Erstellung eines Programmierframeworks zur Visualisierung von Algorithmen im Unterricht

Jan Kramer

12. Oktober 2008

Universität zu Lübeck
Institut für Theoretische Informatik

Studienarbeit

Erstellung eines Programmierframeworks zur Visualisierung von Algorithmen im Unterricht

von
Jan Kramer

Aufgabenstellung und Betreuung:

Prof. Dr. T. Tantau

Lübeck, den 12. Oktober 2008

Kurzfassung

Diese Studienarbeit stellt das Framework zur Algorithmusvisualisierung ALF vor. Die Struktur und die Funktionsweise werden erklärt und die Benutzung des Framework werden beschrieben. Mit dem Konzept des Algorithmischen Labors werden Pinzipien für das Interaktionsdesign und das Interfacedesign von Applikationen zur Algorithmusvisualisierung beschrieben, die mit Hilfe von ALF umgesetzt werden können. Die Entwicklung einer solchen Applikation wird am Beispiel des Algorithmus zur Matrix-Matrix-Multiplikation erläutert.

Abstract

This thesis presents the algorithm-visualization-framework ALF. Its topology and functionality will be explained and it will be described how to use this framework. The introduced concept of algorithm labs provides principles for an algorithm-visualization-applications interaction design and interface design, which can be realized supported by ALF. As an example the implementation of the matrix-matrix-multiplication algorithm is explained.

Inhaltsverzeichnis

1 Einleitung

Diese Studienarbeit beschäftigt sich mit den Möglichkeiten, den Computer als Hilfsmittel zur Vermittlung von Algorithmen und deren Funktionsweise zu benutzen. Es wird das Konzept des Algorithmischen Labors entwickelt und das Algorithmus-Labor-Framework (ALF) vorgestellt, ein Prototyp für ein Framework, welches die Erstellung solcher Algorithmischen Labore erleichtern soll.

1.1 Motivation

Der Informatikunterricht soll die systematische Verarbeitung von Informationen, wie sie an Computern geschieht, vermitteln. Sowohl in der Schule als auch an der Universität wird dazu unter anderem versucht den Lernenden die Funktionsweise von bestimmten Algorithmen zu erklären. So stehen im Informatikunterricht in der Schule meist grundlegende Algorithmen zum Sortieren von Datensätzen auf dem Lehrplan, wie zum Beispiel der Bubble-Sort-Algorithmus. In den Vorlesungen der Universität werden weitere, teilweise komplexere Algorithmen, beispielsweise der Quick-Sort-Algorithmus, vorgestellt und miteinander verglichen. Auch speziellere Algorithmen, die auf bestimmte Datenstrukturen zugeschnitten sind, werden hier behandelt. Ein Beispiel hierfür ist der Algorithmus für das Einfügen in einen AVL-Baum.

Die Lernenden sollen diese Algorithmen verstehen und deren Funktionsweise nachvollziehen können. Um dies zu erreichen, werden häufig eine oder mehrere der folgenden Methoden verwendet. Ein Algorithmus kann vom Lehrenden als Text in Form von Pseudocode oder einer bestimmten Programmiersprache vorgestellt werden. Der Lehrende kann ebenso die Funktionsweise des Algorithmus erklären indem er eine Folienfolge präsentiert, ein Tafelbild erstellt, den Algorithmus in einem Spiel nachstellt oder ihn von den Lernenden eigenständig nachprogrammieren läßt.

Ein Algorithmus ist eine genau definierte Abfolge von Instruktionen und kann daher gut als Text formuliert werden. Ob dies nun in Form einer gewissen Programmiersprache, als Pseudocode oder als natürlichsprachliche Aufzählung von Anweisungen geschieht, hängt davon ab, in welchem Rahmen der Algorithmus diskutiert wird.

Einem Lernenden, der noch keine Vorkenntnisse von Programmiersprachen hat, fällt es natürlich am leichtesten, den Algorithmus zu verstehen, wenn dieser in natürlicher Sprache vom Lehrenden erklärt wird. Aber auch Lernenden mit umfangreichen Programmierkenntnissen kann auf diese Art ein komplexer Algorithmus erklärt werden, wenn der grobe Ablauf wichtig ist und auf kleine Details keine Rücksicht genommen werden soll, weil diese zum Beispiel schon ausreichend behandelt wurden. Diese Art der Vorstellung hat jedoch den Nachteil, dass sehr viel Text gelesen werden muss, wenn jedes Detail des Algorithmus verstanden werden soll oder dass eventuell zum Verständnis wichtige Details fehlen, wenn der Text kurz gehalten werden soll. Um den Text kürzer zu gestalten, wird oft ein Pseudocode benutzt. Dieser Pseudocode kann ganz unterschiedlich ausfallen. Im wesentlichen werden Algorithmusschritte aufgelistet, die noch natürlichsprachlich oder schon eher in Form einer Programmiersprache beschrieben werden. Je mehr die Lernenden über Programmiersprachen wissen, desto mehr sieht so ein Pseudocode häufig wie eine Programmiersprache aus. Auch hierbei können für das Verständnis unnötige Details, sei es weil diese bereits bekannt sind oder weil sie noch zu komplex sind, ausgeblendet werden, indem einzelne Schritte zu einem Schritt zusammengefasst werden. Steht neben dem Verständnis des Algorithmus auch das Erlernen einer Programmiersprache im Vordergrund, so wird der Algorithmus häufig in der entsprechenden Programmiersprache vorgestellt. An solch einem Text kann der Lehrende sehr genau erklären, was zu welchem Zeitpunkt vom Computer ausgeführt wird.

Lernende, die mit ihren Informatikkenntnissen noch weit am Anfang stehen, wird der Zugang zu Algorithmen mit natürlichsprachlichen Texten leichter fallen, während Lernenden mit mehr Erfahrung häufig Pseudocode oder eine Programmiersprache vorgesetzt wird. Die Nachbearbeitung des Unterrichts gestaltet sich hierbei recht einfach, indem der Lernende sich den Text immer wieder durchlesen kann, um die Abfolge des Algorithmus nachzuvollziehen. Jedoch kann es auch vorkommen, dass einem Lernenden der Algorithmus in Textform zu abstrakt und unzugänglich vorkommt und er deshalb die Funktionsweise nicht vollständig versteht.

Sehr viel anschaulicher ist es, wenn der Ablauf eines Algorithmus mit Hilfe einer Folienfolge präsentiert wird. Ein bestimmtes Beispiel wird dafür nach jedem Algorithmusschritt bildlich dargestellt. Durch die Aneinanderreihung dieser Bilder kann die Funktionsweise des Algorithmus erklärt werden. Eine solche Folienfolge kann entweder als Folge von echten Folien existieren, die nacheinander auf den Overheadprojektor gelegt wird, oder als eine Präsentation am Computer erstellt werden, die mittels eines Beamers oder anderer Ausgabemedien präsentiert wird. Die Erstellung solcher Folien ist in jedem Fall aufwendig. Andererseits kann eine einmal erstellte Folienfolge immer wieder verwendet werden und eventuell den Lernenden zugänglich gemacht werden, indem sie kopiert wird oder ins Internet gestellt wird. Neben dem großen Aufwand zur Erstellung von Folienfolgen, gibt es allerdings noch weitere Nachteile. Das gewählte Beispiel zeigt immer einen sehr speziellen Ablauf des Algorithmus und kann deshalb

die Funktionsweise nicht so abstrakt darstellen. Die Präsentation ist außerdem sehr starr. Es kann nur schwer etwas während des Unterrichts abgeändert werden, um zum Beispiel Verständnisfragen zum Ablauf zu klären.

Die Erstellung eines Tafelbilds während des Unterrichts, ermöglicht es, einen Algorithmus etwas flexibler zu beleuchten. Der Unterricht kann zudem etwas interaktiver gestaltet werden, indem die Lernenden bei der Erstellung des Tafelbilds mit einbezogen werden. Dies kann die Konzentration und den Grad des Verständnisses erhöhen, da die Lernenden nicht nur passiv den Stoff vorgetragen bekommen, sondern auch aktiv mitdenken müssen. Da ein solches Tafelbild jedoch relativ spontan entsteht, kann es leicht dazu kommen, dass es unübersichtlich wird. Desweiteren schleichen sich leicht Fehler ein, die die Lernenden später verwirren können. Außderdem läßt sich ein Tafelbild oftmals schwer in die eigenen Unterlagen übernehmen. Die Lernenden können unter Umständen den Stoff also nicht nachbearbeiten.

Die Spielerische Nachstellung des Algorithmusablaufs ist eine weitere Form, den Unterricht aktiv zu gestalten. Die Lernenden führen dazu einzelne Schritte des Algorithmus selbst aus. Sortieralgorithmen könnten so beispielsweise erklärt werden, indem eine Gruppe von Lernenden sich nach den Vorgaben des Algorithmus selbst sortiert. Die Nachbearbeitung fällt bei dieser Methode allerdings besonders schwer, da das gesamte Geschehen im Klassenraum oder dem Hörsaal schwer aufgezeichnet werden kann. Es läßt sich also nicht zu Hause nachvollziehen. Außerdem ist es sehr zeitaufwendig, einen Algorithmus so durchzuspielen.

Häufig sollen Lernende den vorgestellten Algorithmus eigenständig programmieren um diesen besser zu verstehen. Auch diese Methode ist sehr zeitaufwendig. Außerdem ist sie nicht für jedes Publikum von Lernenden gleichermaßen geeignet. Ein Lernender ohne Programmierkenntnisse wird sich mit dieser Aufgabe sehr schwer tun. Auch Lernende mit Programmierkenntnissen können jedoch an dieser Aufgabe scheitern. Dieser Frust ist sicherlich nicht motivationsfördernd. Andererseits hat diese Methode aber auch einen entscheidenden Vorteil gegenüber den Vorigen: Der Lernende kann frei mit dem Algorithmus experimentieren. Während des Programmierens kann er den Algorithmus verändern und die Folgen beobachten. Er kann frei wählen, zu welchen Zeitpunkten im Ablauf des Algorithmus ein Abbild des Zustands ausgegeben werden soll.

1.2 Ziel der Arbeit

Die Vermittlung von Algorithmen geschieht heutzutage meist durch eine Kombination dieser Methoden, wobei sich der Computereinsatz meist nur auf die Anzeige von

Folien oder das Nachprogrammieren beschränkt. Diesem Umstand soll das in dieser Arbeit vorgestellte Konzept des *Algorithmischen Labors* Abhilfe schaffen. Es handelt sich hierbei um Programme, welche die Ausführung eines bestimmten Algorithmus für beliebige Beispieleingaben visualisieren.

Für einige Algorithmen existieren bereits verschiedene Programme, die dies leisten. Die Entwicklung solcher Programme ist jedoch sehr aufwändig und die bereits existierenden Programme unterscheiden sich zum Teil stark in ihrer Bedienung. Algorithmischen Labore sollen daher ein einheitliches und einfaches Bedienkonzept vorweisen können, damit die Lernenden sich auf den Inhalt konzentrieren können und sich nicht sonderlich mit der Bedienung auseinandersetzen müssen. Außerdem soll die Entwicklung eines Algorithmischen Labors möglichst einfach gehalten werden. Deshalb wurde im Rahmen dieser Studienarbeit ein Prototyp für ein Framework entwickelt, welches dem Entwickler große Teile der Entwicklung abnimmt und zudem ein einheitliches Design und Bedienkonzept verschiedener Algorithmischer Labore sicherstellt. Dieses Algorithmische-Labor-Framework (im weiteren kurz *ALF*) wurde auf die Entwicklung eines exemplarischen Algorithmischen Labors hin zugeschnitten.

1.3 Aufbau der Arbeit

Im zweiten Kapitel wir das Konzept des Algorithmischen Labors ausführlich vorgestellt. Dort werden mögliche Gestaltungsprinzipien für Algorithmischen Laboren diskutiert. Der Schwerpunkt liegt dabei auf den Interaktionsformen und der Art der Visualisierung, sowie der bisher im ALF-Frameworks realisierten Umsetzung dieser. Der Aufbau von ALF ist das Thema im dritten Kapitel. Die Zusammenhänge zwischen den einzelnen Klassen, die Benutzung des Frameworks und dessen Erweiterungsmöglichkeiten werden dort erklärt. Als Fallbeispile wird die Entwicklung eines Algorithmischen Labors zum Thema Matrix-Matrix-Multiplikation im vierten Kapitel beschrieben.

2 Konzept des Algorithmischen Labors

Das Konzept des Algorithmischen Labors soll die in der Einleitung erwähnten Methoden zur Vermittlung von Algorithmen ergänzen. In einem realen Labor führen Forscher Experimente durch, indem sie verschiedene Komponenten zusammenstellen und dann beobachten was geschieht. Aus diesen Beobachtungen ziehen sie dann Rückschlüsse über gewisse Zusammenhänge und erschaffen, bekräftigen oder widerlegen Theorien, zu dem Zweck, den Gegenstand der Forschungen besser zu verstehen.

Als ein Algorithmisches Labor bezeichnen wir eine Applikation, die am Computer Komponenten zur Verfügung stellt, um Funktionsweisen bestimmter Algorithmen bei verschiedenen Eingaben zu beobachten. Diese können im Unterricht vom Lehrenden vorgeführt werden oder vom Lernenden zu Hause zur Nachbearbeitung genutzt werden. Auch unerfahrene Lerner ohne Programmierkenntnisse sollen damit umgehen und ihren Kenntnisstand erweitern können.

Da die Idee solcher Algorithmusvisualisierung nicht neu ist, gibt es bereits viele Applikationen, die das Geforderte leisten, sich jedoch voneinander stark in der Bedienung und dem Aussehen unterscheiden. Beim Einsatz solch verschiedener Applikationen in einer Lehrveranstaltung, kann es passieren, dass der Aufwand zum Erlernen der Bedienung der Applikationen zu hoch im Verhältnis zum ursprünglich zu vermittelnden Wissen ist. Deshalb sollen hier Gestaltungsprinzipien gesammelt werden, nach denen ein Algorithmisches Labor aufgebaut sein sollte.

2.1 Interaktionsformen

Als erstes legen wir die Interaktionsformen eines Algorithmischen Labors fest. Diese sollten leicht verständlich sein, damit auch unerfahrene Nutzer einen leichten Zugang dazu haben. Um die effiziente Nutzung während des Unterrichts zu gewährleisten, sollte das Algorithmische Labor möglichst einfach zu bedienen sein.

Das Algorithmische Labor sollte sich mit Hilfe einer Maus (oder eines ähnlichen Zeigeinstruments) und einer Taste darauf bedienen lassen. Für diese Anforderung gibt es verschiedene Gründe. Die Komplexität einer solchen Bedienung ist sehr gering und läßt sich schnell erlernen. Der Benutzer muss sich nicht mit den verschiedenen Auswirkungen unterschiedlicher Tasten auf der Maus oder Tastenkombinationen auf der Tastatur auseinandersetzen. Ein anderer Grund ist, dass der Lehrende während einer Lehrveranstaltung nicht unbedingt in der Nähe des Computers bleiben muss. Mit einer kabellosen Maus oder anderen Alternativen kann er so freier agieren. Möglich wäre beispielsweise auch die Anwendung eines Algorithmischen Labors auf einem Touchscreen. Auch an einem Notebook lassen sich mit einem Touchpad Anwendungen leichter bedienen, die nur eine Taste zur Interaktion vorsehen.

Die Hauptelemente eines Algorithmischen Labors sollen Matten sein. Damit sind Flächen gemeint, die sich ähnlich wie die Fenster eines Betriebssystems verhalten. Diese Matten können verschoben werden, indem sie mit gedrückter Maustaste gezogen werden. Einige Matten lassen sich vom Bildschirm löschen, indem auf einen roten Knopf mit einem Kreuz, der in der rechten oberen Ecke ist, geklickt wird. Durch einen Klick auf eine Matte, wird diese in die oberste Ebene geholt und überdeckt alle anderen Matten. Die Matten passen sich in der Größe ihrem Inhalt an. Der Unterschied zu Fenstern ist, dass die Matten nicht vom Benutzer in der Größe verändert und nicht minimiert werden können. Es gibt Algorithmus-Matten, Eingabe-Matten, Programmablauf-Matten und Kommentar-Matten, die anhand ihres Aussehens und ihres Verhaltens voneinander unterscheidbar sind.

Eine Algorithmus-Matte besitzt einen Titel, der den Namen des Algorithmus angibt, ein oder mehrere Eingabefelder, welche die nötigen Eingaben für den Algorithmus symbolisieren, und einen Start-Knopf. Diese Matten können nicht gelöscht werden.

Eingabe-Matten beinhalten Daten, die für einen Algorithmus als Eingabe dienen können. Durch einen Klick auf einen Kopieren-Knopf lässt sich die gesamt Matte samt Inhalt kopieren. Wird der Mauszeiger über einem Datum einer Eingabe-Matte platziert, so erscheinen Optionen mit denen dieses Datum verändert werden können. Auch die Datenstruktur kann verändert werden, indem Daten hinzugefügt oder gelöscht werden. Auf diese Weise lassen sich beliebige Beispiele erstellen.

Um eine Eingabe-Matte als Eingabe zu nutzen, muss diese auf ein geeignetes Eingabefeld einer Algorithmus-Matte gezogen werden. Die Eingabefelder der Algorithmus-Matten nehmen nur Eingaben an, die der jeweilige Algorithmus auch verarbeiten kann. Dieses Verhalten muss dem Benutzer deutlich gemacht werden.

Im ALF-Framework wird durch farbliche Codierung (siehe Abbildung 2.1) dargestellt, welche Eingabe-Matten auf welche Eingabefelder passen. Außderdem wird dem Benutzer symbolisiert, dass ein Eingabefeld eine Eingabe-Matte aufnehmen würde, in-

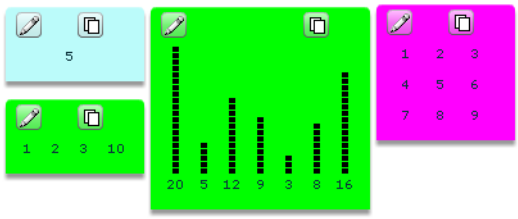

Abbildung 2.1: Eingabe-Matten mit verschiedenen Datenstrukturen als Inhalt

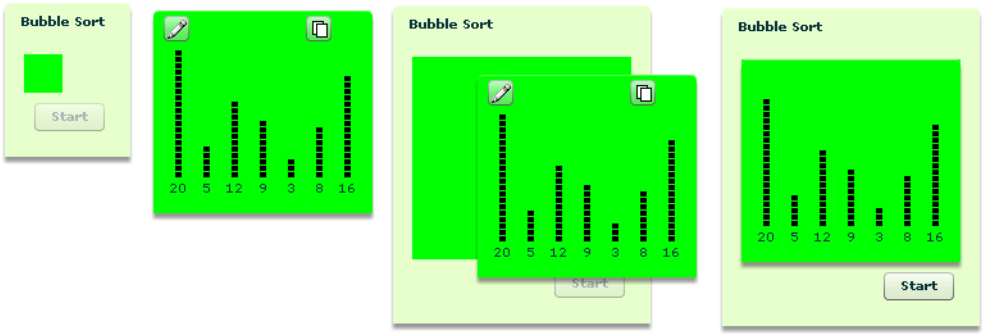

Abbildung 2.2: Eingabe-Matte wird über ein Eingabefeld gezogen

dem sich die Größe des Eingabefeldes der Größe einer darübergezogenen Eingabe-Matte anpasst, vergleiche Abbildung 2.2.

Sind auf einer Algorithmus-Matte alle Eingabefelder belegt, so wird ein Knopf zum starten des Algorithmus aktiviert. Durch einen Klick auf diesen Knopf wird der Algorithmus ausgeführt und eine Programmablauf-Matte wird erstellt, welche eine Liste aller Schritte des Algorithmus, sowie eine Visualisierung des Zustands, in dem sich der Algorithmus beim ausgewählten Schritt befindet, enthält. Mit Zustand des Algorithmus ist hier ein Schnappschuß der Daten, mit denen der Algorithmus arbeitet, nach einem bestimmten Schritt gemeint. Der visualisierte Algorithmuszustand kann aus der Liste der Algorithmusschritte frei ausgewählt werden. Die Abfolge der Schritte wird hintereinander abgespielt. Das Abspielen läßt sich pausieren und in der Geschwindigkeit regeln. Eine Programmablauf-Matte kann auf der Oberfläche bewegt oder gelöscht werden, um Platz für weitere Programmablauf-Matten zu schaffen. Mehrere Matten dieser Art können so zum Vergleich nebeneinander gelegt werden. In Abbildung 2.3 ist eine Programmablauf-Matte von ALF abgebildet.

Für Kommentare, Anweisungen oder Tipps an den Benutzer sind Kommentar-Matten vorgesehen. Diese öffnen sich, wenn sich der Mauszeiger über ihnen befindet, um den ganzen Text anzuzeigen und verkleinern sich automatisch, wenn der Mauszeiger sich nicht mehr über ihnen befindet. Sie können auch in der Größe festgestellt, verschoben

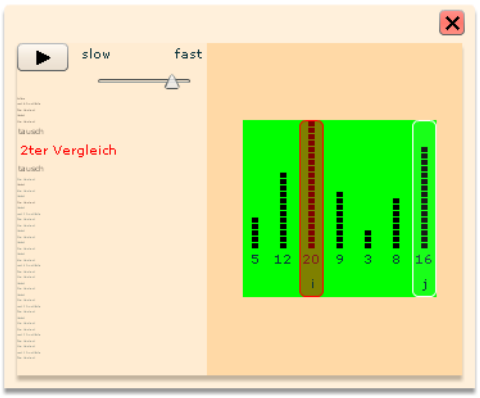

Abbildung 2.3: Programmablauf-Matte

oder geschlossen werden.

In *The Humane Interface* fordert Raskin [1] die Abschaffung von Modi bei Interfaces, um mögliche Modusfehler bei der Anwendung eines Programms zu vermeiden. Die aktuelle Implementierung von ALF besitzt noch einige Modi, welche allerdings abgeschafft werden könnten, um die Bedienung weiter zu vereinfachen und um Verwirrungen vorzubeugen. Der Editiermodus der Eingabematten kann abgeschafft werden, indem die Inhalte dieser Matten immer veränderbar sind. Diese Maßnahme hätte zur Folge, dass die Eingabematten nur noch an ihren Rändern gezogen werden kann. Desweiteren befindet sich auf der Programmablauf-Matte ein Knopf, der je nach Modus das Symbol für Play oder das Symbol für Pause anzeigt. Zeigt dieser Knopf das Pause-Symbol an, ist nicht sofort klar, ob die Programmablauf-Matte gerade im Pause-Modus ist oder ob sie sich durch einen Klick auf den Knopf in diesen versetzen lässt. Dieser eine Knopf sollte in späteren Implementierungen durch zwei Knöpfe, für jedes Symbol einen, ersetzt werden. Dies schafft erstens einen Modus ab und zeigt zweitens klar an, ob die Zustandsfolge gerade abgespielt wird oder ob das Abspielen pausiert wurde.

Das Erscheinungsbild der mit ALF programmierten Applikationen kann noch übersichtlicher gestaltet werden, indem die Knöpfe in der Titelleiste der Matten nur dann sichtbar sind, wenn der Mauszeiger sich über der Matte befindet. Außerdem sollten die Eingabefelder der Algorithmus-Matten und die Titelleisten der Eingabe-Matten eindeutige Beschriftungen tragen, an denen auch ohne die farblich Codierung erkannt werden kann, was zueinander passt.

2.2 Präsentation

Der Erfolg beim Einsatz von multimedialen Hilfsmitteln im Unterricht hängt jedoch nicht nur von der Bedienbarkeit ab. Ebenso muss beachtet werden, dass der Inhalt entsprechend präsentiert wird. Befassen wir uns hier also mit dem Umfang, der Farbgestaltung, der Skalierung und der Aufmerksamkeitssteuerung.

Ein zu umfangreiches Beispiels macht dieses unübersichtlich und schwer verständlich. Der für das Verständnis wichtige Teil kann leicht übersehen werden, wenn den Lernenden ein überbordendes Beispiel zugemutet wird. Ein Beispiel, welches im Unterricht vorgetragen wird, sollte deshalb so gewählt sein, dass wesentliche Funktionsweisen eines Algorithmus deutlich werden, aber nicht durch zu häufige Wiederholung den Lernenden langweilt.

Die Farbgestaltung eines Beispiels sollte der Situation angemessen sein. Im wesentlichen gibt es bei der Anwendung eines Algorithmischen Labors zwei Situationen. Während des Unterrichts sind andere Schwerpunkte bei der Farbgestaltung zu setzen als bei der Verwendung des Algorithmischen Labors zum E-Learning zu Hause.

Im Unterricht werden mittlerweile überwiegend Beamer zur Präsentation genutzt. Somit kann der Bildschirminhalt ausreichend groß dargestellt werden. Allerdings haben Beamer auch Schwächen. Die Farbtreue und der Kontrast der Projektion leiden stark unter dem Umgebungslicht.

Ein ausreichender Kontrast muss daher sichergestellt werden. Texte sollten deshalb möglichst immer schwarz dargestellt werden. Für andere wichtige Objekte ist ebenfalls eine möglichst dunkle Farbe zu wählen. Der Hintergrund ist weiß und andere hintergründige Flächen sind entweder auch weiß oder in sehr hellen Farben. Sollen durch verschiedene Farben Abgrenzungen zwischen Objekten dargestellt werden, so müssen sich diese deutlich voneinander unterscheiden. Sind die gewählten Farben sich zu ähnlich, kann es passieren, dass sie bei der Projektion nicht mehr voneinander unterschieden werden können.

Bei der Nachbearbeitung zu Hause kommt in der Regel kein Beamer zum Einsatz, deshalb muss nicht besonders auf einen hohen Kontrast geachtet werden. Der Hintergrund kann hier also auch farbig gestaltet werden, um die Verwendung des Algorithmischen Labors angenehmer für das Auge zu machen. Zu beachten bleibt jedoch immer noch, dass, laut Herczeg [2], nicht mehr als sechs Farben verwendet werden sollten, wenn diese zur Abgrenzung von verschiedenartigen Objekten dienen sollen. Außerdem sollte nicht unbeachtet bleiben, dass „etwa 9% aller Männer und 0,8 % aller Frauen“ eine Rot-Grün-Sehschwäche haben[3]. Die Farbgestaltung sollte also höchstens eine Hilfe zur Benutzung des Programms darstellen.

Die Größe eines Algorithmischen Labors muss ebenfalls anpassbar sein, damit es gleichermaßen an einem Laptop, einem PC, sowie bei der Projektion über einen Projektor verwendbar ist. Dazu zählt die Schrifftgröße genauso wie die Größe der Applikation selbst.

Durch die Verwendung von Cascading-Styles-Sheets kann die Anpassung der Farben und der Größe eines Algorithmischen Labors komfortabel ohne Änderungen im Code erreicht werden.

Bei komplexen Vorgängen kann es schnell passieren, dass der Lerner die Übersicht verliert und die wichtigen oder interessanten Bereiche aus dem Auge verliert. Mit der Hervorhebung dieser wichtigen Bereiche lässt sich dem vorbeugen. Gängige Methoden einen Benutzer auf etwas aufmerksam zu machen, sind farbliche Markierungen und Animationen.

Ein Bereich, der zu einem bestimmten Zeitpunkt von besonderer Bedeutung ist, kann durch eine farbliche Markierung hervorgehoben werden. Dies kann durch die Umrandung des Bereichs oder dessen komplette Einfärbung geschehen. Auch bestimmten Datentypen kann durch Farbe eine Bedeutung gegeben werden. Ein boolescher Wert kann, in Anlehnung an eine Verkehrsampel, durch zwei übereinanderliegende Kreise dargestellt werden. Ist der Wert true ist der untere Kreis grün und andernfalls der obere Kreis rot eingefärbt. Ein Integer-Wert kann, wenn er als Zählvariable für ein Array benutzt wird, durch eine Umrandung des jeweiligen Array-Eintrags dargestellt werden.

Animationen erregen beim Betrachter immer Aufmerksamkeit. Durch eine einfache Animation, wie ein Aufblinken, kann so ein bestimmtes Ereignis hervorgehoben werden. Außerdem kann eine Animation helfen den Zusammenhang zwischen zwei Darstellungen herzustellen, indem sie die Zwischenschritte zeigt. Die Vertauschung von zwei Array-Einträgen zum Beispiel, könnte dem Betrachter möglicherweise entgehen, wenn ihm nur die Vorher-Nachher-Darstellug dargeboten wird. Wird die Vertauschung jedoch durch eine Animation hervorgehoben, so wird diese Änderung bestimmt nicht übersehen. Ein weiteres Problem, das dadurch behoben wird, ist dass bei der Betrachtug von zwei Zuständen, nicht immer klar ist, wie die Änderung zu Stande gekommen ist.

3 Das Visualisierungssystem ALF

Dieses Kapitel beschreibt das während dieser Studienarbeit entstandene Algorithmus-Labor-Framework (kurz ALF). Dieses Framework soll die Entwicklung von Algorithmischen Laboren vereinfachen, indem es eine Bibliothek von Klassen zur Verfügung stellt, welche die im vorigen Kapitel genannten Anforderungen bereits erfüllen. Der Entwickler muss nur den Algorithmus entsprechend anpassen, eine Darstellung der vom Algorithmus verwendeten Daten erstellen und Beispieleingaben für den Algorithmus anlegen.

Zunächst betrachten wir ALF hinsichtlich der Aspekte der Algorithmenvisualisierung, später wird der Aufbau des Frameworks erläutert. Am Ende dieses Kaptiels werden die Benutzung von ALF und dessen Erweiterungsmöglichkeiten beschrieben.

3.1 Aspekte der Algorithmenvisualisierung

Algorithmenvisualisierung wird von C. A. Bröcker in [4] als abstrakte Darstellung von grundlegenden Operationen und Datenstrukturen eines Programms definiert. Er grenzt sie damit von der Programmvisualisierung ab, welche eine spezifische Implementation visualisiert und bei der Implementierungsdetails sowohl des Programms als auch der verwendeten Datenstrukturen im Vordergrund stehen. Die Beschränkung auf die wesentlichen Schritte soll die Weitergabe von Konzepten und Ideen erleichtern und ein tieferes Verständnis fördern.

Bei der Entwicklung von Systemen zur Algorithmusvisualisierung müssen gewisse Aspekte berücksichtigt werden, von denen einige generell für Softwaresyteme und andere speziell für Algorithmusvisualisierungssysteme gelten. Diese lassen sich in drei Bereiche gliedern: Architektur, Gestaltung und Nutzung[4]

Aspekte der Architektur

Die Algorithmischen Labore, die mit ALF erstellt werden können, sollen in erster Linie als Anschauunsgmaterial in Lehrveranstaltungen dienen. Der Lehrende kann mit ihnen die Funktionsweise eines Algorithmus exemplarisch darstellen oder die Unterschiede mehrerer Algorithmen untereinander herausstellen. Die behandelten Algorithmen sollen sich nicht auf ein bestimmte Themengebiet beschränken, sondern die Möglichkeit bieten in verschiedenen Bereichen der Informatik angewendet zu werden. Das Framework ist also dementsprechend offen für Erweiterungen ausgelegt.

Die Visualiserung des Algorithmus muss Änderungen, die der Algorithmus an den Daten vornimmt, bemerken. Diese Kopplung ist bei ALF nach dem Konzept der interessanten Ereignisse realisiert. Dieses Konzept wurde von Brown im Visualisierungssystem *Balsa* eingeführt. Bei der Ausführungs eines Algorithmus werden nach bestimmten Schritten die Datenstrukturen komplett gespeichert. Nach der Ausführung des Algorithmus liegt so eine komplette Auflistung aller Schritte vor, die von Interesse sind. Hierzu muss der Algorithmus angepasst werden, indem an geeigneten Stellen Funktionsaufrufe eingefügt werden, die ein Abbild der verarbeiteten Daten speichert.

Sieht man sich die Computer-Ausstattung der Universitäten an, so fällt auf, dass es kein einheitliches Bild gibt, was die Nutzung von Betriebssystemen angeht. Auch bei der Studentenschaft kann man nicht davon ausgehen, dass alle dasselbe Betriebssystem benutzen. Eine möglichst hohe Plattformunabhängigkeit ist also nötig, damit der Lehrende das Algorithmische Labor auf dem Computer, der ihm zur Verfügung steht präsentieren kann und damit der Lernende es zur Nachbearbeitung nutzen kann.

Heutzutage ist beinahe jeder Computer mit dem Internet verbunden und irgendein Browser ist darauf installiert. Außderdem ist der Flash-Player, der als Browser-Plug-In für viele Web-2.0-Anwendungen benötigt wird, weit verbreitet. Daher basiert ALF auf dem Flex-Framework, mit welchem Flash-Animationen erstellt werden können, die somit sehr plattformunabhängig sind.

Aspekte der Gestaltung

Obwohl ALF so konstruiert ist, dass es die in Kapitel 2 erwähnten Anforderungen erfüllt, bleibt dem Entwickler eines Algorithmischen Labors noch Raum zur Gestaltung. Der Entwickler kann die Darstellung der Datenstrukturen und der Zustände des Algorithmus frei wählen. So ist es ihm möglich einen Schwerpunkt bei der Visualisierung zu setzen. Auch der Umfang der Visualisierung ist frei wählbar. Es kann

beispielsweise nur die Datenstruktur, die gerade bearbeitet wird angezeigt werden, als auch weitere Hilfsvariablen oder Kommentare. Auch der Grad der Abstraktion kann damit frei gewählt werden.

Aspekte der Nutzung

Jeder Algorithmus wird in ALF als eine Komponente entwickelt und kann daher durch Vererbung abgewandelt und erweitert werden und in verschiedenen Algorithmischen Laboren angewendet werden. Die Adaption von Algorithmen ist recht einfach, da nur ergänzt werden muss, zu welchen Zeitpunkten im Algorithmus Daten gespeichert werden, welche Daten gespeichert werden und wie diese dargestellt werden sollen.

Ein Algorithmisches Labor läßt sich, da es eine Flash-Animation ist, leicht auf jeder Internetseite einbinden und kann somit sehr gut zum E-Learning genutzt werden.

3.2 Struktur des Frameworks

In diesem Abschnitt werden die einzelnen Klassen des Frameworks, ihre Bedeutung und die Beziehung untereinander vorgestellt.

Implementierung der Matten

Die Klasse Mats ist eine abstrakte Oberklasse, die das allgemeine Verhalten der verschiedenen Matten regelt. Eine Matte (Instanz des Klasse Mat) kann bei gehaltener Maustaste verschoben werden. Bei einem einfachen Mausklick auf eine Matte wird diese in den Vordergrund geholt. Matten, deren Eigenschaft closable auf true gesetzt ist, können geschlossen werden.

Eine ContentMat-Instanz kann eine Instanz vom Typ DisplayableType enthalten und verwalten. Eine ContentMat nimmt automatisch die Farbe an, die dem Datentyp den sie enthält zugewiesen wurde. Diese Matten haben zwei zusätzliche Buttons: den Edit-Button und den Copy-Button. Durch einen Klick auf den Edit-Button kann der Inhalt verändert werden. Ein Klick auf den Copy-Button erzeugt eine neue ContentMat, mit einer Kopie des Inhalts.

Die Klasse InputMat ist die Basisklasse für die Algorithmusmatten. Um einen Algorithmus zu visualisieren muss eine neue Klasse geschrieben werden, die von die-

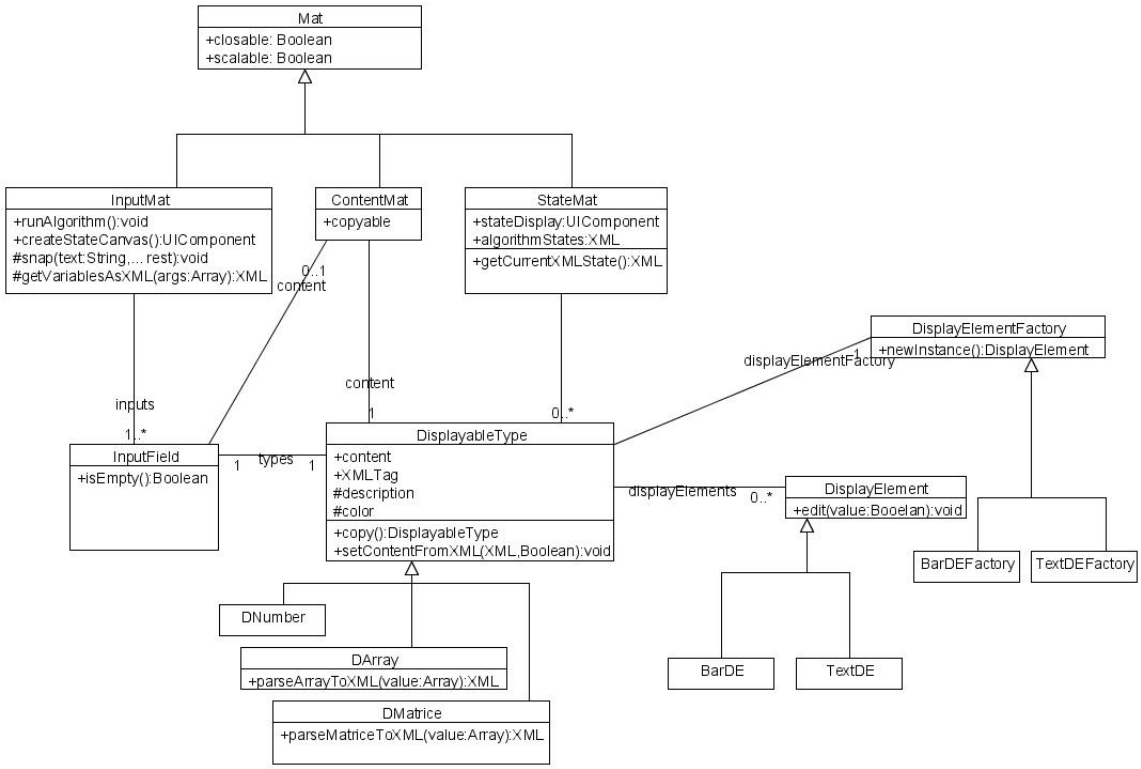

Abbildung 3.1: Klassendiagramm des Frameworks

ser Klasse erbt. Die neue Klasse muss folgende Funtktionen überschreiben: *runAlgorithm()*, *getVariableAsXML()* und *createStateDisplay()*. In die Funktion *runAlgorithm()* muss der Algorithmus geschrieben werden, der nach der Aktivierung durch den Start-Knopf ausgeführt werden soll. Diese Funktion hat keinen Rückgabewert. Für jeden Algorithmusschritt, der gespeichert werden soll, muss in dieser Funktion einmal die Funktion *snap()* aufgerufen werden, der weitere Parameter zur Beschreibung des Schrittes übergeben werden können. Die Funktion *getVariablesAsXML()* legt fest, welche Variablen bei einem Schnappschuß des Zustands gespeichert werden sollen und wie diese formatiert sind. Der Rückgabewert dieser Funktion ist ein XML-String, der alle zu speichernden Werte enthält. Mit der Funktion *createStateDisplay()* wird die Darstellung eines Zustands festgelegt, indem jedem Wert der nach der Ausführung des Algorithmus vorliegenden XML-Liste eine Visualisierung (Unterklassen von *DisplayableType*) zugeteilt wird. Diese Funktion gibt eine Instanz vom Typ UIComponent zurück, in der die entsprechenden Visualisierungenstypen angeordnet sind.

Nach dem Ablauf des Algorithmus in der *InputMat* erzeugt diese eine neue Instanz vom Typ *StateMat*. Dieser neuen Matte wird die Liste der Zustände übergeben. Außerdem wird ihr die durch *createStateDisplay()* erzeugte Komponente übergeben, welche festlegt, wie ein Zustand angezeigt werden soll. Die Aufgabe der *StateMat* ist es die Liste der Zustände anzuzeigen und den aktuell ausgewählten Zustand anzuzeigen. Mit dem Play/Pause-Knopf der StateMat kann das Abspielen der Zustandsfolge gestartet und gestoppt werden. Der Regler neben diesem Knopf regelt die Abspielgeschwindigkeit.

Darstellung von Datenstrukturen

Für die Darstellung von Datenstrukturen sind Unterklassen von *DisplayableType* und *DisplayElement* zuständig. Die Visualisierung für eine bestimmte Datenstruktur wird mit einer Unterklasse von *DisplayableType* beschrieben. Diese Unterklasse (beispielsweise DArray) muss sich um die Anordnung der einzelnen Daten auf dem Bildschirm kümmern. Dazu kann sie über eine oder mehrere Instanzen einer Unterklasse von *DisplayElement* besitzen. Eine solche Unterklasse kümmert sich um die Darstellung genau eines Datums. Beispielsweise kann eine Zahl entweder durch eine Zahl, die Höhe eines Balkens oder einen Grauwert dargestellt werden. Einer *DisplayableType*-Unterklasse wird dafür eine spezifische *DisplayElementFactory*-Unterklasse zugewiesen, die Instanzen der zugehörigen *DisplayElement*-Unterklasse erstellt.

3.3 Benutzung des Frameworks

Um ein Algorithmisches Labor zu entwickeln, muss zunächst für jeden Algorithmus, der darin vorgestellt werden soll, eine eigene Komponente erstellt werden, die von der Klasse *InputMat* abgeleitet wird. Danach wird eine Flex-Applikation erstellt, welche diese Komponenten und ein paar Beispieleingaben enthält. Das Flex-Framework wird in der Dokumentation von Adobe unter [5] ausführlich vorgestellt.

Um die *InputMat*-Unterklasse zu erstellen, muss eine Datei mit der Endung *.mxml* angelegt werden. Der Name der neuen Klasse wird durch den Dateinamen bestimmt. In der Datei muss folgendes stehen.

```
<!-- Algorithmus.mxml -->
<?xml version="1.0" encoding="utf-8"?>
<mats:InputMat xmlns:mats="mats"
            xmlns:mx="http://www.adobe.com/2006/mxml">
    <mats:inputs>
    </mats:inputs>
    <mx:Script>
        <![CDATA[
        ]]>
    </mx:Script>
</mats:InputMat>
```

Mit *xmlns:* können neben *mats* und *mx* noch weitere Namespaces definiert werden. Zwischen den *inputs*-Tags werden die benötigten Eingaben für den Algorithmus festgelegt. Zwischen den *Script*-Tags wird der ActionScript-Code angegeben, um die InputMat durch überschreiben der Funktionen *runAlgorithm()*, *getVariablesAsXML()* und *createStateCanvas()* anzupassen.

```
override public function runAlgorithm():void{}

override protected function
         getVariablesAsXML(args:Array):XML{}

override public function
         createStateDisplay():UIComponent{}
```

In der Funktion *runAlgorithm* wird der Code für den Algorithmus, der nach dem Klick auf den Start-Knopf ablaufen soll, angegeben. Dabei sollte nach jedem *inter-*

essanten Ereignis die Funktion *snap()* aufgerufen werden. Diese Funktion fertigt einen Schnappschuss der Daten an, die der Algorithmus bearbeitet. Dieser Funktion können beliebig viele Parameter mitgegeben werden, wobei der erste Parameter ein String sein sollte, der diesem Schnappschuss einen Namen gibt. Die weiteren Parameter werden der Funktion *getVariablesAsXML()* als *Array* übergeben.

Die Funktion *getVariablesAsXML()* gibt die Daten in einem XML-String zurück, welche bei einem Schnappschuss gespeichert werden sollen. Der XML-String muss als Wurzelelement ein *state*-Tag haben. Die zu speichernden Werte sind nacheinander in folgender Form anzugeben.

```
<Name> {Wert} </Name>
```

Der Wert kann bei den Grundtypen String, Number und Boolean direkt angegeben werden. Soll allerdings der Inhalt einer Datenstruktur gespeichert werden muss diese in dem Format gespeichert werden, dass die entsprechende Visualisierung auch lesen kann. Dazu werden statische Funktionen der jeweiligen Datenvisualisierungen benutzt. Das folgende Beispiel speichert die Werte eines Arrays *array1* und einer anderen Variablen *i*.

```
override protected function
         getVariablesAsXML(args:Array):XML
{
   return <state>
     <A>{DArray.parseArrayToXML(array1)}</A>
     <counter>{i}</counter>
   </state>;
}
```

Variablen, die bei einem Schnappschuss gespeichert werden sollen, müssen aufgrund dieser Architektur in dieser Klasse global deklariert werden.

Damit die StateMat, die nach Ausführung des Algorithmus entsteht, die XML-Daten, die ihr übergeben werden interpretieren kann muss noch die Funktion *createStateDisplay()* überschrieben werden. In dieser Funktion kann der Entwickler festlegen, wie die gespeicherten Schnappschüsse der Daten später dargestellt werden sollen. Die Funktion hat als Rückgabewert eine Instanz vom Typ *UIComponent*. Dies kann ein *Canvas* sein oder irgendeine andere Layout-Komponente des Flex-Frameworks. Dieser Komponente fügen wir Datenvisualisierungen hinzu und ordnen sie darauf an. Außerdem

teilen wir jeder Datenvisualisierung ein XML-Tag zu, welches sie auslesen soll. Ebenso kann hier auch die Darstellungsart gewählt werden. Folgendes Beispiel würde nur das Array anzeigen, welches zuvor unter *A* in der XML-Liste abgescpeichert wurde.

```
override public function createStateDisplay():UIComponent
{
    // erstelle eine neue Instanz
    var stateDisplay:DArray = new DArray();

    // weise 'Balken' als Darstellungsart zu
    stateDisplay.displayElementFactory =
                                    new BarDEFactory();

    // 'A' soll von stateDisplay ausgelesen werden
    stateDisplay.xmlTag = "A";
    return stateDisplay;
}
```

Diese neu erstellte Komponente muss nun in eine Flex-Applikation eingebunden werden. Es muss also eine neue Datei mit der Endung .mxml erstellt werden.

```
<!-- AlgorithmusLabor.mxml -->
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                xmlns:mats="mats"
                xmlns:*="*">
</mx:Application>
```

Mit dem Tag *¡Algorithmus/¿* wird der eben erstellte Algorithmus dieser Applikation hinzugefügt. Durch die Attribute *x* und *y* kann die Position bestimmt werden. Die Attribute *width* und *height* sollten jedoch nicht gesetzt werden, da sich sonst die Matte nicht mehr automatisch an ihren Inhalt anpasst.

Beispieleingaben für den Algorithmus werden mit Hilfe von *ContentMat*-Elementen hinzugefügt werden. Diese müssen einen bestimmten Datenvisualisierungstypen als Kind-Element haben. Die Datenvisualisierungstypen stammen aus dem Package *displayabletypes*. Dies wird der Applikation mit der Namespace-Definition *xmlns:disp= "displayabletypes"* mitgeteilt. Folgendes Beispiel fügt eine einfache Zahl zur Applikation hinzu.

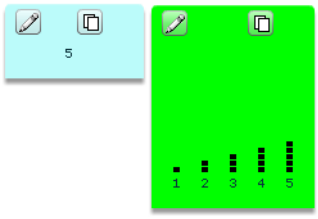

Abbildung 3.2: Verschiedene Beispieleingaben

```
<mats:ContentMat>
    <disp:DNumber>
        5
    </disp:DNumber>
</ContentMat>
```

Werden Beispiele so angelegt, wie im vorigen Beispiel, wird der Inhalt als normaler Text dargestellt. Sollen die Inhalte allerdings anders dargestellt werden, so muss noch eine *DisplayElementFactory* übergeben werden. Außerdem muss dabei der Inhalt gesondert durch den *content*-Tag markiert werden. Das folgende Beispiel erstellt eine ContentMat an der Position 50,0, die ein Array in Balkenform enthält.

```
<mats:ContentMat x="50" y="0">
    <disp:DArray>
        <disp:content>
            [1,2,3,4,5]
        </disp:content>
        <disp:displayElementFactory>
            <dispEl:BarDEFactory/>
        </disp:displayElementFactory>
    </disp:DArray>
</mats:ContentMat>
```

Wenn alle benötigten Beispieleingaben erstellt wurden, kann das Algorithmus-Labor mit dem folgenden Befehl kompiliert werden.

```
mxmlc -library-path+=[Pfad von ALF.swc] [Name der Applikation]
```

Wenn sich die Datei ALF.swc im Pfad ../ALF/ relativ zum Pfad der Applikation befindet, lassen sich die Beispiel-Dateien aus diesem Abschnitt mit folgendem Befehl kompilieren.

```
mxmlc -library-path+=../ALF/ AlgorithmusLabor.mxml
```

Als Ausgabe ensteht die Datei *AlgorithmusLabor.swf* welche entweder direkt mit dem Flasch-Player ausgeführt oder in eine Web-Seite eingebunden und von dort aus gestartet werden kann.

3.4 Erweiterungsmöglickeiten

Da es sich bei ALF nur um einen Prototypen handelt, möchte ich hier beschreiben, wie ALF erweitert werden kann. Mit der Darstellung von Arrays und Zahlen lassen sich nur einige grundlegende Algorithmen visualisieren, das Framework kann jedoch um neue Datenstrukturen erweitert werden, ohne dass es komplett neu geschrieben werden muss. Ebenso kann das Framework um neue Darstellungsweisen der Grundtypen erweitert werden.

Um dem Framework eine neue Datenstrukturvisualisierung hinzuzufügen, die alle Funktionen des Framworks unterstützt, muss eine neue Klasse erstellt werden, die von DisplayableType (oder einer ihrer Subklassen) erbt. Diese neue Subklasse muss sich um die Darstellung des in *content* gespeicherten Inhalts kümmern. Die Komponenten zur Anzeige der Grundtypen müssen strukturiert und verwaltet werden. Außerdem sollte eine Standardfarbe für diese neue Datenstrukturvisualisierung festgelegt werden, welche diese Datenstruktur von den anderen unterscheidbar macht. Zu empfehlen ist noch das Schreiben einer Funktion, die die Datenstruktur als XML-String abspeichert, sowie das Überschreiben der Funktion *setContentFromXML()* um diesen XML-String wieder auszulesen.

Dieses Framework ist so aufgebaut, dass die Darstellung von Grundtypen in den einzelnen Datenstrukturen beliebig ausgetauscht werden kann. Eine neue Darstellungsart eines Grundtypen kann also bei allen bereits bestehenden Datenstrukturen verwendet werden.

Um eine neue Darstellung zu erstellen, muss eine neue Klasse von der Basisklasse DisplayElement abgeleitet werden. Dort wird festgelegt, wie Inhalte dargestellt werden und wie sie zur Laufzeit verändert werden können. Damit die DisplayableType-Unterklassen die neuartige DisplayElement-Unterklasse erstellen können, muss ihnen eine spezielle DisplayElementFactory-Unterklasse übergeben werden.

4 Fallbeispiel: Matrix-Matrix-Multiplikation

Um die Erstellung eines Algorithmus-Labors zu veranschaulichen, wird in diesem Kapitel Schritt für Schritt erklärt, wie eine Demonstration der Matrixmultiplikation erstellt wird.

4.1 Erstellen einer InputMat

Um einen neuen Algorithmus demonstrieren zu können, muss eine neue Klasse erzeugt werden, die von der Klasse *InputMat* erbt. Dazu wird eine Datei mit dem Namen *Matrixmultiplikation.mxml* angelegt, um den Namen der Komponente auf „Matrixmultiplikation“ festzulegen. Der folgende MXML-Code beschreibt eine Klasse, die von *InputMat* erbt und in der Titelleiste mit „Matrixmultiplikation“ beschriftet ist.

```
<?xml version="1.0" encoding="utf-8"?>
<mats:InputMat  xmlns:mats="mats.*"
                xmlns:mx="http://www.adobe.com/2006/mxml"
                title="Matrixmultiplikation">
</mats:InputMat>
```

Die Eingaben, die der Algorithmus zu Berechnung braucht, werden mit dem XML-Tag *¡mats:inputs¿ ¡/mats:inputs¿* festgelegt. Hier geben wir eine Liste von Objekten des Typs DisplayableType oder dessen Unterklassen an. Im Falle dieses Beispiels soll der Algorithmus zwei Eingaben vom Typ DMatrice erwarten. Wir nennen sie *dMatrixA* und *dMatrixB*.

```
<mats:inputs>
    <displayabletypes:DMatrice id="dMatrixA"/>
    <displayabletypes:DMatrice id="dMatrixB"/>
</mats:inputs>
```

Um den Algorithmus festzulegen, der von der InputMat ausgeführt werden soll, muss innerhalb des *¡mx:Script¿* - Tags die Funktion *runAlgorithm()* überschrieben werden. Alle Variablen, die im Algorithmischen Labor angezeigt werden sollen, müssen global, also außerhalb dieser Funktion, definiert werden. Die Funktion *snap()* ruft die Routinen zum Speichern der interessanten Daten auf. Sie sollte nach jeder Ausführung eines wesentlichen Schritts des Algorithmus ausgeführt werden.

```
<mx:Script>
  <![CDATA[

    private var matrixA:Array;
    private var matrixB:Array;
    private var matrixC:Array;
    private var i:Number;
    private var j:Number;
    private var k:Number;
    private var produkt:Number

    override public function runAlgorithm():void
    {
      matrixA = dMatrixA.getMatrice();
      matrixB = dMatrixB.getMatrice();
      matrixC = new Array();

      // Inititalisierung von matrixC mit Nullen
      for (i = 0; i < matrixA.length; i++)
      {
        matrixC[i] = new Array();
        for (j = 0; j < matrixB[i].length; j++)
        {
          matrixC[i][j] = 0;
        }
      }

      snap("Anfang");

      for (i = 0; i < matrixA.length; i++)
        for (j = 0; j < matrixB[i].length; j++)
          for (k = 0; k < matrixB.length; k++)
            {
              produkt = Number(matrixA[i][k]) *
                                Number(matrixB[k][j]);
```

```
                snap("Produkt");
                matrixC[i][j] = matrixC[i][j] + produkt;
                snap("Summe");
              }
    }
  ]]>
</mx:Script>
```

Das Framework arbeitet intern mit einer XML-Liste um die Zustände eines Algorithmus zu speichern. Was in dieser Liste alles gespeichert werden soll, wird mit Hilfe der Funktion getVariablesAsXML() festgelegt. Die Inhalte der Matrizen *A*, *B* und *C* sollen gespeichert werden, sowie die Werte der Zählvariablen *i*, *j* und *k* und das Ergebnis der letzten Multiplikation, welches in der Variablen *produkt* steht.

```
override protected function
         getVariablesAsXML(args:Array):XML
{
  return <state>
    <A> {DMatrice.parseMatriceToXML(matrixA)} </A>
    <B> {DMatrice.parseMatriceToXML(matrixB)} </B>
    <C> {DMatrice.parseMatriceToXML(matrixC)} </C>
    <i> {i} </i>
    <j> {j} </j>
    <k> {k} </k>
    <produkt> {produkt} </produkt>
  </state>
}
```

Damit die gespeicherten Zustände auf der *StateMat* auch angezeigt werden können, muss festgelegt werden, in welcher Form dies geschieht. Dazu erstellen wir eine neue Komponente, welche der *StateMat* übergeben wird. Es werden drei Instanzen von *DMatrice* mit den Namen *a*, *b* und *c* mit Hilfe der *Tile*-Komponente aus dem Flex-Framework angeordnet. Erzeugt werden diese Instanzen, indem sie von den bestehenden Instanzen *dMatrixA* und *dMatrixB* kopiert werden. Dadurch wird der gesamte Stil der bestehenden Instanzen kopiert. Außerdem wird mit der Eigenschaft *xmltag* festgelegt, welchen Teil der XML-Zustandsliste die jeweilige *DMatrice*-Instanz auslesen und darstellen soll.

```
override public function createStateDisplay():UIComponent
{
```

```
  var ret:Tile = new Tile();
  var a:DMatrice = DMatrice(dMatrixA.copy());
  var b:DMatrice = DMatrice(dMatrixB.copy());
  var c:DMatrice = DMatrice(dMatrixA.copy());
  a.xmlTag = "A";
  b.xmlTag = "B";
  c.xmlTag = "C";
  ret.addChild(new Canvas());
  ret.addChild(b);
  ret.addChild(a);
  ret.addChild(c);
  return ret;
}
```

4.2 Erstellen einer Applikation mit Beispieleingaben

Die im vorigen Abschnitt erstellte Klasse muss noch in eine Applikation eingebunden werden. Außerdem werden für die Applikation noch Beispieleingaben benötigt. Dazu legen wir eine neue Datei mit dem Namen *ALMatrixmultiplikation.mxml* und folgendem Inhalt an.

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                xmlns:*="*"
                xmlns:mats="mats.*"
                xmlns:dt="displayabletypes.*">

    <Matrixmultiplikation/>
</mx:Application>
```

Mit diesem Code wird eine Applikation erstellt, welche die vorher erstellte Algorithmus-Matte beinhaltet. Damit diese Applikation auch benutzt werden kann, erstellen wir im nächsten Schritt eine Beispieleingabe, indem wir folgenden Code einfügen.

```
<mats:ContentMat x="100">
  <dt:DMatrice>
    <dt:content>
```

```
      <mx:Array>
        <mx:Array>
          [1,2,3]
        </mx:Array>
        <mx:Array>
          [4,5,6]
        </mx:Array>
        <mx:Array>
          [7,8,9]
        </mx:Array>
      </mx:Array>
    </dt:content>
  </dt:DMatrice>
</mats:ContentMat>
```

Die Farbe aller *DMatrice*-Instanzen in dieser Applikation lässt sich folgendermaßen auf ein pastellartiges violett anpassen.

```
<mx:Style>
  DMatrice {IDColor : #ff7af4;}
</mx:Style>
```

Um zur Anpassung der Farben der Komponenten und der verwendeten Schriftgrößen eine CSS-Datei zu verwenden, wäre folgender Code nötig.

```
<mx:Style source="style.css"/>
```

Zum Schluss kompilieren wir die Dateien mit dem folgendem Befehl. Die Datei *ALF.swc* muss sich dazu im gleichen Verzeichnis befinden oder der Pfad muss mit der Option *-library-path* hinzugefügt werden (siehe Kapitel 3.3).

```
mxmlc ALMatrixmultiplikation.mxml
```

Das Resultat ist die Datei *ALMatrixmultiplikation.swf*, die mit dem Flash-Player ausgeführt werden kann.

5 Zusammenfassung und Ausblick

Der im Rahmen dieser Arbeit erstellte Algorithmische-Labor-Framework-Prototyp ermöglicht es einfache Algorithmen, die mit Arrays und Matrizen arbeiten, mit relativ geringem Aufwand zu visualisieren. Dieses Framework ist offen für Erweiterungen, indem es so angelegt wurde, dass Visualisierungen für andere Datenstrukturen leicht angelegt werden können und mit bestehenden Teilen des Frameworks funktionieren. Neue Klassen, die für die Darstellung einzelner Datenelemente zuständig sind, können auch für bestehende Datenstrukturen verwendet werden. Diese Offenheit macht die Entwicklung mit dem Framework allerdings an mancher Stelle kompliziert. Ein Framework, welches bereits alle gebräuchlichen Datenstrukturen und alle nötigen Darstellungsarten von Grundtypen beinhaltet, könnte geschlossener konzipiert werden und so eine noch einfachere Entwicklung von Algorithmischen Laboren gewährleisten.

Das bisherige Konzept des Algorithmischen Labors beschränkt sich auf die Wiedergabe von Zustandsabfolgen eines Algorithmus. Der Benutzer dieser Labore kann zwar die vorgefertigten Eingaben für den Algorithmus nach seinen Wünschen abändern, den Ablauf des Algorithmus kann er jedoch nicht beeinflussen. Beispiele, die den Stoff der Lehrveranstaltung veranschaulichen, können so schnell erstellt werden und sogar während der Lehrveranstaltung verändert werden.

Die Lernenden können auch zu Hause mit den Algorithmischen Laboren versuchen die Funktionsweise der Algorithmen nachzuvollziehen und zu verstehen. Wie groß der Lernerfolg beim wiederholten passiven Anschauen einer Animation jedoch ist, bleibt fraglich. Ein größerer Lernerfolg wird unter anderem von Faltin [6] erwartet, wenn der Benutzer aktiv den Algorithmus beeinflussen kann oder gar dazu aufgefordert wird, den jeweils nächsten Schritt im Algorithmus zu tätigen.

Literaturverzeichnis

[1] Jef Raskin. *The Humane Interface*. ACM Press, 2000.

[2] Michael Herczeg. *Interaktionsdesign*. Oldenbourg Verlag, München, 2006.

[3] Artikel zu Rot-Grün-Sehschwäche bei Wikipedia. http://de.wikipedia.org/wiki/Rot/Grün-Sehschwäche.

[4] Christoph A. Bröker. *Verteilte Visualisierung geometrischer Algorithmen und Anwendungen auf Navigationsverfahren in unbekannter Umgebung*. PhD thesis, Albert-Ludwigs-Universität, Freiburg im Breisgau, 1999.

[5] Flex Livedocs. http://livedocs.adobe.com/flex/.

[6] Nils Faltin. *Strukturiertes aktives Lernen von Algorithmen mit interaktiven Visualisierungen*. PhD thesis, Carl von Ossietzky Universität Oldenburg, 2002.