Generierung interaktiver Animationen für den Übersetzerbau

Dissertation

Zur Erlangung des Grades eines
Doktors der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

von

Diplom-Informatiker
Andreas Kerren

Juli, 2002

Tag des Kolloquiums:	16. September 2002
Dekan:	Prof. Dr.-Ing. P. Slusallek
Gutachter:	Prof. Dr. R. Wilhelm Prof. Dr. H. Hagen
Vorsitzender:	Prof. Dr.-Ing. G. Weikum

Berichte aus der Informatik

Andreas Kerren

Generierung interaktiver Animationen für den Übersetzerbau

Shaker Verlag
Aachen 2002

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Kerren, Andreas:
Generierung interaktiver Animationen für den Übersetzerbau /
Andreas Kerren. Aachen : Shaker, 2002
(Berichte aus der Informatik)
Zugl.: Saarbrücken, Univ., Diss., 2002
ISBN 3-8322-0899-2

Copyright Shaker Verlag 2002
Alle Rechte, auch das des auszugsweisen Nachdruckes, der auszugsweisen oder vollständigen Wiedergabe, der Speicherung in Datenverarbeitungsanlagen und der Übersetzung, vorbehalten.

Printed in Germany.

ISBN 3-8322-0899-2
ISSN 0945-0807

Shaker Verlag GmbH • Postfach 101818 • 52018 Aachen
Telefon: 02407 / 95 96 - 0 • Telefax: 02407 / 95 96 - 9
Internet: www.shaker.de • eMail: info@shaker.de

Zusammenfassung

Der Einsatz generischer bzw. generativer Verfahren zur Entwicklung und Anwendung interaktiver Lehr- und Lernsoftware wurde in Industrie und Ausbildung bislang kaum untersucht. Diese Arbeit befaßt sich mit der Visualisierung bzw. Animation von Algorithmen und Programmen, insbesondere von komplexen Berechnungsmodellen im Übersetzerbau, sowie mit der Verwendung generierter Animationen in Lehr- und Lernsoftware. Eigenschaften klassischer und aktueller Algorithmenanimationssysteme werden hierzu untersucht und klassifiziert. Ausgehend von den Eigenschaften dieser Systeme wird eine Reifikationstechnik vorgestellt, die der Softwarevisualisierung vollkommen neue Möglichkeiten eröffnet, z. B. die Animationssteuerung der Besuchsfolge von Schleifen oder die Vermischung von post mortem- und live-Visualisierungen. Zusätzlich unterstützt diese Technik zahlreiche mächtige Konzepte, die in klassischen Systemen ebenfalls realisiert wurden, wie etwa die parallele Ausführung von Programmpunkten mitsamt der damit verbundenen Animationen.

Die Technik wurde in einem Framework, genannt Ganimal, implementiert. Es besteht aus der Animationsbeschreibungssprache Ganila mit zugehörigem Compiler und einer Laufzeitumgebung. Ganila erweitert die Programmiersprache Java um Animationsannotationen und ermöglicht so die Spezifikation komplexer, interaktiver Animationen. Mit Ganimal entwickelte Animationen wurden in eine Lehr- und Lernsoftware für ein Themengebiet des Übersetzerbaus integriert, die lokal und über das Internet verwendet werden kann. Im Übersetzerbau oftmals angewandte Techniken zur Generierung von Software wurden dazu genutzt, die in der Lernsoftware enthaltenen Animationen teilweise automatisch zu erzeugen. Dieser generative Ansatz wird in einen lerntheoretischen Rahmen eingeordnet und eine daraus resultierende, neue Form des explorativen Lernens propagiert. Eine Evaluation der Lernsoftware in Form von Lernexperimenten mit über 100 Probanden belegt ihre Lerneffizienz und schließt die Arbeit ab.

Abstract

The use of generic and generative methods for the developement and application of interactive educational software is a relatively unexplored area in industry and education. In this thesis novel concepts for visualization and animation of algorithms or programs are presented, especially the animation of complex computational models for compiler design, as well as the integration of the generated animations in educational software systems. Concepts of classical and recent algorithm animation systems are surveyed and classified. Based on these characteristics, a reification technique is introduced that provides novel possibilities in software visualization, such as animation control of loops or mixing live and post mortem visualization. This technique also supports numerous powerful concepts, that have previously been implemented by classical algorithm animation systems, e. g. parallel execution of program points and associated animations.

The technique has been implemented in a framework, called Ganimal, consisting of the animation description language Ganila, a compiler and a runtime environment. Ganila extends the programming language Java by animation annotations and it supports the specification of complex interactive animations. Animations developed with Ganimal have been embedded into a learning system for compiler design, which can be used locally or online. Advanced techniques for software generation as used in compiler construction have been applied to the automatic generation of animations contained in the learning system. This generative approach is positioned within the scope of several educational theories and as a result a new way of explorative learning is propagated. An evaluation of the learning system based on experiments with more than 100 participants proves its efficiency and concludes this work.

Ausführliche Zusammenfassung

Die Entwicklung interaktiver, multimedialer Lehr- und Lernsoftware hat während der letzten Jahre in Industrie und Ausbildung große Beachtung erfahren. Fördergelder in Millionenhöhe wurden landesweit für die Erforschung und Entwicklung neuer Medien in der Lehre bereitgestellt. Generische bzw. generative Ansätze in der Implementierung und Anwendung dieser Lehr- und Lernsoftware allerdings sind bisher kaum untersucht, geschweige denn realisiert worden. Meist werden Autorensysteme für deren Implementierung eingesetzt, aber auch vermehrt HTML-Editoren. Dabei liegen die Vorteile generischer und generativer Techniken u. a. in einem hohen Grad an Wiederverwendbarkeit von Systemteilen sowie in der Reduzierung hoher Entwicklungskosten.

Die vorliegende Arbeit befaßt sich mit der Visualisierung bzw. Animation von Algorithmen und Programmen, insbesondere von komplexen Berechnungsmodellen im Übersetzerbau. Berechnungsmodelle, wie etwa endliche Automaten oder abstrakte Maschinen, lassen sich über interaktive Animationen besonders anschaulich vermitteln, sei es im Frontalunterricht oder im Selbststudium. Weiterhin diskutiert diese Arbeit die Verwendung automatisch generierter Animationen in Lehr- und Lernsoftware für den Übersetzerbau.

Ausgehend von einer Gegenüberstellung der Eigenschaften klassischer und neuerer Algorithmenanimationssysteme wird eine Reifikationstechnik vorgestellt, die die Assoziation von Metainformationen mit jedem Programmpunkt des zu animierenden Algorithmus unterstützt. Zur Laufzeit der Animation bildet jeder Programmpunkt ein Objekt mit manipulierbaren Eigenschaften. Dieses Verfahren eröffnet der Softwarevisualisierung vollkommen neue Möglichkeiten:

- die Animationssteuerung der Besuchsfolge von Schleifen,
- die Vermischung von post mortem- und live-Visualisierungen,
- die Einbindung alternativer Interesting Events,
- die Auswahl alternativer Codeblöcke und
- die Überprüfung von Invarianten an Programmpunkten.

Jede dieser Eigenschaften kann zur Laufzeit der Algorithmenanimation geändert werden, indem der Anwender entsprechende Programmpunkteinstellungen modifiziert. Zusätzlich unterstützt die Reifikationstechnik zahlreiche mächtige Konzepte, die in klassischen Systemen ebenfalls realisiert wurden, wie z. B.

- Interesting Events,
- verschiedene Sichten auf den auszuführenden Algorithmus,
- parallele Ausführung von Programmpunkten mitsamt der damit verbundenen Animationen oder
- Haltepunkte.

Das Verfahren wurde in einem JAVA-basierten Framework, genannt GANIMAL, implementiert. Es besteht aus der Animationsbeschreibungssprache GANILA mit zugehörigem Compiler und einer Laufzeitumgebung, für die der Compiler entsprechenden Programmcode generiert. GANILA erweitert die Programmiersprache JAVA um Animationsannotationen und ermöglicht die Spezifikation komplexer, interaktiver Animationen. Die Arbeit führt die verschiedenen GANILA-Sprachkonstrukte ein und motiviert anhand vieler Beispiele die Reifikation von Programmpunkten. Für die Übersetzung von GANILA-Programmen in JAVA-Quellcode sind eine Reihe von statischen Analysen notwendig. Nach einer Diskussion dieser Analysen werden Übersetzungsschemata angegeben, die den Übersetzungsprozeß für eine Auswahl an Sprachkonstrukten semiformal beschreiben. Von Seiten der Implementierung sind Analysephase und Codegenerierung gemäß dem sogenannten Besuchermuster (Visitor Pattern) strukturiert. Der generierte JAVA-Quellcode bildet zusammen mit der GANIMAL-Laufzeitumgebung interaktive, multimediale Animationen des Eingabeprogramms. Die Laufzeitumgebung wird komponentenweise anhand eines weiteren Entwurfsmusters, MVC (Model/View/Controller) genannt, strukturiert und erläutert. Sie unterstützt den Animationsentwickler in der Programmierung eigener Sichten auf den Algorithmus und stellt ein Basispaket mit graphischen Primitiven sowie eine Anzahl vordefinierter Standardsichten zur Verfügung. Eine graphische Benutzerschnittstelle dient zur Steuerung der Animation und zur Modifikation der o. g. Programmpunkteinstellungen.

Die Anwendbarkeit des Frameworks wird anhand einer Beispielimplementierung der Animation des Heapsort-Algorithmus verdeutlicht. Mit GANIMAL entwickelte Animationen wurden in eine Lehr- und Lernsoftware für ein Themengebiet des Übersetzerbaus integriert: Das Lernsystem GANIFA ist ein HTML-basiertes elektronisches Textbuch, das die Generierung endlicher Automaten zum Gegenstand hat. Es kann lokal oder über das Internet verwendet werden. Im Übersetzerbau oftmals angewandte Techniken zur Generierung von Software wurden dazu genutzt, die in diesem Lernsystem enthaltenen Animationen teilweise automatisch zu erzeugen. Es ist möglich, sowohl den Generierungsprozeß des über einen beliebigen regulären Ausdruck spezifizierten endlichen Automaten als auch das Akzeptanzverhalten des generierten Automaten für ein beliebiges Eingabewort zu animieren. Dieser generative Ansatz wird in einen lerntheoretischen Rahmen eingeordnet und eine daraus resultierende, neue Form des explorativen Lernens propagiert. Die Arbeit erläutert hierzu grundlegende Theorien des Lernens und ordnet exemplarisch einige Lernsysteme für den Übersetzerbau, einschließlich GANIFA, in vier verschiedene Explorationsstufen ein. Es wird gezeigt, wie der diskutierte generative Ansatz neue Übungsformen unterstützt.

Eine Evaluation der generativen Lernsoftware GaniFA in Form von Lernexperimenten mit über 100 Probanden schließt die Arbeit ab. Teilnehmer vier verschiedener Versuchsgruppen studierten die Theorie zur Generierung endlicher Automaten mit jeweils unterschiedlichen Lernmitteln: im Frontalunterricht, mit einem Lehrbuch sowie mit Hilfe zweier Lernsysteme mit und ohne generativen Teil. Die Ergebnisse belegen, daß GaniFA hinsichtlich der Lerneffizienz besser als der Frontalunterricht und im Vergleich mit dem Lehrbuch nahezu gleich gut abschnitt. Darüber hinaus empfanden die Probanden die Möglichkeit, Animationen über die eingegebenen regulären Ausdrücke selbst zu erzeugen und auf diese Weise nachzuvollziehen, wie der Generierungsprozeß endlicher Automaten funktioniert, als in hohem Maße motivationssteigernd. GaniFA wird in der frei verfügbaren Online-Version mehrmals täglich besucht.

Verschiedene Aspekte dieser Arbeit wurden in mehreren Artikeln, Buchbeiträgen und Tagungsbänden veröffentlicht [DK02a, DGK02, KS02, DKW01, DK01, DK00a, DK02b] sowie auf einer Reihe von internationalen Konferenzen und Workshops vorgestellt.

Extended Abstract

During the past years, the developement of interactive, multimedial learning software has become more and more relevant in industry and education. Research funds of several millions EUR were nationwide supplied for the research and development of new educational media. However, generic and generative approaches for the implementation and application of such educational software systems are neither in the center of actual research interests nor have they been realized frequently. In most cases commercial authoring systems are used for their implementation. Recently, these authoring systems are replaced by so-called HTML-editors. Important advantages of generic and generative techniques over authoring systems are the high level of system reusability and the reduction of development costs.

In this thesis novel concepts for visualization and animation of algorithms or programs are described, especially the animation of complex computational models for compiler design. Computational models, such as finite automata or abstract machines, can be clearly taught by using interactive animations in traditional instruction as well as in self-study. Furthermore, this work presents the integration of automatically generated animations into educational software systems for compiler design.

Based on a short survey of classical and recent algorithm animation systems, a reification technique is introduced that allows arbitrary meta-information to be associated with the program points of the underlying algorithm. During execution, each program point corresponds to an object whose meta-information can be manipulated. As a consequence, this technique provides novel possibilities in software visualization, such as

- the animation control of loops,
- the mixing of live and post mortem visualization,
- the specification of alternative interesting events,
- the choice of alternative code blocks, and
- the visualization of invariants for program points and blocks.

Each of these features can be manipulated at the runtime of algorithm animation by changing settings attached to each program point. This technique also supports numerous powerful concepts that have previously been implemented by classical algorithm animation systems, e. g.

- interesting events,
- several views on the algorithm,
- parallel execution of program points and associated animations or
- breakpoints.

The reification technique has been implemented in a framework, called GANIMAL, consisting of the animation description language GANILA, a compiler, and a runtime environment, in which the compiled programm code is executed. GANILA extends the programming language JAVA by animation annotations and it supports the specification of complex, interactive animations. In this work, different constructs of the GANILA language are introduced and the reification of program points is motivated. To translate GANILA programs into JAVA source code, a number of static analyses are necessary. After a discussion of these analyses some code translation schemes are given, which describe the compilation process with a choice of GANILA constructs in a semi-formal way. Analysis phase and code generation of the implementation are structured in accordance with the so-called visitor pattern. The generated JAVA source code produces the interactive animations for the input program in combination with the runtime environment. The runtime environment, that is precisely described in this thesis, implements another design pattern, called MVC (Model/View/Controller). It supports the animation developer to implement user-definied views on the algorithm and offers a base package which consists of a set of primitive methods for graphical output as well as a number of predefined standard views. A graphical user interface can be used to control the animation and to manipulate the settings.

The applicability of GANIMAL is demonstrated by an example implementation of an algorithm animation for Heapsort. A more complex example represents the embedding of animations developed with GANIMAL into a learning system for compiler design: GANIFA is an HTML-based electronic textbook on the theory of generating finite automata. It can be used locally or online. In the area of compiler construction frequently applied techniques for software generation have been used for the automatic generation of animations the learning system contains. It is possible to visualize both the generation process of the finite automaton specified by a regular expression and the acceptance behaviour of the generated automaton on an arbitrary input word. This generative approach is discussed within the scope of several educational theories and as a result a new way of exploratory learning is propagated. To this end, seminal educational theories are explained and four levels of increasing exploration in compiler design are introduced. For each level one educational system is exemplified. The generative approach corresponds to the highest level of exploration and provides new kinds of excercises.

An evaluation of the generative learning system GANIFA based on experiments with more than 100 participants proves its efficiency and concludes this work. Four learner groups studied the theory of generating finite automata each with the help of different methods: classical instruction, text book and two learning systems with and without generative part. The results prove, that the learning efficiency of GANIFA is higher than

the one of classical instruction and nearly as good as the text book's. The participants esteemed the possibilities to enter regular expressions and to generate animations of finite automata in order to understand how finite automata work. They confirmed that motivation increased because of using this system. The system's freely available online version is usually visited several times a day.

Many aspects of this work have been published in various articles, book contributions and conference proceedings [DK02a, DGK02, KS02, DKW01, DK01, DK00a, DK02b] and were presented at a number of international conferences and workshops.

Danksagung

Mein Dank gilt Prof. Dr. Reinhard Wilhelm, der mir während der letzten vier Jahre an seinem Lehrstuhl ein ausgezeichnetes Umfeld für die Erstellung dieser Arbeit zur Verfügung stellte. Seine Kommentare und Ratschläge eröffneten mir viele neue Erkenntnisse und unterschiedliche Sichtweisen. Weiterhin gab er mir stets großen Freiraum und ermöglichte mir mehrere Reisen zu Workshops und Konferenzen im In- und Ausland, in denen ich meine Arbeiten vorstellen durfte. Als Zweitgutachter dieser Dissertation stellte sich Prof. Dr. Hans Hagen zur Verfügung, dem ich hiermit ebenfalls mein Dankeswort aussprechen möchte.

Ich möchte Dr. Stephan Diehl für seine Betreuung und Ratschläge danken, die er mir in den letzten Jahren angedeien ließ. Diese Arbeit, von der Konzeption über die Implementierungsphase bis hin zur Niederschrift, hat von diesen Momenten, seiner Einsicht und Erfahrung in weitem Maße profitiert. Mein herzlicher Dank gilt auch allen Studenten und Mitarbeitern des Ganimal-Projekts. Insbesondere danke ich Beatrix Braune, die in der Anfangsphase des Projekts viele wertvolle Hinweise zur Gestaltung des Ganimal-Frameworks gab. Gleiches gilt für Thomas Kunze, der weite Teile des graphischen Basispakets programmierte. Ich danke Carsten Görg für seine Implementierung der Layoutalgorithmen für Graphen, einschließlich der Entwicklung des Vorausschauenden Graphlayouts, sowie Torsten Weller für die Programmierung einiger Laufzeitsystemkomponenten und des GaniFA-Applets. Für die effiziente Zusammenarbeit hinsichtlich der Evaluation der in dieser Arbeit vorgestellten Lernsysteme bedanke ich mich neben den bereits genannten Personen ganz besonders bei Julia Kneer und Christoph Glasmacher. Beide leisteten zur Auswertung der erhobenen Daten einen sehr großen Beitrag.

Meine besondere Dankbarkeit geht an meine Partnerin, Ulla Huber, für sorgfältiges Korrekturlesen sowie für ihre Unterstützung, Nervenstärke und Geduld während der Arbeit an meiner Dissertation.

Andreas Kerren

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Generative Lernsoftware für den Übersetzerbau 2
1.2 Das GANIMAL-Framework 4
1.3 Aufbau der Arbeit 6

2 Verwandte Systeme zur Algorithmenanimation **9**
2.1 Klassische Systeme und Konzepte 10
2.2 Spezifikationstechnik 12
2.2.1 Ereignisgetriebene Animation 12
2.2.2 Zustandsgetriebene Animation 13
2.2.3 Visuelle Programmierung 14
2.2.4 Automatische Animation 14
2.3 Visualisierungstechnik 15
2.3.1 3D-Algorithmenanimation 16
2.3.2 Auralisierung 17
2.3.3 Webeinsatz 17
2.4 Sprachparadigmen 18
2.4.1 Imperative Programmiersprachen 18
2.4.2 Funktionale Programmiersprachen 18
2.4.3 Objektorientierte Programmiersprachen 18
2.4.4 Logische Programmiersprachen 19
2.5 Domänenspezifische Animationen 19
2.5.1 Algorithmische Geometrie 20
2.5.2 Nebenläufige Programme 20
2.5.3 Realzeitanimationen 20
2.5.4 Berechnungsmodelle 21
2.5.5 Animation von Beweisen 22

3 Die Animationsbeschreibungssprache GANILA **23**
3.1 Sprachbeschreibung 24
3.1.1 Semantik der GANILA-Konstrukte 25
3.2 Reifikation von GANILA-Programmen 31
3.2.1 Programmpunkteinstellungen 31
3.2.2 Anweisungen 32

3.2.3 Kontrollstrukturen . . . 36
3.2.4 Ausgewählte GANILA-Konstrukte . . . 37
3.2.5 Verwandte Arbeiten und Techniken . . . 48
3.3 Zusammenfassung . . . 49

4 Der Compiler GAJA 51
4.1 Architektur . . . 52
4.2 Generierung des Parsers . . . 55
4.3 Statische Analyse . . . 56
4.3.1 Berechnung von globalen Klasseneigenschaften . . . 56
4.3.2 Berechnung lokaler Eigenschaften von Methoden und Feldern . . . 58
4.4 Generierung von JAVA-Code . . . 61
4.4.1 Ausdrücke . . . 64
4.4.2 Ausdrucksanweisungen . . . 71
4.4.3 Kontrollflußanweisungen . . . 72

5 Die GANIMAL-Laufzeitumgebung 77
5.1 Realisierung einer MVC-Architektur . . . 77
5.2 Graphische Benutzerschnittstelle . . . 79
5.2.1 Steuerung einer Animation . . . 80
5.2.2 Modifikation der Programmpunkteinstellungen . . . 80
5.3 Visualisierungskontrolle . . . 82
5.3.1 Steuerung des Algorithmus . . . 82
5.3.2 Verarbeitung der Programmpunkteinstellungen . . . 84
5.4 Sichten . . . 88
5.4.1 Benutzerdefinierte Sichten . . . 89
5.4.2 Standardsichten . . . 90
5.4.3 Vorausschauendes Layout von Graphen . . . 98
5.4.3.1 Beschreibung des Algorithmus . . . 99
5.4.3.2 Anwendung in der Algorithmenanimation . . . 100

6 Anwendungen 103
6.1 Vorausgegangene Entwicklungen . . . 103
6.1.1 Animation der lexikalischen Analyse . . . 103
6.1.2 Animation der semantischen Analyse . . . 105
6.1.3 Generierung interaktiver Animationen von abstrakten Maschinen 107
6.2 Animation des Heapsort-Algorithmus . . . 111
6.2.1 Implementierungssicht . . . 111
6.2.2 Anwendungssicht . . . 115
6.3 Animation der Generierung endlicher Automaten . . . 117
6.3.1 Elektronisches Textbuch . . . 117
6.3.2 GANIFA-Applet . . . 118
6.3.2.1 Animationen . . . 120
6.3.2.2 Adaptabilität . . . 122

6.3.2.3 Implementierungsaspekte 124

7 Lerntheoretische Aspekte 125
7.1 Terminologie 125
7.2 Lerntheorien 126
7.2.1 Instruktionalismus 127
7.2.1.1 Programmierte Instruktion 127
7.2.1.2 Instruktionstheorie 128
7.2.2 Konstruktivismus 128
7.2.2.1 Mikrowelten und entdeckendes Lernen 129
7.2.2.2 Konstruktive Wissensaneignung 130
7.2.2.3 Situiertes Lernen 131
7.2.3 Schlußfolgerungen 131
7.2.3.1 Lernerkontrolle vs. Programmkontrolle 131
7.2.3.2 Instruktionsmethoden 132
7.3 Lernen mit Multi- und Hypermedia 134
7.3.1 Multimedia 134
7.3.1.1 Medien 135
7.3.1.2 Interaktivität 137
7.3.2 Problemfelder 140
7.4 Stufen explorativen Lernens 141
7.4.1 Lernsoftware für den Übersetzerbau 141
7.4.2 Beschreibung des Lernmodells 143
7.4.3 Fallstudien 146
7.4.3.1 Statischer Ansatz 146
7.4.3.2 Interaktiver Ansatz 147
7.4.3.3 Generativer Ansatz erster Ordnung 148
7.4.3.4 Generativer Ansatz zweiter Ordnung 148

8 Evaluation 151
8.1 Qualitative vs. quantitative Verfahren 152
8.2 Statistische Grundlagen 152
8.2.1 Deskriptive Statistik 153
8.2.2 Schließende Statistik 155
8.3 Vorstudie zum Lernsystem ADLA 157
8.3.1 Einschränkung des Lernstoffs 158
8.3.2 Experimenteller Ablauf 158
8.3.3 Auswertung 159
8.4 Evaluation von GANIFA 161
8.4.1 Experimenteller Ablauf 161
8.4.2 Ergebnis des Lernexperiments 162
8.4.3 Ergebnis der Lehrmittelbewertung 166
8.4.4 Zugriffstatistik des Textbuchs 169
8.4.5 Zusammenfassung 169

9 Abschließende Bemerkungen **171**
9.1 Zusammenfassung 171
9.2 Ausblick 172

A Quellcode der Animation von Heapsort **175**
A.1 Implementierung des Heapsort-Algorithmus 175
A.1.1 Spezifikation in GANILA 175
A.1.2 Generierte Module 178
A.2 Implementierung der Balkensicht 185
A.3 Integration der graphischen Komponenten 186

B Deskriptive Statistik der Evaluation **189**
B.1 Leistungstest 189
B.2 Analyse des Bewertungsfragebogens 196
B.3 Auswertung des Fragebogens zum Lernverhalten 206

Abbildungsverzeichnis

1.1 Entwicklung animierter Generatoren und Übersetzerphasen. 3
1.2 Das GANIMAL-Framework. 5

2.1 Klassifikation der Softwarevisualisierung. 10
2.2 POLKA-Animation des Quicksort-Algorithmus. 11
2.3 Deklarative Spezifikation in ALPHA und die sich ergebende Animation. . 13
2.4 ZEUS3D-Animation des SSSP-Algorithmus. 16
2.5 PARADE-Animation eines Programms mit Threads. 21

3.1 Ausschnitt der GANILA-Grammatik. 24
3.2 Graphische Benutzerschnittstelle zur Modifikation der PPEs. 33

4.1 Überblick über die Architektur des Compilers GAJA. 52
4.2 Struktur des Besuchermusters. 54
4.3 Überblick über die Generierung des GANILA-Parsers. 55
4.4 Typhierarchie von GANILA. 57
4.5 Auszug aus dem dekorierten AST des Codebeispiels Example1. 62
4.6 Notation für eine annotierte Syntax. 64

5.1 Die GANIMAL-Laufzeitumgebung. 78
5.2 Die graphische Benutzerschnittstelle des GANIMAL-Frameworks. 79
5.3 Position der GUI in der GANIMAL-Laufzeitumgebung. 81
5.4 Kontrolle zur Algorithmensteuerung. 83
5.5 Kontrolle über die Verarbeitung und Ausführung von IEs. 86
5.6 Sichten im GANIMAL-Laufzeitsystem. 89
5.7 Bildschirmaufnahme der CodeView. 92
5.8 Bildschirmaufnahme der EventView. 92
5.9 Bildschirmaufnahme der HtmlView. 94
5.10 Die HtmlView mit einem eingebettetem JAVA-Applet. 94
5.11 Bildschirmaufnahme der InvariantView. 97
5.12 Bildschirmaufnahme der GraphView. 97
5.13 Ad hoc- und Vorausschauendes Layout einer Graphanimation. 101

6.1 Äquivalenz von Übergangsdiagramm und NEA (ADLA). 104
6.2 Statische Animation eines Deklarationsanalysators (ADSA) 106

6.3 Visualisierung der Überprüfung der Kontextbedingungen (AdsA). 106
6.4 Momentaufnahme einer animierten abstrakten Maschine (Ganimam). . . 108
6.5 Interaktion der Systemkomponenten von Ganimam. 109
6.6 Auszug aus einer Maschinenspezifikation. 110
6.7 GUI mit AST des Heapsort-Algorithmus. 116
6.8 Ganimal-Algorithmenanimation von Heapsort. 116
6.9 Definitionenfenster. 119
6.10 Bildschirmaufnahme des elektronischen Textbuchs. 119
6.11 Animation des Algorithmus $NEA \rightarrow DEA$. 121
6.12 Animation des Algorithmus $DEA \rightarrow minDEA$. 121

7.1 Zusammenhang zwischen Spezifikation und Generierung. 143
7.2 Stufe 1 – Statischer Ansatz. 147
7.3 Stufe 2 – Interaktiver Ansatz. 147
7.4 Stufe 3 – Generativer Ansatz erster Ordnung. 148
7.5 Stufe 4 – Generativer Ansatz zweiter Ordnung. 149

8.1 Deskriptive Statistik der Leistungsindizes. 159
8.2 Streudiagramme für die Frage S13 und die Leistungsindizes. 160
8.3 Histogramme der Indexmittelwerte. 163
8.4 Leistungsdifferenzen zwischen den verschiedenen Lehrmethoden. 165
8.5 Resultate der Bewertungsfragen B1 und B2. 166
8.6 Resultat der Bewertungsfrage B5. 168
8.7 Resultate der Bewertungsfragen B6, B8 und B10. 168
8.8 Resultate der Bewertungsfragen B11 und B12. 168

Tabellenverzeichnis

5.1 Modifizierbarkeit der PPEs über die GUI. 82

6.1 Parameterübersicht. 123

7.1 Lernparadigmen und Softwaretypologie. 134
7.2 Übersetzerphasen. 142
7.3 Vier Stufen explorativen Lernens. 146

8.1 α- und β-Fehler bei statistischen Entscheidungen. 156
8.2 Deskriptive Statistik der Leistungsindizes. 163
8.3 Effektgrößen für den Vergleich von GaniFA mit den übrigen Gruppen. . 163
8.4 Signifikante Korrelationen. 165
8.5 Signifikantes Ergebnis zweier Kovarianzanalysen. 165