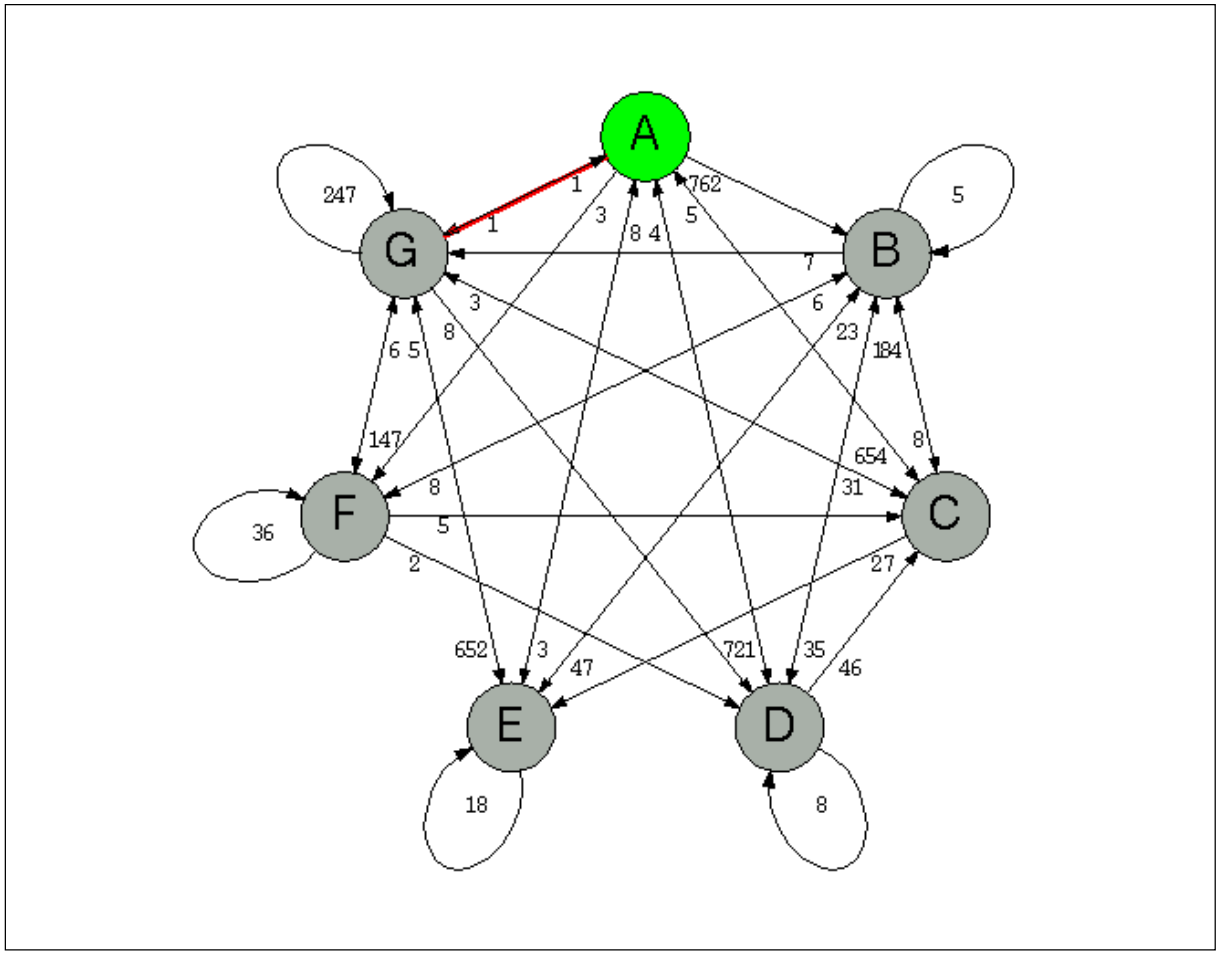
Statusbericht 09/2003

Das interaktive Skript

Automatische Überprüfung und Hilfestellung zu Vorlesungs−begleitenden Übungen

Klaus von der Heide, Norman Hendrich
Universität Hamburg
Fachbereich Informatik



E-LEARNING CONSORTIUM Hamburg

Inhaltsverzeichnis

Kontakt:
Prof. Dr. Klaus von der Heide
Universität Hamburg
Fachbereich Informatik
Vogt-Kölln-Str. 30
D 22 527 Hamburg

http://tams-www.informatik.uni-hamburg.de

1 Einführung

1.1 Rahmen des Projekts: Das interaktive Lehrbuch

Überblick

Ein *interaktives Lehrbuch* bzw. *interaktives Skript* vereinigt die textuelle Beschreibung der zu lernenden Zusammenhänge und Sachverhalte mit interaktiven elektronischen Werkzeugen zur Darstellung und Anwendung dieser Sachverhalte — und zwar der Anwendung nicht nur auf die im Lehrbuch selbst integrierten Beispiele, sondern vielmehr auf beliebige, vom Lernenden auch selbst veränderbare oder hinzugefügte Anwendungsfälle. Dadurch wird das Lehrbuch erweiterbar und kann jederzeit auch an ursprünglich gar nicht vorgesehene oder vorhersehbare zukünftige Entwicklungen angepasst werden. Es bietet damit ideale Voraussetzungen zur Unterstützung des *life-long learning* und zum produktiven dauerhaften Einsatz während des Berufslebens.

Das dem Projekt zugrunde liegende Konzept des interaktiven Lehrbuchs hat folgende Zielsetzungen:

Problemlösung

- Die Spanne zwischen klassischem Lehrbuch und Anwendung wird zunehmend größer, weil die Komplexität der behandelten Themen sich nur noch mit Rechner-gestützten Systemen beherrschen lässt. Durch die harmonische Integration von klassischem Lehrtext mit einem zugrunde liegenden, universellen Softwaresystem zur Problemlösung kann das interaktive Lehrbuch sehr anwendungsspezifisch werden, ohne dabei jedoch auf ein Anwendungsgebiet festgelegt zu sein.

Nachhaltigkeit

- Der Rückblick auf die letzten Jahre der Softwareentwicklung zeigt, dass für nachhaltige Entwicklungen nur einfache und standardisierte Datenformate verwendet werden sollten. Beim Einsatz proprietärer Formate muss jederzeit damit gerechnet werden, nach einem Versionswechsel auf ältere Datensätze nur noch eingeschränkt oder eventuell überhaupt nicht mehr zugreifen zu können. Inhalte für das interaktive Lehrbuch basieren daher im Wesentlichen auf annotierten Texten im einfachen ASCII-Format.

Anpassbarkeit

- Ein klassisches Lehrbuch spiegelt immer nur die Sicht (und Absicht) des jeweiligen Autors wider. Um sich ein objektiveres Bild zu verschaffen, greifen viele Studierende und Lehrende deshalb parallel auf mehrere Lehrbücher zurück. Wegen des hohen Erstellungsaufwands im Falle von E-Learning Content stehen geeignete alternative Materialien bisher aber nur selten zur Verfügung. Daraus ergibt sich die Forderung, dass der Inhalt des interaktiven Lehrbuchs von den Lehrenden individuell nach ihren eigenen Vorstellungen ausgerichtet und geändert werden kann — und zwar mit Hilfe eines langfristig und auf allen Plattformen verfügbaren Werkzeugs. Dies betrifft sowohl die Auswahl als auch die Inhalte der Texte, Abbildungen, Animationen, Audioausgaben, Simulationen, usw.

Exploration

- Die Integration von interaktiven Elementen in das interaktive Lehrbuch hilft dabei, Interpretationsschwierigkeiten oder Verständnislücken durch aktive Exploration des Lehrstoffs zu überwinden. Viele Inhalte und insbesondere Graphiken im interaktiven Lehrbuch werden deshalb erst zur Laufzeit mit

vom Benutzer individuell einstellbaren Parametern erzeugt und angezeigt. Ein Studierender kann auf diese Weise grundsätzlich nachvollziehen, wie es zu den Graphiken und Ergebnissen kommt.

1.2 Kurzbeschreibung des Projekts

Integrierte Übungen

Es liegt nahe, auch *Übungsaufgaben* in das oben beschriebene interaktive Lehrbuch zu integrieren. Während eine derartige Integration bei klassischer Lernsoftware im Sinne des *Computer Based Training* [28] sehr aufwendig sein kann, stellt das interaktive Lehrbuch mit seiner Softwareplattform bereits alle Werkzeuge zur Verfügung, um die Lernenden bei der Bearbeitung der Aufgaben zu unterstützen und zu motivieren.

Überprüfung und Hilfestellung

Die Erforschung und Erprobung dieser Techniken ist der Inhalt des vorliegenden Projekts. Ziel ist eine Softwarebibliothek, die für viele Typen von Übungsaufgaben eine automatische Überprüfung der Lösungen erlaubt und bei Fehlern geeignete Hilfestellung anbietet. Die Studierenden bekommen dadurch sofort eine Rückmeldung über ihren Lernfortschritt bzw. Hinweise auf noch verbliebene Fehler. Zusammen mit der nahtlosen Integration der Aufgaben in das Skript wird die Hemmschwelle zur Bearbeitung der Übungsaufgaben gesenkt, was die Studierenden zu einer intensiveren Beschäftigung mit dem Lehrstoff einlädt.

Methodenkompetenz

Das primäre Ziel der Übungen ist dabei nicht die Vermittlung von Faktenwissen, sondern die Verbesserung der Fertigkeiten bei der Auswahl und dem Einsatz von Methoden. Die im Projekt erzielten Ergebnisse werden sich deshalb auch auf viele andere Fachgebiete mit ähnlicher Methodik übertragen lassen, etwa die angewandte Mathematik, Experimentalphysik oder die Ingenieurwissenschaften.

1.3 Gliederung dieses Reports

Einführung

Der Rest dieses Kapitels befasst sich mit einer Einführung in das Projekt, die im folgenden Abschnitt 1.4 mit einer Erläuterung des *interaktiven Skripts* beginnt. Abschnitt 1.5 präsentiert eine kurze Übersicht über den Stand der Technik bei der automatischen Überprüfung von Übungsaufgaben, bevor in Abschnitt 1.6 die für das Projekt geplanten Anwendungsfälle skizziert werden.

Klassifikation und Analyse

Inhalt von Kapitel 2 ist eine Klassifikation und Analyse der gängigen Übungsaufgaben zur technischen Informatik. Dazu werden in Abschnitt 2.1 der Lehrstoff der technischen Informatik im Grundstudium und Abschnitt 2.2 die zugehörigen Übungen vorgestellt. Die Auswertung in Abschnitt 2.3 zeigt, welche Typen von Übungsaufgaben besonders häufig vorkommen, während sich Abschnitt 2.4 mit der Frage nach einer geeigneten Repräsentation der Daten beschäftigt. In Abschnitt 2.5 werden Verfahren vorgestellt, mit denen die Übungsaufgaben der verschiedenen Klassen überprüft werden können, bevor Abschnitt 2.6 auf die Möglichkeiten zur Hilfestellung eingeht.

Kapitel 3 begründet die für das Projekt gewählte Software-Infrastruktur. Die Details zur Implementierung inklusive Klassenhierarchien und -dokumentation werden den Schwerpunkt des nächsten Projektsberichts bilden.

1.4 Das interaktive Skript

Definition

Bei dem *interaktiven Skript* handelt es sich um Lernsoftware, die erläuternde Texte mit integrierten interaktiven Komponenten verknüpft. Es handelt sich dabei aber nicht um eine bloße multimediale Aufbereitung des Lehrstoffes. Vielmehr baut das Skript eine virtuelle Welt auf, deren zugrunde liegende Modelle für die Lernenden offen liegen und veränderbar sind. So werden alle Graphiken, Videos und Audioausgaben immer mit diesen Modellen per Simulation berechnet. Das vorhandene Material umfasst derzeit vier vollständige Vorlesungen und zusätzliche Kapitel für Praktika und Übungen mit zusammen über 1000 interaktiven Funktionen, die auf Grundlage von Matlab als Programmierumgebung realisiert sind [23].

Zur Anzeige der Texte und zum Aufruf der interaktiven Funktionen der Skripte dient ein selbstentwickelter Browser, *mscriptview*, der auch erweiterte Funktionen zur Navigation und Suche innerhalb der umfangreichen Materialien umfasst. Im Unterschied zu vielen anderen E-Learning Umgebungen gibt es keine Trennung zwischen einem Editor zur Content-Erstellung und einem reinen Viewer. Vielmehr erlaubt *mscriptview* den vollen Zugriff auf alle im Material enthaltenen Funktionen, so dass die Lernenden die Skripte auch selbst bearbeiten und nach ihren Wünschen verändern oder erweitern können. Tatsächlich stellen die Skripte damit gleichzeitig auch eine hochwertige Programmbibliothek dar, die später im Berufsleben jederzeit produktiv genutzt werden kann.

Hypertext, Formeln, Medien

Im Moment unterstützt der *mscriptview*-Browser neben einfachem Text auch diverse Varianten von formatiertem Text und Formeln. Die Darstellung funktioniert plattformübergreifend; über Einstellungen in der Konfigurationsdatei können die zu verwendenden Schriftarten ausgewählt werden, um eine optimale Qualität zu erreichen. Die im Text eingebetteten Querverweise können vom Benutzer einfach angeklickt werden und verhalten sich damit genau wie von den bekannten HTML-Browsern gewohnt. Durch den Aufruf der entsprechenden Matlab-Funktionen können außerdem alle Medientypen verwendet werden, die auf der zugrunde liegenden Plattform überhaupt unterstützt werden. Dies sind auf jeden Fall statische Abbildungen, via Matlab berechnete Graphiken und 2D-/3D-Plots, Animationen, und die Audioausgabe. Zum Abspielen von Videos kann auf externe Medienplayer wie den Windows Mediaplayer, Quicktime oder Xine zugegriffen werden.

Zur Darstellung von statischen Abbildungen gibt es zwei Alternativen. Dazu enthält der Browser eine Funktion, die zum Anzeigen der Bilder ein externes Programm aufruft. Dabei kann es sich sowohl um einen kleinen Bildbetrachter als auch ein ausgewachsenes Bildverarbeitungsprogramm wie Photoshop handeln. Die zweite Variante besteht im Aufruf der Matlab-Funktionen *imread* und *image*, mit der die gängigen Bildformate angezeigt werden können. Beim Einsatz zusammen mit der Student-Edition von Matlab ist aber die in dieser Version vorhandene Größenbeschränkung der Datenstrukturen zu berücksichtigen.

Interaktion

Wesentlich wichtiger als die Einbettung „statischer“ Medien ist die Verwaltung der eingebetteten Matlab-Funktionen, die den eigentlichen Kern des interaktiven Skripts ausmacht. Dazu erkennt der Browser die im Skript enthaltenen Codezeilen und fasst diese intern zu zusammengehörigen Blöcken zusammen. Beim ersten Mausklick in einen solchen Textblock wird dieser optisch hervorgehoben und intern in einen Puffer kopiert. Jedes weitere Anklicken des bereits markierten Blocks

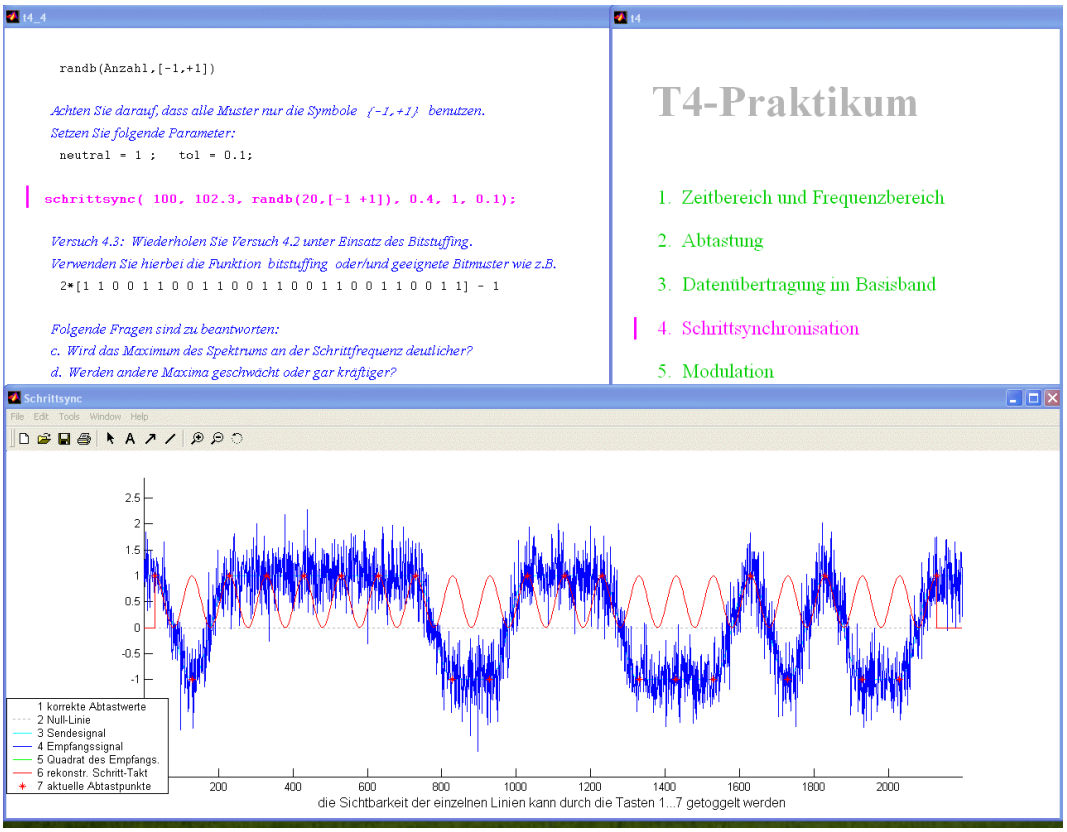


Abbildung 1: *Das interaktive Skript: Matlab-Simulation der Schrittsynchronisation bei der seriellen Datenübertragung aus dem T4-Praktikum der Autoren. Links ist ein Ausschnitt aus dem Skript mit integrierten Übungsaufgaben dargestellt, rechts die übergeordnete Seite zur Navigation. Die Graphiken werden abhängig von den vom Benutzer eingestellten Parametern innerhalb von wenigen Sekunden neu berechnet.*

führt dann dazu, dass der entsprechende Code als Matlab-Skript ausgeführt wird. Zusammen mit der Option, den markierten Programmcode direkt im Matlab-Editor zu bearbeiten, eröffnet dies Interaktionsmöglichkeiten, die weit über den gewohnten Umfang hinausgehen. Tatsächlich eröffnet sich dem Anwender eine virtuelle Welt, deren zugrunde liegende Modelle und Regeln offen liegen und die beliebig modifiziert werden können — alle Voraussetzungen für entdeckendes Lernen sind damit erfüllt. Insbesondere können nicht nur die Parameter für die Funktionen vom Benutzer eingestellt werden, sondern die Funktionen selbst können jederzeit verändert werden.

Als Beispiel für dieses Konzept zeigt Abbildung 1 eine Simulation der Schrittsynchronisation bei der Datenübertragung aus unserem Praktikum *Technische Informatik 4*. Der Plot wird bei Anklicken der vorher als aktiv ausgewählten *schrittsync*-Funktion dynamisch aus den Eingabedaten und zufällig generiertem Rauschen erzeugt. Bei Interesse können die Studierenden den Quelltext im Editor öffnen und damit auch die zugrunde liegende Funktion untersuchen.

Natürlich ist für dieses Interaktionskonzept nicht unbedingt Matlab als Plattform erforderlich. Tatsächlich basieren die aus Matlab und Mathematica bekannten *Notebooks* auf einem ähnlichen Konzept, wenn auch mit anderen Schwerpunkten. Im Prinzip eignet sich jede Softwareumgebung, die die interaktive Programmentwicklung unterstützt und das Ausführen von Text (-dateien) als Code erlaubt. Ein aktuel-

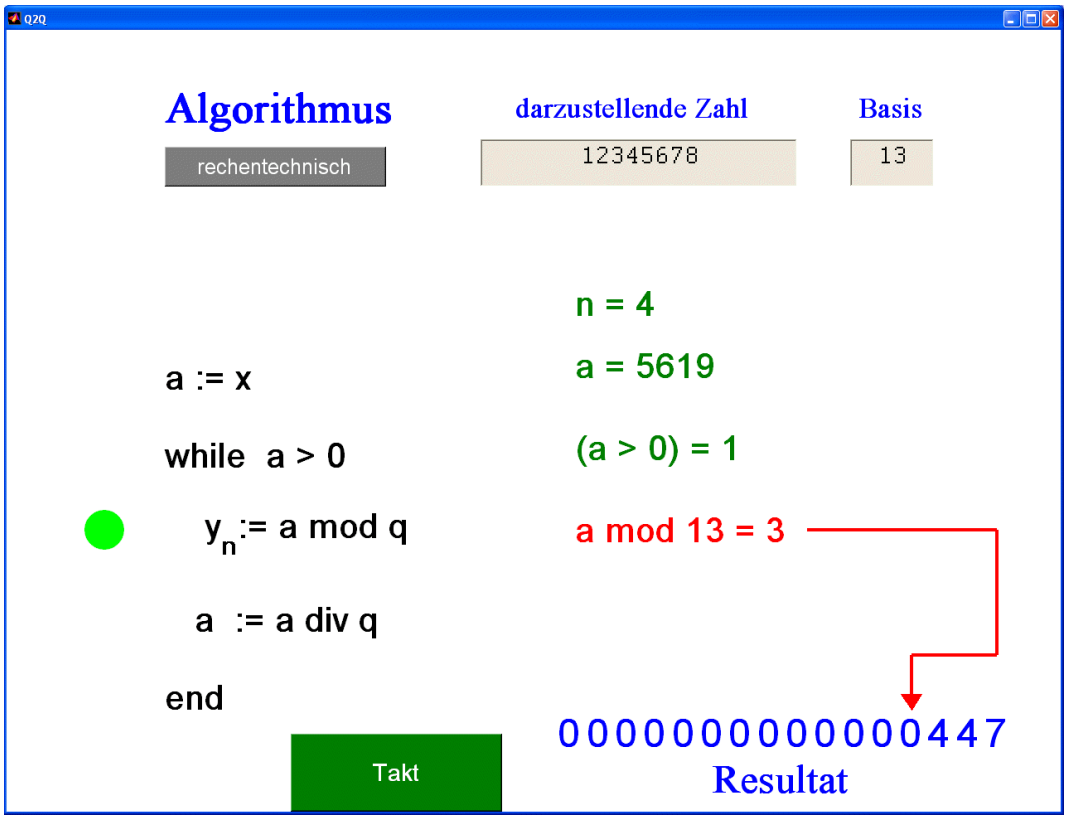


Abbildung 2: *Integration von Applets mit eigener Benutzeroberfläche in das interaktive Skript. Das Matlab-Applet zur Vorlesung T1 demonstriert die Darstellung und Umrechnung von Integerzahlen in verschiedenen Zahlensystemen (z.B. Binär- und Dezimalsystem). Während des Umrechnung werden die gerade aktiven Rechenschritte hervorgehoben und die zugehörigen Zwischenergebnisse angezeigt.*

les Beispiel liefert das *Jython Environment for Students* (JES) des Georgia Institute of Technology [16], das auf den Programmiersprachen Java und Jython [18] aufsetzt. Wegen der Vielzahl der bereits vordefinierten Funktionen eignet sich Matlab natürlich hervorragend für Anwendungen in der technischen Informatik.

Applets

Obwohl sich die Darstellung des Stoffes im Browser für viele Anwendungsfälle sehr gut eignet, kann eine eigene Benutzeroberfläche in manchen Fällen die Bedienung der jeweiligen Software stark erleichtern. Ähnlich zur Einbettung interaktiver Flash- oder Java-Applets in Webseiten lassen sich auch in Matlab Applikationen mit eigener Benutzeroberfläche schreiben und in das interaktive Skript einbetten. Als Beispiel zeigt Abbildung 2 ein Applet zur Umrechnung von Integerzahlen zwischen verschiedenen Basen. Die üblichen GUI-Elemente wie Buttons oder Textfelder werden dabei mit interaktiv angepassten Graphiken (hier dem Pfeil) kombiniert, um die einzelnen Schritte während der Umrechnung zu illustrieren. Derartige Applets sind auch ideal, um die — möglicherweise recht komplexen — Details der Implementierung zunächst vor dem Benutzer zu verbergen, etwa im Grundstudium oder für Studierende, die sich noch nicht mit Matlab auskennen.

Auch komplexe Editoren und Simulatoren lassen sich in das interaktive Skript integrieren, da Matlab problemlos externe Programme aufrufen kann. Im Rahmen des Projekts wird über die Matlab-Java Schnittstelle unter anderem ein bestehender Schaltungssimulator [11] eingebunden werden.

Eingebettete Übungen

Da den Skripten der volle Funktionsumfang von Matlab zur Verfügung steht, können auch Übungsaufgaben inklusive der automatischen Überprüfung in das Skript integriert werden. Die Aufgabentexte selbst werden einfach als gewöhnlicher Text geschrieben. Außerdem wird zusätzlicher Matlab-Code in das Skript mit aufgenommen, der auf die Eingaben der Studierenden wartet und diese anschließend gleich auswertet. Je nach Art der Aufgabenstellung können die Eingaben in eigens dafür vorgesehene Benutzeroberflächen und Dialogfenster oder einfach in die Matlab-Shell erfolgen. Ziel des Projekts ist es, für möglichst viele Typen von Aufgaben geeignete Software zur Überprüfung bereitzustellen und darüber hinaus auch Hilfestellungen zu generieren, um die Studierenden bei der Bearbeitung dieser eingebetteten Übungen zu unterstützen. Im Gegensatz zur üblichen Praxis, lediglich einfache Auswahl-Fragen (Ja-/Nein, Multiple-Choice, Zuordnungen) zuzulassen, sind über den Zugriff auf passende Algorithmen auch weit komplexere Fragestellungen möglich.

Eine Klassifikation der in Frage kommenden Typen von Aufgaben mit den zugehörigen Verfahren zur Überprüfung wird in Kapitel 2.2 entwickelt. Als erster Hinweis auf die Möglichkeiten der automatischen Hilfestellung mag das folgende Beispiel der Minimierung von Schaltfunktionen dienen. Die vom Studenten berechnete Lösung wird als Text (String) in ein Matlab-Dialogfenster eingegeben, von der Software intern aber direkt als Funktion behandelt. Im ersten Schritt erfolgt die Überprüfung auf Korrektheit einfach durch Berechnung der Funktionswerte für alle möglichen Eingangswerte. Anschließend wird die Struktur der Formel untersucht, um weitere geforderte Eigenschaften zu überprüfen, etwa die Anzahl der Terme. Falls beim ersten Test falsche Ausgangswerte gefunden werden, liefern die zugehörigen Eingangswerte direkt die Gegenbeispiele, für die eine Nachbesserung erforderlich ist. Damit steht auch sofort eine geeignete Hilfestellung zur Verfügung.

Natürlich sollte die im Projekt erstellte Software nicht ausschließlich Textmeldungen erzeugen, sondern in die jeweils zugrunde liegenden Skripte und Simulatoren integriert werden. Ein Beispiel dafür ist es, Kurzschlüsse oder offene Eingänge in einem Schaltplan direkt durch Hervorhebung zu markieren, so dass die Studierenden diese Probleme auf einen Blick erkennen können.

Content-Creation

Ein weiterer wesentlicher Punkt betrifft den *content creation* Prozess für die interaktiven Skripte. Anders als in einigen E-Learning Frameworks ist für das Erstellen von neuen Materialien keine besondere Editor-Software erforderlich. Tatsächlich handelt es sich bei den Skripten um Matlab-Programme, deren Kommentarzeilen vom *mscriptview*-Browser zur Darstellung des erläuternden Textes ausgewertet wird. Die Formatierungsanweisungen werden dabei genau wie in anderen bekannten Markup-Sprachen (z.B. SGML, XML, HTML, TEX) als Annotationen direkt in den Text der Seite integriert. Für erfahrene Anwender ist die Erstellung von annotierten Texten nach kurzer Eingewöhnung mindestens so effizient wie die Verwendung von WYSIWIG-Editoren.

Wegen der Beschränkung auf das einfache ASCII-Textformat können die interaktiven Skripte inklusive der Matlab-Funktionen mit jedem beliebigen Texteditor erstellt werden, so dass erfahrene Anwender weiterhin mit ihrem Lieblingseditor arbeiten können und keine Umgewöhnung erforderlich ist. Im Zusammenhang mit der Fehlersuche in den eingebetteten Matlab-Skripten ist natürlich der in Matlab integrierte Editor zu empfehlen, der die üblichen Komfortmerkmale wie Syntax-Highlighting bietet und nahtlos mit dem Matlab-Debugger zusammenspielt.

1.5 State of the Art — Software, Tools, Umgebungen

Im Rahmen dieses Reports ist eine vollständige Bestandsaufnahme der E-Learning-Landschaft selbstverständlich unmöglich. Statt dessen werden in diesem Abschnitt die Konzepte zur Integration von Übungen in elektronische Lehrmaterialien beispielhaft an einem aktuellen Lehrbuch sowie den vom E-Learning Consortium Hamburg zur Verfügung gestellten Frameworks dargestellt.

Das erste Beispiel liefert die kürzlich erschienene *Interactive Multimedia Introduction to Signal Processing* von U. Karrenberg [21], ein durch Software unterstütztes Lehrbuch, das thematisch gut in das Umfeld des hier bearbeiteten Projekts passt. Das gedruckte Buch verwendet jeweils Doppelseiten, um nebeneinander den erläuternden Text und die zugehörigen Graphiken und Blockschaltbilder zu präsentieren. Auf der Begleit-CD zum Buch ist der vollständige Buchtext im PDF-Format ebenfalls enthalten; die Blockschaltbilder sind jetzt aber aktive Verweise auf eine Simulationsumgebung, die ein interaktives Ausprobieren der einzelnen Schaltungen erlaubt. Jedes Thema wird damit unmittelbar durch ein Experiment ergänzt, das letztlich auch als zugehörige Übungsaufgabe interpretiert werden kann. Die dazu eingesetzte Software (Dasylab) ähnelt Simulink und ist in der auf der CD enthaltenen, leicht eingeschränkten Version für die Studierenden kostenlos. Leider ist die Software aber auf die graphischen Blockschaltbilder und den Satz der mitgelieferten Komponenten beschränkt, was eigene Experimente der Studierenden erschwert. Insgesamt ähnelt das Konzept des Buches stark den in diesem Projekt untersuchten interaktiven Skripten; die komplett vorbereiteten Experimente bedeuten aber eine gröbere Granularität als die hier propagierten kleinen (Matlab-) Funktionen.

Übungsaufgaben in E-Learning-Frameworks

Die Integration von Übungsaufgaben in E-Learning Materialien ist natürlich keine neue Idee. Wegen der Typenvielfalt der möglichen Aufgabenstellungen und diverser Probleme bei der konkreten Implementierung werden aber üblicherweise nur einige wenige, besonders einfach umzusetzende Arten von Aufgaben zugelassen. Als Beispiel zählt die Kurzbeschreibung der auch im Rahmen des ELCH-Projekts verfügbaren Lernplattform CLIX [33] die folgenden vom System unterstützten Typen von Übungsaufgaben auf:

- Ja/Nein-Fragen und Multiple-Choice-Fragen
- Zuordnungs-Fragen, Reihenfolgen
- Numerische Zahlenwerte
- Lückentexte
- Image-Maps
- Fließtext

All diesen Aufgabentypen bis auf die Kategorie Fließtext ist gemeinsam, dass die Antwort sofort durch einen trivialen Vergleich mit der korrekten Musterlösung überprüft werden kann. Bei den Multiple-Choice und Zuordnungs-Fragen reicht es vollständig aus, die Antwort als richtig oder falsch einzuordnen; eine feinere Einteilung in Zwischenwerte oder die Bewertung von fast richtigen Teillösungen ist kaum notwendig.

Ganz anders sieht es in der Kategorie Fließtext aus, bei denen die Unterstützung durch das Framework sich auf das reine Abspeichern der von den Studierenden

eingegebenen Texte beschränkt. Da die Lernplattform keine über den Datentransfer hinausgehenden Funktionen bereitstellt, muss die Durchsicht und Korrektur wie üblich durch die Lehrpersonen erfolgen.

Maßnahmen zur automatischen Hilfestellung bei falschen Antworten zu Lösungen der ersten Kategorien sind selten; üblich dagegen ist es, die weitere Navigation innerhalb des Lehrmaterials solange einzuschränken, bis die im Material eingestreuten Übungen korrekt gelöst sind. Es ist im Einzelfall abzuwägen, ob diese Behinderung wirklich zu einem gründlicheren und erneuten Durcharbeiten des Stoffes führt — oder zu wildem Durchprobieren der angebotenen Multiple-Choice Antworten. Zumindest aus Sicht der Autoren kann eine derartige Beschränkung der Navigationsmöglichkeiten leicht Frust auslösen und die weitere Beschäftigung mit dem Material erschweren.

Frameworks (bisher) für technische Informatik unbrauchbar

In Bezug auf die Anwendung in der technischen Informatik ist die obige Liste der unterstützten Aufgabentypen völlig unzureichend. Die im nächsten Kapitel vorgenommene Analyse und Klassifikation der Übungsaufgaben zur technischen Informatik wird zeigen, dass nur ein Bruchteil der üblichen Aufgaben sich in diese Kategorien einordnen lässt (vgl. Seite 15). Zum Beispiel umfassen die insgesamt 58 Übungsaufgaben zu den Vorlesungen T1/T2 des Autors im Jahr 2002 nicht eine einzige Ja/Nein-, Multiple-Choice-, oder Zuordnungsfrage [8]. Von den immerhin 22 Fragen nach Zahlenwerten entfallen fast alle auf Details zur Zahlenrepräsentation, die mit der von den Frameworks unterstützten Dezimaldarstellung ebenfalls nicht bearbeitet werden können.

Trotzdem ist die Situation nicht hoffnungslos. Vielmehr lassen sich viele der Aufgabentypen auf eine kleine Anzahl von Klassen zurückführen, für die mit geeigneter Repräsentation der Daten und mit einfachen Algorithmen durchaus die automatische Überprüfung durchführbar ist.

1.6 Use-Cases

Derzeit sind für die Integration der Übungsaufgaben in die interaktiven Skripte und die automatische Überprüfung drei verschiedene Einsatzgebiete vorgesehen. Diese sind als UML-Anwendungsdiagramme in Abbildung 3 zusammengefasst.

Häusliche Bearbeitung der Übungen

Die erste und zentrale Aufgabe ist es, die Studierenden bei der häuslichen Nachbereitung der Vorlesung und der Bearbeitung der Übungsaufgaben zu unterstützen. Dabei interagieren die Studierenden auf ihren eigenen PCs mit den verschiedenen Komponenten des interaktiven Skripts. Je nach Aufgabe können dies kleine Applets mit eigener Benutzeroberfläche, externe Programme wie ein Schaltungssimulator, der Browser für das interaktive Skript mit Matlab-Dialogfenstern oder auch die Matlab-Shell selbst sein. Um möglichst kurze Antwortzeiten für die Überprüfung erreichen zu können, wird die Software lokal auf dem Rechner der Studierenden ablaufen, was angesichts der Rechenleistung aktueller PCs problemlos möglich ist. Falls kein eigener PC vorhanden ist, können die Studierenden die Aufgaben selbstverständlich auch auf den Rechnern der Universität bearbeiten.

Für das Abgeben bzw. Einsenden der (bereits überprüften) Lösungen ist zusätzlich eine Client/Server-Architektur vorgesehen, bei der nur Teile der Software inklusive der Benutzeroberfläche lokal ausgeführt werden, während andere Komponenten

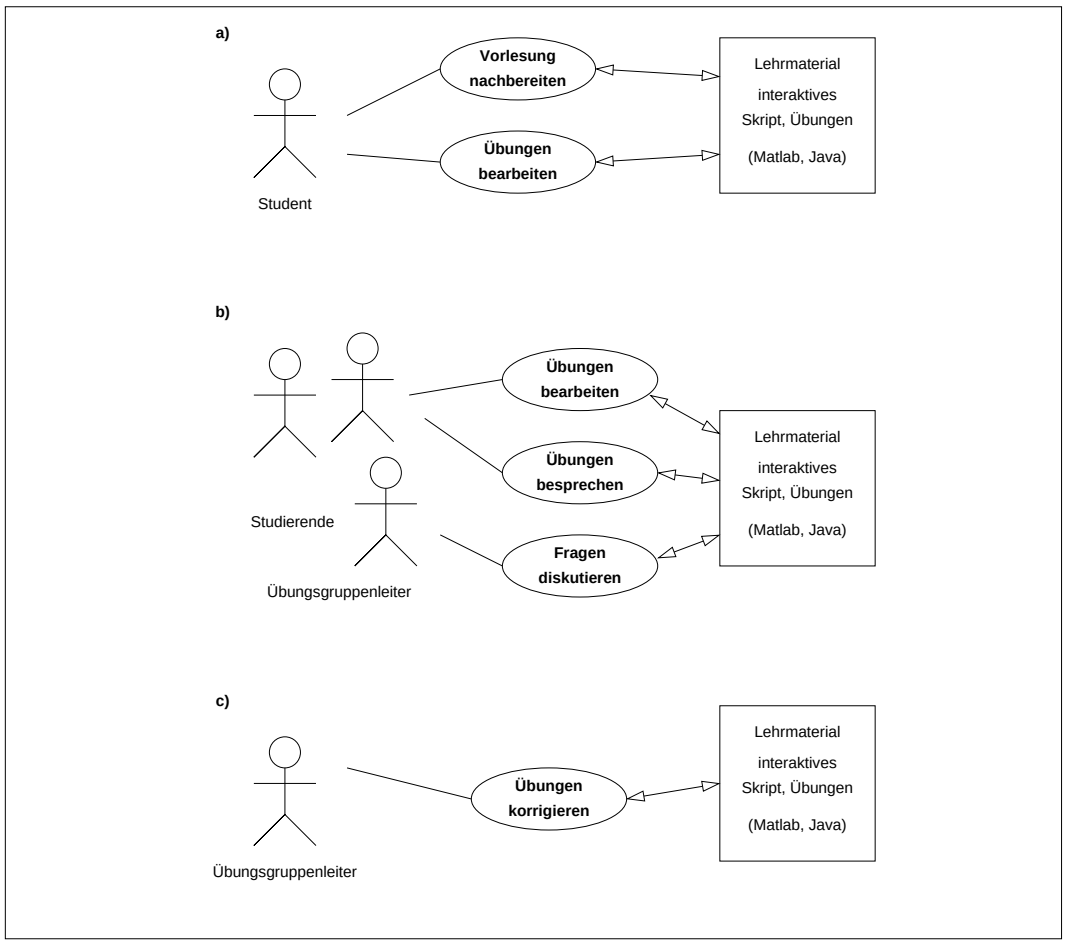


Abbildung 3: *Anwendungsdiagramme (Use-Cases) für die Integration von Übungsaufgaben in das interaktive Skript: (a) Bearbeitung der Übungsaufgaben mit Rechnerunterstützung durch den einzelnen Studenten, (b) Einsatz während klassischer Präsenzübungen, (c) Unterstützung der Übungsgruppenleiter durch automatische Korrektur und Klassifikation der Aufgaben.*

auf dem Server verbleiben. Diese Trennung wird auch für Aufgaben zum Einsatz kommen, bei denen eine lokale Installation der Software zum Beispiel aus Lizenzgründen nicht in Frage kommt.

Gruppenarbeit

Natürlich müssen die Studierenden die Aufgaben nicht alleine angehen, sondern können auch in Gruppen zusammenarbeiten, wobei nur ein einzelner PC benötigt wird. Im Rahmen des Projekts ist keine spezielle Unterstützung für Gruppenarbeit geplant. Dies ist auch nicht notwendig, da passende *groupware*-Softwarelösungen von mehreren Herstellern angeboten werden bzw. teilweise bereits zum Standardumfang der E-Learning Frameworks gehören.

Einsatz während der Übungen

Die zweite Anwendung ergibt sich beim Einsatz der Software innerhalb der klassischen Präsenzübungen. Angesichts der (am Fachbereich Informatik) üblichen Größe der Übungsgruppen von gut 20 Teilnehmern bedeutet dies, dass geeignete Infrastruktur wie Notebook und Beamer auch in den Übungsräumen zur Verfügung stehen muss. Unter dieser Voraussetzung ist es möglich, dass die Studierenden ihre zu Hause erarbeiteten Lösungen nicht nur an der Tafel sondern auch mit dem Skript vorführen. Während der Übung entwickelte Lösungsansätze lassen sich sowohl von den Studierenden als auch von den Gruppenleitern direkt ausprobieren,

wobei die Werkzeuge zur Überprüfung in vollem Umfang zur Verfügung stehen. Natürlich werden die Gruppenleiter in den meisten Fällen noch wesentlich bessere Unterstützung leisten können, als die automatisch generierten Hilfestellungen es vermögen. Nicht zuletzt erlaubt das interaktive Skript dem Übungsgruppenleiter, während der Übungstermine aufkommende Fragen mit Hilfe des Skripts zu klären und Antworten nicht nur mündlich zu geben, sondern zum Beispiel auch durch kurzerhand neuberechnete Graphiken zu illustrieren.

Unterstützung der Gruppenleiter

Das dritte Einsatzgebiet betrifft die Unterstützung der Gruppenleiter bei der Korrektur der von den Studierenden eingesandten Lösungen. Das Skript erlaubt dabei nicht nur das Vorsortieren in korrekte und falsche Lösungen, sondern liefert auch Daten für die Anmerkungen; zum Beispiel alle Eingangsbelegungen, bei denen die eingesandte Schaltfunktion falsche Ausgangswerte liefert. Je nach Aufgabenstellung ist eventuell sogar das Erkennen von typischen Fehlern wie etwa ausgelassenen Rechenschritten oder kleineren Rechenfehlern möglich.

2 Übungen zur technischen Informatik — Versuch einer Klassifikation

Inhalt dieses Kapitels ist eine Klassifikation der Übungsaufgaben zur technischen Informatik, aus der sich ein Überblick über die im Rahmen des Projekts zu erstellenden Verfahren zur automatischen Überprüfung ableiten lässt. Ausgehend vom Grundstudiums-Lehrstoff der technischen Informatik, der in Abschnitt 2.1 zusammengefasst wird, lassen sich mehrere Klassen von typischen Übungsaufgaben unterscheiden, die in Abschnitt 2.2 beschrieben werden. Abschnitt 2.3 liefert Hinweise darauf, welche der Aufgabentypen besonders häufig vorkommen und deshalb auch bevorzugt im Projekt bearbeitet werden sollten. Algorithmen zur automatischen Überprüfung für die einzelnen Kategorien von Aufgaben werden in Abschnitt 2.5 vorgestellt, während Abschnitt 2.6 die entsprechenden Ansätze für automatisch generierte Hilfestellungen zusammenfasst. Schließlich wird in Abschnitt 2.7 noch auf das in der Praxis zu erwartende Problem von Täuschungsversuchen eingegangen.

2.1 Lehrstoff der technischen Informatik

Themengebiete

Obwohl sich die Informatik- und Elektrotechnik-Fachbereiche durchaus in der Gewichtung der einzelnen Themen unterscheiden, gibt es doch einen gemeinsamen Konsens über den Lehrstoff der technischen Informatik im Grundstudium. Die folgende Aufstellung entstammt dem Studienführer des Fachbereichs Informatik [4] an der Universität Hamburg. Im einzelnen werden in den Vorlesungen *Technische Informatik 1 bis 4* die folgenden Themengebiete behandelt:

T1: Digitale Systeme 1

- Grundbegriffe der Informationsverarbeitung: Information, Repräsentation, Kommunikation, Zahlensystem, System-Begriff, Codierung
- Arithmetiken: Ganzzahl- und Gleitkomma-Arithmetik
- Hardwarebeschreibungssprachen, Entwurfsebenen, Verhaltens- und Strukturbeschreibung
- Schaltnetze: Boolesche Algebra, Schaltnetzsynthese, Zeitverhalten
- Schaltwerke: Moore- und Mealy-Modell, Entwurf von getakteten Schaltwerken, Flipflops
- von-Neumann-Rechner: Architektur, Befehlssatz, Assemblierung

T2: Digitale Systeme 2

- Elektrotechnische Grundlagen: lineare Netze
- Nichtlineare Bauelemente: Halbleiterdiode, Transistoren
- Digitale Schaltungstechnik: Logik-Glieder, Bus-Systeme
- Strukturen von Halbleiterspeichern: Arten der Adressierung, Realisierung statischer und dynamischer Speicher
- Programmierbare Bausteine: Arten der Programmierung, interne Struktur, Anwendung

- Herstellung integrierter Schaltungen: Lithographie, chemische und physikalische Prozess-Schritte
- Wirtschaftliche Aspekte der Hochintegration: Fehlermechanismen und Chipausbeute, Entwurfskomplexität und Entwurfsstile, Chiptest

T3: Rechnerorganisation und Betriebssysteme

- Grundzüge der Rechnerarchitektur: Entwurfsprinzipien, Befehlssätze, Adressierung, Speicherhierarchie
- Prozessorimplementation: Aufbau moderner Prozessoren, Pipelining, Parallelisierung und weitere effizienzsteigernde Maßnahmen
- Ein- und Ausgabe: Peripheriegeräte, Schnittstelle zu Prozessoren, Bussysteme, Interrupt-Behandlung
- Grundelemente von Betriebssystemen: Prozesse, Interprozesskommunikation, Betriebsmittelverwaltung, Dateisysteme
- Prozess-Scheduling: Ziele, klassische Verfahren, Termin-Scheduling, Bewertung
- Schutzmechanismen: Schutzmodelle, Zugangskontrolle, kryptographische Ansätze

T4: Parallelverarbeitung und Kommunikationsnetze

- Parallelrechner: Grundlagen der Parallelverarbeitung, Architekturkonzepte und Realisierungen
- Informationstheorie und Codierungstheorie: Informationsgehalt, Redundanz, fehlererkennende und fehlerkorrigierende Codes
- Rechnerarchitekturkonzepte und Referenzmodelle: Dienst, Protokoll, Schichtenstruktur, Adressierung, Standardarchitekturen, Internet-Architektur
- Übertragungstechnik und Datenübertragung: physikalische Übertragungswege, Signalübertragung, ge gesicherte Datenübertragung über Punkt-zu-Punkt Verbindungen
- Vermittlungsnetze: Vermittlungstechniken, Wegeermittlung, Flusskontrolle, Internet-Protokolle
- Lokale Rechnernetze: Zugriffskontrolle in Bussystemen und Ringnetzen, LAN-Standards
- Modellierungs-, Analyse-, und Entwurfsverfahren: Überblick über Ziele und Verfahren zur Planung, Bewertung und zum Management von Netzen
- Unterstützung verteilter Anwendungen: Transportdienste und Transportsysteme, anwendungsorientierte Dienst- und Verwaltungskonzepte

2.2 Klassifikation der Übungsaufgaben

Aufgabenvielfalt oder Klassen von Aufgaben?

Angesichts der großen Themenvielfalt des Lehrstoffs in den oben skizzierten Grundvorlesungen zur technischen Informatik ist eine ähnliche Vielfalt auch für die zugehörigen Übungsaufgaben zu erwarten. Es stellt sich daher die wichtige Frage, ob eine Einteilung der Übungsaufgaben in Kategorien oder Klassen möglich ist, für die sich gemeinsame grundlegende Ansätze zur Überprüfung und Hilfestellung finden lassen. In diesem Abschnitt wird erläutert, dass die untersuchten Übungsaufgaben tatsächlich zwanglos zu einer von lediglich sechs wichtigen Klassen von Aufgaben zugeordnet werden können. Dies erlaubt die Entwicklung von generischen Algorithmen, deren Grundfunktionen jeweils auf eine ganze Gruppe von Übungsaufgaben angewendet werden können. Gegenüber der vollständigen Neuentwicklung der Software für jede einzelne Aufgabe ist also nur noch eine Spezialisierung der gemeinsamen Basisfunktionalität erforderlich, was auch die spätere Anwendung der im Projekt erstellten Algorithmen auf weitere Anwendungsgebiete aus anderen Themenbereichen stark erleichtern wird.

Als Grundlage für diese Analyse dienten an erster Stelle die Übungen zur T-Vorlesung des Autors, zu denen sowohl die Musterlösungen als auch die Erfahrungen der Übungsgruppenleiter und Feedback der Studenten vorlagen:

- K. von der Heide, Vorlesung *Technische Informatik 1 und 2* im WS 2002 und SS 2003 [6, 7]. Alle Aufgabenblätter stehen zusammen mit den vollständig ausgearbeiteten Musterlösungen für die Studenten zum Download zur Verfügung [8].

Quellen

Als weitere Quellen wurde eine Reihe von bekannten Lehrbüchern in Hinblick auf die enthaltenen Übungsaufgaben untersucht. Die Auswahl konzentrierte sich dabei auf jene Standardwerke, zu denen neben den Übungsaufgaben auch die zugehörigen Musterlösungen zur Verfügung standen. Besondere Erwähnung verdienen die folgenden Bücher:

- W. Schiffmann & R. Schmitz, *Technische Informatik* [24]. Diese Bücher sind im deutschen Sprachraum sehr beliebt und derzeit bereits in der vierten, überarbeiteten Auflage erhältlich. Neben den beiden Bänden des Lehrbuchs gibt es einen zusätzlichen dritten Band [26] mit über 100 vollständig gelösten Übungsaufgaben.
- A. S. Tanenbaum, *Structured Computer Organization* [32], ein international anerkanntes Standardwerk, das sich nach einer gründlichen Überarbeitung für die vierte Auflage durch eine aktuelle Stoffauswahl auszeichnet. Die Lösungen vieler Übungsaufgaben aus dem Buch sind für Professoren (nach Anmeldung) über die Website des Prentice-Hall Verlags zugänglich.
- D. A. Patterson & J. L. Hennessy, *Computer Organization and Design: the Hardware/Software-Interface* [13]. Der Schwerpunkt dieses Buchs liegt auf der Vermittlung der Grundprinzipien von (RISC-) Mikroprozessorsystemen. Mehrere Anhänge enthalten aber weitere Kapitel mit grundlegenen Themen, so dass auch Zahldarstellung, Arithmetik, und der Entwurf von Schaltnetzen behandelt werden.

Das in diesen Büchern behandelte Material umfasst alle Themen der im letzten Abschnitt genannten Vorlesungen T1 bis T3. Absichtlich nicht berücksichtigt wurden die Inhalte und Übungsaufgaben zur Vorlesung T4, da diese derzeit in einem eigenen ELCH-Projekt untersucht werden.

Sechs zentrale Klassen von Übungsaufgaben

Natürlich setzen die verschiedenen Lehrbücher unterschiedliche Akzente, und auch die Anforderungen an die Studenten unterscheiden sich deutlich, nicht zuletzt in Bezug auf die notwendige Mathematik. Zum Beispiel werden die Übungsaufgaben in [13] nach ihrem Schwierigkeitsgrad eingeteilt, der von einfachen Aufgaben mit einer angenommenen Bearbeitungsdauer von weniger als einer Minute bis zu aufwendigen Aufgaben reicht, die umfangreiche Programmierarbeit erfordern. Insgesamt ergeben sich aber klare Überschneidungen, aus denen sich die folgenden sechs Kategorien von Übungsaufgaben für das untersuchte Themengebiet ableiten lassen, die in Abbildung 4 zusammengestellt sind:

- Die Klasse der *Auswahlaufgaben* umfasst Fragestellungen, bei denen die Lösung durch Auswahl aus vorgegebenen Alternativen erfolgen kann, darunter die gängigen Ja/Nein- und Multiple-Choice- Aufgaben, aber auch Zuordnungsaufgaben oder Image-Maps. In allen Fällen ist die automatische Überprüfung durch trivialen Vergleich mit einer Musterlösung möglich; aus diesem Grund wurden auch die Lückentexte mit in diese Kategorie mit aufgenommen.

- Die nächste Klasse enthält alle Aufgaben, deren Antwort aus *Zahlenwerten* besteht. Neben den üblichen Integer- und Gleitkommazahlen sind im Rahmen der technischen Informatik auch diverse Varianten, darunter die Binär- oder Hexadezimalformate und die Komplementdarstellungen, zu berücksichtigen.

- Die Kategorien der *Formeln* umfasst lineare und nichtlineare Gleichungen, Gleichungssyteme, logische Ausdrücke und die Boole'sche Algebra. Wegen der ähnlichen Strategie zur Auswertung fallen auch Funktions- und Wertetabellen in diese Klasse.

- Die Klasse der *Diagramme* umfasst eine Vielfalt von verschiedenen Untergruppen. Im Rahmen der technischen Informatik sind besonders die Schaltpläne, aber auch Zustandsdiagramme von Automaten und Flussdiagramme von Algorithmen wichtig. Ebenfalls in Abbildung 4 mit aufgenommen wurden weitere Diagrammtypen, die in anderen Anwendungsgebieten häufig vorkommen.

- Die Klasse der *Text*-Aufgaben bereitet für die automatische Überprüfung die größten Schwierigkeiten. Wie in Abschnitt 1.5 angedeutet wurde, dürfte in den meisten Fällen eine manuelle Korrektur durch die Übungsgruppenleiter erforderlich sein.

- In die letzte Klasse der *Programme* fallen alle Aufgaben, bei denen die Softwareentwicklung im Mittelpunkt steht. Unabhängig von der jeweiligen Abstraktionsebene oder der geforderten Programmiersprache ist eine automatische Überprüfung durch Übersetzen und Ausführen der Programmtexte möglich.

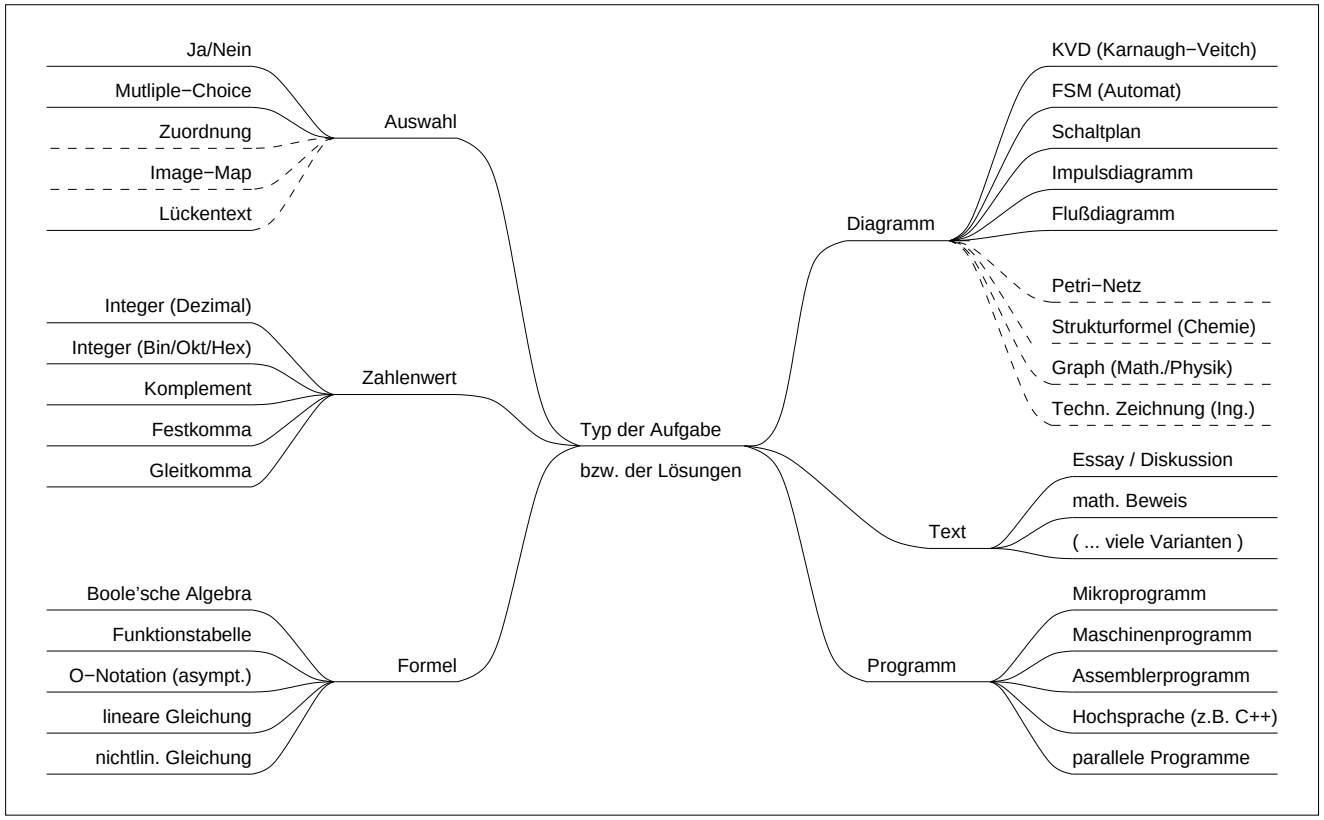


Abbildung 4: *Klassifikation der Übungsaufgaben zur technischen Informatik. Die sechs Hauptklassen dürften auch auf andere mathematisch naturwissenschaftliche Fächer übertragbar sein. Für fast alle Kategorien bis auf freie Texte ist eine automatische Überprüfung möglich.*

2.3 Auswertung und Statistik

Gewichtung der Aufgabentypen

Obwohl sich alle der untersuchten Übungsaufgaben in eine der oben aufgestellten sechs Klassen einordnen lassen, bleiben noch eine Reihe von Fragen offen. Insbesondere erlaubt die Klassifikation noch keine Aussage darüber, mit welcher Gewichtung und Häufigkeit die verschiedenen Typen von Übungsaufgaben in den einzelnen Vorlesungen bzw. Lehrbüchern vorkommen. Es ist zu vermuten, dass sich die relative Gewichtung der Typen von Übungsaufgaben beträchtlich voneinander unterscheiden können. Gründe dafür sind die Eigenheiten und Gewohnheiten des jeweiligen Fachgebiets, aber auch die subjektiven Themenschwerpunkte der Stoffauswahl und die Vorlieben der Autoren.

Für drei der betrachteten Materialien wurde deshalb auch die Häufigkeit der verschiedenen Aufgabentypen analysiert. Abbildung 5 fasst diese Ergebnisse als Tortendiagramme zusammen, so dass die zahlenmässig häufigsten Klassen von Aufgaben auf einen Blick zu erfassen sind.

Zahlenwerte und Formeln

In allen betrachteten Fällen decken die beiden Kategorien *Zahlenwerte* und *Formeln* einen großen Anteil der Übungsaufgaben ab. Offensichtlich handelt es sich hier um besonders lohnende Kandidaten für eine zügige Implementierung der Algorithmen zur automatischen Überprüfung, die zumindest im Fall der Zahlenwerte auch noch geradlinig umsetzbar ist.

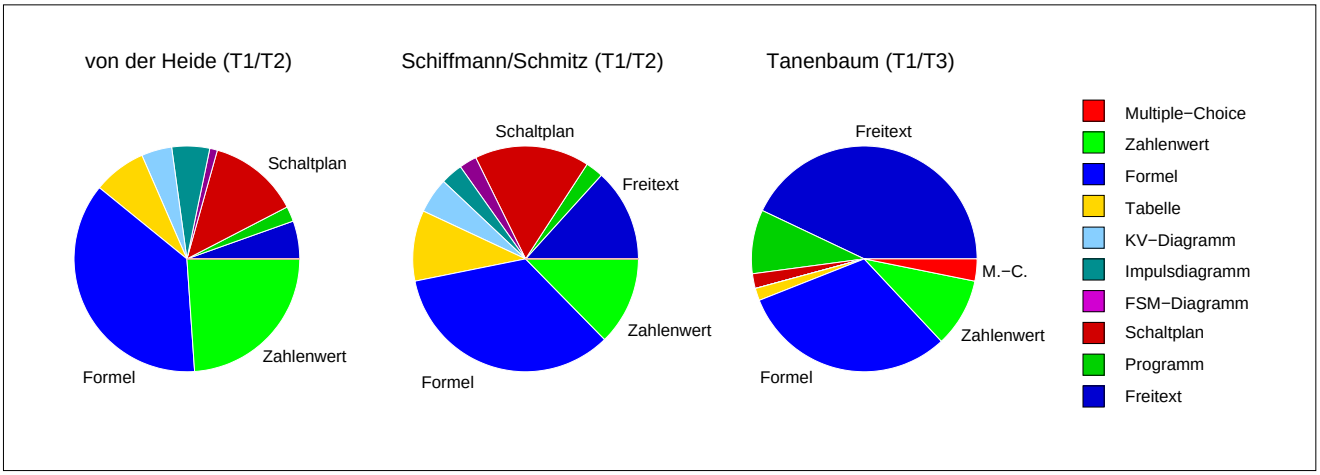


Abbildung 5: *Gewichtung der Übungsaufgaben in den betrachteten Vorlesungen bzw. Lehrbüchern. Unterschiede ergeben sich durch die Stoffauswahl (T1/T2 bzw. T1/T3) und die Gewichtung einzelner Themen. Einfache Auswahl-Fragen fehlen fast völlig. Ein großer Teil der Fragen kann durch geeignete Parser für Zahlenwerte und Formeln sowie die Unterstützung für Schaltpläne abgedeckt werden.*

Auffällig ist außerdem der hohe Anteil von Aufgaben der Kategorie *freier Text* zu [32], der sich aus den zu vielen Aufgaben geforderten Diskussionen oder ausführlichen Begründungen ergibt. Wie sich im nächsten Abschnitt zeigen wird, eignen sich diese Aufgaben beim derzeitigen Stand der Texterkennungsverfahren leider nicht für die automatische Überprüfung bzw. Hilfestellung. Für einen möglichst umfangreichen Einsatz der im Projekt entwickelten Tools wird daher eine andere Auswahl der Übungsaufgaben notwendig sein.

Der gegenüber [32] deutlich höhere Anteil der *Schaltplan*-Aufgaben in [8] und [26] ist direkt durch die unterschiedliche Themenauswahl (T1/T3 vs. T1/T2) bedingt.

Fast keine Auswahl-Aufgaben

Erst auf den zweiten Blick wird deutlich, dass in den betrachteten Quellen *Auswahl*-Aufgaben entweder überhaupt nicht oder nur außerordentlich selten vorkommen. Für das betrachtete Themengebiet erweist sich also die Unterstützung genau dieser Klasse von Aufgaben in den üblichen E-Learning Frameworks als vollkommen nutzlos — insgesamt könnten weniger als 3 Prozent aller untersuchten Aufgaben durch diese Frameworks bearbeitet werden. Natürlich darf dies nicht verwundern, da keine der Quellen ausdrücklich auf eine automatische Überprüfung der Übungsaufgaben hin ausgerichtet ist, sondern vielmehr die Korrektur und Erläuterung durch die Übungsgruppenleiter vorausgesetzt wird und zusätzlich ausführliche Musterlösungen bereitgestellt werden.

2.4 Repräsentation der Daten

Problem der Handschrifterkennung

Im herkömmlichen Übungsbetrieb reichen die Studierenden ihre Lösungen und Antworten handgeschrieben auf Papier zur Überprüfung durch die Übungsgruppenleiter ab und bekommen die Aufgaben mit ebenfalls handgeschriebenen Anmerkungen später korrigiert zurück. Leider lässt sich dieses Verfahren noch nicht zuverlässig automatisieren, da selbst bei der reinen Texterkennung noch mit einer inakzeptabel hohen Fehlerrate gerechnet werden muss. Zudem wäre nicht nur die Erkennung von Textdokumenten, sondern darüber hinaus auch die Interpretation von Formeln, Diagrammskizzen und Schaltplänen erforderlich, was die verfügbaren Programme schlichtweg überfordert.

Deshalb ist eine einheitliche Repräsentation der von den Studenten eingereichten Lösungen in elektronischer Form notwendig. Eine Unterstützung der strukturierten Dateiformate der gängigen Textverarbeitungsprogramme (insbesondere Word `.doc` und `.rtf`, Acrobat `.pdf`) ist im Rahmen des Projekts ebenfalls nicht möglich. Der Hauptgrund dafür ist die hohe Komplexität der Dokumentenformate, die das Erstellen von ausreichend leistungsfähigen und robusten Parsern sehr erschwert. Die bekannten Probleme beim Austausch von Dokumenten zwischen unterschiedlichen Programmen sind ein deutliches Indiz für die in diesem Zusammenhang zu erwartenden Schwierigkeiten.

Zusätzlich gibt es in den meisten Programmen völlig unterschiedliche Ansätze, um komplexe Inhalte zu repräsentieren. Zum Beispiel könnte eine Formel einerseits über den zugehörigen Formeleditor erstellt werden, andererseits aber „von Hand“ durch Platzierung einzelner lateinischer und griechischer Buchstaben zusammen mit Ziffern und mathematischen Symbolen aufgebaut werden. Der Parser stände dann vor der Aufgabe, diese Kombination als Formel zu erkennen und aus den Layoutpositionen der einzelnen Zeichen wieder zu rekonstruieren.

Unterstützte Datenformate

Daraus folgt, dass die mögliche Repräsentation möglichst auf einfache und offen dokumentierte Dateiformate beschränkt werden muss. Es wird angestrebt, die folgenden Dokumententypen zu unterstützen:

- ASCII Textdateien (`.txt`) mit unformatiertem oder einfach formatiertem Inhalt, zum Beispiel für Zahlenwerte, mathematische Formeln, logische Ausdrücke. Dieses Format wird auch für die Repräsentation von frei formuliertem Text eingesetzt.

- ASCII Textdateien mit Tabellen, zum Beispiel Funktionstabellen oder State-Machine Beschreibungen.

- Matlab Quelltexte (`.m` Dateien) mit ausführbaren Skripten oder Funktionsdefinitionen, die jeweils die Lösung zu einer einzelnen Aufgabe enthalten.

- Textdateien mit Programm-Quelltexten, zum Beispiel für Aufgaben zur Assemblerprogrammierung. Das genaue Format der Datei wird durch die zugehörigen Tools (Assembler, Compiler, usw.) festgelegt.

- Hades Designdateien (`.hds`) als Repräsentation für Schaltnetze und Schaltwerke.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE uebungsaufgabe SYSTEM "aufgabe.dtd">
<student-name="Max Muster" />
<matrikelnummer="9870815" />

<aufgabe nummer="1.2">
  <zwischenrechnung>
      1998 : 2 = 999 Rest           0
       999 : 2 = 499 Rest          1
       499 : 2 = 249 Rest         1
       249 : 2 = 124 Rest        1
       124 : 2 =  62 Rest       0
        62 : 2 =  31 Rest      0
        31 : 2 =  15 Rest     1
        15 : 2 =   7 Rest    1
         7 : 2 =   3 Rest   1
         3 : 2 =   1 Rest  1
         1 : 2 =   0 Rest 1
      (1998)_10 =        (11111001110)_2
  </zwischenrechnung>

  <ergebnis>
      11111001110
  </ergebnis>
</aufgabe>
```

Abbildung 6: *Beispiel für die Repräsentation einer Übungsaufgabe mit XML-Darstellung der Verwaltungsinformationen sowie reinem ASCII-Text für Zwischenrechnung und Endergebnis einer Aufgabe zur Zahldarstellung (Umrechnung vom Dezimalsystem in das Dualsystem).*

- Als Dateiformat für Diagramme wird zunächst FIG (`.fig`) zum Einsatz kommen, da ein geeigneter Editor zur Verfügung steht. Zusätzlich ist die Unterstützung des SVG-Standards (scalable vector graphics) geplant, kann aber noch nicht garantiert werden.
- In der letzten Projektphase ist geplant, zusätzlich HTML Formularstrukturen (Eingabefelder, Auswahl-Menüs, usw.) als Eingabeformat zu unterstützen. Dies wird die Integration in Web-basierte Applikationen oder E-Learning Plattformen erleichtern.

Metadaten In einem Gesamtsystem zur automatischen Korrektur der Übungsaufgaben müssen neben den eigentlichen Nutzdaten — den von den Studenten erstellten Lösungen der Übungsaufgaben — natürlich auch Metainformationen zu den einzelnen Datensätzen verwaltet werden, u.a. die Nummer der Aufgaben und die Namen und Matrikelnummern der Studenten. Angesichts der zunehmenden Verbreitung von auf XML [34] basierenden Dokumentenformaten und Tools bietet es sich an, auch für

dieses Projekt die Metadaten in einer XML-Notation zu repräsentieren. Ein Beispiel für dieses Konzept zeigt Abbildung 6 mit den XML-Verwaltungsinformationen sowie den konkreten Nutzdaten mit Zwischenrechnung und Endergebnis einer Übungsaufgabe zum Thema Zahldarstellung. Da nur wenige XML-Tags unterstützt werden müssen, wird in der ersten Projektphase ein eigener Java-basierter Parser eingesetzt werden, der ohne XSLT-Unterstützung auskommt. Details zu dieser Architektur werden im nächsten Projektbericht dokumentiert. Ein späterer Umstieg auf einen leistungsfähigeren Parser oder die Integration in eines der kommerziellen E-Learning Frameworks ist bei Bedarf jederzeit möglich.

2.5 Verfahren zur Überprüfung der Übungsaufgaben

Anforderungen

Wie bereits im Abschnitt 1.6 erläutert wurde, soll die automatische Überprüfung der Übungsaufgaben sowohl während der häuslichen Bearbeitung der Aufgaben durch die Studierenden als auch bei der Korrektur durch die Gruppenleiter zum Einsatz kommen. In beiden Szenarien soll die Software innerhalb von wenigen Sekunden eine Antwort liefern, ob oder inwieweit die abgegebene Lösung bereits korrekt ist. Der Schwerpunkt liegt im ersten Fall auf einem schnellen Feedback für die Studierenden, während im zweiten Fall eine möglichst hohe Genauigkeit der Bewertung notwendig ist.

Verfahren

Dazu müssen die relevanten Daten aus ihrer externen Repräsentation, etwa als ASCII-Textdatei, extrahiert und in eine interne Darstellung umgesetzt werden, die dann im nächsten Schritt abhängig von der Art der Aufgabe analysiert und bewertet wird. In diesem Abschnitt werden die Algorithmen und Techniken beschrieben, die bisher als aussichtsreiche Kandidaten für die automatische Überprüfung ausgewählt worden sind. Eine tabellarische Übersicht über diese Verfahren zeigt Abbildung 7 auf Seite 27. Für die einzelnen Klassen der Übungsaufgaben ergeben sich diese Varianten:

- *Auswahl*-Aufgaben (Ja-/Nein, Multiple-Choice, Zuordnungen): Für diese Kategorie von Aufgaben erfolgt die Überprüfung trivial durch Vergleich mit der vorher bekannten Musterlösung. Die einzige Komplikation ergibt sich für Multiple-Choice Fragen, bei denen mehrere Antworten angekreuzt werden müssen, so dass auch teilweise richtige Antworten möglich sind. Der zugehörige Parser wird eine Einstellung enthalten, ob derartige Lösungen als komplett falsch gewertet werden sollen.
- *Zahlenwerte*: Auch für diese Klasse von Aufgaben genügt ein einfacher Wertevergleich für die Überprüfung auf Korrektheit. Dies gilt jedenfalls für die üblichen Integerzahlen in Dezimaldarstellung, deren Überprüfung auch von den gängigen Frameworks bereits abgedeckt wird. Für die anderen in der technischen Informatik häufigen Fragestellungen werden erweiterte Parser benötigt, unter anderem für Integerzahlen in anderen Zahlensystemen wie Binär-, Oktal-, oder Hexadezimalsystem, Komplementdarstellung, Festkommadarstellung, und Gleitkommadarstellung. Bei der Festkommadarstellung wäre zu überlegen, im Zahlenformat auch eine Syntax für periodische Brüche (etwa 1/3 im Binärsystem) zu definieren. Für Anwendungen in der Physik und den Ingenieurwissenschaften sollten darüber hinaus nicht nur die reinen Zahlenwerte sondern auch die (SI-) Einheiten unterstützt werden.

- *Formeln*: Für diese Kategorie von Aufgaben bietet sich auf den ersten Blick die symbolische Verarbeitung an, wie sie von den üblichen Mathematik-Programmierumgebungen direkt unterstützt wird. Für die Überprüfung ist allerdings zu berücksichtigen, dass die Darstellung selbst elementarer Formeln nicht unbedingt eindeutig ist und die Vereinfachung der Ausdrücke durch die Software auch nicht zuverlässig zu eindeutigen Ergebnissen führt.

  Als Alternative kann die Überprüfung auch durch Auswertung der Formeln für geeignet gewählte Eingangswerte erfolgen. Da die im Grundstudium üblichen Aufgaben keine komplexen Probleme behandeln, ist in vielen Fällen sogar die vollständige Überprüfung (*complete enumeration*) möglich. Ein typisches Beispiel bieten die Aufgaben zur Boole'schen Algebra oder Schaltfunktionen, bei denen selten mehr als acht Variablen vorkommen, so dass die Überprüfung aller möglichen Eingabekombinationen in weniger als einer Sekunde erfolgen kann. Falls die vollständige Auswertung nicht in Frage kommt, können natürlich auch vom Autor vorgegebene oder zufällig generierte Eingangswerte für die Tests der Formeln ausgewertet werden. Da bei einigen Aufgaben (z.B. dem Entwurf von Filtern) keine eindeutig beste Lösung existiert, kann der Vergleich der Ausgangswerte auch unter Berücksichtigung von zulässigen Fehlerintervallen erfolgen.

  Bei der Behandlung von logischen Ausdrücken wird häufig konkret nach bestimmten Darstellungen gefragt, insbesondere den so genannten Normalformen. In diesen Fällen ist die Überprüfung trivial, da die Darstellung nach lexikalischem Sortieren der Terme eindeutig ist. Die Umwandlung in eine Normalform bietet eine weitere Möglichkeit zur Überprüfung beliebiger Boole'scher Funktionen.

- *Diagramme*: Wegen der Vielfalt der möglichen Diagrammtypen ist eine weitere Unterteilung dieser Klasse von Aufgaben notwendig. Für die immer noch häufig vorkommenden Karnaugh-Veitch Diagramme (KVD) stehen uns bereits selbstentwickelte Applets zur Verfügung, die die interaktive Eingabe der Funktionen und die interaktive Minimierung unterstützen. Für die Überprüfung müssen also lediglich die Eingaben der Studenten aus den Applets ausgelesen und mit den Musterlösungen verglichen werden. Ähnlich sieht die Situation in Bezug auf endliche Automaten (finite state machines, FSM) aus, für die ebenfalls bereits Applets vorhanden sind. Da in den Übungsaufgaben nur sehr kleine Automaten entworfen werden müssen, stellt eine vollständige Simulation keinerlei Problem dar. Alternativ dazu könnten die im Zusammenhang mit der formalen Verifikation von Hardware entwickelten Techniken des Model-Checking verwendet werden, um die Korrektheit der entworfenen Automaten zu überprüfen, da diese Algorithmen direkt Hinweise auf noch bestehende Fehler liefern.

  Als Werkzeug zur Überprüfung von Hardwarestrukturen wie Schaltnetzen und Schaltwerken auf Gatterebene und optional der Register-Transfer-Ebene wird das von uns bereits seit einiger Zeit im Praktikum Technische Informatik eingesetzte Hades-Framework zum Einsatz kommen. Hades umfasst einen graphischen Schaltplaneditor und erlaubt die interaktive Simulation bereits während der Eingabe; zusätzlich stehen Hilfsmittel zur Visualisierung zur Verfügung. Zur Überprüfung der von den Studierenden entworfenen Schal-

tungen können sowohl die vollständige Simulation mit allen möglichen Eingabedaten als auch die Technik der Signaturanalyse benutzt werden. Dabei wird die zu testende Schaltung mit gezielt ausgewählten oder pseudozufälligen Eingabedaten betrieben, während die Ausgangswerte der Schaltung von einem Register zu einer Signatur verrechnet werden, die anschließend mit der zuvor generierten Signatur der Musterlösung verglichen werden kann.

Zur Darstellung anderer Diagrammtypen wird ein geeigneter Editor benötigt. Hier ist im Einzelfall zu entscheiden, ob ein universeller Graphikeditor geeignet ist, oder doch lieber eine speziell auf die jeweilige Anwendung optimierte Software zum Einsatz kommen sollte. Mit *jfig* steht uns ein in Java implementierter 2D-Graphikeditor zur Verfügung, der ein offenes Dateiformat verwendet, skriptfähig ist, und über das Java-Interface auch von Matlab und den interaktiven Skripten heraus gesteuert werden kann. Als Ersatz lassen sich auch die Handle-Graphics Funktionen von Matlab nutzen, obwohl der zugehörige Graphikeditor nur wenige Zeichenoperationen und -hilfen bereitstellt. Auf diese Weise könnten auch andere der in der Klassifikation aufgezählten Diagrammtypen wie Flussdiagramme oder Petri-Netze repräsentiert werden, wenn keine geeigneten Spezialeditoren benötigt werden.

Für die in einigen Übungsaufgaben geforderten *Impulsdiagramme* steht leider bisher kein geeigneter Editor zur Verfügung. Wie die meisten Simulatoren stellt auch das Hades-Framework lediglich einen Viewer bereit, der zur Anzeige der während der Simulation aufgezeichneten Impulsdiagramme dient, aber keine Edit-Funktionen zur interaktiven Eingabe der Daten enthält. Es gibt zwar einige derartige Editoren (auch als Freeware), aber die Bedienung ist verhältnismässig kompliziert, so dass der zusätzliche Lernaufwand zur Einarbeitung für die Studierenden zu hoch wäre. Deshalb werden wir für diese Übungsaufgaben einfache Templates als ASCII-Textdateien bereitstellen, in denen die Studierenden dann mit einem Texteditor die (zeitdiskreten) Impulsdiagramme eintragen können. Die Überprüfung der Lösung ist damit auf einen trivialen Textvergleich reduziert.

- *Programme*: Hier genügt es, die Programme auszuführen und die Ausgaben mit den bekannten Musterlösungen zu vergleichen. Selbst bei exakt spezifiziertem Format der erwarteten Programmausgaben sollte der Vergleich gegenüber kleinen Abweichungen tolerant sein, um die Studierenden nicht allzu stark einzuschränken bzw. frühzeitig auf Formatfehler (im Gegensatz zu inhaltlichen Fehlern) hinzuweisen. Da dieser Themenbereich bereits gut erforscht ist, kann darüber hinaus auf zusätzliche Tests zurückgegriffen werden, etwa um aus der Übereinstimmung struktureller Merkmale auf ein „Abschreiben“ der Programme zu schließen.

  Während die selbstgeschriebenen Programme zum Test der Korrektheit auf den Rechnern der Studierenden ohne weiteres gestartet werden können, müssen bei einer Server-basierten Realisierung die üblichen Risiken von nicht vertrauenswürdigem Code berücksichtigt werden. Andernfalls könnten die von den Studenten eingeschickten Programme durch versehentlich — oder eventuell absichtlich — eingebaute Fehler durchaus Schaden anrichten. Da die für die Übungen einzusendenden Programme normalerweise aber nur einfache Textausgaben erzeugen, lassen sich die Programme in einer gesicherten

Umgebung und mit beschränkten Zugriffsrechten ausführen. Dies ist mit geringem Aufwand umzusetzen, da die betroffenen Softwarekomponenten in Java implementiert werden, wo entsprechende Sicherheitsmechanismen zur Verfügung stehen.

- *Freie Texte*: Für diese Klassen von Aufgaben wird es nur in Ausnahmefällen möglich sein, eine automatische Überprüfung zu gewährleisten, da dies ein echtes Textverständnis erfordern würde. Die gängigen Verfahren zur Textklassifikation, etwa die häufig zur Erkennung von Spam-Mails eingesetzen Bayes-Filter, erlauben nur eine äußerst grobe Klassifikation und arbeiten selbst für ein Vorsortieren der Übungsaufgaben nicht zuverlässig genug. Der Einsatz fortgeschrittener Verfahren der Sprachverarbeitung ist im Rahmen des Projekts nicht zu leisten, dürfte aber wegen der potentiell enormen Anwendungsvielfalt ein sehr interessantes Forschungsgebiet für weitere Projekte bieten.

Geeignete Auswahl der Übungsaufgaben

Trotz der Vielzahl der skizzierten Verfahren zur automatischen Überprüfung der Aufgaben wird der Erfolg des Projekts nicht zuletzt auch von einer geeigneten Auswahl bzw. Formulierung der Übungsaufgaben abhängen. Ein gutes Beispiel für diesen Aspekt bieten zwei auf den ersten Blick sehr ähnliche Aufgaben aus den Übungen zur Vorlesung T1 im WS2002/2003:

> **Aufgabe T1-2.5**: Finden Sie einen zyklisch-einschrittigen Code mit 12 Codewörtern. Ein solcher Code könnte z. B. für eine Winkelcodierung in 30^o-Schritten benutzt werden.

Lösung nicht eindeutig

Da eine korrekte Lösung zu dieser Aufgabe aus einer Liste mit den zwölf Codewörtern besteht, erscheint die automatische Überprüfung der Lösung trivial durch Vergleich der Codewörter mit einer Musterlösung möglich zu sein. Das Problem mit diesem Ansatz ist jedoch, dass die Aufgabe nicht nur eine einzige sondern vielmehr eine Vielzahl von korrekten Lösungen zulässt, die zwar in Bezug auf die Anforderungen gleichwertig sind, sich aber nicht direkt (etwa durch Umsortieren) ineinander überführen lassen. Eine vollständige Auflistung aller möglichen korrekten Lösungen wäre zwar möglich, ist wegen der großen Anzahl der Lösungen aber nicht praktikabel. Damit dürfte die automatische Überprüfung der Lösung zu dieser Aufgabe bereits die meisten am Markt erhältlichen E-Learning Frameworks überfordern, selbst wenn diese über die nötigen Parser für Binärzahlen verfügen.

aber leicht überprüfbar

Trotzdem ist für diese Aufgabenstellung eine Überprüfung der Lösung auf Korrektheit problemlos mit einem einfachen Algorithmus realisierbar. Dazu wird zunächst getestet, ob die Lösung genau 12 unterschiedliche Codeworte der Länge 4 bit umfasst. Ausgehend vom ersten Codewort wird anschließend überprüft, ob sich die nachfolgenden Codeworte jeweils um nur 1 bit unterscheiden. Auch eine interaktive Hilfestellung ist für diese Aufgabe leicht zu implementieren, da ein passender Assistent die wesentlichen Merkmale der Lösung (Anzahl, Länge und Hamming-Distanz der einzelnen Codewörter) mit geringem Aufwand überprüfen kann. Daraus lassen sich dann sofort hilfreiche Fehlermeldungen und Tips ableiten.

Trotz der sehr ähnlichen Aufgabenstellung erscheint eine automatische Überprüfung für die direkt vorangestellte Aufgabe dagegen praktisch unmöglich:

> **Aufgabe T1-2.4**: Erläutern Sie, warum es keinen zyklisch-einschrittigen Code mit ungerader Zahl von Codewörtern geben kann.

In diesem Fall besteht die korrekte Antwort in einem Hinweis darauf, dass sich die Parität aufeinanderfolgender Codewörter unterscheiden muss, dies aber bei ungerader Anzahl von Codewörtern zu einem Widerspruch führt. Angesichts der Vielzahl der möglichen Formulierungen dürfte eine automatische Auswertung des Lösungstextes selbst die derzeit fortschrittlichsten Werkzeugen der Textanalyse überfordern. Allenfalls wäre eine grobe Vorauswahl denkbar, indem ein Parser gezielt nach Schlüsselworten wie „Parität“ oder „Widerspruch“ sucht.

2.6 Verfahren zur Hilfestellung

Mögliche Hilfestellungen

Die ebenfalls im Projekt geplante Realisierung einer automatischen Hilfestellung für die Studenten ist wesentlich aufwendiger zu realisieren. Während die Überprüfung auf Korrektheit in den meisten Fällen durch Vergleich mit einer bekannten Musterlösung erfolgen kann, sind für (nützliche) Hilfestellungen bei mangelhaften Lösungen einerseits ein Verständnis der Fehlerursache und andererseits daraus abgeleitete konkrete Verbesserungsvorschläge notwendig. Diese wiederum müssen schonend vermittelt werden, um die Studierenden nicht abzuschrecken sondern zu motivieren, sollten die weitere Bearbeitung der Aufgaben aber auch nicht zu stark vereinfachen.

Der klassische Ansatz, den Lehrstoff in der Art eines regelbasierten KI-Systems zu repräsentieren, scheitert schon am großen Stoffumfang und der in Abschnitt 2.1 angedeuteten Themenvielfalt. Die Umsetzung innerhalb des Projekts wird sich daher auf einige ausgewählte Themen beschränken müssen, während für die übrigen Aufgabenstellungen einfachere Verfahren zu entwickeln sind. Zum Glück erlauben viele der im letzten Abschnitt beschriebenen Algorithmen nicht nur die reine Überprüfung der Lösungen auf Korrektheit, sondern liefern bei Fehlern auch weitere Informationen, die sich für die Hilfestellung verwenden lassen.

Einige Beispiele für die Vielfalt der möglichen Hilfestellungen in den verschiedenen Anwendungsfällen bietet die folgende Liste:

- Elektrotechnische Grundlagen: Überprüfen Sie noch einmal gründlich die Polarität der Spannungsquellen in Ihrer Berechnung!
- Zahldarstellung: Überlegen Sie sich, welche Ziffern für die Zahldarstellung im Oktalsystem überhaupt zulässig sind!
- Boole’sche Algebra und Schaltfunktionen: Was passiert mit Ihrem Schaltnetz unter der Eingangsbelegung $a_0 = 0$ und $a_1 = 1$?
- Schaltwerksentwurf: Wie viele Flipflops benötigt man mindestens, um einen Automaten mit den geforderten sieben Zuständen zu realisieren?

- Die Resetleitung ist noch nicht an allen Flipflops angeschlossen!
- Rechnerarchitektur: Im dritten Schritt des Mikroprogramms wird der Datenbus von zwei Treibern getrieben; das führt zu einem Kurzschluss und in der Simulation zu undefinierten Werten.
- von-Neumann Rechner: Ihr Programm liefert die falschen Ergebnisse für die folgenden Eingabedaten: (Liste mit Eingabedaten)

Hilfreiche Gegenbeispiele

Ein wichtiger Aspekt der obigen Liste sind die Gegenbeispiele, so dass die Fehlersuche direkt auf die (hoffentlich wenigen) problematischen Stellen eingegrenzt werden kann, an denen die aktuelle Lösung noch zu falschen Ausgangswerten führt. Bei mehreren der oben aufgezählten Verfahren zur Überprüfung der Lösungen fallen geeignete Gegenbeispiele als Nebenprodukt des Tests ab, etwa bei der vollständigen Simulation einer Schaltfunktion oder dem Test eines Programms mit geeigneten Eingangswerten. Für die Überprüfung von Automaten und Schaltwerken muss wegen der mit der Anzahl der Zustände und Übergänge extrem schnell ansteigenden Komplexität eventuell auf effizientere Algorithmen ausgewichen werden. Hier bieten die im Kontext der formalen Verifikation von Hardwarestrukturen entwickelten Model-Checking Algorithmen einen interessanten Ansatz, der für die in den Übungsaufgaben vorkommenen Automaten auf jeden Fall ausreicht.

aber nicht nur Textmeldungen

Natürlich sollte die im Projekt erstellte Software nicht ausschließlich Textmeldungen erzeugen, sondern in die jeweils zugrunde liegenden Skripte und Simulatoren integriert werden. Ein Beispiel dafür ist es, Kurzschlüsse oder offene Eingänge in einem Schaltplan direkt durch Hervorhebung zu markieren, so dass die Studierenden diese Probleme auf einen Blick erkennen können.

- Für *Auswahlfragen* ergibt sich die kuriose Situation, dass sich die Überprüfung durch einen Wertevergleich zwar sehr einfach gestaltet, gute Hilfestellungen dagegen praktisch unmöglich zu erzeugen sind, da aus einer falsch angekreuzten Lösung nicht auf die eigentliche Fehlerursache zurückgeschlossen werden kann. Hier könnten höchstens anwendungsspezifische Regeln oder Erfahrungen aus zurückliegenden Semestern eingesetzt werden, um häufige Fehlerursachen zu identifizieren und daraus Tips abzuleiten.

  Auf der anderen Seite ist die Unterstützung von Auswahl-Aufgaben durch Hilfestellung auch nicht unbedingt erforderlich, da dieser Aufgabentyp in den betrachteten Übungen kaum vorkommt. Natürlich werden Multiple-Choice Aufgaben wegen der einfachen Korrektur gerne in Klausuren eingesetzt; eine Hilfestellung für die Studierenden dürfte in dieser Situation aber ebenfalls nicht notwendig sein.

- Für Übungsaufgaben aus der Klasse der *Zahlenwerte* sind mehrere Techniken der Hilfestellung denkbar. Dies betrifft zunächst eine Reihe von Plausibilitätstests zum Wertebereich, Vorzeichen, der tatsächlich versus der erwarteten Zahldarstellung, sowie einfache Tests auf kleinere Rechen- oder Schreibfehler wie Dreher oder ausgelassene Ziffern. Die zugrunde liegenden Parser werden dazu um Funktionen ergänzt, mit denen sich diese Tests nach Bedarf ein- oder abschalten lassen.

Schließlich liefert die Überprüfung der Einheiten zu einer physikalischen Größe oft Hinweise auf Rechenfehler oder falsche Ansätze; eventuell kann durch Rekonstruktion von fehlenden Einheiten auch auf die übersprungenden oder fehlerhaften Rechenschritte zurückgeschlossen werden. Hierzu könnten sogar schematische Tests eingesetzt werden, die einfach eine Reihe von typischen Rechenoperationen berücksichtigen; auch hier könnten aber Erfahrungswerte aus vorherigen Lehrveranstaltungen direkt in die Software übernommen werden.

- Für den wichtigen Bereich der *Formeln* sind verschiedene Ansätze der Hilfestellung zu untersuchen. Einige grundlegende Tests ergeben sich bei symbolischer Verarbeitung durch die Überprüfung, ob alle für die korrekte Lösung benötigten Variablen überhaupt schon (und in richtiger Struktur) in der eingegebenen Lösung vorkommen. Auch die numerische Auswertung anhand entweder zufällig oder von Hand ausgewählter Eingangswerte für die Variablen ist sinnvoll und liefert Hinweise auf Funktionsverlauf und Skalierung der Lösung. Es ist allerdings zu beachten, dass die derzeit verfügbaren Algorithmen zur Vereinfachung einer gegebenen Funktion nicht ausreichen, um eine eindeutige Repräsentation zu erzeugen.

  Wie bereits erwähnt wurde, eignen sich Fehler, die im Verlauf der vollständigen Auswertung einer Formel gefunden werden, als Gegenbeispiele und helfen, noch vorhandene Probleme gut zu lokalisieren. Ähnliches gilt bei Umwandlung in Normalformen, da sich aus der dann eindeutigen Darstellung ebenfalls hilfreiche Tips abgeleiten lassen. Schließlich können einfache Plausibilitätstests („die Funktion wird niemals Null“, „für $x = \pi/2$ ergibt sich eine Division durch Null“, usw.) wertvolle Zusatzinformationen liefern.

  Beim Spezialfall der *logischen Ausdrücke* inklusive der gängigen Übungsaufgaben zur Minimierung von Schaltfunktionen ergeben sich zusätzliche Möglichkeiten. Wegen der binären Variablen ist für praktisch alle in den Übungen vorkommenden Probleme eine vollständige Überprüfung der Lösungen durch Vergleich mit der Musterlösung möglich. Ein einfaches Auflisten der dabei gefundenen Fehler erlaubt eine gezielte Nachbesserung. Durch geeignete Vorgabe der Musterlösung sind auch erweiterte Hilfestellungen möglich, etwa der direkte Hinweis auf die noch nicht vollständig minimierten Terme einer zweistufigen UND-ODER Realisierung der geforderten Funktion.

- Beim Entwurf von *Schaltfunktionen* und *Schaltwerken* werden darüber hinaus zwei weitere Techniken zum Einsatz kommen — Simulation und Plausibilitätstests. In vielen Fällen wird eine vollständige Simulation der Schaltung praktibel sein, die sämtliche Eingangsmöglichkeiten überprüft. Falls dabei Fehler in der Schaltung entdeckt werden, eignen sich die jeweiligen Eingangsdaten wieder zur Eingrenzung der Fehlerursache und können auch gezielt im Graphikeditor hervorgehoben werden. Dagegen liefert die im vorigen Abschnitt beschriebene Simulation mit pseudozufälligen Eingabedaten zwar die Information über die Korrektheit, aus einer nicht übereinstimmenden Signatur kann aber leider nicht auf die Fehlerursache zurückgeschlossen werden.

  Deshalb wird die Simulation durch einige recht einfache Plausibilitätstests ergänzt, mit denen viele häufige Fehler schnell entdeckt und aufgezeigt wer-

den können. Beim Entwurf digitaler Schaltungen lässt sich zum Beispiel überprüfen, ob die notwendige Mindestanzahl von Gattern und Flipflops verwendet wird, ob offene Eingänge vorkommen, ob Reset- und Taktleitungen korrekt beschaltet sind, usw. Auch Hinweise auf Leitungen, die während der Simulation undefiniert bleiben oder niemals ihren Wert wechseln, können wertvolle Hilfestellungen bei der Fehlersuche abgeben. Aufwendigere Tests sind natürlich möglich, erfordern aber vermutlich speziell für die jeweilige Aufgabe maßgeschneiterte Funktionen. Ein Beispiel wäre die Überprüfung, ob in einem Mikroprogramm die Write-Enable-Signale überhaupt und/oder in der richtigen Reihenfolge aktiviert werden. Erfahrungen mit dem Praktikum Technische Informatik zeigen, dass die Studierenden schon durch einfache Hinweise auf solche Problemursachen viel Zeit beim Bearbeiten der Aufgaben sparen können.

- Für *Freie Texte* dürften sich Hilfestellungen ohne ein echtes Textverständnis nur in Ausnahmefällen generieren lassen, etwa durch Hinweise auf im Text erwartete aber noch nicht vorkommende Schlüsselwörter.

2.7 Maßnahmen gegen Täuschungsversuche

Mißbrauch der Hilfestellungen

Auch wenn die Unterstützung der Studierenden durch die automatische Hilfestellung das eigentliche Ziel des Projekts darstellt, sollten die vom System generierten Hinweise sich natürlich nicht dazu eignen, die eigentliche Aufgabenstellung einfach zu umgehen. Die Studierenden sollen schließlich motiviert werden, sich wirklich um ein Verständnis zu bemühen, anstelle durch Tricks auf bequemen Abkürzungen zur Lösung zu gelangen.

Zum Beispiel könnte bei der Überprüfung von Zahlenwerten sehr einfach ein Hinweis generiert werden, ob der aktuelle Eingabewert größer oder kleiner als die korrekte Lösung ist. Das wiederum würde einen Täuschungsversuch erlauben, in dem ein Student ausgehend von einem zufälligen Anfangswert die Antworten anhand der Hinweise der Hilfestellung modifiziert, bis der korrekte Wert gefunden ist. Dies ließe sich sogar per Skript automatisieren, was zwar die Programmierkenntnisse, nicht jedoch die eigentlich zu vermittelnden Lehrinhalte vertiefen würde.

Eine detaillierte Diskussion dieser Problematik und Lösungsansätze durch Einsatz einer Client-/Server-Architektur bzw. Quotierung der Anfragen an das System wird im nächsten Projektbericht entwickelt.

Art der Aufgaben bzw. Antworten	Ansatz zur Überprüfung	Ansatz zur Hilfestellung
Multiple-Choice, Zuordnungsfragen	Wertevergleich	anwendungsspezifisch
Zahlenwerte	Wertevergleich für Integerzahlen, binär/okt/dez/hex/etc., Brüche, Gleitkomma, phys. Einheiten	Plausibilitätstests, Wertebereich, Dreher/Off-by-One, Vorzeichenfehler, Einheiten-Überprüfung
Formeln, logische Ausdrücke	Parser für Formeln, numerische Auswertung, symbolische Verarbeitung,	vollständige Enumeration, Normalformen, Plausibilitätstests
Funktions- und Wertetabellen	Parser für Tabellen, dann Wertevergleich	vollständige Enumeration, Normalformen, Plausibilitätstests
State-Machines	Simulation (Matlab, Java)	Tests der Struktur, Plausibilitätstests
Schaltfunktionen, Schaltwerke	Simulation, Signaturanalyse	vollständige Simulation, Plausibilitätstests
Impulsdiagramme	Diagrammvergleich	vollständige Analyse
Lückentexte	Wertevergleich	Sprachverarbeitung
Freitext	Textklassifikation, Bayes-Filter statistische Verfahren	statistische Auswertung, Schlüsselwortsuche, Sprachverarbeitung
Graphiken	Graphikeditor mit offeneme Dateiformat	strukturelle Merkmale, Plausibilitätstests
Programme	Programmausführung und Überprüfung der Ausgaben	anwendungsspezifisch

Abbildung 7: *Übersicht über die Ansätze zur automatischen Überprüfung und zur Hilfestellung für wichtige Klassen von Übungsaufgaben.*

3 Plattformen und Implementierung

Matlab Als Plattform für die bisher erstellten interaktiven Skripte dient Matlab, das international führende Softwaresystem für numerische Mathematik mit Schwerpunkt auf Anwendungen in technisch-orientierten Fachgebieten [23]. Mit dem Begriff Matlab werden dabei sowohl das Gesamtsystem als auch die zugrunde liegende Programmiersprache bezeichnet, die sich durch eine besonders einfache Formulierung von Matrixoperationen auszeichnet. Als Ergänzung zum Grundsystem stehen diverse Erweiterungen bereit, die applikationsspezifische Funktionen in so gennannten Toolboxes bzw. Blocksets zusammenfassen. Die Software steht nicht nur für Windows, sondern auch für alle verbreiteten Varianten von Unix zur Verfügung (u.a. Linux, Solaris, MacOS X) und erlaubt damit den plattformunabhängigen Einsatz.

Zwar können Matlab-Skripte über die so genannten *Notebook*-Funktionen direkt in Microsoft-Word Dokumente eingebettet werden, wobei alle Formatierungsmöglichkeiten von Word zur Verfügung stehen und auch eine Schnittstelle zu Excel-Tabellen vorgesehen ist. Leider ist für diese Erweiterung zwingend die Vollversion von Word erforderlich, da andere Office-Software nicht unterstützt wird. Wegen dieser Abhängigkeit und dem Verlust der Plattformunabhängigkeit eignet sich die *Notebook*-Schnittstelle daher weniger für die interaktiven Skripte dieses Projekts.

mscriptview Als Abhilfe dient *mscriptview*, ein selbstentwickelter Browser, der auf die in Matlab integrierten Funktionen zur Textformatierung zurückgreift und bei Bedarf modular erweitert werden kann. Da Formeln in einer TEX-ähnlichen Syntax direkt in die Skripte integriert werden können, ergibt sich für diesen wichtigen Aspekt gegenüber dem Formeleditor aus Word sogar eine vereinfachte Bedienung. Allerdings reichen die Möglichkeiten zur Formatierung komplexer Formeln bei weitem nicht an die umfangreichen Funktionen von TEX heran.

Lizenzbedingungen Ein Hindernis für den breiten Einsatz von Matlab in Lehrveranstaltungen und im Kontext von E-Learning betrifft die Kosten und die Lizenzbedingungen, die zudem in der Vergangenheit von MathWorks, Inc. mehrfach geändert worden sind. Eine Vollversion der so genannten Matlab-Suite mit den grundlegenden Komponenten Matlab, Simulink und Symbolic Math Toolbox kostet derzeit immerhin 1400 €. Für weitere wünschenswerte Komponenten wie die Signal-Processing oder Image-Processing Toolboxes entstehen zusätzliche Kosten. Im Rahmen der *Student Version* sind diese Komponenten bei vollem Funktionsumfang für einen deutlich reduzierten Preis von 87 € erhältlich, wobei auch hier für zusätzlich benötigte Toolboxes weitere Kosten entstehen. Die immer noch im Abverkauf erhältliche *Student Edition* basiert auf der älteren Version Matlab 5.3 und beinhaltet bereits die Signal Processing Toolbox, limitiert aber die Grösse von Datenstrukturen auf maximal 32767 Elemente, was für Audio- und Bildverarbeitung eine empfindliche Einschränkung darstellt. Wegen der großen Verbreitung dieser Version werden die im Rahmen des Projekts entwickelten Skripte soweit wie möglich auch mit der Student Edition einsetzbar sein, um möglichst viele Anwender zu erreichen.

Octave Zwar existiert mit OCTAVE [19] ein Klon von Matlab, der im Rahmen und unter den Lizenzbedingungen der GPL inklusive der Quelltexte frei verfügbar ist. Leider eignet sich Octave aber nicht als Grundlage für das interaktive Skript, da nur die Sprachelemente von Matlab 4 unterstützt werden und daher weder höhere Datenstrukturen noch die komfortable Graphikschnittstelle zur Verfügung stehen. Da die

Weiterentwicklung von Octave derzeit nur sehr schleppend erfolgt, ist im Rahmen der Projektlaufzeit keine Verbesserung zu erwarten.

Im Vergleich mit anderen bekannten Softwarepaketen für symbolische und numerische Mathematik (Mathematica, Maple, MuPAD, usw.) bietet Matlab die beste Unterstützung für technische Anwendungen wie Signalverarbeitung, Bildverarbeitung, Datenkommunikation. Dafür mangelt es, gerade im Vergleich mit den umfangreichen Möglichkeiten der Mathematica *Notebooks*, an Unterstützung zur Darstellung von Formeln und der graphischen Aufbereitung von Dokumenten. Die Symbolic Math Funktionen von Matlab basieren übrigens auf den Algorithmen aus Maple und decken alle Anforderungen dieses Projekts ab.

Java

Neben Matlab wird Java die zweite wichtige Programmierplattform für das Projekt bilden. Die Vorteile von Java wie das durchdachte und saubere Objektkonzept, die umfangreichen Klassenbibliotheken auch von Drittherstellern und die Unterstützung durch eine Vielzahl von Tools sind bekannt und brauchen hier nicht weiter erläutert zu werden. Nicht zuletzt ist Java für alle wesentlichen Desktop- und Serverplattformen frei verfügbar und bietet ideale Voraussetzungen für die Realisierung von Netzwerk- oder Client/Server-Applikationen.

Als klassische compilierte Programmiersprache eigenet sich Java auf der anderen Seite weniger für eine interaktive Umgebung wie das interaktive Skript, da nach jeder Änderung des Quellcodes ein erneutes Compilieren erforderlich ist. Dieses Problem wird dadurch entschärft, dass die aktuellen Java-Entwicklungsplattformen eine inkrementelle Entwicklung erlauben, indem die sehr kurzen Compilerläufe im Hintergrund erfolgen und Klassen während der Entwicklung jederzeit geändert werden können. Zusätzlich existiert für mehrere Skriptsprachen eine Java-Anbindung, die eine interaktive Umgebung für den Zugriff auf alle unter Java verfügbaren Klassenbibliotheken zur Verfügung stellt.

Scripting

Unter anderem gilt dies auch für Matlab selbst, dass seit Version 5.3 eine Java-Schnittstelle beinhaltet, die in Version 6 noch einmal deutlich aufgewertet wurde. Dabei kann direkt aus Matlab-Funktionen heraus auf beliebige Java-Klassen und -Methoden zugegriffen werden, wodurch eine Vielzahl bestehender Klassenbibliotheken auch für Matlab zur Verfügung steht. Ein Beispiel bietet die Ansteuerung des von uns in Java-entwickelten Hades-Frameworks zur Simulation digitaler Schaltungen aus Matlab heraus, das sich auf diese Weise nahtlos in die interaktiven Skripte zur technischen Informatik einbetten lässt.

Während der Zugriff auf Java-Objekte aus Matlab heraus sehr gut gelöst ist, ist der umgekehrte Weg leider nicht so einfach möglich. Dies bedeutet für das Projekt, dass Matlab sich nur mühsam (auf Umwegen über Objektkommunikation oder temporäre Dateien) als serverbasiertes Hilfsmittel zur Überprüfung der Übungsaufgaben eignet.

Wegen der besonders nahtlosen Integration in das Objektkonzept von Java und einer Reihe von weiteren Vorteilen wird im Projekt außerdem die Sprache Jython [18] für das Skripting von Java-Applikationen eingesetzt werden. Dabei handelt es sich um eine in Java selbst implementierte Variante der modernen, objektorientierten Skriptsprache Python, die frei verfügbar ist und sich ideal zum interaktiven Skripting von Java-Applikationen eignet.

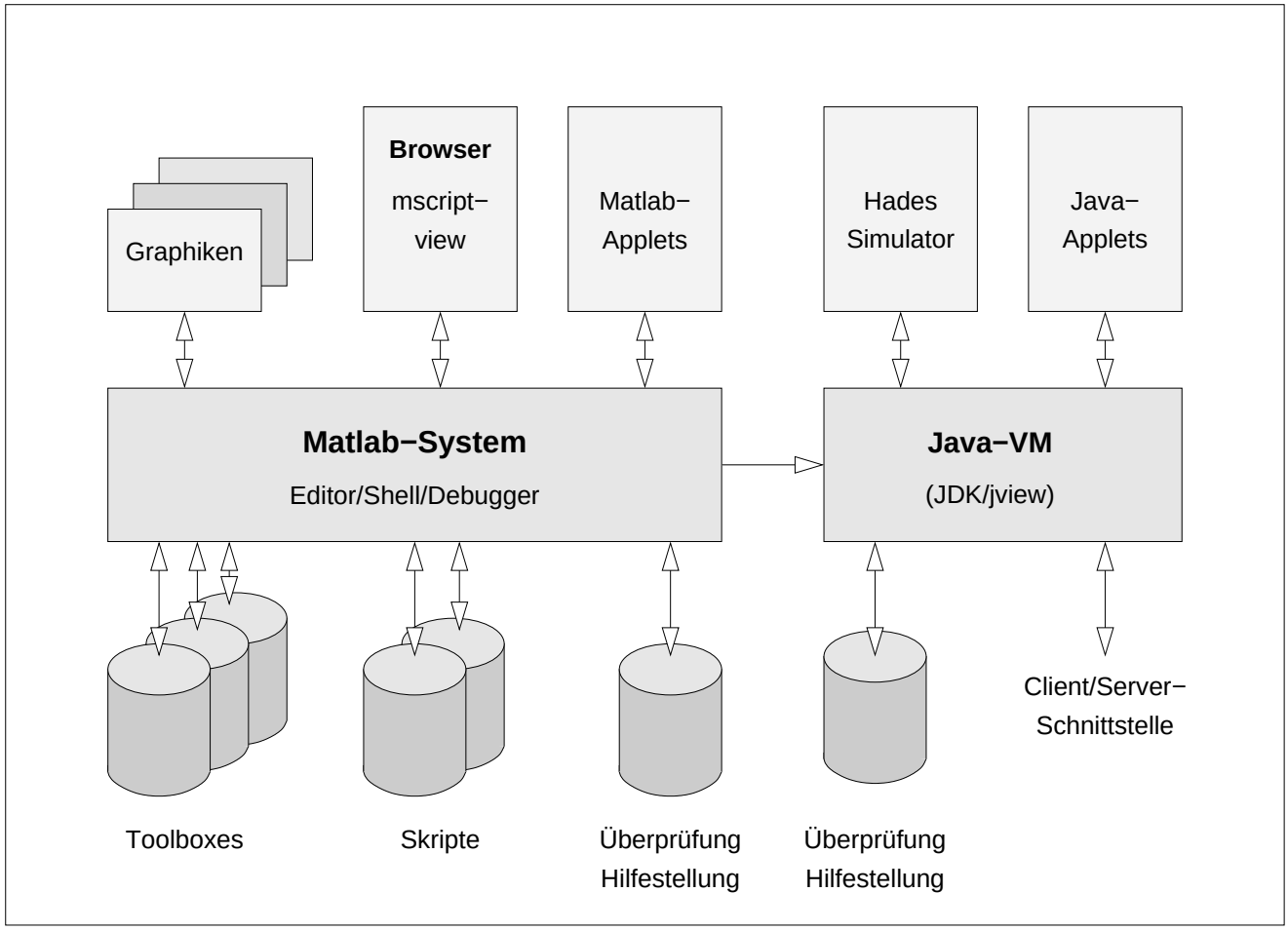


Abbildung 8: *Die Software-Architektur des geplanten Gesamtsystems mit dem Browser zur Darstellung der interaktiven Skripte sowie den Funktions- und Klassenbibliotheken mit Algorithmen zur automatischen Überprüfung der integrierten Übungsaufgaben.*

Software-Architektur

Eine Übersicht über die im Projekt gepante Software-Architektur ist in Abbildung 8 dargestellt. Die interaktiven Skripte sind als Matlab-Funktionsbibliotheken realisiert und setzen auf dem Matlab-Kernsystem auf, während die Darstellung und Interaktion über den Browser *mscriptview* erfolgt. Einige der benötigten Funktionen zur automatischen Überprüfung der integrierten Übungsaufgaben werden ebenfalls in Matlab erstellt, zum Beispiel die Auswertung logischer Ausdrücke und die Verarbeitung von symbolischen Funktionen. Andere Komponenten werden in Java erstellt und über die Matlab-Java Schnittstelle angebunden, darunter die Simulation digitaler Schaltungen im Hades-Framework. Die gesamte Software kann problemlos lokal auf den Rechnern der Studierenden installiert werden, so dass die Skripte auch ohne Netzwerkzugang offline bearbeitet werden. Trotzdem ist eine Client-Server Kommunikation jederzeit möglich, etwa um die Lösungen der fertig bearbeiteten Aufgaben an den Server der Universität zur Bewertung einzusenden.

4 Zusammenfassung

Das interaktive Skript verfolgt den Ansatz, erläuternden Text nicht nur mit statischen Medien aufzubereiten sondern direkt mit interaktiv ausführbarem Programmcode zu kombinieren. Dabei können nicht nur die Parameter sondern auch die zugrunde liegenden Funktionen selbst vom Anwender modifiziert und ergänzt werden, womit ideale Voraussetzungen für entdeckendes Lernen gegeben sind. Als Software-Plattform für unsere Vorlesungen zur technischen Informatik dient Matlab, das Prinzip lässt sich aber mit jeder Programmierumgebung umsetzen, die das interaktive Ausführen von Programmtexten oder Skripten erlaubt.

Die Integration von Übungsaufgaben in die interaktiven Skripte erlaubt nicht nur eine Kontrolle des Lernfortschritts, sondern ist eine zentrale Stütze des didaktischen Konzepts. Inhalt und Ziel des vorliegenden Projekts ist die Realisierung von Verfahren, um die Studierenden beim Bearbeiten der Übungsaufgaben zu unterstüzen.

Die in diesem Report zusammengestellten Algorithmen erlauben für viele Typen von Übungsaufgaben eine automatische und schnelle Überprüfung der Lösungen, so dass die Studierenden sofort auf falsche Ansätze oder Flüchtigkeitsfehler hingewiesen werden, und die Übungsgruppenleiter bei der Korrektur der Aufgaben entlastet werden. Als Nebenprodukt der Überprüfung ist es möglich, bei falschen Lösungen auf die Fehlerursache zurückzuschließen und daraus gezielt Hilfestellungen abzuleiten, ohne jedoch die Lösungen selbst vorwegzunehmen.

Da die im Projekt entwickelten Algorithmen und Verfahren eine Vielzahl von möglichen Übungsaufgaben abdecken und die Software ausreichend flexibel ausgelegt ist, werden sich die Ergebnisse auch zur Unterstützung von Übungen und Praktika in anderen Fachgebieten einsetzen lassen.

Literatur

[1] R. E. Bryant, D. R. O'Hallaron, *Computer Systems, A Programmer's Perspective*, Prentice Hall, 2003, ISBN 0-13-034074-X

[2] U. Bentlage (Hrsg.), *E-Learning: Märkte, Geschäftsmodelle, Perspektiven*, Verl. Bertelsmann-Stiftung, 2002, ISBN 3-89204-547-7

[3] U. Dittler (Hrsg.), *E-Learning, Einsatzkonzept und Erfolgsfaktoren des Lernens mit interaktiven Medien* (2. Auflage), Oldeenbourg Verlag, 2003, ISBN 3-486-27398-1

[4] Fachbereich Informatik der Universität Hamburg, *Studienführer Informatik 2002/2003*, www.informatik.uni-hamburg.de/

[5] R. Hartenstein, *Standort Deutschland — Wozu noch Mikro-Chips?*, IT Press Verlag, 1994, ISBN 3-9298-1400-5

[6] K. v. d. Heide, *Vorlesung Technische Informatik 1*, Universität Hamburg, FB Informatik, WS2002, tams-www.informatik.uni-hamburg.de/lehre/ws2002/t1Vor/

[7] K. v. d. Heide, *Vorlesung Technische Informatik 2*, Universität Hamburg, FB Informatik, SS2003, tams-www.informatik.uni-hamburg.de/lehre/ss2003/vorlesungen/T2/

[8] K. v. d. Heide, M. Grove, *Übungsaufgaben und Musterlösungen zur Vorlesung Technische Informatik*, Universität Hamburg, FB Informatik, SS2003, tams-www.informatik.uni-hamburg.de/onlineDoc/

[9] K. v. d. Heide, M. Grove, N. Hendrich, B. Wolfinger, *Praktikum Technische Informatik 1-4*, Universität Hamburg, FB Informatik, tams-www.informatik.uni-hamburg.de/onlineDoc/

[10] N. Hendrich, *Hades Tutorial*, tams-www.informatik.uni-hamburg.de/applets/hades/archive/tutorial.pdf

[11] N. Hendrich, *A Java-based framework for simulation and teaching*, Proc. 3rd European Workshop on Microelectronics Education, EWME-2000, 285–288, Aix en Provence, 2000

[12] N. Hendrich, *Automatic checking of students' designs using built-in selftest methods*, Proc. 3rd European Workshop on Microelectronics Education, EWME-2002, Baiona, 2002

[13] J. L. Hennessy, D. A. Patterson, *Computer organization and design: the hardware/software interface*, Morgan Kaufmann, 1998, ISBN 1-558-60491-X

[14] J. Hugunin, *Python and Java — The Best of Both Worlds*, Proc. 6th Internation Python Conference, San Jose, 1997,

[15] D. Jansen (Hrsg.), *Handbuch der Electronic Design Automation* Hanser, 2001 ISBN 3-446-21288-4

[16] *Jython Environment for Students*, Georgia Institute of Technology, 2002 http://coweb.cc.gatech.edu/cs1315/814

[17] W. Jutzi, *Digitalschaltungen, eine Einführung*, Springer, 1995 ISBN 3-540-593312-4

[18] Jython project homepage, www.jython.org

[19] J.W. Eaton and others, *GNU Octave Project*, www.octave.org

[20] S. Pedroni & N. Rappin, *Jython Essentials*, O'Reilly & Associates, Inc., 2002, ISBN 0-596-00247-5

[21] U. Karrenberg, *An interactive multimedia introduction to signal processing*, Springer 2002, ISBN 3-540-43509-3

[22] The MathWorks, Inc., *Matlab Version 5 User's Guide*, 1997 ISBN 0-13-272550-9

[23] The MathWorks, Inc., *Matlab Version 6 User's Guide*, 2002

[24] W. Schiffmann, R. Schmitz, *Technische Informatik 1*, Springer Verlag, 2001, ISBN 3-540-42170-X

[25] W. Schiffmann, R. Schmitz, *Technische Informatik 2*, Springer Verlag, 1999, ISBN 3-540-

[26] W. Schiffmann, R. Schmitz, *Übungsbuch zur Technischen Informatik 1 und 2* (2. Auflage), Springer Verlag, 2001, ISBN 3-540-42171-8

[27] R. W. Schmidt, *Java Network Launching Protocol & API Specification*, JSR-56, Sun Microsystems, Inc., 2001,

[28] R. Schulmeister, *Grundlagen hypermedialer Lernsysteme: Theorie – Didaktik – Design*, Oldenbourg, 1997, ISBN 3-486-24419-1

[29] R. Schulmeister, *Virtuelle Universität — virtuelles Lernen*, Oldenbourg, 2001, ISBN 3-486-25742-0

[30] B. Simon, *E-Learning an Hochschulen: Gestaltungsräume und Erfolgsfaktoren von Wissensmedien*, Lohmar, 2001 ISBN 3-89012-886-6

[31] Sun Microsystems, Inc., *Java WebStart Developer's Guide*, java.sun.com/products/javawebstart/

[32] A. S. Tanenbaum, *Structured Computer Organization, 4th. Edition*, Prentice Hall, 1999 ISBN 0-13-020435-8

[33] imc information multimedia communication AG, *Clix 4 Campus* Lernplattform, http://www.im-c.de/homepage/clix_campus.htm

[34] World wide web consortium, *Extensible Markup Language (XML)*, http://www.w3.org/XML/