Algorithmen lernen mit interaktiven Visualisierungen durch Betrachten, Nachgehen und Experimentieren

Nils Faltin

C. v. Ossietzky Universität Oldenburg
Abt. Computer Graphics & Software-Ergonomie
D – 26111 Oldenburg

Faltin@Informatik.Uni-Oldenburg.DE

Dieser Artikel definiert drei Arten von interaktiven Visualisierungen für das Lernen von Algorithmen: Betrachten (Animation), Nachgehen (Nachmachen der Schritte) und Experimentieren (mit einer Simulation des Algorithmus). Für sie werden aus didaktischer und software-ergonomischer Sicht Gestaltungshinweise gegeben und Beispiele genannt. Schließlich wird eine SW-Architektur skizziert, die es erleichtern soll, die drei Arten von Visualisierung zu entwickeln.

Einführung

Dieser Artikel versucht die folgenden Fragen zu klären:

- Wie kann man mit interaktiven Visualisierungen Algorithmen lernen?
- Welche Rolle kann das Lernmedium Computer dabei spielen?
- Welche neuen Möglichkeiten bieten dabei insbesondere grafische Benutzungsoberflächen?

Als Einstieg soll ein Beispiel aus der Grundschule dienen: Der erste (formale) Algorithmus den ein Kind typischerweise lernt ist das schriftliche Addieren von Dezimalzahlen. Die Lehrkraft zeigt den Schülern schrittweise den Ablauf an einem Beispiel und erklärt Regeln, wie z.B. der Übertrag zu handhaben ist. Die Ziffern (Datenobjekte) werden an der Tafel in einem bestimmten, tabellenartigen Muster angeordnet. Das Tafelbild wird schrittweise weiterentwickelt, der Algorithmus wird also in Einzelschritten dynamisch visualisiert. Die Schüler *betrachten* diese dynamische Visualisierung und lernen so den Algorithmus kennen.

In einer zweiten Phase sollen sie den Algorithmus selbst ausführen, indem sie eine Beispielaufgabe auf Papier rechnen. Dabei ist es hilfreich, wenn das tabellenartige Muster schon in einem Arbeitsblatt vorgegeben ist und dort die zu addierenden Zahlen bereits eingetragen sind. Das Abarbeiten des Algorithmus durch das *Nachgehen* der zuvor gesehenen Schritte hat mehrere didaktische Funktionen. Zunächst dient es der Selbstkontrolle, ob der Algorithmus verstanden wurde. Weiterhin prägt sich der Ablauf ins Gedächtnis ein und kann nach mehreren Durchläufen schließlich automatisch (rou-

tiniert) ausgeführt werden. Eine Aufgabe soll auch zum Lernen motivieren. Sie fordert die intellektuelle Fähigkeit der Schüler heraus und bringt sie von einer aufnehmenden (rezeptiven) Haltung zu einer aktiven Arbeit. Wird die Aufgabe richtig bearbeitet, so kann dies als Erfolg zum weiteren Lernen motivieren. Ein falsches Ergebnis dient als Hinweis, dass der Algorithmus noch nicht verstanden wurde oder noch nicht sicher beherrscht wird.

In diesem Beispiel sieht man zwei Formen der Visualisierung von Algorithmen für das Lernen. Zunächst wird die Veränderung der Datenstruktur visualisiert und der Lerner *betrachtet* dies rezeptiv. In der zweiten Phase führt der Lerner die Schritte des Algorithmus auf einer Visualisierung der Datenstruktur selbst aus. Da er sozusagen den Fußstapfen des zuvor gesehenen Standardalgorithmus nachgeht, soll dies als *Nachgehen* bezeichnet werden.

Die genannten didaktischen Prinzipien lassen sich auch auf das Lernen von Algorithmen in späteren Stufen der Ausbildung, z.B. an der Hochschule anwenden. Dort geht es dann um Algorithmen, die von Computern ausgeführt werden sollen. Hat der Lerner gelernt, welche Schritte der Algorithmus durchführt so dürfte es ihm leichter fallen, den Algorithmus mathematisch zu analysieren und zu implementieren.

Soll der Lerner aber befähigt werden einen Algorithmus um neue Funktionen zu erweitern, so wird eine Lernform benötigt, die die promblemlösenden Fähigkeiten fördert. Die Aufgabe kann dann lauten, den Pseudocode für eine Funktion des Algorithmus selbst zu entwickeln. Eine Funktion besteht aus Aufrufen von anderen, bereits vorhandenen Funktionen. Die Effekte einer selbst gewählten Aufrufkette kann der Lerner durch *Experimentieren* mit einer interaktiven visuellen Simulation des Algorithmus erforschen.

Die Lernformen *Betrachten*, *Nachgehen* und *Experimentieren* dienen dem Erlernen von Algorithmen mit interaktiven Visualisierungen. In den folgenden Abschnitten werden die Lernformen, insbesondere unter didaktischem und software-ergonomischem Blickpunkt, beleuchtet. Schließlich wird eine Softwarearchitektur vorgestellt, die es erleichtern soll, interaktive Visualisierungen für diese Lernformen zu entwickeln.

Lernform „Betrachten“

Bei der Lernform „Betrachten“ ist der Ablauf des Algorithmus fest vorgegeben. Vom Lerner wird im wesentlichen erwartet, dass er die Schritte des Algorithmus in einer Visualisierung (Algorithmen-Animation) betrachtet. Es wird nicht erwartet, dass er Schritte voraussagt oder selbst Teile des Algorithmus entwickelt. Von den drei Lernformen ist „Betrachten“ die mit Abstand verbreitetste. Einen guten Überblick über Forschungsarbeiten zu Algorithmen-Animation gibt das Werk von Stasko et al [1].

Gestaltungshinweise

Aus didaktischer und software-ergonomischer Sicht möchte ich nun einige Gestaltungshinweise für Animationen geben, die für die Lernform „Betrachten“ entwickelt werden. Peter Gloor hat eine ähnliche Liste mit Schwerpunkt auf software-ergonomischen Kriterien entwickelt [2]. Die Animation sollte möglichst viele der Kriterien erfüllen.

Funktionale Struktur. Der Algorithmus wird in Funktionen aufgeteilt. Jede nichttriviale Funktion wird in einer eigenen Animation erklärt. Bereits gelernte Funktionen können in einer Animation als Bausteine (Funktionsaufrufe) verwendet werden und werden nur noch als ein Schritt angezeigt.

Lehrtext. Mit der Animation ist ein Lehrtext verbunden, der in der Art eines Lehrbuchs mit Text und Bildern den Algorithmus erklärt. Alternativ kann dies auch durch einen vorherigen Vortrag vermittelt werden.

Anzeige des Pseudocode. Zusätzlich zu den Datenstrukturen wird der Pseudocode des Algorithmus angezeigt. Die aktuelle Befehlszeile wird hervorgehoben. So kann der Lerner den Pseudocode und die Aktionen auf den Datenstrukturen miteinander in Beziehung setzen.

Erklärung der Schritte. Die Schritte des Algorithmus werden textuell oder per Sprachausgabe kommentiert, damit der Lerner versteht, was der Algorithmus gerade tut.

Interessante Eingabedaten. Viele Algorithmen zeigen bei bestimmten Eingabedaten ein besonderes Verhalten. Der Sortieralgorithmus Quicksort zeigt z. B. bei vorsortierten Eingabedaten ein anderes Laufzeitverhalten als bei zufällig gemischten Eingabedaten. Dem Lerner werden die verschiedenen Sätze von Eingabedaten zur Auswahl gestellt werden.

Kleine Datenmenge. Die Animation arbeitet auf einer kleinen Zahl von Datenobjekten. Die Anzahl ist gerade so gewählt, dass das wesentliche am Algorithmus noch zu erkennen ist. Große Datenmengen erschweren die Wahrnehmung und das Verständnis. Dabei bringen sie im Normalfall keine zusätzlichen Erkenntnisse. Kognitive Psychologen haben in Tests ermittelt, dass Menschen typischerweise 7-9 Einheiten („chunks“) im Arbeitsgedächtnis halten können. Deshalb sollten nicht mehr als 7 Objekte an einem Algorithmusschritt aktiv beteiligt sein.

Wenige Schritte. Analog werden nur soviele Wiederholungen von Schleifendurchläufen gezeigt, wie für Wahrnehmung und Verständnis nötig sind. Ansonsten wird der Lerner schnell gelangweilt und seine Aufmerksamkeitskapazität unnötig verbraucht.

Flexible Ablaufsteuerung. Die Animation des Algorithmus kann ähnlich wie bei einem Videorecorder gesteuert werden: Einzelschritte vor und zurück, Abspielen, Pause, Stopp, Sprung an Anfang und Ende. Aus didaktischer

Sicht ist es besonders wichtig in Einzelschritten vor und zurück schalten zu können. Gerade Anfänger, die den Algorithmus neu lernen sind mit einem durchlaufenden Film meist überfordert. Da normale Programme nicht „rückwärts“ laufen können stellt dies besondere Anforderungen an die Software-Architektur der interaktiven Visualisierung. Auf eine mögliche Lösung wird im letzten Abschnitt eingegangen.

Interessante Ereignisse. Ein Schritt umfasst dabei ein „interessantes Ereignis“ des Algorithmus. Das kann z. B. eine Aktion des Algorithmus oder das Erreichen wichtiger Zwischenergebnisse sein. Im Gegensatz zu diesem Prinzip machen manche SW-Visualisierungs-Systeme grafische Operationen zu Schritten. Das Konzept der „interesting events“ wurde von Marc Brown entwickelt [8].

Fokussierung der Aufmerksamkeit. Der Blick des Lerners wird auf die Teile der Darstellung gelenkt, die sich als nächstes verändern werden. Dazu kann der Autor diese Objekte z.B. blinken oder ihre Größe ändern lassen. So wird verhindert, dass der Betrachter wichtige Schritte nur als diffuse Bewegung wahrnimmt.

Weiche Übergänge. Veränderungen auf den Datenstrukturen werden möglichst mit weichen, fließenden Bewegungen gezeigt. Werden z.B. zwei Elemente eines Array vertauscht, so können sie sich jeweils auf einem Halbkreisbogen bewegen und so „Plätze tauschen“. Dies erleichtert die Wahrnehmung des Vorgangs. Damit beim Lerner nicht der Eindruck entsteht, in einem Computer würden sich die Daten „fließend bewegen“ sollte im Begleittext darauf hingewiesen werden, wie die Veränderung der Daten implementiert wird. Vertauschung wird z. B. über „plötzliche“ Kopiervorgänge im Dreieckstausch realisiert. Es war eines der wesentlichen Ziele bei der Entwicklung des SW-Visualisierungssystems Tango, Animationen mit weichen Übergängen zu ermöglichen [7].

Beispiel

Mir ist keine Algorithmen-Animation bekannt, die alle genannten Prinzipien umsetzt. Die Autoren der folgenden Animation des Bubblesort-Algorithmus (Abb. 1) wollten eine ästhetische Animation entwickeln, die die Metapher „Luftblasen“ (Bubbles) aufgreift [3]. Sie haben dabei die folgenden Prinzipien umgesetzt:

- Weiche Übergänge
- Flexible Ablaufsteuerung
- Anzeige des Pseudocodes
- Kleine Datenmenge
- Lehrtext

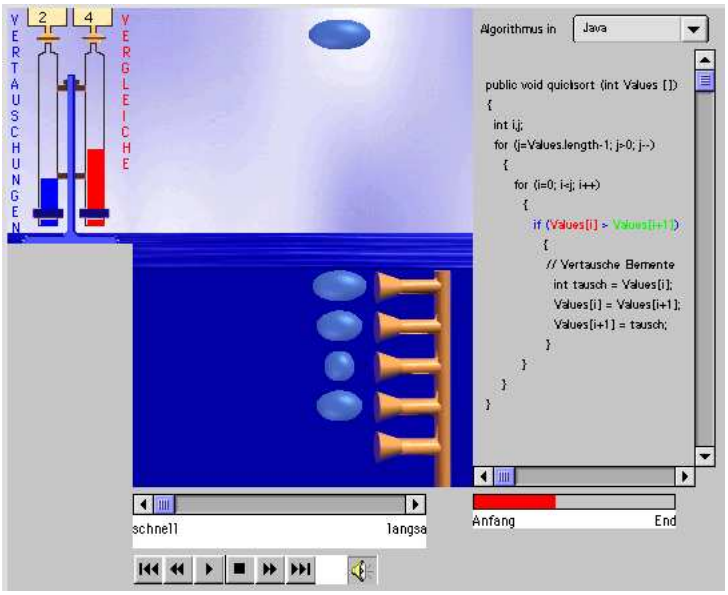

Abb. 1 - Lernen durch Betrachten: Bubblesort-Animation

Lernform „Nachgehen“

Nachdem ein Algorithmus durch „Betrachten“ gelernt worden ist, sollte das Wissen durch Übung überprüft und vertieft werden. Bei der Lernform „Nachgehen“ soll der Lerner mehrfach den nächsten Schritt des Algorithmus voraussagen. Es gibt dabei nur einen „richtigen“ nächsten Schritt, und zwar den, den der Standardalgorithmus als nächstes ausführen würde. Dazu interagiert der Lerner mit einer Visualisierung der Datenstruktur über eine grafische Benutzungsoberfläche. Es scheint nur wenige Projekte zu geben die versucht haben, interaktive Visualisierungen zu entwickeln, die diese Lernform unterstützen [5], [4].

Gestaltungshinweise

Die Gestaltungshinweise für Animationen, die ich für das Lernen durch „Betrachten“ gegeben habe, gelten bis auf den Punkt „Anzeige des Pseudocode“ auch für das Lernen durch „Nachgehen“. Zusätzlich sollten noch die folgenden Gestaltungshinweise beachtet werden.

Kein Pseudocode. Da der Lerner den nächsten Schritt eigenständig voraussagen soll, wird kein Pseudocode angezeigt. Insbesonder der Charakter der Überprüfung des Wissens würde sonst unterlaufen.

GUI nutzen. Die Möglichkeiten einer grafischen Benutzungsoberfläche (GUI) werden soweit möglich und sinnvoll für die Interaktion genutzt. Insbesondere kann der Lerner Datenobjekte in der Grafik der Visualisierung direkt anwählen (z.B. mit der Maus anklicken). Ein Schritt eines Algorithmus besteht oft im Aufruf einer Funktion. Die Funktionen können z.B. durch Schaltflächen (Buttons) dargestellt werden. Der Lerner ruft eine Funktion auf, indem er erst die Objekte (aktuelle Parameter) in der Grafik anklickt und dann die Schaltfläche der Funktion.

Bedienung vereinfachen. Die Bedienung der Aufgabe ist möglichst einfach gestaltet. Es ist einfach erkennbar, welche Funktionen angeboten sind und welche Objekte anwählbar sind. Der Lerner muss also Objekte und Funktionen nur auswählen. Bedienungsfehler sollten als solche gemeldet werden. Die Bedienung sollte beschrieben sein (z.B. als Online-Hilfe oder im Begleittext).

Lösung erschweren. Es werden z. B. mehr Funktionen angeboten, als tatsächlich benötigt werden. Auch sind möglichst alle Objekte der Visualisierung anwählbar. Dadurch wird es erschwert, den nächsten Schritt zu erraten.

Fehlermeldungen und gestufte Hilfe. Wählt der Lerner eine falsche Funktion oder falsche Parameter, so kann es sinnvoll sein zu begründen, warum dieser Funktionsaufruf jetzt keinen Sinn macht. Meist wird es aber reichen zu melden, dass der Algorithmus einen anderen Schritt ausführen würde. Es sollte aber eine gestufte Hilfe abrufbar sein, die einen Hinweis gibt, was zu tun ist und als nächste Stufe den Schritt direkt angibt.

Beispiel

Ein Beispiel für die Lernform „Nachgehen“ ist das folgende Applet aus der Lernsoftware zum Binomial Heap Algorithmus (Abb. 2) [4]. In ihm sollen zwei Listen von Binomialbäumen zu einer Ergebnisliste verschmolzen werden. Dies wird ähnlich dem schriftlichen Addieren durchgeführt. Der Lerner muss genau die Schritte des Standardalgorithmus einhalten, kann aber über die Schaltfläche „Tipp“ eine einstufige Hilfe abrufen. Die oben genannten Kriterien sind im wesentlichen erfüllt.

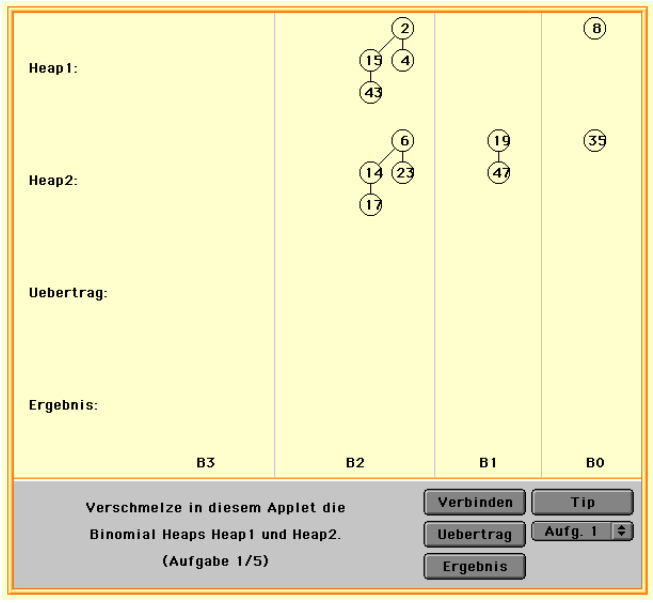

Abb. 2 – Lernen durch Nachgehen: Verschmelzen-Funktion

Lernform „Experiment“

Die *Lernform „Experiment“* soll es dem Lerner ermöglichen den Algorithmus „nachzuentdecken“. Der Algorithmus wird dem Lerner nicht als ein fertiges Werk vermittelt, sondern dieser soll einen Teil der Arbeit der Entwicklung und Entdeckung wiederholen. Er wird dadurch in die Rolle eines Algorithmenentwicklers versetzt.

Die Lernform „Experiment“ muss, wie andere Lernformen auch, in einen größeren didaktischen Kontext eingebettet werden. Für die Gestaltung von Lernsoftware zu Algorithmen habe ich hierzu das didaktisches Konzept „Strukturierte aktive Lernen von Algorithmen (SALA)“ entwickelt [6].

Nach meinem Verständnis besteht ein Algorithmus aus den folgenden Strukturen:

1. Statische und dynamische *Struktur der Daten* (z.B. Listen, Verzeigerung, Records/ Structs)
2. Mathematische und logische Bedingungen zwischen Datenobjekten, den sog. *Invarianten*
3. Funktionale Struktur
4. *Pseudocode* der einzelnen Funktionen

Die *funktionale Struktur* eines Algorithmus gibt an, in welche Funktionen der Algorithmus aufgeteilt ist und welche Funktionen einander aufrufen. Für jede Funktion muss angegeben sein:

- *Zweck* der Funktion
- *formale Parameter* (Typ und Rolle)
- *Start-Invarianten*, die bei Aufruf der Funktion gelten müssen, damit die Funktion arbeiten kann.
- *Arbeits-Invarianten*, die während der Arbeit der Funktion beachtet werden müssen. Teils drückt sich darin die „Strategie“ aus, nach der die Funktion arbeitet.
- *Ziel-Invarianten*, die beim Ende der Funktion gelten müssen. Diese leiten sich aus dem Zweck der Funktion ab und aus den Voraussetzungen (zeitlich) nachfolgender Funktionen.

Man kann nicht erwarten, dass ein Lerner den Algorithmus selbst entwickelt (nachentdeckt), wenn er nicht zuvor wesentliche Teile vermittelt bekommt. Ich schlage vor, die Punkte 1 bis 3 (Struktur der Daten, Invarianten und funktionale Struktur) zu vermitteln und den Lerner den Pseudocode selbst entwickeln zu lassen.

Der Pseudocode bestimmt, welche Schritte der Algorithmus auf einem konkreten Satz Daten ausführt. Umgekehrt kann sich der Lerner durch Ausprobieren von Schritten auf Daten zum allgemein funktionierenden Pseudocode vortasten. Dabei ist zu vermuten, dass ihm ein zielloses Probieren wenig weiterhilft, sondern dass er sich eine Strategie überlegen muss, die er durch Experimentieren erproben und verfeinern kann.

Verglichen mit dem Lernen durch „Nachgehen“ arbeitet der Lerner beim „Experimentieren“ mit einer echten Simulation, bei der aus einer Situation heraus in der Regel alternative Schritte möglich sind. Es kann durchaus mehrere Wege zum Ziel geben. Der Fortschritt der Bearbeitung sollte sich in der Visualisierung zeigen, so dass der Lerner sieht wo „die Arbeit erledigt ist“ und wie nah er dem Ziel gekommen ist.

Die Simulation ermöglicht dem Lerner zu experimentieren. Dazu stellt sie mehrere Funktionen zur Verfügung, die auf den Datenobjekten angewendet werden können. Sie führt aber einen Funktionsaufruf nur dann aus, wenn die Arbeits- und Start-Invarianten dies erlauben.

Gestaltungshinweise

Die Benutzungsoberfläche ähnelt der Lernform „Nachgehen“. Zusätzlich zu den bei „Nachgehen“ genannten Prinzipien sollten die folgenden Gestaltungshinweise beachtet werden:

Inhaltsbezogene Fehlermeldungen. Versucht der Lerner eine Funktion aufzurufen, deren Start-Invarianten nicht erfüllt sind oder würde dadurch eine Arbeits-Invariante verletzt, so muss die interaktive Visualisierung dies mit einer Fehlermeldung abweisen. Die Fehlermeldung sollte den Grund in Bezug auf die Invarianten angeben.

Fortschrittsanzeiger. Der Lerner kann erkennen, wie nah er dem Ziel gekommen ist. Die Visualisierung kann dazu z.B. anzeigen, welcher Anteil der Daten die Ziel-Invariante bereits erfüllt.

Undo. Es kann durchaus sein, dass der Lerner Schritte tätigt, die zwar die Invarianten nicht verletzen, aber nicht zum Ziel führen. Über einen Undo-Mechanismus kann der Lerner einzelne Schritte zurücknehmen.

Beispiel

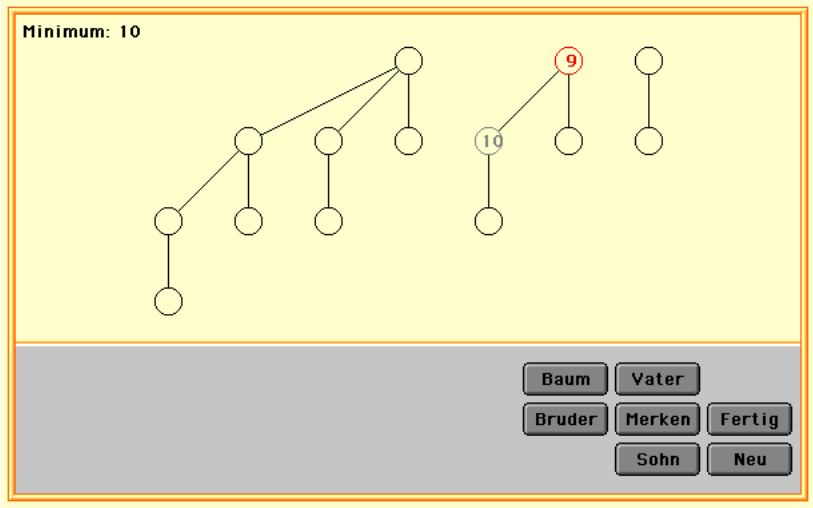

Abb. 3 Lernen durch Experimentieren: „Minimum Suchen“-Funktion

Abbildung 3 zeigt einen Screenshot einer Simulation der „Minimum suchen“-Funktion aus dem Binomial-Heap Algorithmus [4]. Der Lerner kann frei von Baum zu Baum und von Knoten zu Knoten navigieren um das Minimum zu suchen. Sobald ein Knoten besucht wird, wird sein Wert sichtbar. Aufgrund der Invarianten reicht es die Wurzeln abzusuchen und sich den kleinsten Wert zu merken. Diese Simulation verzichtet weitgehend auf Fehlermeldungen, Fortschrittsanzeiger und Undo. Stattdessen wird am Ende gesagt, ob das Minimum gefunden wurde und ob dies der effizienteste Weg war.

SW-Architektur für interaktive Visualisierungen

In unserer Arbeitsgruppe werden schon seit einigen Jahren Algorithmen-Visualisierungen für Lernzwecke entwickelt. Während die Visualisierungen zunächst das Lernen durch „Betrachten“ unterstützten ([9],[3]), haben wir im Projekt „Medienunterstütztes

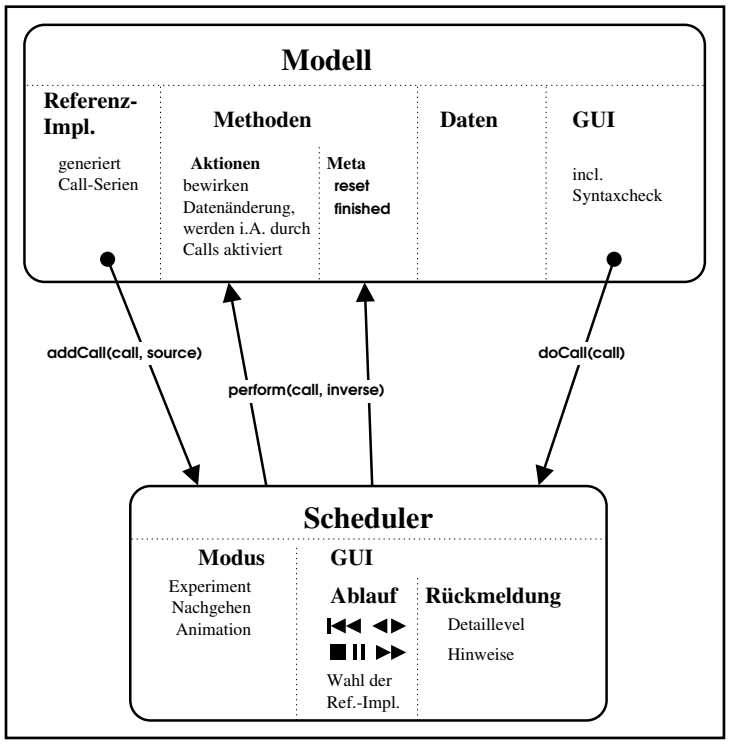

Abb. 4 – SW-Architektur

Studium der Informatik“ ([10]) das Lernen durch „Experimentieren“ untersucht und Algorithmen-Simulatoren dafür entwickelt. Jede der Lernformen „Betrachten“, „Nachgehen“ und „Experimentieren“ hat einen didaktisch sinnvollen Einsatzbereich. Es kann daher sein, dass für einen Algorithmus alle drei Arten der interaktiven Visualisierung entwickelt werden sollen.

Daher arbeiten wir zur Zeit an einer SW-Architektur, die dem Autor erleichtern soll die drei Arten der Visualisierung zu erstellen. Die Teile, die unabhängig vom Algorithmus sind, sollen dem Autor als Modul bereitgestellt werden. Außerdem soll eine programmtechnische Beschreibung des Algorithmus und einer „allgemeinen“ Visualisierung die automatische Erzeugung der drei Arten der Visualisierung erlauben.

Ein erster Entwurf soll hier skizziert werden. Er wurde in Java prototypisch implementiert und mit einer einfachen Testaufgabe erprobt.

Die Architektur realisiert die interaktive Visualisierung für eine Funktion des Algorithmus (Abb. 4).

Die zentrale Idee ist die Trennung der Visualisierung in ein *Modell* und einen *Scheduler*. Funktionen des Algorithmus werden nicht direkt aufgerufen, sondern die *Aufrufe* (engl. *call*) laufen über die Zwischeninstanz des Schedulers.

Im *Modus Animation* (Lernen durch „Nachgehen“) arbeitet die Visualisierung in der folgenden Weise. Bei der Initialisierung generiert die sog. *Referenzimplementierung* der Funktion eine Serie von Aufrufen und leitet diese an den Scheduler, der sie in einer Liste speichert. Der Lerner kann über die Ablaufsteuerung (GUI des Schedulers) die Aufrufe ausführen lassen (*perform*). Der Scheduler reicht die Aufrufe als Objekte an die Aktions-Methoden des Modells. Diese berechnen selbst die nötigen Daten, mit denen ihr eigener Aufruf rückgängig gemacht werden kann, inklusive des inversen Aufruf-Objekts. Das Inverse Aufruf-Objekt wird als Rückgabewert an den Scheduler gereicht, der dies in einer Inverse-Liste speichert. Führt der Lerner einen Schritt rückwärts aus, so wird das entsprechende Inverse Aufruf-Objekt an das Modell geschickt.

Eine Aktionsmethode arbeitet in folgenden Phasen:

1. **Regelprüfung**. Die Start-Invariante wird geprüft. Ist sie verletzt, gibt die Methode eine Fehlermeldung aus und endet.
2. **Invers-Daten bestimmen**. Die Methode bestimmt die Daten, die nötig sind, um den Effekt der Methode umzukehren. Dazu wird auch die dafür nötige Aktionsmethode bestimmt.
3. **Änderungen animieren**. Die Veränderung der Datenstruktur wird, wenn möglich, (animiert) in der Visualisierung dargestellt.
4. **Daten ändern**. Die in der Visualisierung sichtbaren Daten werden auch intern im Modell verändert.
5. **Zustand darstellen**. Wenn nötig wird nun der neue Zustand der Modell-Daten in der Visualisierung dargestellt.

Die Daten werden durch die GUI des Modells visualisiert. Sie nimmt auch die Interaktionen des Benutzers entgegen, mit denen er Funktionen aufruft und prüft auf einfache syntaktische Fehler (z.B. Anzahl und Typ der Parameter).

Im *Modus „Nachgehen“* vergleicht der Scheduler die Aufrufe des Benutzers mit denen der Referenzimplementierung und lässt nur die gleichen Schritte wie die Referenzimplementierung zu.

Im *Modus „Experimentieren“* ignoriert der Scheduler die Aufrufe der Referenzimplementierung und leitet alle Aufrufe des Lerners an die Aktionsmethoden durch. Da auch hier intern inverse Call-Objekte erzeugt werden, kann der Lerner über den Scheduler Schritte zurücknehmen (Undo).

Zusammenfassung und Ausblick

Die Interaktive Visualisierung von Algorithmen sollte kein Selbstzweck sein, sondern im Blick auf den Einsatzzweck gestaltet werden. Sollen interaktive Visualisierungen für das Lernen von Algorithmen verwendet werden, so ist die didaktische Einbettung von großer Bedeutung. Als Teil der didaktischen Einbettung wurden in diesem Artikel die Lernformen „Betrachten“, „Nachgehen“ und „Experimentieren“ vorgestellt. Wäh-

rend es viele Veröffentlichungen zu technischen Fragen rund um die Animation von Algorithmen gibt, werden didaktische Fragestellungen nur selten behandelt. Hier ist sicher noch Forschungsbedarf gegeben.

Neue Lernformen wie „Nachgehen“ und „Experimentieren“ müssen sich in der Praxis bewähren. Dazu braucht es zunächst noch mehr Lernmaterial (z.B. Lernsoftware), die nach diesen Methoden gestaltet ist. Anschließend ist der Lernerfolg dieser Vermittlungsmethoden zu evaluieren.

Abschließend möchte ich die Leser ermutigen, sich anhand der genannten Lernsoftware ([4], [10])ein eigenes Bild davon zu machen, was die neuen Lernformen leisten können. Die Lernsoftware kann mit normalen Webbrowsern aus dem Internet geladen und ausgeführt werden.

Quellen

[1] John T. Stasko, John B. Domingue, Marc H. Brown und Blaine A. Price (Hrsg.): *"Software Visualization"*. 1998. MIT Press, Cambridge, Massachusets.

[2] Peter Gloor: „User Interface Issues for Algorithm Animation“ in: [1], S. 145-152.

[3] A. Barbu, M. Dromowicz, X. Gao, M. Köster und C. Wolf: *Lernsoftware „Sortieren mit Bubblesort und Quicksort“*. 1998. http://www-cg-hci.informatik.uni-oldenburg.de/~musik/lp_musik.html#label19 (letzer Zugriff: 4.3.2000).

[4] Karsten Block: *"Lernsoftware zum Binomial Heap Algorithmus"*. 1999. http://www-cg-hci.informatik.uni-oldenburg.de/~musik/lp_musik.html#label2 (Zuletzt abgerufen 4.3.2000).

[5] Linda Stern, Harald Sondergaard and Lee Naish: *"A Strategy for Managing Content Complexity in Algorithm Animation"*. PP 127-130 in: Bill Manaris (Ed.), Proceedings of the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education - ITiCSE '99. June 1999. ACM Press, New York.

[6] Nils Faltin: *"Gestaltung von Lernprogrammen zu Algorithmen für aktives Lernen mit virtuellen Brettspielen"*. 1999. Tagungsband der Tagung "Informatiktage 1999" der Gesellschaft für Informatik, 12. - 13. November 1999 in Bad Schussenried. Konradin Verlag. Preprint-Version: http://www-cg-hci.informatik.uni-oldenburg.de/~musik/Gest_LP/

[7] John Stasko: *„Smooth Continous Animation for Portraying Algorithms and Processes“*. S. 103-118 in [1].

[8] Marc Brown: *"Algorithm Animation"*. 1987. MIT Press, Cambridge.

[9] Karsten Block, Tammo Freese, Palle Klante et al.: *Lernsoftware "Computergrafik Interaktiv - Grafiti"*. http://www-cg-hci.informatik.uni-oldenburg.de/~musik/lp_musik.html#label1 . 1999. (Zuletzt abgerufen 4.3.2000).

[10] Peter Gorny und Nils Faltin: *"Das Projekt Medienunterstütztes Studium der Informatik (MuSIK)"*. 2000. http://www-cg-hci.informatik.uni-oldenburg.de/~musik/ (Zuletzt abgerufen 4.3.2000).