Eine Lerneinheit für die multimediale Lehre von Algorithmen und Datenstrukturen

Peter Aschenbrenner
peter@informatik.unibw-muenchen.de
Fachgebiet Echtzeitsysteme
Technische Universität Darmstadt

Abstract: Es gibt viele Ansätze, um Algorithmen und Datenstrukturen zu unterrichten. Unsere Lerneinheit unterstützt die Präsenzlehre durch einen konfigurierbaren, modularisierten Foliensatz, in dem Algorithmen und Datenstrukturen aus der Sicht der Softwaretechnik dargestellt werden und durch eine interaktive generische Visualisierungsumgebung, deren Instanzen aus visuellen Modellen (UML-Diagrammen) über Graphgrammatiken in einem halbautomatischen Prozess erzeugt werden können. Die Haupt-Vorteile dieses Ansatzes sind die leichte Anpassbarkeit an neuen oder abgewandelten Lehrstoff und bei den generierten Visualisierungen zusätzlich die Unterstützung von flüssiger Animation, Interaktivität und visuellem Debugging.

1 Einführung

Das Projekt MuSofT, ein BMBF-gefördertes Gemeinschaftsprojekt sieben deutscher Universitäten, hat es sich zum Ziel gemacht, multimediale Lerneinheiten für die Lehre der Softwaretechnik zu entwickeln und bereitzustellen (siehe auch [DE02]). Deshalb auch der Name MuSofT: Multimedia in der Softwaretechnik. MuSofT besteht aus 11 Teilprojekten, die Lerneinheiten entwickeln mit aufeinander abgestimmten Themen wie Anforderungsanalyse, Architekturen, Muster, Informationssysteme, Management und andere mehr.

In diesem Papier wird unser Teilprojekt vorgestellt, in dessen Rahmen wir eine multimedial aufbereitete Lerneinheit zum Thema “Algorithmen und Datenstrukturen” entwickeln,

- in der die Modellierung von Datenstrukturen und Algorithmen nach softwaretechnischen Gesichtspunkten mit grafischen Notationen (Teilen der UML) gezeigt wird
- und die in verschiedenen Umfängen für ganz unterschiedliche Studiengänge eingesetzt werden kann.

Um insbesondere den letzten Punkt zu gewährleisten, wurde der Vorlesungsanteil der Lerneinheit in logische Blöcke aufgeteilt und modularisiert. Abschnitt 2 beschreibt den zugrundeliegenden Lehrstoff und die gefundene Modularisierung samt Abhängigkeiten.

Will man das klassische Vorlesungsthema der “Algorithmen und Datenstrukturen” multimedial unterstützen, so bietet sich insbesondere die Animation und Visualisierung der behandelten Standardalgorithmen und -datenstrukturen an. Man findet zwar heute - besonders im Internet - eine große Anzahl von fertigen Animationen und Werkzeugen, die

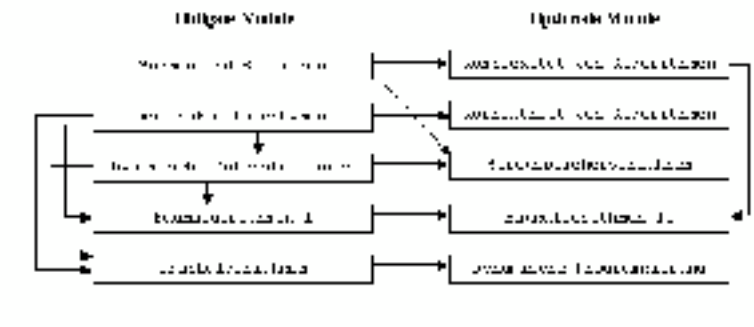

Abbildung 1: Die Modularisierung des Lehrstoffes unserer Lerneinheit

deren Erstellung vereinfachen (z.B. [CCA03], [Ani03], [Sor03], [Bub03a]). Einen kurzen (zwangsläufig unvollständigen) Überblick über solche und andere parallele Bestrebungen gibt Abschnitt 3. Allerdings bestehen in nahezu allen Fällen Einschränkungen in der Ausnutzung des möglichen Potentials der multimedialen Lehre von Algorithmen und Datenstrukturen. Solche Stolpersteine, auch aus dem Blickwinkel der Lernpsychologie, beschreibt Abschnitt 4 etwas genauer.

Unser Paradigma für die Visualisierungsumgebung (als Konsequenz aus diesen Erkenntnissen) wird in Abschnitt 5 erläutert, während Abschnitt 6 die verwendeten Werkzeuge und die technische Implementierung beschreibt. Abschnitt 7 schließlich zeigt mögliche Szenarien und bisherige Erfahrungen auf und Abschnitt 8 fasst noch mal die wichtigsten Punkte zusammen und gibt einen Ausblick auf kommende Entwicklungen.

Alles in allem stehen in diesem Papier der Gesamtüberblick über die Lerneinheit, das dahinterliegende Konzept und einige Anwendungsbeispiele im Vordergrund; wer sich speziell für den Transformationsprozess von der UML Spezifikation einer neuen Datenstruktur zu einer Instanz unserer Visualisierungsumgebung interessiert, sei auf [AS03] verwiesen.

2 Inhalt und Modularität der Lehrveranstaltung

In unserer Lerneinheit werden “Algorithmen und Datenstrukturen aus der Sicht der Softwaretechnik” vermittelt. Diese grobe Spezifikation stellt einen Rahmen dar, innerhalb dem verschiedene konkrete Ausprägungen verwirklichen werden sollen. So wurde der Vorlesungsteil unserer Lerneinheit bereits im Jahr 2002 an der Universität der Bundeswehr München erprobt, wobei es notwendig war, innerhalb der gleichen Veranstaltung zwei Gruppen von Studenten zu unterrichten: Studierende der Informatik, die die ganze vierstündige Veranstaltung besuchten und Studierende der Elektrotechnik, die nur jede zweite Doppelstunde besuchten und denen dabei natürlich trotzdem ein in sich geschlossener Lehrstoff zu bieten war.

Wir entschieden uns daraufhin, eine Aufteilung des Lehrstoffes entsprechend Abbildung 1 vorzunehmen. Die Kästen stehen dabei für Lehrblöcke in Form von konfigurierbaren Fo-

liensätzen, Flash-Animationen (die im Sommersemester 2003 im Rahmen eines Softwarepraktikums an der Technischen Universität Darmstadt zum ersten Mal erprobt werden, siehe [htt03]) und einer hochgradig konfigurierbaren Visualisierungsumgebung, die in diesem Papier noch ausführlich beschrieben wird. Die Pfeile in Abbildung 1 repräsentieren eine "ist Vorraussetzung für"-Beziehung. So müssen z.B. zuerst dynamische Datenstrukturen besprochen worden sein, bevor man über Graphalgorithmen spricht. Wie man ebenso sieht, ist kein Modul auf der linken Seite von einem der rechten Seite abhängig. Diese Tatsache hat es erlaubt, der oben beschriebenen Gruppe der Elektrotechnik-Studenten genau die links aufgeführten Themen zu lehren.

3 Related Work

Im Bereich der Softwarevisualisierung und auch insbesondere der Animation von Algorithmen und Datenstrukturen gibt es so viele Bestrebungen, Projekte und Werkzeuge, dass eine schnelle Orientierung unmöglich erscheint (eine Übersicht über aktuelle Visualisierungsbestrebungen im deutschsprachigen Raum gibt [Bub03b], für eine allgemeine Einführung in die Softwarevisualisierung siehe z.B. [SDBP98]).

Noch unübersehbarer ist die Anzahl existierender Animationen, insbesondere im Internet; in der Einführung wurden bereits 4 willkürlich herausgegriffene Beispiele genannt. Auffallend für uns war, dass die meisten existierenden Visualisierungen entweder fest verdrahtete Funktionalität haben und viel Aufwand in die Animation gesteckt wurde oder dass diese zwar erweiterbar oder sogar generierbar sind, aber dafür entweder keine flüssigen Animationen bieten (sondern diskrete Sprünge) oder so gut wie keine Interaktivität besitzen.

Deshalb haben wir uns in der ersten Projektphase entschieden, verschiedene Visualisierungs-Paradigmen zu vergleichen, auch bereits im Hinblick auf unsere speziellen Erfordernisse, die in den nächsten Abschnitten beschrieben und begründet werden. Aus Platzgründen sei hier nur erwähnt, dass wir uns nach einer Evaluation von diversen Animationsbaukästen, graphischen Entwicklungsumgebungen verschiedener Art (regelbasiert, UML-basiert, für Benutzerschnittstellen u.ä.), Simulationsumgebungen, Graphgrammatikwerkzeugen, Bibliotheken für die Visualisierung von Graphen und graphischen Debuggern dafür entschieden haben, eine Lösung mit Graphgrammatiken zu implementieren. Für einen detaillierteren Überblick über diese Evaluation sei auf [AS03] verwiesen. Der kommende Abschnitt fasst zunächst die Erfahrungen einiger lernpsychologischer Studien im Animationsumfeld zusammen, bevor wir als Konsequenz daraus unser Konzept darlegen.

4 Stolpersteine und Erfolgsfaktoren bei der Visualisierung von Datenstrukturen im Lernprozess

Lernen geschieht über die Sinneskanäle. Multimedia stellt Informationen in mehreren Repräsentationen gleichzeitig zur Verfügung. Eine einfache Vermutung wäre also, dass der Lerneffekt sich durch multimediale Lernmaterialien automatisch verbessert, weil mehrere Sinneskanäle angesprochen werden (siehe z.B. [Bal90]). Hierbei wird aber einerseits übersehen, dass ein überfrachtetes oder schlecht synchronisiertes Informationsangebot den

Lerneffekt auch schmälern kann und andererseits, dass für das Lernen und Verstehen aus kognitionspsychologischer Sicht weniger die angesprochenen Sinneskanäle, sondern die internen Codierungen und Verarbeitungsprozesse des Lerners ausschlaggebend sind (für eine genauere Diskussion siehe [Wei95]). Man spricht hier auch vom Unterschied zwischen Codierung und Modalität.

Allerdings lassen sich die intern ablaufenden kognitiven Prozesse durch die äußerliche Repräsentation des Lernstoffes bis zu einem gewissen Grad lenken: Wenn etwa einem Lerner, der sonst textuell angebotene Begriffe intern auditiv repräsentiert, zum richtigen Zeitpunkt zusätzlich ein Bild angeboten wird, kann dieses intern als zusätzliche visuelle Repräsentation dienen. Ein anderes Beispiel sind bewegte Bilder: Diese lassen sich intern nur visuell repräsentieren. [Wei95] empfiehlt in diesem Zusammenhang u.a.

- die Information (ohne Überlastung !) gut synchronisiert auf mehrere Sinnesmodalitäten zu verteilen (das spricht für den Einsatz multimedialer Techniken)
- dem Lerner durch Interaktivität eine Verankerung der Lerninhalte zu ermöglichen (vgl. mit der kinästhetischen Modalität “tun”).

Ein Ansatz, der die gleichzeitige Codierung von Information in Bild und Erklärung (auditiv oder textuell) erforscht hat, ist die Theorie der Doppelcodierung von Paivio [Pai86]; darauf basiert u.a. die unten erwähnte Untersuchung von Mayer und Anderson.

Ein für den Lernerfolg immer wieder als entscheidend genannter Aspekt ist der Motivationsgewinn durch multimediale Techniken. Hier muss allerdings unterschieden werden zwischen einem diffusen Fesseln der Aufmerksamkeit durch (zu)viel sensorischen Input und der klaren Lenkung einer gerichteten Aufmerksamkeit auf das Objekt des Lernens.

Bisherige Untersuchungen der Wirksamkeit der rechnergestützten Visualisierung ergeben durchaus gemischte Ergebnisse. Während Mayer und Anderson fast durchweg positive Ergebnisse erzielten ([MA91, MA92]) waren die Resultate von Rieber et al. ([RBA90]) eher ernüchternd. Stasko et al. und Lawrence ([BCS96, LBS94]) fanden zumindest teilweise signifikante Verbesserungen. Interessanterweise gibt es eine gemeinsame Basis von genannten Erfolgsfaktoren:

1. Das Lernziel sollte auch für den Lernenden klar sein, sonst wird evtl. nur die Animation an und für sich gelernt.
2. Anfänger und Kinder profitieren mehr von Animationen (das wird dadurch erklärt, dass Animationen beim Aufbau neuer interner Repräsentationen helfen können).
3. Die Ausnutzung von Interaktivität brachte generell deutlich bessere Lernresultate.
4. Die besten Ergebnisse wurden erzielt, wenn Studenten selber Animationen entwickelten (siehe [Sta97]).

Unsere Schlussfolgerungen aus diesen Erkenntnissen und damit auch eine Abgrenzung zu den existierenden, in Abschnitt 3 erwähnten Ansätzen stellt der kommende Abschnitt dar.

5 Das Paradigma unserer Visualisierungsumgebung

Die Essenz aus den letzten beiden Abschnitten könnte man zusammenfassen wie folgt:

- Existierende Visualisierungen sind meist schwer bis gar nicht erweiterbar oder sie unterstützen keine kontinuierlichen Animationen und/oder keine Interaktivität.

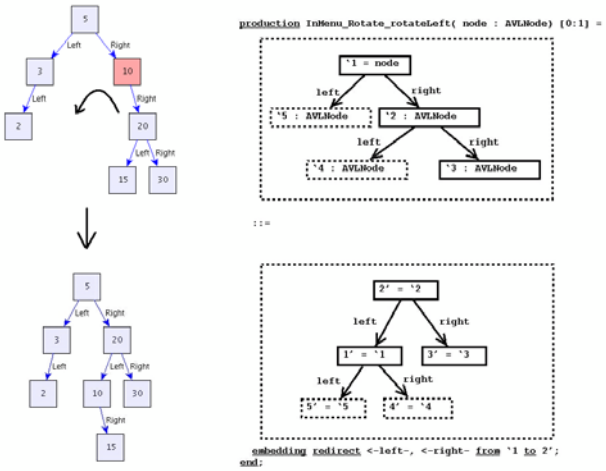

Abbildung 2: AVL Linksrotation. Links: Übliche Darstellung, Rechts: PROGRES Produktion.

- Zwei der für den Lernerfolg aussichtsreichsten Eigenschaften multimedialer Lehre sind aber gerade gute, möglichst flüssige Synchronisation der Modalitäten und Interaktivität (bzw. gleich selber tun).

Unser Ansatz will deswegen folgende Punkte unter einen Hut bringen: Konfigurierbarkeit, flüssige Animationen, Interaktivität und eigenes graphisches Debugging selbst geschriebener Programme der Studierenden. Die Frage ist nun, wie das erreicht werden kann.

In der herkömmlichen Lehre von Datenstrukturen in Informatiklehrbüchern werden Schnittstellenoperationen wie z.B. die Linksrotation in einem AVL-Baum (siehe Abbildung 2 - links, bzgl. AVL-Bäumen vgl. auch Abschnitt 7) oft als Aufeinanderfolge zweier Zustände dieser Datenstruktur dargestellt. Wenn man nun sowohl den Zustand der Datenstruktur vor der Schnittstellenoperation als auch den Zustand danach jeweils als Graph auffasst, kann man die Operation selbst auf ganz natürliche Weise als Graphtransformation auffassen. Deswegen bieten sich Graphen und Graphtransformationen als Beschreibungsmittel für Datenstrukturen, Schnittstellenoperationen und Algorithmen und damit als Spezifikationsmittel für unsere Visualisierungsumgebung an.

Bezüglich des Stichwortes der Spezifikation sei noch gesagt, dass man im Sinne der UML jede Graphtransformation als (minimal angereichertes) Paar von Objektdiagrammen auffassen kann. Eine genauere Diskussion dieses Übergangs von einer UML Spezifikation einer Datenstruktur zu Graphtransformationen findet der interessierte Leser in [AS03].

Zu betonen ist auch, dass dieser Vorgang der Spezifikation für den Dozenten (zumindest implizit) sowieso anfällt und dass nur dieser sich bis zu einem gewissen Grad mit Graphgrammatiken beschäftigen muss, nämlich wenn er neue Instanzen der Visualisierungsumgebung bauen und nutzen will. Für die Studierenden bleiben diese Strukturen also - genauso wie der Rest der Implementierung - in jedem Fall transparent.

6 Funktionsweise und Aufbau der Visualisierungsumgebung

Der nächste Schritt ist nun, aufbauend auf den vorliegenden graphgrammatischen Strukturen eine Art interaktiven Graphbrowser zur Verfügung zu stellen, der es ermöglicht, konkrete Übungsaufgaben (wie in Abschnitt 7 beschrieben) ablaufen zu lassen.

Dazu muss es möglich sein, folgende Funktionalitäten zu bieten:

- Definition beispielhafter Instanzen einer Datenstruktur
- Grundfunktionalitäten wie Auswählen, Löschen, ... von Knoten und Kanten
- Konfiguration von Farbschemata und Layoutalgorithmen
- Generierung von Menüs aus den Schritten des zu lehrenden Algorithmus (die als Graphtransformationen vorliegen)
- Definition von Übungsaufgaben, ebenso über Menüpunkte ansprechbar
- Eine Parametrisierung dieser Grundfunktionen mit knotenwertigen u.a. Parametern
- (Fehler-)Meldungen als Reaktionen auf Aktionen des Nutzers

Eine Kombination von Werkzeugen, die diese Anforderungen teilweise unterstützt und den Rest über Erweiterbarkeit ermöglicht, ist die Nutzung des Graphtransformationswerkzeugs PROGRES [PRO03] zusammen mit dem Java-Rahmenwerk UPGRADE2 [UPG03].

Dabei liefert PROGRES eine visuelle Spezifikationsumgebung für die Erstellung von Graphenszenarien, eine ergänzende Programmiersprache (siehe z.B. [Sch96]) und die Laufzeitumgebung für die Ausführung der Graphenalgorithmen inklusive der Graphdatenbank. Eine Beispielspezifikation der Linksrotation ist in Abbildung 2 - rechts zu sehen. Angewandt auf Abbildung 2 - links würde dem Knoten mit Wert 10 auf der rechten Seite ’1 entsprechen (der “Vaterknoten” der Linksrotation); der in der PROGRES-Produktion optionale Knoten ’5 dagegen existiert im konkreten Beispiel nicht.

UPGRADE2 bietet die Transformation in die Java Welt und ein gewisses Maß an Standardfunktionalität wie automatische Erzeugung von Menüs und Standard-Eventhandling. Der bereits in UPGRADE2 enthaltene JViews ([JVi03])-basierte Prototyp hat allerdings für unsere Zwecke nicht ausgereicht, so dass wir uns entschieden haben, stattdessen eine eigene Lösung basierend auf yFiles ([yfi03]) zu bauen. Gründe für den Wechsel zwischen diesen beiden Swing-basierten Graphvisualisierungs- Bibliotheken waren die große Anzahl an Layoutalgorithmen in yFiles, die Möglichkeit, flüssige Animationen mit den angebotenen Layoutmorphern zu nutzen und Lizenzgründe. Zusätzlich entwickeln wir eigene, neue Java-Klassen (in UPGRADE2 Nomenklatur eine “Extension”).

Abbildung 3 gibt eine Übersicht über das Zusammenspiel der beschriebenen Komponenten. Innerhalb des großen Kastens sind diejenigen Teile rechts dargestellt, die mit PROGRES realisiert sind, während die zu UPGRADE2 gehörigen Teile sich links befinden.

Im folgenden soll anhand Abbildung 3 kurz das Prinzip des Konfigurationsprozesses einer neuen Instanz der Visualisierungsumgebung unserer Lerneinheit beschrieben werden: Als erstes spezifiziert der Dozent eine Datenstruktur (etwa einen AVL-Baum) als PROGRES-Knotentyp, Schnittstellenoperationen (wie Einfügen eines neuen Elementes, Linksrotation, ...) als Graphtransformationen und benötigte Hilfsprozeduren als PROGRES- Produktionen oder -Tests (ein Beispiel für eine solche Hilfsprozedur wäre ein Test ’IsBalanced’, der in der Übung verwendet wird, um die Balance eines Teilbaums manuell zu überprüfen).

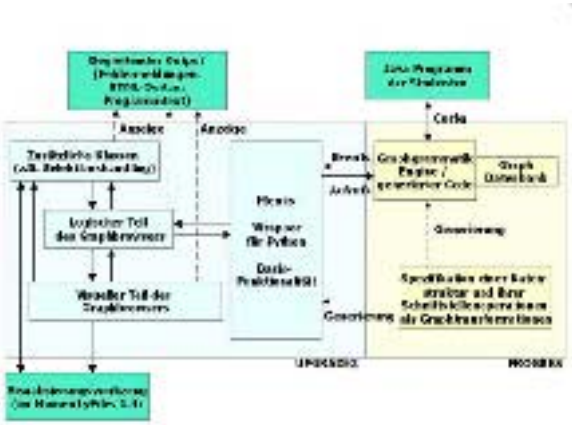

Abbildung 3: Aufbau der Visualisierungsumgebung unserer Lerneinheit

Aus dieser Spezifikation wird als nächstes in einem automatisierten Verfahren Code generiert, der sowohl die Basis für eine weitere Generierung von UPGRADE2-Datenstrukturen für Menüs etc. darstellt als auch die notwendige Grundlage für die PROGRES- Laufzeitumgebung, die später die Graphtransformationen ausführt, in der Graphdatenbank speichert, Aufrufe von UPGRADE2 entgegennimmt und Events für UPGRADE2 generiert.

Die restlichen Komponenten seien anhand eines kleinen Beispielablaufs charakterisiert: Wenn der Lernende etwa im Rahmen einer Übung zum manuellen Aufbau eines AVL-Baumes (eine genauere Beschreibung dieser AVL-Übung folgt in Abschnitt 7) einen gerade neu erzeugten Knoten mithilfe einer Produktion 'AddLeft' als linken Sohn eines schon im Baum befindlichen Knotens einfügen will, kann er zuerst den zukünftigen Vater-Knoten mit der Maus markieren, wobei der Selektionsmanager (als eine der 'zusätzlichen Klassen' in Abbildung 3) über eine yFiles-Funktionalität den markierten Knoten identifiziert, mithilfe des logischen Teils des Graphbrowsers in eine PROGRES- / UPGRADE2-Knotennummer umrechnet und für die weitere Verwendung in einer entsprechenden UPGRADE2-Datenstruktur vormerkt. Das gleiche passiert nun mit dem neuen linken Sohn. Falls die vorher erwähnte PROGRES-Produktion 'AddLeft' wirklich zwei Knoten als Parameter erwartet und zwar zuerst den Vater und dann den neuen linken Sohn, kann der Lernende nun im Menu 'AddLeft' aufrufen. In diesem Fall verpackt die Basisfunktionalität des generierten UPGRADE2 Prototypen diesen Aufruf in einen Aufruf der entsprechenden PROGRES-Produktion und übergibt die zugehörigen Parameter. Die Graphgrammatik-Engine führt die Graphtransformation auf ihrer Datenbank aus und liefert dementsprechende Events zurück, etwa im Erfolgsfall ein 'AddEdge'-Event für die neu hinzukommende Kante. Der Layoutmanager (im visuellen Teil des Graphbrowsers) erfährt davon mithilfe der Basisfunktionalität des Prototypen und führt mit yFiles-Funktionalität einen flüssigen Animationsschritt aus, in dem der neue linke Sohn an seinen ihm nun zustehenden Platz im Graphen (links unter seinem Vater) bewegt wird.

Im Fehlerfall wäre im Rahmen des begleitenden Outputs eine Fehlermeldung ausgegeben

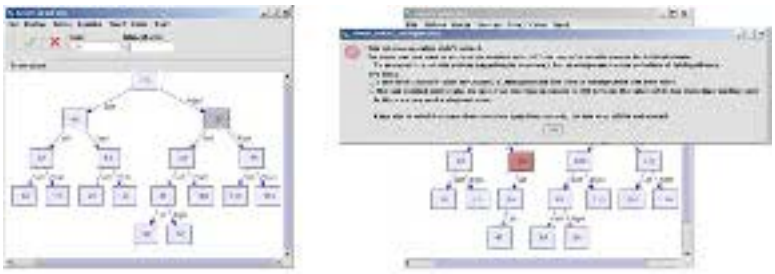

Abbildung 4: Screenshots der AVL-Instanz. Links: Der korrekte Balance-Faktor eines Knotens ist gesucht. Rechts: Eine falsche Rotation wurde gewählt.

worden und der Graph hätte sich nicht verändert. Die bisher nicht beschriebene Komponente 'Java-Programm der Studenten' bezieht sich auf die geplante Möglichkeit, mithilfe der Visualisierungsumgebung eigene Programme graphisch debuggen zu können.

7 Lernszenarien - Beschreibung und Evaluation

Der erste praktische Einsatz unserer Visualisierungsumgebung erfolgte im Rahmen der Übungen zur Vorlesung "Einführung in die Informatik II" im Wintertrimester 2003 an der Universität der Bundeswehr München.

Dort wurde an zwei Tagen (mit einer Gruppe von ca. 15 Studenten) ein aus der Vorlesung bekanntes Thema jeweils ca. eine Stunde lang anhand einer interaktiven Übungsaufgabe vertieft. Parallel dazu wurde eine gleich große Kontrollgruppe mit einer Papier-Aufgabe des selben Lernstoffes konfrontiert. Die Betreuung und beobachtende Evaluierung erfolgte durch zwei Mitarbeiter des MuSofT-Projektes. Nach Beendigung der Aufgabe füllten beide Gruppen einen Evaluationsbogen aus, der den Lernfortschritt messen sollte. Die MuSofT-Gruppe füllte zusätzlich einen Fragebogen aus, der Selbsteinschätzung und Motivation vor und nach der Übung überprüfte. Im folgenden wird die erste Aufgabe (AVL-Bäume) kurz beschrieben, die zweite (Dijkstra's kürzeste Wegesuche) nur angerissen:

Ein AVL-Baum (für eine Einführung siehe z.B. [Tim01]) ist ein binärer Suchbaum, der mithilfe von Balancierungs-Operationen (sogenannten "Rotationen") symmetrisch genug bleibt, um z.B. effektives Suchen von Elementen in einer Menge zu erlauben. Während der interaktiven Übung war es von den Studierenden u.a. gefordert, einen Beispielablauf von Einfüge-Operationen, die in flüssiger Animation alle Rotationen illustrierten, zu beobachten, Höhe und Balancierungs-Faktor mehrerer innerer Knoten richtig anzugeben (siehe Abb. 4 - links), einen falsch einsortierten Knoten zu finden, den Baum durch Umhängen des fehlerhaften Knotens zu reparieren, die nun unbalancierte Stelle zu finden, durch manuellen Aufruf geeigneter Rotations-Operationen auch dieses Problem zu beheben (siehe Abb. 4 - rechts) und das Verhalten bei Einfügen weiterer Knoten zu testen.

Die beobachtende Evaluation zeigte, dass die Studenten schnell lernten, das Tool zu bedienen, sie hatten Spaß und arbeiteten konzentriert. Andererseits lasen sie Fehlermeldungen

nicht richtig und es bestand die Tendenz, das gewünschte Ergebnis unter Umgehung der geforderten Schritte zu erreichen, so wurde etwa versucht, den Baum durch Einfügen neuer Knoten zu rebalancieren anstatt durch explizite Rotationen.

Die Auswertung der Fragebögen zeigte, dass beide Gruppen ein gutes Verständnis von AVL-Bäumen gewonnen hatten, wobei die MuSofT-Gruppe besser darin war, etwa die Höhe eines Beispiel-Baums anzugeben oder die Stelle, wo ein Beispiel-Baum unbalanciert ist. Die Stärken der Vergleichsgruppe lagen in Fragen wie die nach der Höhe des leeren Baums oder wie viele Rotationen nach einer Einfüge-Operation maximal notwendig sind. Auch wenn durch die sehr eng liegenden Resultate und die Kleinheit der Gruppen kein signifikantes Ergebnis hergeleitet werden kann, könnte man doch eine Stärke der MuSofT-Gruppe bei Fragen erahnen, die mit der visuellen Darstellung am Bildschirm zusammenhingen und eine Stärke der anderen Gruppe bei eher theoretischen Fragen. Ausnahmen davon gab es bei Teilthemen, die unterschiedlich intensiv geübt worden waren.

Die Auswertung des Motivationsfragebogens zeigte, dass die Studenten in unserer MuSofT-Gruppe die graphische Darstellung und die Animation ansprechend fanden, den Lernvorgang als intensiv empfanden, dass es Spaß machte und verständlich war.

Die zweite Übung behandelte Dijkstra's kürzeste Wegesuche (siehe etwa [Ein03]), einen Algorithmus, der es durch eine Breitensuche erlaubt, kürzeste Pfade in einem (un-)gerichteten Graphen mit attributierten Kanten zu finden. In vielen Fassungen des Algorithmus wird mit 3 Farben gearbeitet, die die Menge der Knoten in 3 Gruppen aufteilen; diese Tatsache haben wir für die Visualisierung genutzt und mit den Ampelfarben rot-gelb-grün gearbeitet.

Übungsaufgaben waren u.a. das Finden kürzester Pfade zu gegebenen Knoten und Überlegen und nachmaliges Testen folgender Fragen: Welcher Knoten wird als erster betrachtet ? Welche neue Farbgebung müsste nach dem nächsten Schritt entstehen, welcher Knoten ist nun der Ausgangsknoten des nächsten Schrittes ? Wann würde der Algorithmus genau stoppen, wenn der kürzeste Weg zu Knoten X (einem definierten Knoten) gesucht wäre ?

Die "beobachtende Evaluation" ergab (genau wie die Auswertung des Motivationsfragebogens) sehr ähnliche Erkenntnisse wie bei der AVL- Übung, allerdings eine genauere Bearbeitung der Vorgaben. Die Auswertung der Verständnis-Fragebögen zeigte ein leicht bis viel besseres Abschneiden der MuSofT-Gruppe bei den meisten Fragen.

8 Zusammenfassung / Ausblick

Unsere Lerneinheit unterstützt die Lehre von Algorithmen und Datenstrukturen aus der Sicht der Softwaretechnik mit zwei Medien: Einem konfigurierbaren Foliensatz als Grundlage für eine Präsenzveranstaltung und einer aus visuellen Spezifikationen generierbaren interaktiven Visualisierungsumgebung. Beide wurden bereits eingesetzt und bis zu einem gewissen Grad evaluiert.

Weiterhin geplant sind eine ausführlichere Evaluierung, die Hinzunahme neuer Features (wie Undo-Funktionalität, das erwähnte, aber noch nicht implementierte graphische Debugging und die Nutzung von Python-Skripten für das Ablaufen-Lassen ganzer Beispiel-Sequenzen), die Erstellung weiterer interaktiver Übungen (wie doppelt verkettete Liste und das Spannbaum-Problem) und die Weitergabe der Materialien an interessierte Dozenten.

Literatur

[Ani03] http://www.informatik.uni-siegen.de/db/animations.php3?lang=en, März 2003.

[AS03] Peter Aschenbrenner and Andy Schuerr. Generating Interactive Animations from Visual Specifications. *submitted for publication for the HCC-VLFM '03*, 2003.

[Bal90] Steffen-Peter Ballstaedt. Integrative Verarbeitung bei audiovisuellen Medien. In *Böhme-Dürr , K./Emig,J./Seel, N (Hrsg.): Wissensveränderung durch Medien. München: Saur.*, 1990.

[BCS96] M. Byrne, R. Catrambone, and J. Stasko. Do Algorithm Animations Aid Learning? Technical Report GIT-GVU-96-18, 1996.

[Bub03a] http://olli.informatik.uni-oldenburg.de/fpsort/Animation.html, März 2003.

[Bub03b] http://www.softwarevisualisierung.de, Mai 2003.

[CCA03] http://www.cs.hope.edu/ alganim/ccaa/, März 2003.

[DE02] Ernst-Erich Doberkat and Gregor Engels. MuSofT - Multimedia in der Softwaretechnik. *Informatik Forschung und Entwicklung*, 17(1):41–44, 2002.

[Ein03] http://www2.informatik.unibw-muenchen.de/Lectures/WT2002/INF2/index.html, Jan. 2003.

[htt03] http://www.es.tu-darmstadt.de/index.php?content=lehre/pl/uebersicht.html&language=deutsch&menu=Lehre/SP, Mai 2003.

[JVi03] http://www.ilog.com/products/jviews/, März 2003.

[LBS94] Andrea Lawrence, Albert Badre, and John T. Stasko. Empirically Evaluating the Use of Animations to Teach Algorithms. In *Proceedings of the 1994 IEEE Symposium on Visual Languages, St. Louis, MO*, pages 48–54, 1994.

[MA91] R. Mayer and R. Anderson. Animations need narrations: An experimental test of a dual-coding hypothesis. In *Journal of Educational Psychology*, pages 484–490, 1991.

[MA92] R. Mayer and R. Anderson. The instructive animation: Helping students build connections between words and pictures in multimedia learning. In *Journal of Educational Psychology*, pages 444–452, 1992.

[Pai86] A. Paivio. *Mental representations. A dual coding approach.* New York: Oxford University Press, 1986.

[PRO03] http://www-i3.informatik.rwth-aachen.de/research/projects/progres/, März 2003.

[RBA90] L. Rieber, M. Boyce, and C. Assad. The Effects of Computer Animation on Adult Learning and Retrieval Tasks. In *Journal of Computer-Based Instruction*, pages 46–52, 1990.

[Sch96] Andy Schürr. Introduction to the Specification Language PROGRES. In *in: M. Nagl (ed.): Building Thightly-Integrated (Software) Development Environments: The IPSEN Approach, LNCS 1170*, pages 248–279. Springer Verlag Berlin, 1996.

[SDBP98] John Stasko, John Domingue, Marc Brown, and Blaine Price. *Software Visualization*. MIT Press, 1998.

[Sor03] http://www.db.fmi.uni-passau.de/Sommercamp2001/Unterlagen/SortDemo.html, März 2003.

[Sta97] J. Stasko. Using Student-Built Algorithm Animations as Learning Aids. *SIGCSEB: SIGCSE Bulletin (ACM Special Interest Group on Computer Science Education)*, 29, 1997.

[Tim01] Timothy Budd. *Classic Data Structures in Java*. Addison-Wesley, 2001.

[UPG03] http://www-i3.informatik.rwth-aachen.de/upgrade/, März 2003.

[Wei95] Bernd Weidenmann. Multimedia, Multicodierung und Multimodalität im Lernprozeß. In *Arbeiten zur Empirischen Pädagogik u. Pädagogischen Psychologie, Gelbe Reihe, München*, 1995.

[yfi03] http://www.yworks.de/, März 2003.