-Code einer Vertauschungs-Methode

```
*{ *IE_MakeTemporary(1,i);
   help1 = a[i];
*} *||
*{ *IE_MakeTemporary(2,j);
   help2 = a[j];
*}

*{ *IE_MoveTemporary(j,1);
   a[j] = help1;
*} *||
*{ *IE_MoveTemporary(i,2);
   a[i] = help2;
*}
```

6.2 Parallele Ausführung

In GANILA ist es möglich, Programmpunkte durch `*{ ... }*` zu gruppieren und Programmpunkte bzw. Gruppen von Programmpunkten durch den Operator `*||` parallel auszuführen. In Programm 1 werden zuerst die Zuweisungen zu `help1` and `help2` parallel ausgeführt, dann die Zuweisungen zu `a[i]` und `a[j]` und ebenso ihre entsprechenden Interesting Events. Als Resultat werden ihre Animationen parallel dargestellt. Zu Beachten ist, daß wenn wir nur eine Hilfsvariable `help` zum Vertauschen verwenden würden, Datenabhängigkeiten eine parallele Ausführung nicht erlauben würden. D.h. der Algorithmus muß hier leicht umgeschrieben werden, damit die gewünschte Animation möglich ist.

6.3 Vorausschauendes Graphenlayout

Oft führen Animationen für Algorithmen, die Graphen verändern, d.h. Knoten oder Kanten hinzufügen oder löschen, zu Konfusionen, weil nach jeder Änderung ein neues Layout des aktuellen Graphen berechnet wird. In diesem neuen Layout werden Knoten zu unterschiedlichen Koordinaten bewegt, ohne daß der Algorithmus diese Knoten verändert hat. Daher ist es für den Anwender nicht sofort klar, welche Änderungen des Graphen durch den Algorithmus vorgenommen wurden. GANILA unterstützt Mechanismen für vorausschauendes Graphenlayout, d.h. ein Graph wird zur Zeit t_1 gezeichnet, wobei Informationen darüber verwendet werden, wie dieser Graph zu einem späteren Zeitpunkt t_2 aussehen wird.

Programm 2 Skizze des GANILA-Code für vorausschauendes Graphenlayout einer "RA→NEA" Algorithmenanimation

```
⋮
*RECORD;
  Code fuer die Umwandlung eines regulaeren
  Ausdrucks in einen nichtdeterministischen Automaten
*REPLAY;
*PLAY;
  Rest des Programms
⋮
```

Abbildung 6 zeigt, wie dieser Mechanismus verwendet werden kann, um die Umwandlung eines regulären Ausdrucks zu einem nichtdeterministischen Automaten (RA→NEA) zu animieren. In Programm 2 ist eine Skizze des entsprechenden GANILA-Codes angegeben. Die Instruktionen `*RECORD` und `*PLAY` werden verwendet, um einen bestimmten Modus zur Ausführung der graphischen Primitive auszuwählen. Im PLAY Modus werden alle interesting events unmittelbar ausgeführt. Im RECORD Modus werden Änderungen im internen Zustand vorgenommen, allerdings ohne graphische Ausgabe. Alle Interesting Events werden

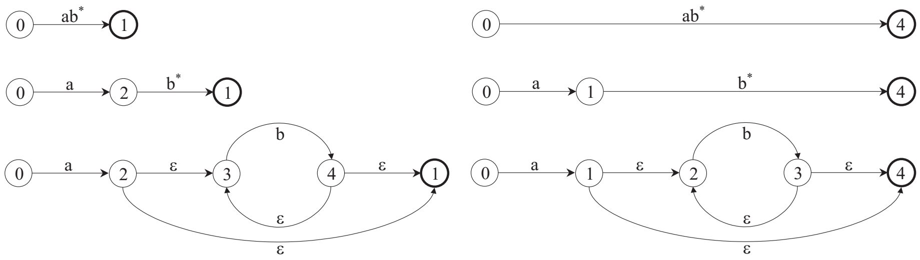


Abbildung6. Ad-Hoc und vorausschauendes Graphenlayout für endliche Automaten

dabei gespeichert. Die Instruktion `*REPLAY` ruft alle gespeicherten Events mit dem endgültigen Zustand als zusätzliches Argument wieder auf. Nun werden alle Zustandsänderungen ausgeführt und graphischer Output erzeugt.

6.4 Break- und Backtrackpunkte

Programmpunkte können mit `*BREAK` als Breakpunkte markiert werden. Wenn die Ausführung des Algorithmus solch einen Breakpunkt erreicht, wird die Ausführung unterbrochen und der Benutzer kann z.B. den gegenwärtigen Zustand untersuchen, fortfahren oder den Algorithmus schrittweise ausführen.

Backtrackpunkte werden mit `*SAVE` markiert. Wenn die Ausführung eines Algorithmus einen solchen Punkt erreicht, kopiert das System den aktuellen Zustand und speichert ihn in einer History.

Backtrackpunkte sind ein Mittel für Rückwärtsausführung eines Algorithmus oder Wiederholung eines Algorithmus von einem vorhergehenden Punkt aus, unter Umständen mit geänderten Einstellungen. Ein anderes Mittel ist die Wiederholung aller vom Startpunkt des Algorithmus aufgezeichneten Interesting Events. Dies ist zwar eine zeitintensivere, aber weniger speicherverzehrende Alternative, die vom System angeboten wird.

7 Abschlußbemerkungen

Wir haben die meisten Teile des Basispakets implementiert und eine formale Beschreibung des GANILA Compilers GAJA erstellt. Die GANILA Quellen des Heapsort Beispiels wurden durch den GAJA-Compiler in ein Java Programm konvertiert. Eine prototypische Implementierung des Compilers, sowie Beispielvisualisierungen, die durch den Compiler erzeugt werden (d.h. Heapsort und Berechnungsmodell der endlichen Automaten) werden zum Zeitpunkt des Workshops verfügbar sein. Mehr Informationen über das GANIMAL Projekt und weitere Beispiele sind unter [8] zu finden.

Das GANIMAL Framework und insbesondere GANILA bieten eine Menge von mächtigen Features. Wir erwarten, daß die beachtliche Investitionen an Zeit und Anstrengungen bei deren Design und Implementierung sich auszahlen werden, wenn wir beginnen, Inhalte für die Lernsoftware als solche zu kreieren. Letztes Jahr haben wir in Experimenten mit Studenten eine Lernsoftware über endliche Automaten ohne generativen Teil evaluiert. Wir beabsichtigen, die gesammelten Ergebnisse mit denen der Evaluation unserer neuen Lernsoftware zu vergleichen, die wir mit Hilfe des GANIMAL Frameworks erstellt haben.

Literatur

1. B. Braune, S. Diehl, A. Kerren, R. Wilhelm. *Animation of the Generation and Computation of Finite Automata for Learning Software.* To appear in Proceedings of Workshop of Implementing Automata WIA'99, Potsdam, Germany, July, 1999.
2. M. H. Brown. *Algorithm Animation.* MIT Press, 1987.
3. M. H. Brown. *Zeus: A System for Algorithm Animation and Multiview Editing.* In IEEE Workshop on Visual Languages, pp. 4-9, 1991. Also appeared as SRC Research Report 75.
4. S. Diehl, T. Kunze, A. Placzek. *GANIMAM: Generation of Interactive Animations of Abstract Maschines.* In Proceedings of "Smalltalk und Java in Industrie und Ausbildung STJA'97" (in German), pp. 185-190, Erfurt (Germany), 1997.
5. S. Diehl, T. Kunze. *Visualizing Principles of Abstract Maschines by Generating Interactive Animations.* In Proceedings of Workshop on Principles of Abstract Machines, Pisa, Italy, 1998.
6. S. Diehl, T. Kunze. *Visualizing Principles of Abstract Machines by Generating Interactive Animations.* To appear in Future Generation Computer Systems, Elsevier, 1999.
7. S. Diehl, A. Kerren. *Increasing Explorativity by Generation.* To appear in Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications, EDMEDIA-2000, AACE, Montreal, Canada, 2000.
8. http://www.cs.uni-sb.de/GANIMAL
9. http://www.cs.uni-sb.de/RW/users/ganimal/GANIFA
10. A. Kerren. *Animation of the Semantical Analysis.* Master Thesis (in German), University of Saarland, Saarbrücken (Germany), 1997.
11. A. Kerren. *Animation of the Semantical Analysis.* In Proceedings of „8. GI-Fachtagung Informatik und Schule INFOS99" (in German), pp. 108-120, Informatik aktuell, Springer, 1999.
12. http://www.mpi-sb.mpg.de/~marco/java.html.
13. R. Schulmeister. *Foundations of Hypermedial Learning Systems.* Addison Wesley (in German), Bonn (Germany), 1996.
14. J. T. Stasko. *A Framework and System for Algorithm Animation.* Computer, 18(2), pp. 258-264, 1990.
15. J. T. Stasko. *The Path-Transition Paradigm: A Practical Methodology for Adding Animation to Program Interfaces.* Journal of Visual Languages and Computing (1), pp. 213-236, 1990.
16. J.T Stasko, J. Domingue, M.H. Brown, B.A. Price. *Software Visualization.* The MIT Press, 1997.
17. A. Y. Tal, D. P. Dobkin. *Visualization of Geometric Algorithms.* IEEE Transactions on Visualization and Computer Graphics, Vol. 1, Num. 2, 1995.
18. R. Wilhelm, D. Maurer. *Compiler Design: Theory, Construction, Generation.* Addison-Wesley, 1995.