

Das interaktive Skript

*Automatische Überprüfung und Hilfestellung
zu Vorlesungs–begleitenden Übungen*

Software–Dokumentation

HTML–Skript und Tools

Norman Hendrich

Universität Hamburg

Fachbereich Informatik



Universität Hamburg



Inhaltsverzeichnis

1	Einführung	1
1.1	Rahmen des Projekts: Das interaktive Lehrbuch	1
1.2	Kurzbeschreibung des Projekts	2
1.3	Klassifikation der Übungsaufgaben	3
1.4	Gliederung dieses Berichts	4
2	Software für das interaktive Skript	5
2.1	Interaktive Skripte als HTML	5
2.2	Varianten	9
3	Tools	11
3.1	Installation und Konfiguration	11
3.1.1	Systemanforderungen	11
3.1.2	Download	12
3.1.3	Dateistruktur	12
3.2	Applet-Security	14
3.3	Die Matlab- und Jython-Applets	17
3.3.1	MatlabApplet	17
3.3.2	MCCApplet	17
3.3.3	JythonApplet	18
3.3.4	HTML-Einbettung	19
3.4	Matlab-Console	20
3.5	Jython-Console	22
3.6	Matlab Component Runtime	24
3.7	T1-Shell	26
3.8	T1-Server	28
3.9	Erzeugen der Funktionsliste für t1shell und t1server	30
3.10	MScript2HtmlConverter	32
3.11	TeXtoPNGConverter	34
3.12	Simulator T1-Hades	38
3.13	PRIMA-Simulator	40
3.14	Yield-Demonstration	42
	Literaturverzeichnis	45

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/forschung/interaktives-skript/>

1 Einführung

Die vorliegende Software-Dokumentation beschreibt die im Rahmen des Projekts „Automatische Überprüfung und Hilfestellung zu Vorlesungs-begleitenden Übungen“ des Hamburger Sonderprogramms E-Learning (ELCH) entwickelten Programme und Werkzeuge für das Matlab-gestützte interaktive Skript. Die parallel entwickelten Java-Applets zur Überprüfung von Übungsaufgaben werden in einem separaten Report vorgestellt [14].

Anders als die bisherigen Projektberichte dient die vorliegende Dokumentation nicht nur zur Beschreibung der Konzepte und Architekturen, sondern gleichzeitig als Tutorial und als Spezifikation der Programme. Die Beschreibung der Installation und Bedienung der verschiedenen Softwarekomponenten wendet sich an alle Anwender des interaktiven HTML-Skripts.

Zielgruppe

Darüberhinaus sind natürlich alle Lehrkräfte und Software-Entwickler angesprochen, die am Einsatz von Matlab in Web-basierten Umgebungen oder E-Learning Plattformen interessiert sind. Die Beschreibungen zur Content-Erstellung und Erweiterung der Skripte setzen Grundkenntnisse in Matlab, Java und HTML voraus. Auch die Erweiterung der bestehenden Software ist problemlos möglich, erfordert aber fundierte Matlab- und Java-Kenntnisse.

Als Einführung fasst der folgende Abschnitt 1.1 noch einmal kurz die wesentlichen Aspekte eines *interaktiven Lehrbuchs* zusammen, während Abschnitt 1.2 die Einbettung von Übungsaufgaben motiviert. In Abschnitt 1.3 wird noch einmal die in der ersten Projektphase erarbeitete Klassifikation der Übungsaufgaben wiederholt.

Einführung

Die Gliederung des vorliegenden Projektberichts wird in Abschnitt 1.4 erläutert. Für eine ausführliche Diskussion der mit den interaktiven Lehrbüchern verfolgten Konzepte sei auf den ersten Projektbericht [11] verwiesen.

Als Ergänzung der Beschreibungen wird in den folgenden Kapiteln bei Bedarf auf kurze Programmbeispiele und Skripte zurückgegriffen, um die zugrundeliegenden Konzepte zu erläutern. Dabei werden auch die Schnittstellen und Parameter der verschiedenen Funktionen bzw. Programme vorgestellt, so dass der Bericht gleichzeitig als Softwaredokumentation genutzt werden kann. Die entsprechenden Abschnitte dienen zur Vertiefung und können beim Lesen ohne weiteres übersprungen werden.

Vertiefung

1.1 Rahmen des Projekts: Das interaktive Lehrbuch

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 jederzeit selbst veränderbare oder hinzugefügte Anwendungsfälle. Dadurch wird das Lehrbuch erweiterbar und kann 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.

Konzept

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 Rechnergestützten Systemen beherrschen lässt. Durch die harmonische Integration von klassischem Lehrtext in ein zugrunde liegendes, universelles 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.

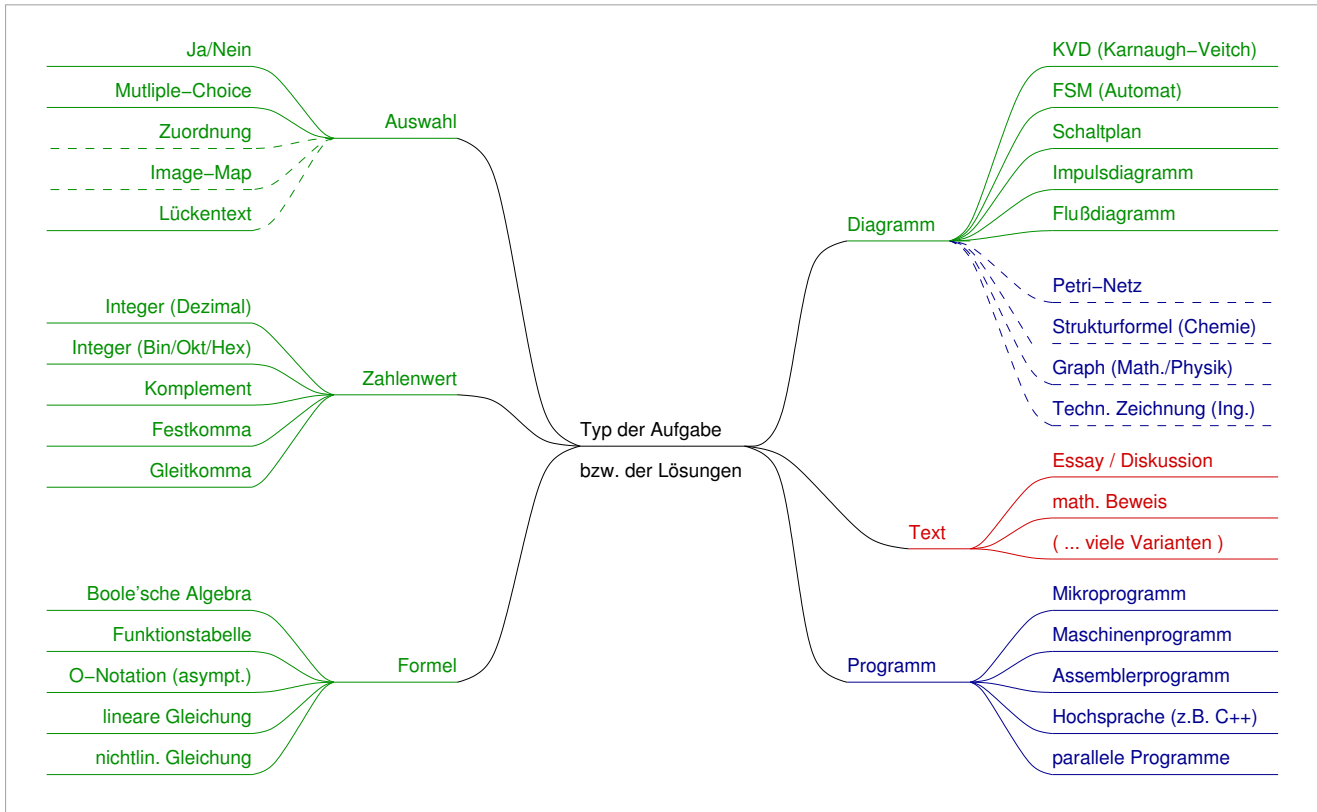


Abbildung 1: Klassifikation der Übungsaufgaben zur technischen Informatik. Die sechs Hauptklassen dürften auch auf andere mathematisch naturwissenschaftliche Fächer übertragbar sein. Für fast alle Kategorien bis auf freie Texte ist eine automatische Überprüfung möglich.

1.3 Klassifikation der Übungsaufgaben

An dieser Stelle ist es notwendig, die in der ersten Projektphase erarbeitete Klassifikation der Übungsaufgaben zur technischen Informatik noch einmal zu wiederholen, da die späteren Kapitel häufig auf diese Klassifikation zurückgreifen. Die Auswertung mehrerer Vorlesungsskripte und klassischer Lehrbücher ergab die in Abbildung 1 dargestellte Einteilung in sechs große Klassen von Aufgabentypen. Diese Klassen von Aufgaben dürften sich auch auf die meisten anderen technisch-naturwissenschaftlichen Fächer übertragen lassen, allerdings eventuell mit anderer Gewichtung und Häufigkeit der einzelnen Aufgabentypen.

Wie bereits im ersten Projektbericht erläutert wurde, ist es beim derzeitigen Stand der Technik zur Texterkennung und Sprachverarbeitung nicht einmal ansatzweise möglich, von den Studierenden eingesandte freie Texte sinnvoll auszuwerten [11]. Im Rahmen des Projekts wird diese Kategorie deshalb von vornherein ausgeklammert; derartige Aufgaben müssen wie bisher von den Übungsgruppenleitern von Hand korrigiert werden. Statt dessen ist geplant, die Textaufgaben zumindest soweit möglich durch gleichwertige Fragestellungen zu ersetzen, die der automatischen Überprüfung besser zugänglich sind.

1.4 Gliederung dieses Berichts

Kapitel 2 Das folgende Kapitel 2 erläutert noch einmal die für das Projekt gewählte Software-Architektur. Als eigentliche interaktive Softwareumgebung kommen dabei sowohl Matlab als auch die Kombination von Jython und Java zum Einsatz.

Kapitel 3 Kapitel 3 beschreibt die verschiedenen Werkzeuge zur Entwicklung und zum Deployment der HTML-Version des interaktiven Matlab-Skripts. Dies betrifft sowohl die Umsetzung der Matlab-Dateien nach HTML mit dem *MScript2HtmlConverter*, als auch das Erzeugen der vorcompilierten Matlab-Programme *t1shell* und *t1server* für das lizenzfreie Skript.

Ab Abschnitt 3.11 werden weitere im Rahmen des Projekts erstellte Java-Programme vorgestellt, insbesondere die speziell an die Vorlesung T1 angepasste Version unseres Hades-Frameworks zur digitalen Simulation.

Der Bericht schließt mit einem Literaturverzeichnis.

2 Software für das interaktive Skript

Die Grundlage für das Konzept der interaktiven Skripte ist eine Softwareplattform, die ein interaktives Ausführen von im Skript selbst eingebettetem Programmcode gestattet. Die von uns gewählte Softwarearchitektur basiert auf einer Kombination der beiden Plattformen *Matlab* und *Java*. Da gegenüber dem letzten Projektstand zwei wichtige Erweiterungen implementiert bzw. erprobt werden konnten, sollen diese in den nächsten Abschnitten vorgestellt werden.

Im Rahmen des Projekts wurden mehrere Varianten untersucht, von denen sich die folgenden vier Kombinationen als besonders geeignet herausgestellt haben:

- Browser *mscriptview*. Zum Lesen der Skripte ist eine Matlab-Installation mit Lizenz erforderlich, aber die Content-Erstellung ist besonders einfach. Für Details zu diesem Ansatz sei auf den ersten Projektbericht [11] verwiesen. *mscriptview*
- HTML-Browser mit Applet-Schnittstelle zu Matlab. Texte und passive Elemente wie Graphiken sind uneingeschränkt nutzbar, aber für das Ausführen der aktiven Elemente ist eine Matlab-Installation mit Lizenz erforderlich. Diese Architektur wird in Abschnitt 3.3.1 und 3.4 vorgestellt. *HTML mit MatlabApplets*
- HTML-Browser mit Applet-Schnittstelle zu vorcompilierten externen Applikationen. In Abschnitt 3.3.2 wird erläutert, wie sich auch unsere Matlab-Funktionen auf diese Weise nutzen lassen. *HTML mit MCCApplets*
- HTML-Browser mit Applet-Schnittstelle zu Jython als Skriptsprache an Stelle von Matlab. Die zugehörige JythonConsole wird in Abschnitt 3.5 vorgestellt. Die Software ist frei verfügbar, aber bisher sind nur wenig Skripte umgesetzt. *HTML mit JythonApplets*

Diese Aufteilung ist in Abbildung 2 noch einmal veranschaulicht. Natürlich können die verschiedenen Ansätze auch kombiniert werden; so setzen die Skripte zur Vorlesung T1 weiterhin auf aktive Matlab-Elemente, während die meisten Algorithmen zur Überprüfung von Übungsaufgaben in Java/Jython implementiert wurden.

Die im zweiten Projektbericht erläuterte Variante mit einem selbstentwickelten, um zusätzliche Syntaxelemente erweiterten XHTML-Browser wurde übrigens nicht weiterverfolgt, da sich die Implementierung als zu aufwendig erwies [12]. Das Problem der aufwendigen Content-Erstellung in HTML konnte durch den in Abschnitt 3.10 beschriebenen Konverter gelöst werden.

2.1 Interaktive Skripte als HTML

Die im obigen Abschnitt skizzierte Softwarearchitektur basiert auf Matlab und dem zugehörigen *mscriptview*-Browser zum Anzeigen der Skripte. Gerade für Veranstaltungen im Hauptstudium hat sich diese Kombination sehr gut bewährt, zumal fast alle Studierenden wegen des hohen Nutzwerts eine eigene Matlab-Lizenz erwerben. Dagegen kann dies beim Einsatz im Grundstudium, aber auch für nicht technisch-mathematisch ausgerichtete Themengebiete, kaum vorausgesetzt werden. Damit ergibt sich ein Problem, denn bei Einsatz von *mscriptview* ist selbst zum passiven Durchblättern und Lesen der Skripte bereits eine vollständige Matlab-Installation notwendig.

Es stellt sich daher die Frage, ob und welche alternativen Plattformen zur Darstellung der Skripte genutzt werden können. Besonders attraktiv erscheint dabei die Umsetzung der interaktiven Skripte mittels HTML-Dateien. Dies ermöglicht die Darstellung der Skripte

HTML

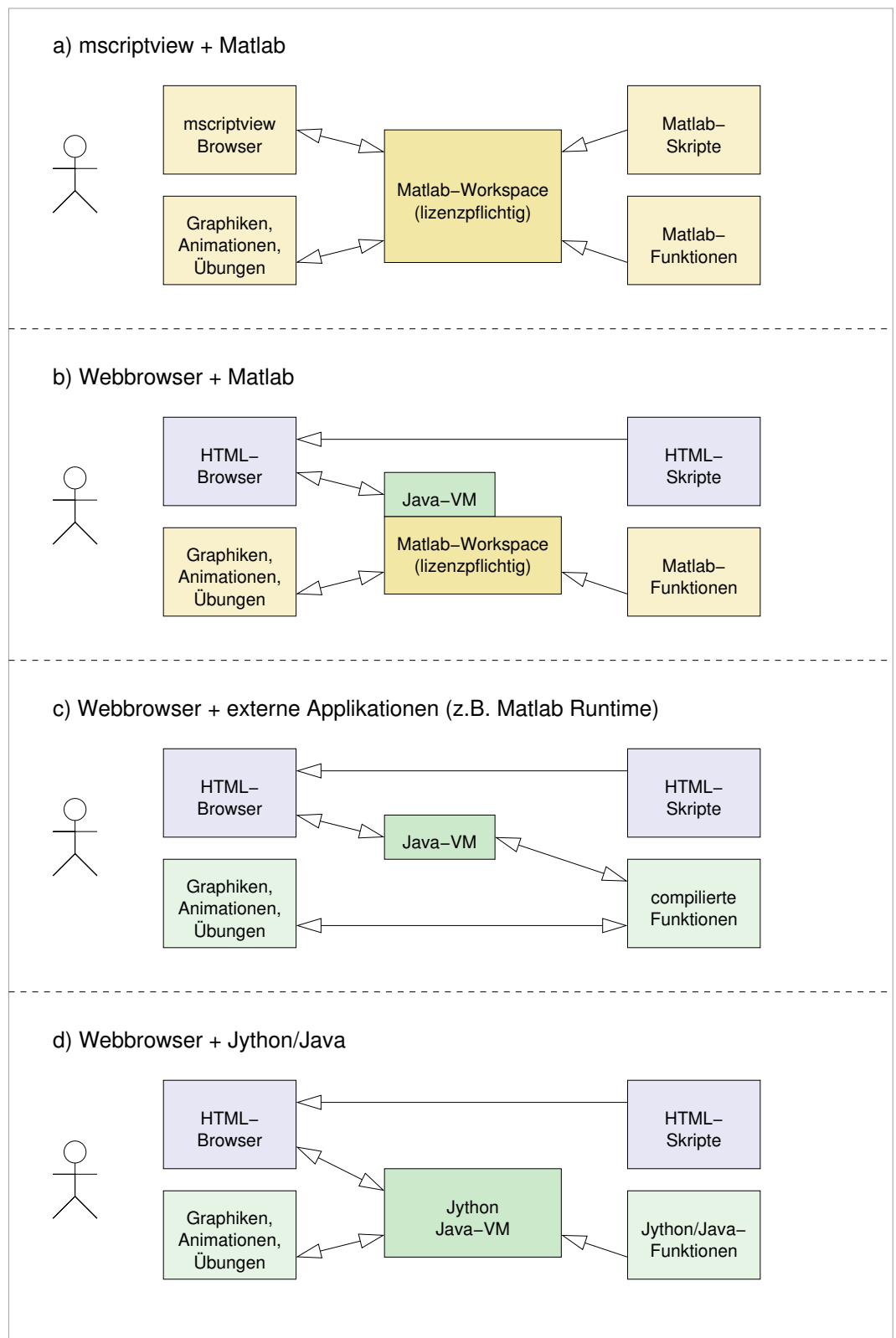


Abbildung 2: Die Abbildung zeigt vier mögliche Varianten zur Realisierung interaktiver Skripte mit den Plattformen Matlab sowie Java/Jython: a) Verwendung von Matlab und Darstellung mit mscriptview-Browser, b) Einbindung von Matlab-Funktionen in HTML via Applets, c) Aufruf vorcompilierter (Matlab-) Applikationen aus HTML via Applets, d) Einbindung von Jython-Applets. Natürlich können die verschiedenen Varianten auch kombiniert werden.

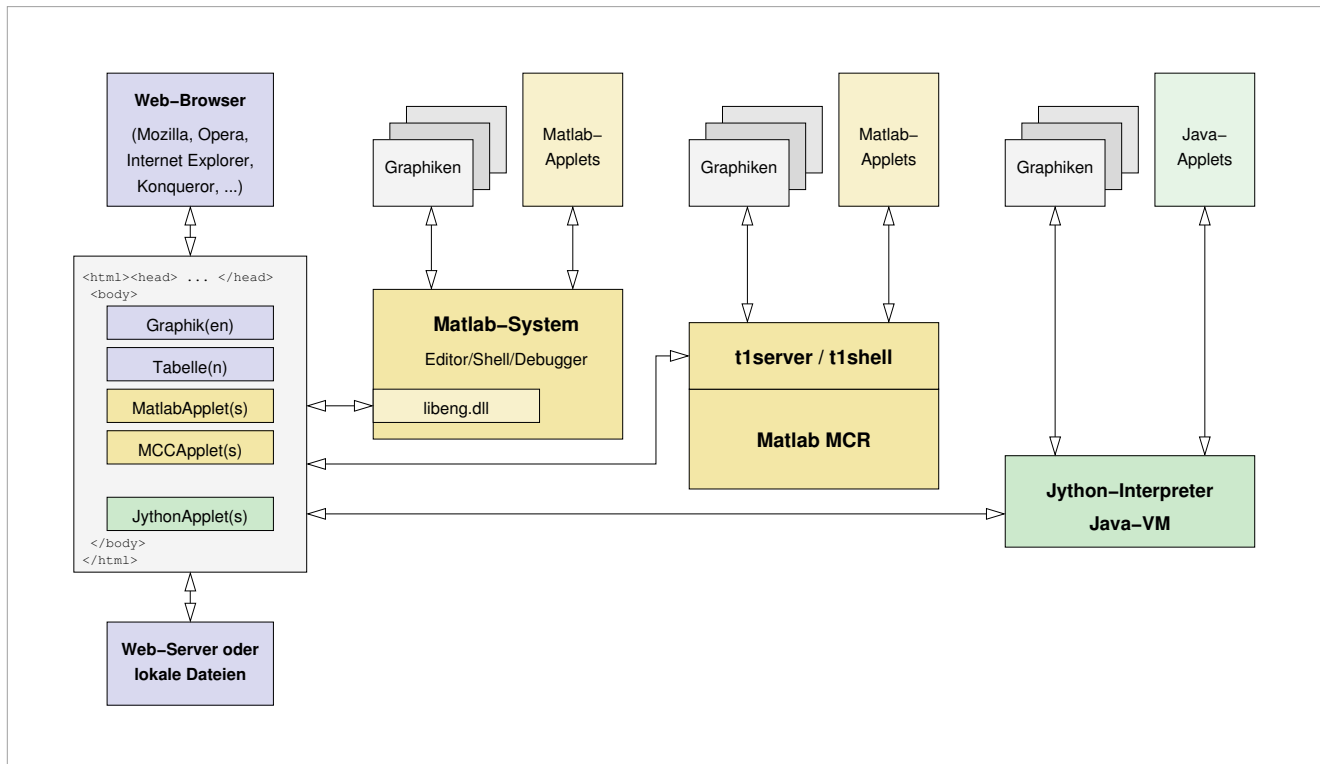


Abbildung 3: Software-Architektur der HTML-basierten interaktiven Skripte. Die Inhalte werden als HTML-formatierte Webseiten aufbereitet und können mit jedem gewöhnlichen Browser angezeigt werden. Die aktiven Elemente sind als einfache Java-Applets realisiert und kommunizieren mit dem als externe Applikation laufenden Matlab-System bzw. dem Jython-Interpreter.

mit einem gewöhnlichen Web-Browser wie dem Internet Explorer oder Mozilla, wobei alle Zusatzfunktionen der Browser zur Verfügung stehen und der Anwender seine gewohnte Umgebung vorfindet. Auch die Integration in bestehende Web-Anwendungen oder E-Learning Frameworks wird deutlich erleichtert. Mehrere Vorteile der Verwendung eines HTML-Browsers sind offensichtlich:

Vorteile...

- Die interaktiven Skripte können auch sauber angezeigt werden, wenn Matlab nicht zur Verfügung steht — dann allerdings passiv ohne die Interaktionsmöglichkeiten.
- Bereitstellen der gewohnten Benutzeroberfläche inklusive der Voreinstellungen für Schriftarten, Schriftgrößen, Farben.
- Zugriff auf alle HTML- bzw. XHTML-Objekttypen wie eingebettete Abbildungen, Tabellen, Formulare und interaktive Objekte (Applets).
- Hyperlinks auf externe Webseiten und Ressourcen, Zugriff auf Suchmaschinen, komfortable Bookmarkverwaltung.
- Aufruf externer Hilfsapplikationen wie etwa Postscript- oder PDF-Viewern.
- Möglichkeit zum Ausdrucken der Skripte inklusive aller Formatierungen und eingebetteten Graphiken.
- Integration in bestehende HTML-Plattformen oder E-Learning-Frameworks.
- Mögliche Integration in Content-Management Systeme und Nutzen bestehender HTML- oder XML-Editoren.
- Zugriff auf Sicherheitsfunktionen wie verschlüsselte Datenübertragung, Verwalten von Benutzerpasswörtern, Überprüfung digital signierter Inhalte.

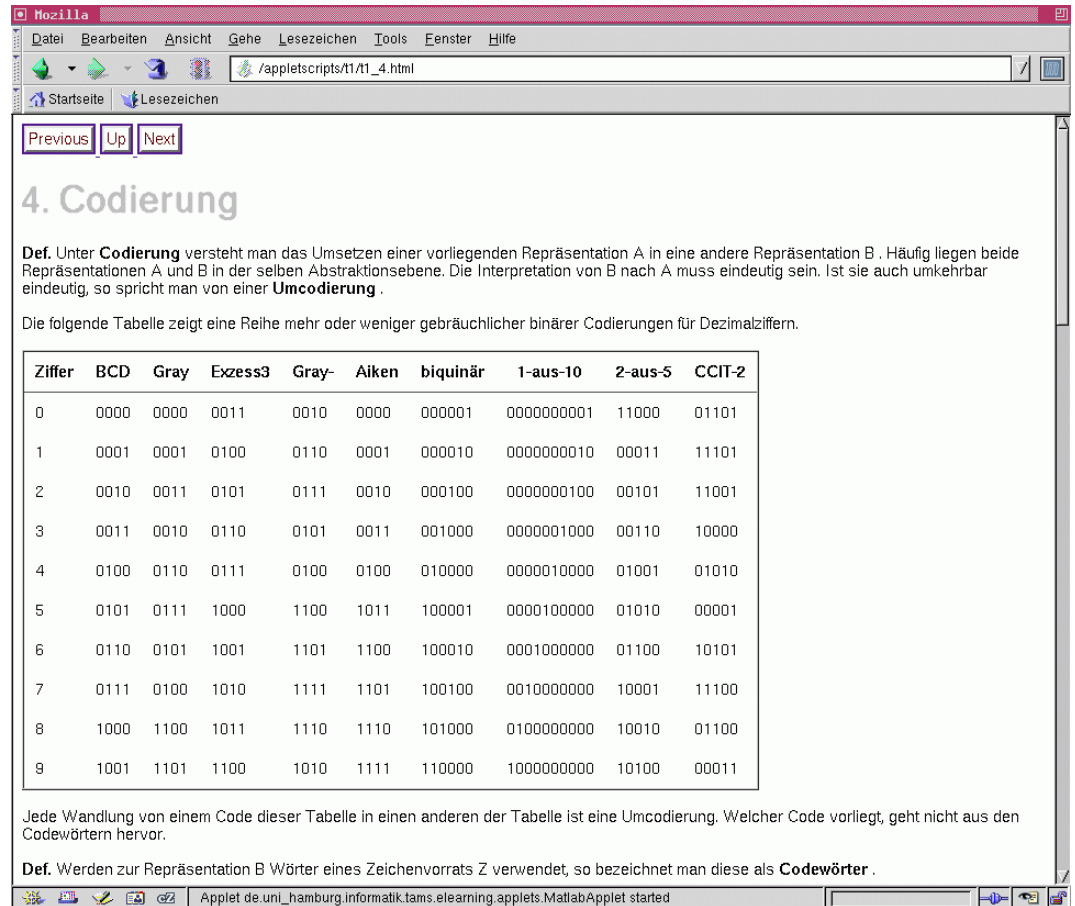


Abbildung 4: Darstellung des interaktiven HTML-Skripts in einem Webbrowser. Die Seite wurde automatisch mit *MScript2HtmlConverter* aus dem original Matlab-Skript konvertiert. Dabei werden alle Textformatierungen übernommen und Tabellen vollständig nach HTML übersetzt, während Formeln als Applets oder Abbildungen eingebettet werden.

... und Nachteile

Auf der anderen Seite sind auch einige Nachteile zu verzeichnen. An erster Stelle steht dabei die teilweise deutlich aufwendigere Content-Erstellung im HTML-Format. Die Lösung besteht in der automatischen Umsetzung unserer vorhandenen Skripte vom *mscriptview*-Format nach HTML. Der zugehörige Konverter wird in Abschnitt 3.10 vorgestellt.

Ein besonders ärgerliches Problem bereitet die immer noch wenig standardkonforme Darstellung einiger HTML-Konstrukte durch die verschiedenen Browser. Zum Beispiel unterstützen alle getesteten Browser (u.a. Mozilla, Internet Explorer, Opera, Konqueror) nur (disjunkte) Teilmengen der in HTML 4 definierten mathematischen Symbole und Operatoren. Dies führt dazu, dass einige der Formelzeichen als Bilder bzw. Applets eingebunden werden müssen, obwohl sie eigentlich direkt in HTML definiert sind. Abhängig von Betriebssystemversionen und Druckertreibern können sich auch beim Ausdrucken Probleme ergeben.

Schließlich stehen in HTML nicht alle Funktionen von Matlab bzw. *mscriptview* zur Verfügung. Zum Beispiel stellt der *mscriptview*-Browser eine spezielle hierarchische Suchfunktion bereit, die in gängigen HTML-Browsern nicht verfügbar ist. (Natürlich könnte man die entsprechenden HTML-Seiten auch durch eine gewöhnliche Suchmaschine wie Google klassifizieren lassen oder auf serverbasierte Tools ausweichen).

Da alle aktuellen Browser zumindest HTML 4.0 bzw. XHTML inklusive CSS unterstützen, ist die Integration passiver Abbildungen, Tabellen, und der üblichen HTML-Formulare überhaupt kein Problem. Spezielle Anforderungen an die Formatierung inklusive Blocksatz

The screenshot shows a Mozilla browser window displaying a web page titled "9.3 Beschreibung von Schaltwerken". The page content includes text about describing circuits using transition and output function tables, and a state diagram. A MatLabConsole window is overlaid on the page, showing code for a finite state machine simulation. The state diagram shows four states: A, B, C, and D. State D is highlighted in green. Transitions are labeled with '1' and '2'. A red arrow points from state D to state A.

Abbildung 5: Auch die vorhandenen Simulatoren mit eigener Benutzeroberfläche sind in die HTML-Seiten integriert. Die Abbildung zeigt den Simulator für Schaltwerke (FSMs, Finite State Machines).

und pixelgenauer Anordnung lassen sich mit Style-Sheets realisieren. Damit bleibt für die konkrete Umsetzung nur noch die Frage übrig, wie sich die für das Konzept der interaktiven Skripte notwendigen Programmtexte integrieren lassen.

Obwohl das Interaktionskonzept zunächst nur auf dem einfachen Anklicken der eingebetteten Codeblöcke basiert, erscheint eine Realisierung mit JavaScript oder etwa als Flash-Plugin kaum möglich. Aus Gründen der Portabilität bleibt damit nur die Umsetzung mit Java-Applets, die direkt in die HTML-Seiten eingebettet werden. Die Applets übernehmen in diesem Konzept eine doppelte Funktion: erstens die Darstellung der interaktiven Elemente in den HTML-Seiten selbst und zweitens die Interaktion mit dem Benutzer und die Ausführung der ausgewählten und angeklickten Programme.

Interaktionskonzept

2.2 Varianten

Wie die Übersicht in Abbildung 2 auf Seite 6 zeigt, sind dabei wiederum mehrere Varianten möglich. In der Variante b) dient das Applet nur als Vermittler und übergibt den ausgewählten Code an die darunterliegende Plattform zur Ausführung. Beim Einsatz von Matlab bedeutet dies, dass weiterhin eine Matlab-Installation und Lizenz notwendig ist. Auch in Variante c) dienen die Applets nur als Vermittler und rufen externe Applikationen auf. Wie in Abschnitt 3.3.2 näher erläutert wird, lassen sich auf diese Weise insbesondere auch vorcompilierte Matlab-Funktionen aufrufen, für die keine Matlab-Lizenz erforderlich

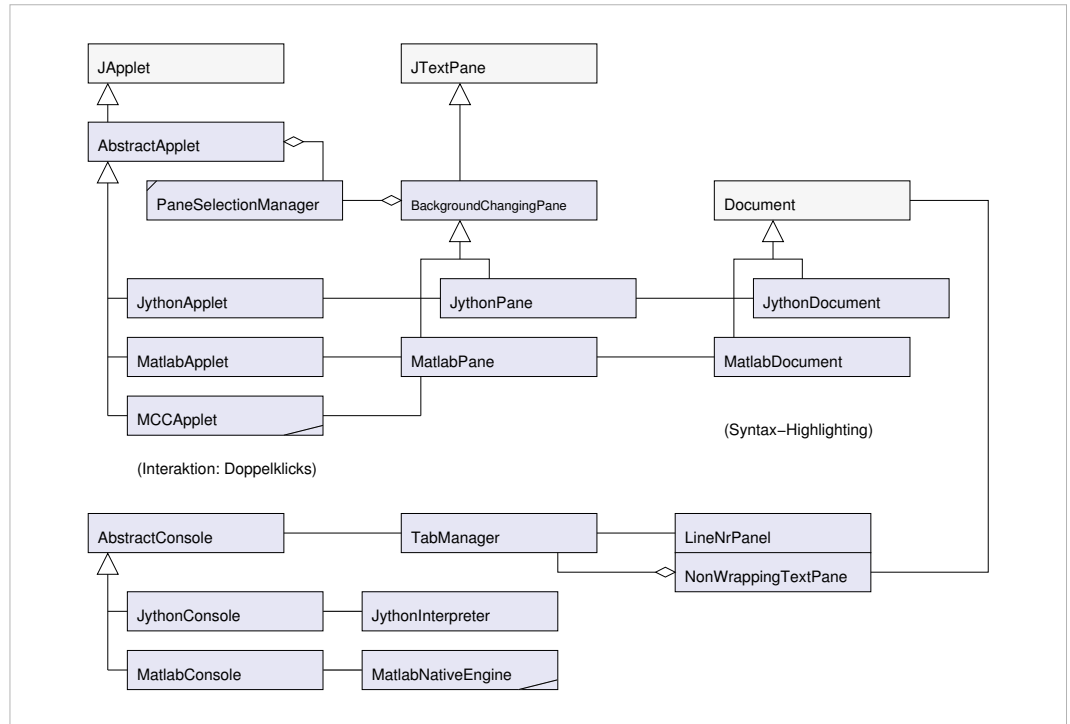


Abbildung 6: Die Klassenhierarchie für das interaktive HTML-Skript mit den verschiedenen Applets und den zugehörigen Consolen. Das optionale Syntax-Highlighting in den Applets und Editorfeldern der Consolen erfolgt über die Syntaxdefinitionen in den Unterklassen *MatlabDocument* bzw. *JythonDocument*.

ist. In Variante d) schließlich übernimmt die Java-Umgebung über den Jython-Interpreter die Ausführung der ausgewählten interaktiven Elemente, siehe Abschnitt 3.5.

Wegen der einheitlichen Softwarearchitektur des interaktiven HTML-Skripts ergeben sich viele Gemeinsamkeiten zwischen den drei Varianten. Entsprechend können in unserer Implementierung viele Grundfunktionen in gemeinsame Basisklassen ausgelagert werden, die dann von den einzelnen Applets und Serverkomponenten geeignet verfeinert werden. Die derzeit realisierte Klassenhierarchie ist in Abbildung 6 dargestellt.

Applets Auf Seite der Applets dient *AbstractApplet* als gemeinsame Basisklasse, die unter anderem mit Hilfe von *PaneSelectionManager* das jeweils aktive Applet auswählt und mit Hilfe der *BackgroundChangingPane* optisch hervorhebt. Während *MatlabApplet* zur Interaktion mit der *MatlabConsole* dient, die ihrerseits über JNI und die C-Schnittstelle auf ein installiertes Matlab zugreift, führt *MCCApplets* Netzwerkzugriffe auf den zugehörigen *tlserver*-Webserver aus (siehe Seite 28). Entsprechend kommuniziert *JythonApplet* mit der *JythonConsole*. Die Darstellung der Skriptcodes in den Applet-Blöcken übernehmen die zugehörigen Klassen *MatlabPane* und *JythonPane*, wobei das Syntax-Highlighting über die entsprechenden *Document*-Typen *MatlabDocument* und *JythonDocument* realisiert wird. Alle Applets realisieren auf einen Doppelklick und alternativ auch auf Shift-Klick, da die aktuellen Java-Versionen nur sehr schnelle Doppelklicks als solche erkennen.

Consolen Die Verwaltung der verschiedenen Editorfenster übernimmt *AbstractConsole* in Verbindung mit dem *TabManager*, wobei sich insbesondere die Realisierung der Undo/Redo-Funktionen als sehr fehlerträchtig herausstellte. Die zugehörigen *Document*-Typen für die *MatlabConsole* und *JythonConsole* können direkt von den entsprechenden Applet-Klassen übernommen werden.

3 Tools

In diesem Kapitel werden die verschiedenen Software-Werkzeuge vorgestellt, die im Rahmen des Projekts für das interaktive HTML-Skript realisiert wurden. Der erste Abschnitt 3.1 beschreibt den Download und die Installation der Programme. Um die Anwender vor den Gefahren von unkontrolliert aus dem Internet heruntergeladenen Programmen zu schützen, verfügen Java-Applets über ausgefeilte Sicherheitsmechanismen. Die Konfiguration der einzelnen Sicherheitseinstellungen für herkömmliche und für signierte Applets wird in Abschnitt 3.2 erläutert.

*Installation,
Einstellungen*

Mit Abschnitt 3.3 beginnt die Beschreibung der einzelnen Applets und der zugehörigen Consolen- bzw. Serverprogramme. Dabei werden zunächst die Konzepte und die verschiedenen Varianten der Applets vorgestellt, danach die zugehörigen *Consolen* als Entwicklungsumgebungen und die vorcompilierten Matlab-Programme *tIshell* und *tIserver*. Diese Abschnitte sind wegen der Hinweise zur Installation und zu möglichen Fehlerquellen nicht nur für die Entwickler sondern auch für die Anwender (Schüler, Studierende) des interaktiven Skripts von Interesse.

Bedienung

Die Werkzeuge zur Erstellung des interaktiven Skripts werden ab Abschnitt 3.9 beschrieben und mit den verschiedenen Optionen vorgestellt. Diese Abschnitte setzen Grundkenntnisse der Matlab- und Java-Programmierung voraus und wenden sich an die Entwickler der interaktiven Skripte.

*Content-
Erstellung*

Abschnitt 3.12 beschreibt die speziell an die Vorlesung T1 angepasste Version *T1-Hades* unseres Simulations-Frameworks *Hades*, während in Abschnitt 3.13 der Simulator für unseren Demonstrationsrechner *PRIMA* vorgestellt wird.

*T1-Hades,
Prima*

3.1 Installation und Konfiguration

Dieser Abschnitt enthält verschiedene Hinweise und Tips zur Installation und Konfiguration der Matlab-Toolboxen und Java-Klassen für das interaktive Skript. Die in den Beispielen gewählten Einstellungen sind dabei nur als Vorschläge zu betrachten, insbesondere die Pfade der Verzeichnisse können natürlich frei gewählt und an eigene Vorlieben bzw. Vorgaben des jeweiligen Systems angepasst werden. Bitte beachten Sie auch die Hinweise in den *README*-Dateien zu den einzelnen Softwarekomponenten, die mögliche Änderungen und Verbesserungen gegenüber den hier beschriebenen Einstellungen dokumentieren.

3.1.1 Systemanforderungen

Voraussetzung für den Einsatz der Matlab-basierten Skripte ist eine lokal installierte oder über das Netzwerk zugreifbare Version von Matlab. Alle Funktionen wurden unter Matlab 7.x und Matlab 6.5.1 getestet, die meisten Skripte sollten aber auch unter älteren Versionen inklusive der Matlab 5.3 Student Edition funktionieren. Dies gilt insbesondere auch für den *mscriptview*-Browser zum Anzeigen der originalen interaktiven Matlab-Skripte.

Matlab

Die HTML-Varianten der interaktiven Skripte benötigen einen aktuellen Webbrowser mit aktivierter Java-Unterstützung. Da die für die Applets verwendeten Swing-Komponenten in älteren Java-Versionen nicht zur Verfügung stehen, wird ein JDK/JRE 1.4 oder höher benötigt. Die Skripte wurden unter Windows XP und Linux unter anderem mit folgenden Webbrowsern getestet: Microsoft Internet Explorer, Mozilla, Firefox, Opera, Konqueror.

HTML und Java

Die lizenzfreie Version des interaktiven Skripts basiert auf der Matlab Component Runtime (Seite 24), die derzeit für Windows XP, Linux (2.4.x und 2.6.x) und diverse Unix-Varianten zur Verfügung steht. Ältere Versionen von Windows werden von Mathworks nicht

*Matlab Component
Runtime*

unterstützt und von uns nicht getestet. Wir bieten die zugehörigen Programme *t1shell* und *t1server* derzeit nur für Windows XP und Linux an, eine MacOS X Version wäre bei ausreichendem Interesse aber möglich.

Jython Für den Einsatz der auf Jython basierenden Skripte eignet sich jeder kompatible Webbrowser mit Java JDK/JRE 1.4.2 oder höher. Die eventuell erforderlichen Sicherheitseinstellungen der Java-Umgebung werden in Abschnitt 3.2 beschrieben.

3.1.2 Download

Die folgenden Komponenten des interaktiven Skripts stehen auf unserem Webserver unter der URL <http://tams-www.informatik.uni-hamburg.de> zum Download zur Verfügung. Bitte folgen Sie von der Startseite aus den Hyperlinks zur jeweiligen Veranstaltung, z.B. [/lehre/ws2004/vorlesungen/t1](#) für die Vorlesung T1 im WS'2004:

- Webseiten mit eingebetteten Applets, unter anderem für die Überprüfung der Übungsaufgaben oder den PRIMA-Simulator.
- Archivdateien mit den Matlab-Toolboxen; dies sind *msv.zip* mit dem *mscriptview*-Browser und Hilfsfunktionen sowie *t1.zip*, *t2.zip*, etc. mit den Funktionen für die Vorlesungen Technische Informatik 1, Technische Informatik 2, usw.
- Versionen der Matlab Component Runtime für Windows XP und Linux
- Die vorcompilierten Matlab-Programme *t1shell* und *t1server*
- Archivdatei *t1skript.zip* mit den HTML-Dateien, den Applets (*elearningApp.jar*) und Consolen (*elearningAppCon.jar*) sowie weiteren benötigten JAR-Archiven für das interaktive Skript.
- Archivdateien *mmkh-tools.jar* mit den Werkzeugen zur Content-Erstellung.

Wir planen, eine CD-ROM mit allen benötigten Tools und einem Installer für Windows XP und Linux anzubieten, eventuell auch als vorbereitetes ISO-Image zum Download. Bitte beachten Sie die Hinweise auf unserem Webserver zur Verfügbarkeit.

3.1.3 Dateistruktur

Die folgende Abbildung 7 zeigt die Dateistruktur für das interaktive HTML-Skript. Alle Dateien liegen unterhalb eines gemeinsamen, frei wählbaren Verzeichnisses.

Skript Das `html`-Unterverzeichnis enthält dabei alle Dateien zum Anzeigen des HTML-Skripts im Webbrowser. Die eigentlichen HTML-Dateien liegen mitsamt der zugehörigen Abbildungen und Icons innerhalb von drei (in der Abbildung nicht gezeigten) Unterverzeichnissen von `html/skript`, die den drei Varianten des interaktiven Skripts (passiv, Matlab-Applets, MCCApplets) entsprechen. Die JAR-Archive für die Applets sind im `html/lib`-Verzeichnis versammelt.

Das `jars`-Verzeichnis enthält die verschiedenen JAR-Archive für die Java-Programme und Werkzeuge und das `matlab`-Verzeichnis die verschiedenen Matlab-Toolboxen. Die vorcompilierten Matlab-Programme und die Matlab Component Runtime liegen im `MCR`-Verzeichnis. Natürlich ist es alternativ auch möglich, die Matlab-Toolboxen in eines der bestehenden Toolbox-Verzeichnisse zu integrieren.

Übungen Das `uebung`-Verzeichnis versammelt die Aufgabenblätter, Musterlösungen und Applets für die automatische Überprüfung der Übungsaufgaben.

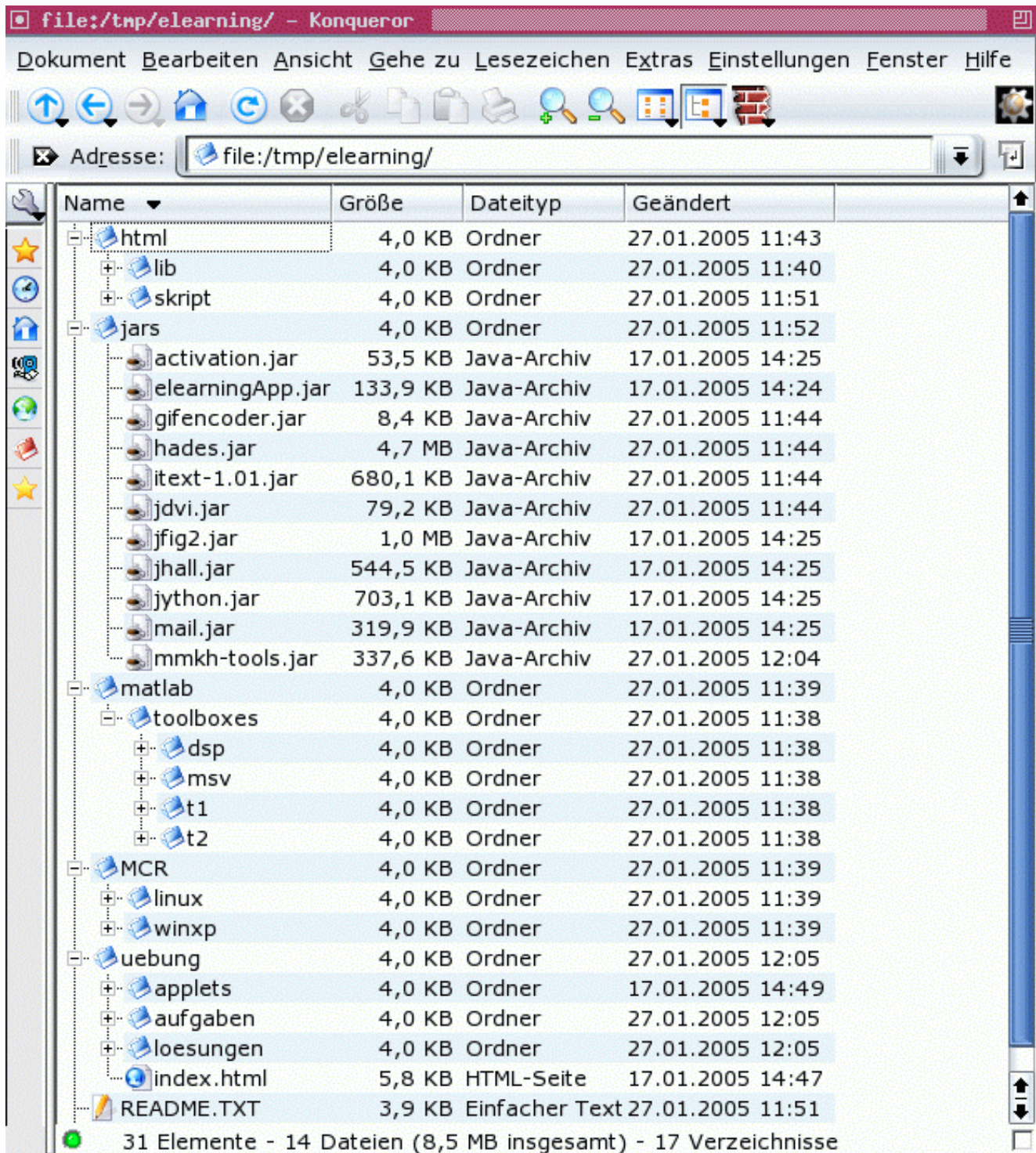


Abbildung 7: Vorschlag und Beispiel der Dateistruktur für das interaktive Skript.

Für das Ausführen der diversen Java-Applets sind keine weiteren CLASSPATH-Einstellungen notwendig, da die zugehörigen JAR-Archive auf dem Webserver bzw. nach Auspacken der *t1skript.zip* Datei bereits an den richtigen Pfaden liegen. Die eventuell erforderliche Konfiguration der Sicherheitseinstellungen wird in Abschnitt 3.2 beschrieben.

Die Schritte zur Installation der vorcompilierten Matlab-Applikationen werden in Abschnitt 3.7 für *t1shell* und in Abschnitt 3.8 für *t1server* beschrieben. Da alle benötigten Funktionen fest in die Programme hineincompiliert sind, ist keine weitere Konfiguration erforderlich.

Matlab-Path Beim Einsatz von *mscriptview* oder der *MatlabConsole* (Abschnitt 3.4) müssen die benötigten Matlab-Funktionen aus *msv.zip* und *t1.zip* in den Matlab-Pfad eingetragen werden. Dies gelingt am einfachsten interaktiv im Matlab-Workspace, kann aber auch durch direktes Editieren der `pathdef.m`-Datei erfolgen. Für eine saubere Trennung der Funktionen ist es empfehlenswert, separate Verzeichnisse `msv/` und `t1/` (usw.) für die einzelnen Tool-boxen anzulegen. (Natürlich müssen diese Funktionen auch für das Neu-Compilieren der vorcompilierten Programme wie *t1shell* im Matlab-Pfad enthalten sein. Zusätzlich werden die Funktionen zum Erstellen der Funktionsliste benötigt, siehe Abschnitt 3.9.)

CLASSPATH Für Java-Applikationen, die über Java Webstart oder unseren Installer gestartet werden, also zum Beispiel unseren *T1-Hades-Simulator* oder die *JythonConsole*, ist keine weitere Konfiguration erforderlich, da der Installer alle notwendigen Schritte übernimmt.

Beim manuellen Download einzelner JAR-Archive und auch für die Content-Erstellung ist dagegen die Konfiguration der Java CLASSPATH-Einstellungen notwendig. Sofern keine Versionskonflikte mit bereits installierten Java-Archiven zu befürchten sind, können alle JAR-Archive durchaus in das globale *Java Extension*-Verzeichnis kopiert werden und stehen danach ohne weitere Einstellungen allen Java-Programmen zur Verfügung. Das entsprechende Verzeichnis findet sich unterhalb des Java JRE-Verzeichnisses (zum Beispiel `/opt/jdk1.4.2/jre` oder `c:\jdk1.5.0\jre`) unter dem Pfad `$JRE/lib/ext/`.

Falls die globale Installation in das Extension-Verzeichnis nicht möglich ist, müssen alle benötigten JAR-Archive einzeln in die CLASSPATH Variable eingetragen werden. Dies kann natürlich durch ein geeignetes Skript automatisiert werden, zum Beispiel:

```
set CLASSPATH=%CLASSAPTH%;c:\temp\elearningAppCon.jar;\
c:\temp\jython.jar;c:\temp\jfig2.jar;c:\temp\jhelp.jar;\
c:\temp\activation.jar;c:\temp\mail.jar;c:\temp\mmkh-tools.jar;
```

3.2 Applet-Security

Sandbox Um die Anwender vor den Gefahren von unkontrolliert aus dem Internet heruntergeladenen Programmen zu schützen, verfügen Java-Applets über ausgefeilte Sicherheitsmechanismen. Das zugrundeliegende *Sandbox*-Konzept basiert auf der Idee, unbekanntes Applets zunächst alle sicherheitsrelevanten Funktionen wie Datei- oder Netzwerkzugriffe zu verbieten. In der Grundeinstellung dürfen Applets nur Netzwerkzugriffe auf denselben Server ausführen, von dem sie ursprünglich geladen wurden. Jedes von einem Applet aus geöffnete Dialogfenster wird mit einem entsprechenden Hinweis (*Applet-Window*) gekennzeichnet, um den Benutzer zu warnen. Andere Netzwerkzugriffe, Zugriffe auf lokale Dateien, oder selbst der Versuch zum Drucken werden mit einer *SecurityException* abgebrochen.

Ohne die Möglichkeit, Dateien zu lesen, zu speichern oder zu drucken, ist der Nutzen von Applets natürlich stark eingeschränkt. Deshalb gibt es in Java zwei unterschiedliche Wege, bestimmten Applets nachträglich erweiterte Rechte zuzuweisen — und zwar durch die Verwendung digital signierter Applets oder alternativ durch benutzerdefinierte Einträge in einer Konfigurationsdatei (`.java.policy`). Bitte beachten sie zusätzlich zu den folgenden Hinweisen auch eventuelle Release-Notes oder Updates zu der auf ihrem Rechner verwendeten Java-Version.

Signierte Applets Digital signierte Applets sind die erste und sowohl für die einzelnen Anwender als auch die Anbieter einfachste Variante, um Applets mit erweiterten Zugriffsrechten auszustatten. Dazu werden die JAR-Archive mit den benötigten Java-Klassen vom Anbieter mit einer digitalen Signatur versehen, die vom Anwender überprüft werden kann. Wenn die Überprüfung erfolgreich verläuft, ist die Urheberschaft und die Unversehrtheit der Java-Klassen

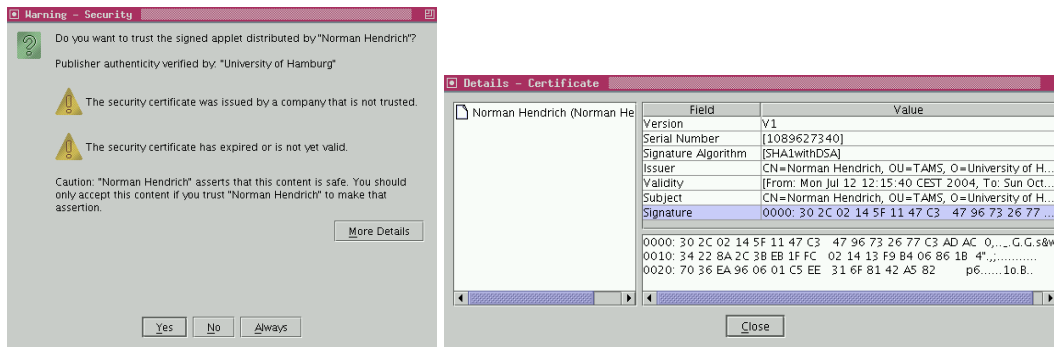


Abbildung 8: Nachfrage der Java Umgebung vor dem Ausführen eines signierten Applets. Nach Anklicken des Details-Buttons werden die einzelnen Informationen des verwendeten Zertifikats angezeigt.

garantiert. In diesem Fall fragt die Java Umgebung beim Anwender nach, ob das entsprechende Applet ausgeführt werden soll, siehe Abbildung 8 (links). Sobald der Anwender zustimmt, verfügt das Applet über alle Rechte einer lokal gestarteten Applikation.

Auf der anderen Seite kann jeder beliebige Anbieter seine eigenen Applets mit sogenannten *self-signed* Zertifikaten digital signieren. Deshalb ist entweder die genaue Überprüfung der detaillierten Signatur durch den Anwender (Abbildung 8 rechts) oder eine zusätzliche Zertifizierung durch eine unabhängige Zertifizierungsstelle erforderlich. Leider sind Zertifikate von glaubwürdigen kommerziellen Unternehmen wie Verisign oder TrustCenter so teuer, dass ein Einsatz in unserem Projekt nicht in Frage kommt. Wir setzen daher soweit möglich auf unsignierte Applets mit entsprechend reduzierten Zugriffsrechten (kein Drucken, kein Dateizugriff, keine Benutzerkonfiguration) oder auf *self-signed* Zertifikate mit dem oben in Abbildung 8 gezeigten Schlüssel. Diese Applets können durch einfaches Bestätigen der Sicherheitsabfrage problemlos (mit erweiterten Rechten) gestartet werden.

*self-signed
Zertifikate*

Alternativ erlaubt die Java Virtual Machine dem Anwender, einzelnen Applets von vorher genau definierten Webservern gezielt bestimmte Zugriffsrechte zuzuteilen. Dies geschieht durch Einträge in der Java-Konfigurationsdatei `.java.policy` im Home-Verzeichnis eines Benutzers. Auf diese Weise können verschiedene Benutzer auf demselben Rechner durchaus völlig unterschiedliche Berechtigungen für Applets erteilen.

*Individuelle
Konfiguration*

Für eine vollständige Liste der möglichen *Permission*-Einträge sei auf die Dokumentation der Java Umgebung verwiesen. Zum Einsatz der Java-Applets für das interaktive Skript sind zumindest die folgenden Einträge notwendig, die den von unserem Webserver (*tams-www*) geladenen Applets alle Netzwerkzugriffe, das Drucken und den Zugriff auf alle *Properties* erlauben. Außerdem werden Dateizugriffe auf ein Verzeichnis `.elch` im Homeverzeichnis des angemeldeten Benutzers zugelassen, die für die Konfigurationsdaten (Name, email-Adressen, etc.) des jeweiligen Benutzers benötigt werden. Die notwendigen Einträge sehen so aus:

```
/* Interaktives Skript T1 vom Webserver */
grant codeBase "http://tams-www.informatik.uni-hamburg.de/lehre/ws2004/-" {
  permission java.net.SocketPermission "*", "connect,accept,listen,resolve";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.util.PropertyPermission "*", "read,write";
  permission java.io.FilePermission "${user.home}${/}.elch${/}-", "read,write";
};
```

Sofern die unsignierten Applets von einem anderen Webserver, einem anderen Pfad (etwa `/lehre/ws2005`) oder aus einer lokalen Datei (wie `file://c:/temp/lehre/ws2005`)

geladen werden, sind entsprechende zusätzliche Einträge in der `.java.policy` Datei notwendig. Aufgrund der Funktionsweise mit weitgehendem Einsatz von Java-Reflection benötigt auch der Jython-Interpreter zusätzliche Berechtigungen beim Start via Applets. Die Webseiten unter `/applets/jython/` auf unserem Webserver enthalten zusätzliche Informationen.

Weitere Hinweise zur `.java.policy`-Datei:

- Überschreiben Sie auf keinen Fall bestehende Einträge in Ihrer `.java.policy` Datei. Fügen Sie statt dessen einfach die obigen Zeilen am Ende der Datei an.
- Die `.java.policy` Datei wird nur bei Bedarf angelegt. Falls in Ihrem Homeverzeichnis (z.B. `c:\Dokumente und Einstellungen\Benutzername`) bisher noch keine `.java.policy` Datei existiert, können Sie diese Datei mit einem Editor neu anlegen.
- Dateien, deren Name mit einem Punkt anfängt (wie `.java.policy`), können im Windows-Explorer und mit einigen Windows-Programmen nur schlecht oder gar nicht bearbeitet werden. Verwenden Sie notfalls die Eingabeaufforderung (`cmd.exe`) und DOS-Befehle (wie `rename` oder `copy`), um die Datei anlegen oder bearbeiten zu können.
- Sie müssen Ihren Browser neu starten, damit die Java-VM neue Einträge in der `.java.policy` Datei auch übernimmt.
- Der offizielle Weg zum Bearbeiten der `.java.policy` Datei ist das Java Programm `policytool`. Aufgrund fehlender Hilfe und einer sehr eigenwilligen GUI dürfte ein normaler Texteditor aber die bessere Wahl darstellen.

3.3 Die Matlab- und Jython-Applets

In allen Varianten unseres interaktiven HTML-Skripts übernehmen Java-Applets die Darstellung der aktiven Skriptblöcke und die Behandlung der Benutzereingaben. Den drei Varianten der darunterliegenden Softwareplattform (Matlab, vorcompiliertes Matlab, Jython) entsprechen die drei Applet-Klassen *MatlabApplet*, *MCCApplet* und *JythonApplet*. Für alle Applets gelten die folgenden gemeinsamen Eigenschaften:

Konzept

- Das User-Interface der Applets wird mit Swing realisiert, wobei die Anzeige des Matlab- bzw. Jython-Skriptcodes über selbstentwickelte Unterklassen von *JTextPane* erfolgt (siehe Abbildung 6 auf Seite 10). Für diese Klassen ist ein Java JDK/JRE 1.4.2 oder neuer erforderlich.
- Der vom Applet angezeigte Skriptcode kann jederzeit vom Anwender direkt im Applet selbst editiert werden; die üblichen Operationen wie Ausschneiden, Kopieren, Einfügen und Undo sind vorhanden.
- Nach einem einfachen Klick auf das Applet wird dieses durch Umschalten der Hintergrundfarbe als aktiv markiert.
- Ein Doppelklick auf das Applet sorgt für die Ausführung des aktuell im Applet angezeigten Skriptcodes. Da das JDK/JRE nur sehr schnelle Doppelklicks erkennt, kann alternativ auch die Kombination Shift+Klick benutzt werden.
- Die Größe der Applets muss im HTML-Code über die *width* und *height* Parameter fest vorgegeben werden. Der *MScript2HtmlConverter* berechnet die erforderliche Größe aus dem vom Applet anzuzeigenden Matlab- bzw. Jython-Skriptcode.
- Der im Applet anzuzeigende Skriptcode wird zeilenweise an das Applet übergeben, wobei der HTML-Parameter *nlines* die Anzahl der Zeilen enthält. Jede der Zeilen wird über einen eigenen Parameter *line1*, *line2* usw. übergeben.

Sonderzeichen (außer Buchstaben und Ziffern) werden nach der Java-Konvention für Unicode-Zeichen in der Hexadezimal-Schreibweise als `\uxxxx` kodiert, etwa `\u0020` für das Leerzeichen oder `\u000a` für Carriage-Return. Bei manueller Eingabe der Applet-Parameter ist diese Umkodierung nur für HTML-Sonderzeichen sowie führende Leerzeichen und Tabulatoren notwendig, zumal in Jython die Gruppierung über Einrückung erfolgt.

- Die JAR-Datei *elarningApp.jar* enthält die benötigten Java-Klassen für alle Applets.
- Siehe Seite 14 für die Auswahl zwischen signierten und unsignierten Applets.

3.3.1 MatlabApplet

Ein *MatlabApplet* dient zum Anzeigen, Editieren und Ausführen von Matlab-Code in Verbindung mit einer lokal installierten Version von Matlab. Nach einem Doppelklick überträgt das Applet den aktuellen Skriptcode über einen Netzwerkzugriff (Port 8211) an die *MatlabConsole*, siehe Seite 20. Dabei wird ein proprietäres Protokoll auf Basis von TCP verwendet [37]. Beim Aufruf der *MatlabApplets* von einem Webserver müssen die Berechtigungen zum Ausführen der entsprechenden TCP-Operationen (Nameserver-Lookup, Zugriff auf Port 8211) erteilt werden, siehe Abschnitt 3.2. Falls die *MatlabConsole* noch nicht gestartet wurde, fordern die Applets über ein entsprechendes Dialogfenster dazu auf.

3.3.2 MCCApplet

Ein *MCCApplet* dient zum Anzeigen, Editieren und Ausführen von Matlab-Code in Verbindung mit einem lokal gestarteten *t1server*-Prozess, siehe Seite 28. Nach einem Doppelklick

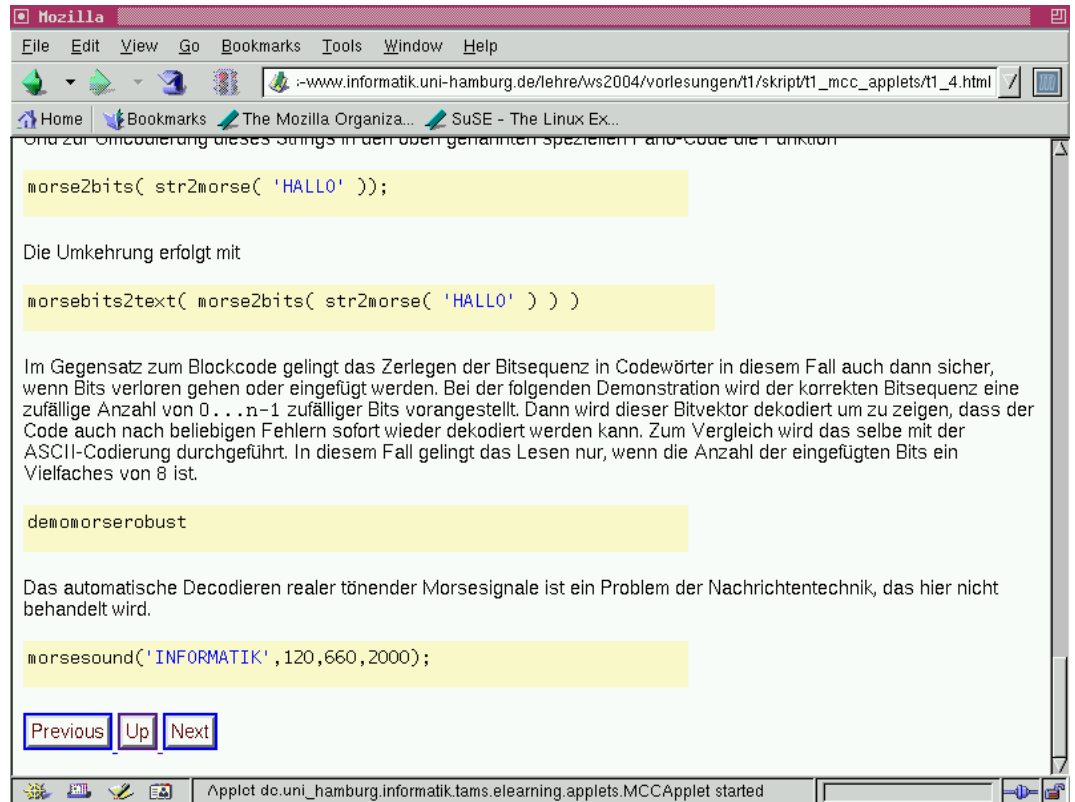


Abbildung 9: Eine HTML-Seite des interaktiven Skripts mit eingebetteten MCCApplets. Nach einem Doppelklick in einem der Applets wird der zugehörige Matlab-Code über einen http-Zugriff an den im Hintergrund laufenden t1server-Prozess übergeben und von diesem ausgeführt. Der Text in den Applets ist vom Anwender editierbar. Die Einbettung in den HTML-Code mit den benötigten Applet-Parametern wird auf der nächsten Seite beschrieben.

überträgt das Applet den aktuellen Skriptcode über einen http-Netzwerkzugriff (Port 8888) an den *t1server*. Beim Aufruf der *MatlabApplets* von einem Webserver müssen die Berechtigungen zum Ausführen der entsprechenden TCP-Operationen (Nameserver-Lookup, Zugriff auf Port 8888) erteilt werden, siehe Abschnitt 3.2. Falls der *t1server* noch nicht gestartet wurde, fordern die Applets über ein entsprechendes Dialogfenster dazu auf.

3.3.3 JythonApplet

Ein *JythonApplet* dient zum Anzeigen, Editieren und Ausführen von Jython-Code in Verbindung mit einem lokal gestarteten *JythonInterpreter*. Nach einem Doppelklick überträgt das Applet den aktuellen Skriptcode über einen Netzwerkzugriff (Port 8210) an die *JythonConsole*, siehe Seite 20. Dabei wird ein proprietäres Protokoll auf Basis von TCP verwendet [37]. Falls die *JythonConsole* noch nicht gestartet wurde, fordern die Applets über ein entsprechendes Dialogfenster dazu auf. Alternativ ist es möglich, die *JythonConsole* über Java Webstart nachträglich herunterzuladen und zu starten.

Die Variante *JythonApplet2* ist noch experimentell und wird von uns noch nicht in Skripten eingesetzt. Ein *JythonApplet2* greift direkt über Java-Funktionsaufrufe anstelle der Netzwerkfunktionen auf die *JythonConsole* zu. Den Vorteilen der besseren Performance und zusätzlichen Funktionen steht der Nachteil gegenüber, dass der *JythonInterpreter* innerhalb der Browser-Sandbox läuft.

3.3.4 HTML-Einbettung

Da wir die HTML-Dateien der interaktiven HTML-Skripte mit dem ab Seite 33 beschriebenen *MScript2HtmlConverter*-Programm automatisch erzeugen, sind nur in Ausnahmefällen manuelle Nacharbeiten erforderlich. Natürlich ist es jederzeit möglich, die HTML-Dateien mit einem geeigneten Editor zu bearbeiten, neu zu erstellen, oder einen eigenen Konverter zu schreiben.

*Applet-
Parameter*

Das folgende Codebeispiel zeigt einen Ausschnitt aus der Datei *t1_4.m* (Kapitel 4 des T1-Skripts) mit dem originalen Matlab-Funktionsaufruf der Funktion `str2morse`:

```
% Zur Umcodierung von ASCII nach Morse nehmen wir die Funktion
% {\fontname{Courier}str2morse} :
%
str2morse('HALLO')
%
```

Der zugehörige HTML-Code für ein *MCCApplet* aus der Webseite *t1_x.y.html* demonstriert die Einbettung der Applets und die Parameterübergabe des zugehörigen Skriptcodes:

```
<p>
  Zur Umcodierung von ASCII nach Morse nehmen wir die Funktion
  <tt>str2morse</tt> :
</p>
<p>
<applet
  code="uni_hh.elearning.applets.MCCApplet"
  archive="lib/elearningApp.jar" codebase="../../"
  alt="The applet support of your browser is disabled!"
  width="500" height="35"
>
  <param value="str2morse\u0028\u0027HALLO\u0027\u0029" name="line1">
  <param value="1" name="nlines">
</applet>
</p>
```

Die Hilfsklasse *NameMangler* dient zum interaktiven Umkodieren der Sonderzeichen. Die Methoden *encodeWithUnicodeEscapes* und *decodeUnicodeEscapes* können selbstverständlich auch direkt von Java-Programmen aus aufgerufen werden:

NameMangler

```
java uni_hh.elearning.util.NameMangler -encode "str2morse('HALLO')"
str2morse\u0028\u0027HALLO\u0027\u0029

java uni_hh.elearning.util.NameMangler -decode \
  "str2morse\u0028\u0027HALLO\u0027\u0029"
str2morse('HALLO')
```

3.4 Matlab-Console

Konzept Die *MatlabConsole* dient als Bindeglied zwischen den *MatlabApplets* des interaktiven HTML-Skripts und Matlab, wobei die eigentlichen Zugriffe auf Matlab über die C-basierte Schnittstelle (*libeng.dll*) erfolgen. Durch die Möglichkeit zum interaktiven Zugriff auf alle Rechenfunktionen von Matlab kann die *MatlabConsole* gleichzeitig aber auch als Alternative zum originalen Matlab Workspace für die Entwicklung und Erprobung von Matlab-Skripten und -Funktionen eingesetzt werden. In beiden Anwendungsfällen ist eine vorherige Matlab-Installation (Vollversion oder Student Version) erforderlich. Nur die in Abschnitt 3.3.2 beschriebene Variante des HTML-Skripts mit *MCCApplets* kommt ohne Matlab-Lizenz aus und benötigt lediglich die kostenfreie Matlab-Component Runtime.

User-Interface Das User-Interface der *MatlabConsole* besteht aus drei Teilen, dem Editorbereich, dem Logfenster mit den Ausgaben der Matlab-Engine, und der Eingabezeile zum interaktiven Absetzen von Matlab-Befehlen. Der Editorbereich verwaltet eine beliebige Anzahl von Texteditoren für Matlab-Code oder beliebige Textdateien. Jedes Editorfenster unterstützt neben der interaktiven Texteingabe die üblichen Grundfunktionen (Ausschneiden, Kopieren, Einfügen, Suche, usw.) und Syntax-Highlighting für Matlab-Funktionen.

Gegenüber dem direkten Aufruf von Matlab-Funktionen aus den Applets heraus bietet das Zwischenschalten der *MatlabConsole* die folgenden Vorteile:

- Beim Zugriff auf Matlab über die C-Schnittstelle (*libeng.dll*) stehen zunächst nur die eigentlichen Rechenfunktionen zur Verfügung, die übrigen GUI-Elemente wie Editor und Workspace dagegen nicht. (Nur die Windows-Version erlaubt eingeschränkten Zugriff auf den Workspace, die Unix-Versionen dagegen nicht.) Die *MatlabConsole* kann als Alternative zum Workspace dienen.
- Das Kernkonzept des interaktiven Skripts fordert, dass der Anwender Funktionsaufrufe jederzeit nach seinen eigenen Ideen abändern und ausprobieren darf. Zwar können die Funktionsaufrufe auch direkt in den *MatlabApplets* editiert werden, aber schon die fest in die Webseiten hineinkodierte Größe der Applets ist eine empfindliche Einschränkung. Die Editor-Komponenten in der *MatlabConsole* sind demgegenüber wesentlich leistungsfähiger und besser bedienbar.
- Die aufwendige Kommunikation von Java über JNI-Aufrufe und die C-Schnittstelle zu Matlab wird von den Applets auf eine Applikation verlagert. Damit entfällt auch die browser-abhängige, teilweise sehr aufwendige Konfiguration der Sicherheitseinstellungen für Java-Applets.
- Die *MatlabConsole* erlaubt den Aufruf und das Fernsteuern von Matlab-Funktionen aus Java-Programmen heraus. Insbesondere ist es möglich, Swing-basierte User-Interfaces zu erstellen, die weit über die Möglichkeiten der in Matlab enthaltenen *uicontrol*-Komponenten hinausgehen. Demgegenüber sind die bisher von Mathworks für diesen Zweck bereitgestellten Funktionen weiterhin experimentell und nicht stabil nutzbar.

Aufruf Beim Standard-Aufruf der *MatlabConsole* ohne Parameter wird das User-Interface sofort angezeigt und erlaubt das interaktive Arbeiten mit Matlab; ein Zugriff über TCP/IP ist in diesem Fall jedoch nicht möglich. Beim Aufruf der *MatlabConsole* mit dem Parameter `-server` wird dagegen zunächst nur ein Serverprozess gestartet, der auf dem voreingestellten TCP-Port 8211 (fest hineincompiliert) auf Anfragen von den *MatlabApplets* wartet. In diesem Fall wird das User-Interface erst nach der ersten Anfrage von einem Applet aus angezeigt:

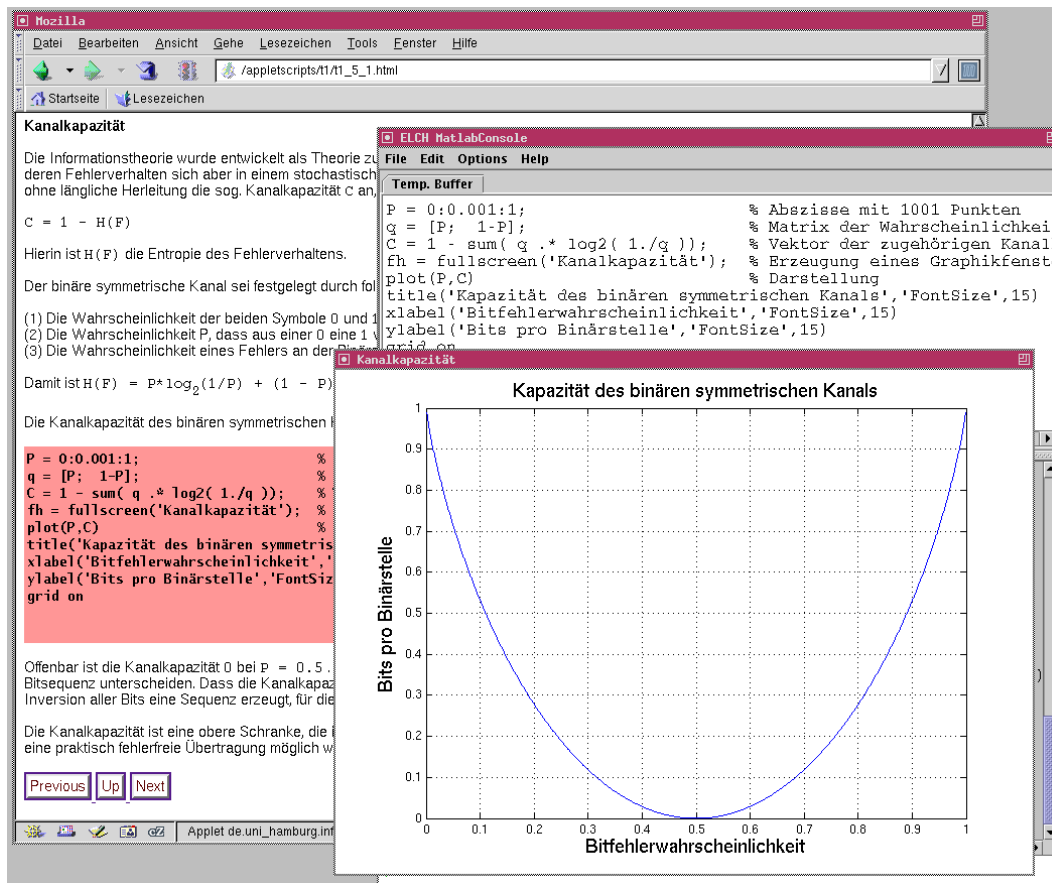


Abbildung 10: Eine HTML-Seite des interaktiven Skripts mit eingebettetem Matlab-Applet. Nach einem Doppelklick im Applet wird der zugehörige Matlab-Code über die MatlabConsole (mittleres Fenster) und die C-Schnittstelle an Matlab übergeben und dort ausgeführt. Im Beispiel berechnet Matlab die Kanalkapazität des gestörten binären Kanals und zeigt diese als Plot in einem neuen Fenster an. Der aktuell ausgewählte Matlab-Code wird in der MatlabConsole in einem eigenen Editorfeld angezeigt und kann dort nach Belieben editiert, abgespeichert und erneut ausgeführt werden.

```
java uni_hh.elearning.console.MatlabConsole
java uni_hh.elearning.console.MatlabConsole -server
```

In beiden Varianten müssen vor dem Aufruf von MatlabConsole die benötigten Einstellungen und Umgebungsvariablen gesetzt werden. Dies sind der Java CLASSPATH, am einfachsten mit einem Verweis auf das *elearningAppCon.jar*-Archiv sowie der Pfad zur *MatlabJNI*-Library (*libMatlabJNI.dll* bzw. *libMatlabJNI.so*).

Für den Zugriff auf die Matlab C-Schnittstelle sind weitere, betriebssystemabhängige Einstellungen erforderlich. Unter Windows werden die notwendigen Einstellungen bei der Matlab-Installation in die Windows-Registry eingetragen; bei nachträglichen Änderungen der Konfiguration müssen diese eventuell über zusätzliche Umgebungsvariablen berücksichtigt werden. Auf Unix-Systemen muss zumindest der Pfad zur Matlab-Bibliothek *libbeng.so* mit der C-Schnittstelle gesetzt werden:

```
setenv MATLABROOT /opt/matlab7
setenv LD_LIBRARY_PATH $MATLABROOT/bin/glnx86
```

3.5 Jython-Console

Konzept Analog zur oben beschriebenen MatlabConsole dient die *JythonConsole* als Bindeglied zwischen den *JythonApplets* des interaktiven HTML-Skripts und dem Jython-Interpreter. Dabei ist jedoch kein externes Programm notwendig, sondern der Interpreter zur Ausführung der von den Applets übergebenen Funktionen läuft in derselben Java-Umgebung wie die Jython-Console. Durch die Möglichkeit zum interaktiven Zugriff auf alle Jython-Funktionen und Bibliotheken kann die JythonConsole auch als Entwicklungsumgebung für die Entwicklung und Erprobung von Jython-Skripten und -Funktionen eingesetzt werden.

User-Interface Das User-Interface der JythonConsole entspricht ebenfalls der bereits vorgestellten Organisation von MatlabConsole mit dem Editorbereich, einem Logfenster mit den Ausgaben des Jython-Interpreters, und der Eingabezeile zum interaktiven Absetzen von Jython-Befehlen. Der Editorbereich verwaltet eine beliebige Anzahl von Texteditoren für Jython-Code oder andere Textdateien. Jedes Editorfenster unterstützt neben der interaktiven Texteingabe die üblichen Grundfunktionen (Ausschneiden, Kopieren, Einfügen, Suche, usw.) und Syntax-Highlighting für Jython-Funktionen.

Gegenüber dem direkten Aufruf von Jython-Funktionen aus den Applets heraus bietet das Zwischenschalten der JythonConsole wiederum die folgenden Vorteile:

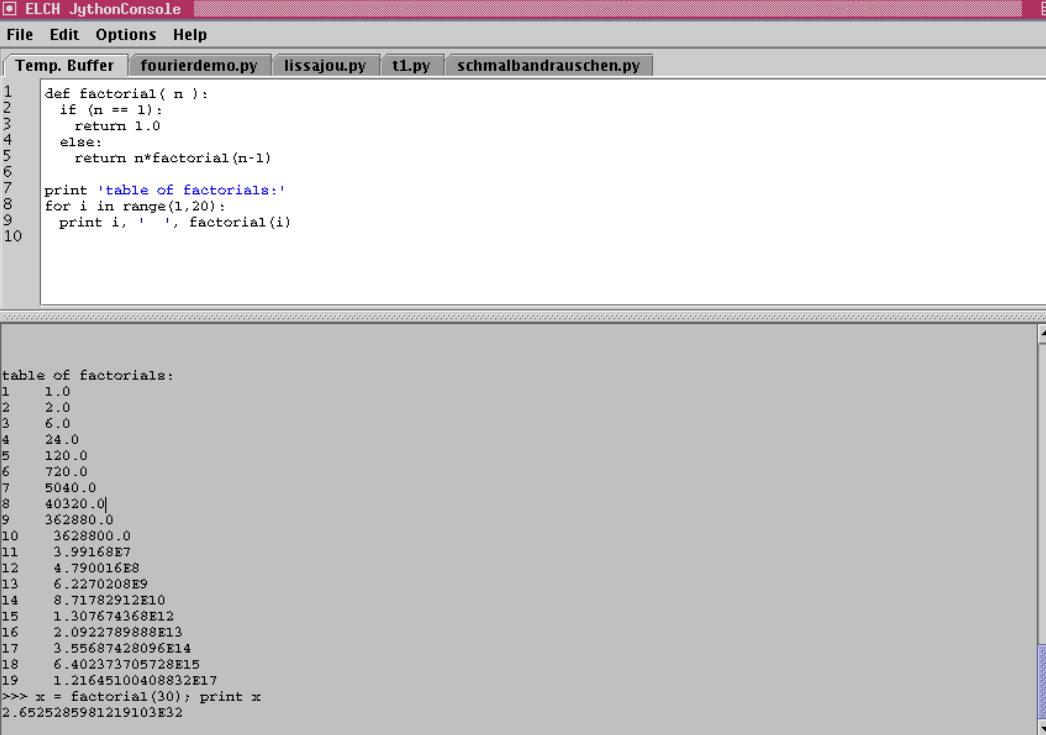
- Die JythonConsole integriert einen vollwertigen Editor mit dem Jython-Interpreter. Zusammen mit den in Jython selbst vorhandenen Debugfunktionen eignet sich die JythonConsole damit als eigenständige Entwicklungsumgebung zur Entwicklung und zum Austesten von Jython-Skripten oder -Klassen.
- Das Kernkonzept des interaktiven Skripts fordert, dass der Anwender Funktionsaufrufe jederzeit nach seinen eigenen Ideen abändern und ausprobieren darf. Zwar können die Funktionsaufrufe auch direkt in den *JythonApplets* editiert werden, aber schon die fest in die Webseiten hineinkodierte Größe der Applets ist eine empfindliche Einschränkung. Die Editor-Komponenten in der *JythonConsole* sind demgegenüber wesentlich leistungsfähiger und besser bedienbar.

Aufruf Beim Standard-Aufruf der JythonConsole ohne Parameter wird das User-Interface sofort angezeigt und erlaubt das interaktive Arbeiten mit Jython; ein gleichzeitiger Zugriff über TCP/IP ist in diesem Fall jedoch nicht vorgesehen. Beim Aufruf der JythonConsole mit dem Parameter `-server` wird dagegen zunächst nur ein Serverprozess gestartet, der auf dem voreingestellten TCP-Port 8210 (fest hineincompiliert) auf Anfragen von den Jython-Applets wartet. In diesem Fall wird das User-Interface erst nach der ersten Anfrage von einem Applet aus angezeigt:

```
java uni_hh.elearning.console.JythonConsole
java uni_hh.elearning.console.JythonConsole -server
```

In beiden Varianten müssen vor dem Aufruf von JythonConsole die benötigten Einstellungen um Umgebungsvariablen gesetzt werden. Dies sind der Java CLASSPATH, am einfachsten mit einem Verweis auf die JAR-Archive *elearningAppCon.jar* sowie *jython.jar*. Sofern zusätzlich zu Java-Klassen auch Python-Module referenziert werden sollen (optional), muss der Pfad zu den entsprechenden Python-Lib Verzeichnissen gesetzt sein. Dies kann aber auch nachträglich im Jython-Interpreter durch Aufrufe von `sys.path.insert()` bzw. `sys.path.append()` erfolgen:

```
sys.path.insert( 0, '/opt/jython/lib' )
sys.path.insert( 3, '/home/username/python/lib' )
```

The screenshot shows a window titled "ELCH JythonConsole" with a menu bar (File, Edit, Options, Help) and several open tabs: Temp. Buffer, fourierdemo.py, lissajou.py, t1.py, and schmalbandrauschen.py. The editor displays a Python function for calculating factorials and a loop to print a table of factorials from 1 to 20. The output window below shows the resulting table of factorials and the result of a manual factorial(30) call.

```
1 def factorial( n ):
2     if (n == 1):
3         return 1.0
4     else:
5         return n*factorial(n-1)
6
7 print 'table of factorials:'
8 for i in range(1,20):
9     print i, ' ', factorial(i)
10
```

```
table of factorials:
1  1.0
2  2.0
3  6.0
4  24.0
5  120.0
6  720.0
7  5040.0
8  40320.0
9  362880.0
10 3628800.0
11 3.99168E7
12 4.790016E8
13 6.2270208E9
14 8.71782912E10
15 1.307674368E12
16 2.0922789888E13
17 3.55687428096E14
18 6.402373705728E15
19 1.21645100408832E17
>>> x = factorial(30); print x
2.6525285981219103E32
```

Abbildung 11: Die Jython-Console integriert einen Editor mit einem Jython-Interpreter. Sie kann als eigenständige Applikation oder alternativ auch von Applets aus gestartet werden. Jedes Editorfenster unterstützt die üblichen Editorfunktionen inklusive Syntax-Highlighting. Bereits während der Eingabe können die Skripte oder Klassen jederzeit im Interpreter ausgetestet werden. Im Beispiel wurden die Editorfenster verkleinert, um mehr Platz für die Ausgaben des Jython-Interpreters im Logfenster zu gewinnen.

3.6 Matlab Component Runtime

Sowohl beim Einsatz unseres *mscriptview*-Browsers als auch beim Arbeiten mit der in Abschnitt 2.1 beschriebenen HTML-Version des interaktiven Skripts mit integrierten *Matlab-Applets* ist eine Matlab-Installation und -Lizenz notwendig. Natürlich kann trotz des sehr guten Preis-Leistungs-Verhältnisses der *Student Version* nicht vorausgesetzt werden, dass alle Studierenden sich Matlab privat beschaffen. Auf der anderen Seite ist es kaum praktikabel, alle vorhandenen Matlab-Funktionen in einer anderen Programmiersprache neu zu implementieren.

- Matlab-Compiler* Erst der seit kurzem verfügbare Matlab-Compiler aus Matlab Version 7 erlaubt einen eleganten Ausweg aus diesem Dilemma, da die fertig compilierten Matlab-Funktionen lizenzkostenfrei (gegenüber Mathworks) weitergegeben werden dürfen. Das dahinterstehende Funktionsprinzip ist neuartig. Die zu compilierende Funktion wird mitsamt aller abhängigen Funktionen verschlüsselt und in eine sogenannte CTF-Datei (*component technology file*) verpackt. Beim Programmstart werden die Funktionen aus der CTF-Datei im Hauptspeicher entschlüsselt und von der zugehörigen *Matlab Component Runtime* ausgeführt. Auf Grundlage dieses Ansatzes wurde im Rahmen des Projekts ein Konzept entwickelt, mit dem die Matlab-Funktionen aus der T1-Vorlesung und der *mscriptview*-Umgebung mit nur zwei vorcompilierten Programmen zur Verfügung gestellt werden können [13].
- t1shell* Das erste Programm, *t1shell*, wird in Abschnitt 3.7 ab Seite 26 beschrieben. Es dient als eigenständiges Hauptprogramm und erlaubt den Aufruf der im Skript referenzierten Matlab-Funktionen über ein einfaches User-Interface. Dabei sind zunächst die im Skript verwendeten Funktionsparameter voreingestellt, diese können aber vom Anwender nach Belieben verändert werden. Da ohne den Matlab-Workspace auch keine Hilfe zur Verfügung steht, integriert *t1shell* eigene Hilfetexte für alle Funktionen.
- t1server* Das zweite Programm, *t1server*, wird in Abschnitt 3.8 ab Seite 28 beschrieben. Es wird als Serverprozess gestartet und bearbeitet die Anfragen von den zugehörigen *MCCApplets* aus den Webseiten des interaktiven HTML-Skripts.
- MCR* Für das Ausführen von vorcompilierten Matlab-Programmen ist die vorherige Installation der sogenannten *Matlab Component Runtime* (MCR) erforderlich, die bisher für Windows XP, Linux 2.4+, MacOS X und diverse Unix-Varianten verfügbar ist. Auf allen Plattformen wird die MCR nur einmal benötigt und kann anschließend für beliebig viele compilierte Applikationen genutzt werden. Erst bei Versionswechseln des Matlab-Compilers muss eventuell auch die MCR aktualisiert werden.

Tatsächlich handelt es sich bei der MCR um ein eingeschränktes Matlab, das zwar über alle mathematischen und Graphikfunktionen verfügt, bei dem aber der interaktive Workspace abgeschaltet ist. Neben dem Debugger fehlen auch die integrierten Hilfsfunktionen und der Matlab-Editor, so dass eine Softwareentwicklung mit der MCR nicht möglich ist. Natürlich können die compilierten Programme problemlos ausgeführt werden, die Anwender können diese Programme aber nicht mehr selbst modifizieren und ohne Matlab-Compiler (und Vollversion) auch keine eigenen Programme erstellen.

Diese Einschränkung ist allerdings zu verkraften, da fast alle unserer bestehenden Funktionen sich weitgehend über Parameter beeinflussen und steuern lassen. Das ursprüngliche Ziel einer vollständig interaktiven und vom Benutzer anpassbaren Umgebung für spielerisches und entdeckendes Lernen bleibt also weitgehend erhalten, zumal in der aktuellen Version der MCR auch der Zugriff auf die *eval*-Funktion und damit die interaktive Auswertung von Matlab-Befehlen möglich ist.

Die für *t1shell* und *t1server* benötigte MCR kann von unserem Webserver frei heruntergeladen werden. Bitte die Webseite für die Vorlesung besuchen,

*Download und
Installation*

<http://tams-www.informatik.uni-hamburg.de/lehre/ws2004/vorlesungen/t1>

und von dort den Links zur Windows bzw. Linux-Version der MCR folgen. Wegen der Dateigröße von derzeit knapp 90 MByte ist eine schnelle Internetanbindung zu empfehlen; alternativ werden wir auch eine CD-ROM mit allen für die Vorlesung benötigten Dateien anbieten. Bitte vor Einsatz der MCR die Lizenzbedingungen (*license.txt*) durchlesen und akzeptieren.

Während die Windows-Version über einen Installer verfügt, der alle benötigten Pfade und Einstellungen in die Registry schreibt, muss die Linux-Version von Hand ausgepackt und konfiguriert werden. Hierzu bitte die Installationshinweise von Mathworks (*MCRInstaller.txt*) beachten. Die Datei *mcr.csh* von unserem Webserver kann als Vorlage für die notwendigen Einstellungen dienen.

```
# sample Linux csh (tcsh) script for MCR setup, edit as needed.
#
# Usage: unpack MCRInstaller.zip to a directory, edit the MCR_ROOT
# line to match that directory. Then do the following:
#
# prompt> /bin/csh -f
# prompt> source mcr.csh
# prompt> ./t1shell
# prompt> ./t1server
# prompt> firefox /media/dvd/skript/t1_mcc_applets/t1.html
#
# edit the following line to the MCR_ROOT directory you chose when
# unpacking the MCRInstaller.zip (Or mkdir /tmp/mcr-test and unzip
# the MCRInstaller.zip there).
#
setenv MCR_ROOT /tmp/mcr-test/v71
setenv XAPPLRESDIR $MCR_ROOT/X11/app-defaults
#
# uncomment the following line if you shell already has
# LD_LIBRARY_PATH set to something...
#
setenv LD_LIBRARY_PATH
#
# paths required by the MCR on Linux
#
setenv LD_LIBRARY_PATH \
$MCR_ROOT/bin/glnx86:\
$MCR_ROOT/runtime/glnx86:\
$MCR_ROOT/sys/os/glnx86:\
$MCR_ROOT/sys/java/jre/glnx86/jre1.4.2/lib/i386/client:\
$MCR_ROOT/sys/java/jre/glnx86/jre1.4.2/lib/i386:\
$MCR_ROOT/sys/opengl/lib/glnx86:\
${LD_LIBRARY_PATH}
#
```

3.7 T1-Shell

Konzept Das Programm *t1shell* erlaubt den direkten Aufruf aller Matlab-Funktionen aus unserem T1-Skript über eine eigene graphische Oberfläche. Für jede Funktion können entweder die Default-Parameter aus dem Skript übernommen oder vom Anwender eigene Parameter eingetragen und ausprobiert werden, so dass exploratives Lernen in vollem Umfang möglich ist. Dabei ist keine Matlab Installation oder Lizenz erforderlich, denn die vorcompilerten Versionen für Windows und Linux benötigen lediglich die jeweilige kostenfreie Matlab Component Runtime. Zweiter Vorteil von *t1shell* ist die einfache Installation. Nach Installation der Matlab Component Runtime (siehe Seite 25) einfach die drei benötigten Dateien herunterladen, in ein beliebiges Verzeichnis kopieren, und das Programm starten.

```
t1shell.exe / t1shell      % Windows / Linux-Programm
t1shell.ctf              % verschlüsselte Matlab-Funktionen
t1shell.mat              % Funktionsliste, Hilfetexte
```

Auf der anderen Seite ist *t1shell* nicht in das HTML-Skript integriert, so dass der Anwender beim Arbeiten zwischen seinem Webbrowser mit den Beschreibungen und *t1shell* mit den interaktiven Funktionen wechseln muss.

Aufruf Zum Starten bitte die entsprechende Binärdatei aufrufen, unter Windows also *t1shell.exe* und unter Linux/Unix einfach *t1shell*. Unter Windows sollte der MCR-Installer alle notwendigen Pfade und Einstellungen in die Registry eingetragen haben. Beim Auspacken der MCR unter Linux/Unix werden dagegen keine Dateien in die Systemverzeichnisse (wie */usr/bin* oder */usr/lib*) geschrieben, so dass die notwendigen Pfadangaben vor Programmstart explizit in der Shell gesetzt werden müssen. Beim Einsatz der C-Shell *cs* kann unser Skript *mcr.csh* als Ausgangspunkt dienen:

```
> /bin/csh -f              % C-Shell starten
> cd /tmp/t1shell         % Verzeichnis mit t1shell Dateien
> source mcr.csh         % Setup der MCR
> ./t1shell              % Programm starten
```

Übrigens sollte das Programm auch unter Windows aus einer Shell (*cmd.exe*) heraus gestartet werden, da einige Funktionen ihre Ausgaben direkt in die Standardausgabe schreiben; dies gilt insbesondere auch für eventuelle Fehlermeldungen. Falls das Programm nicht startet, bitte die Einstellungen auf falsche oder unvollständige Pfade überprüfen und gegebenenfalls korrigieren. Der allererste Programmstart von *t1shell* kann durchaus mehrere Sekunden dauern, da die Matlab Component Runtime in diesem Fall zunächst einmal das CTF-Archiv auspacken und entschlüsseln muss, wobei automatisch die benötigten Unterverzeichnisse angelegt werden. Falls *t1shell* nach eventuellen Updates nicht mehr startet, muss dieses Verzeichnis eventuell von Hand gelöscht werden, um eine Aktualisierung der ausgepackten Dateien zu erzwingen.

Bedienung Nach Programmstart erscheint ein Fenster mit mehreren Auswahl- und Textfeldern, siehe Abbildung 12. Die Bedienung von *t1shell* erfolgt in drei Schritten:

1. Auswahl der gewünschten Funktion in der Auswahlliste. Die Funktionen sind dort zusammen mit den im Skript verwendeten Parametern enthalten; die Liste kann entweder alphabetisch nach den Funktionsnamen oder aber nach der Reihenfolge im Skript sortiert werden.

Anklicken des *Info*-Buttons liefert Informationen zur gerade ausgewählten Funktion, unter anderem die Anzahl der Eingabe- und Ausgabeparameter dieser Funktion.

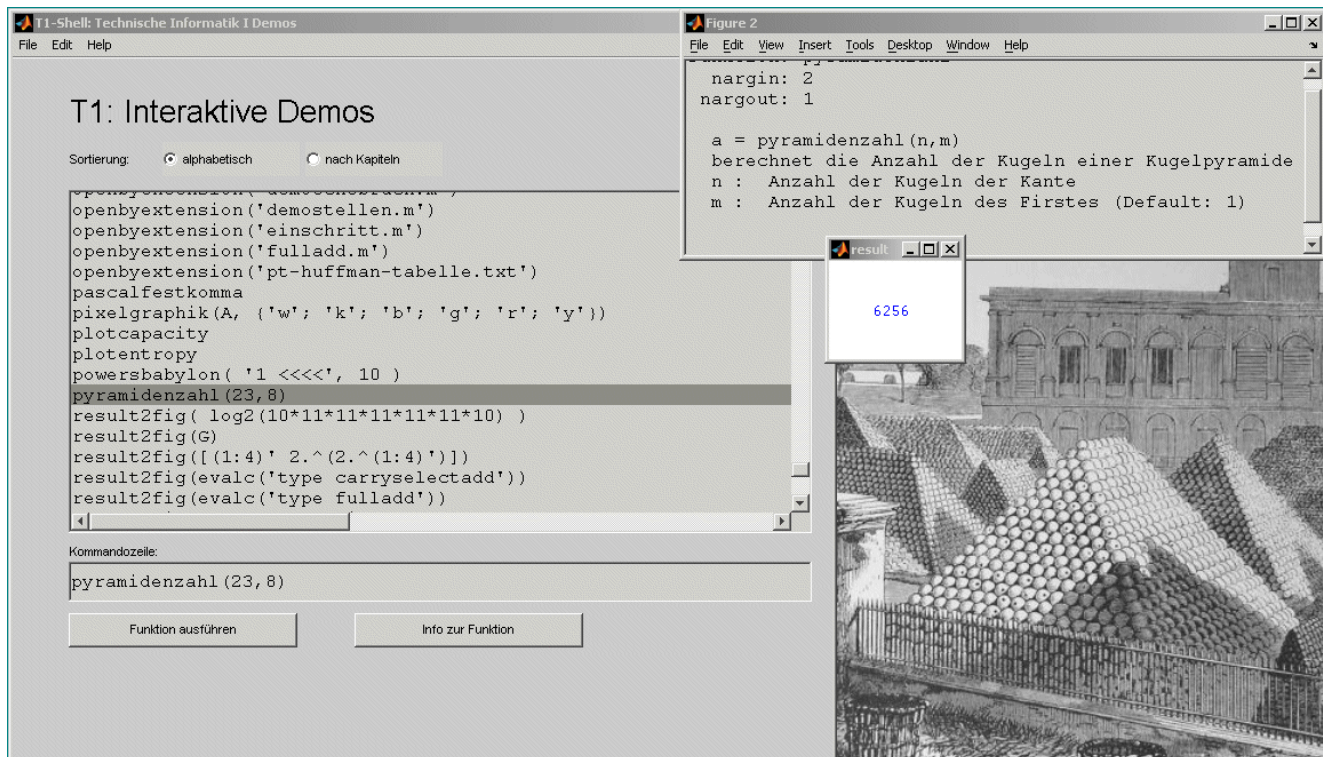


Abbildung 12: Beispiel für den Einsatz von *t1shell* als Standalone-Programm: Auswahl der Funktion `pyramidenzahl` und Anzeige der zugehörigen Hilfe. Jede Funktion kann mit den vorgegebenen oder vom Benutzer individuell ausgewählten Parametern aufgerufen werden. Zum Beispiel enthält die vordere Pyramide des Bildes mit 23 Schichten und 8 Kugeln Firstlänge insgesamt 6256 Kanonenkugeln.

2. Durch Doppelklicken eines Eintrags in der Funktionsliste kann die jeweilige Funktion direkt ausgeführt werden.
3. Eintragen anderer Eingabeparameter für die ausgewählte Funktion in das Textfeld. Alle Parameter können in Matlab-Syntax eingegeben werden, so dass auch Matrizen oder Funktionen als Parameter möglich sind. Einige Beispiele:

```
[0:0.01:2*pi]
{ 'a'; 'b'; 'c'; 'd' }
str2morse( 'INFORMATIK' )
```

Durch Anklicken des *Ausführen*-Buttons wird die ausgewählte Funktion mit den eingestellten Parametern aufgerufen.

Sofern die Funktionsliste noch einige Funktionen enthalten sollte, die in vorcompilierten Programmen nicht sinnvoll ausgeführt werden können, melden Sie solche Probleme bitte an uns. Aus lizenzrechtlichen Gründen erlaubt *t1shell* nur das Aufrufen der Funktionen aus der T1-Vorlesung und verweigert die Ausführung anderer Funktionen. Bitte beachten Sie vor Einsatz von *t1shell* auch die Lizenzbedingungen der MCR.

3.8 T1-Server

Konzept Das Programm *t1server* erlaubt den Aufruf aller Matlab-Funktionen aus der T1-Vorlesung über eine Netzwerkschnittstelle. Dabei ist keine Matlab Installation oder Lizenz erforderlich, denn die vorcompilierten Versionen für Windows und Linux benötigen lediglich die jeweilige kostenfreie Matlab Component Runtime. In Verbindung mit den zugehörigen *MCC-Applets* (siehe Seite 17) kann erstmals eine lizenzfreie Version des interaktiven HTML-Skripts realisiert werden (Variante b in Abbildung 3 auf Seite 7). Nach Anklicken eines Applets überträgt dieses den ausgewählten Matlab-Code an den Serverprozess, der wiederum die zugehörige Matlab-Funktion ausführt.

Anders als *t1shell* verfügt *t1server* nicht über eine eigene graphische Benutzeroberfläche, sondern wird ausschließlich über Netzwerkzugriffe gesteuert. Um die aufwendige Entwicklung eines eigenen Protokolls zu vermeiden, basiert *t1server* auf dem http-Protokoll. Der auszuführende Matlab-Befehl wird dabei direkt als Base64-kodierte URL übergeben, etwa `http://localhost:8888/YXNjaW1tYXQ=` zum Aufruf der Funktion `asciimat`. Auf diese Weise können auch Sonderzeichen und mehrzeilige Matlab-Aufrufe problemlos umgesetzt werden.

Installation Nach Installation der Matlab Component Runtime (siehe Seite 25) bitte einfach die beiden benötigten Dateien herunterladen und in ein beliebiges Verzeichnis kopieren:

```
t1server.exe / t1server      % Windows / Linux-Programm
t1server.ctf                % verschlüsselte Matlab-Funktionen
```

Aus Sicherheitsgründen bearbeitet *t1server* nur http-Anfragen, die vom aktuellen Rechner (*localhost*) ausgehen, während Anfragen von externen Rechnern ignoriert werden. Trotzdem besteht in Multiuser-Umgebungen ein Sicherheitsrisiko, da *t1server* bisher noch nicht über Funktionen zur Benutzer-Authentifizierung verfügt. Jeder lokal angemeldete Benutzer kann http-Anfragen an Port 8888 abschicken, die im Kontext des *t1server*-Prozesses bearbeitet werden. Unter Linux/Unix-Systemen ist es deshalb ratsam, das *t1server* Programm nicht unter der eigenen Benutzerkennung sondern mit reduzierten Rechten (z.B. User *nobody*) zu installieren und auszuführen.

Server starten Zum Starten bitte die entsprechende Binärdatei aufrufen, unter Windows also *t1server.exe* und unter Linux/Unix einfach *t1server*. Unter Windows sollte der MCR-Installer alle notwendigen Pfade und Einstellungen in die Registry eingetragen haben. Beim Auspacken der MCR unter Linux/Unix werden dagegen keine Dateien in die Systemverzeichnisse (wie `/usr/bin` oder `/usr/lib`) geschrieben, so dass die notwendigen Pfadangaben vor Programmstart explizit in der Shell gesetzt werden müssen. Beim Einsatz der C-Shell *cs* kann unser Skript *mcr.csh* als Ausgangspunkt dienen:

```
> /bin/csh -f                % C-Shell starten
> cd /tmp/t1server          % Verzeichnis mit t1server Dateien
> source mcr.csh            % Setup der MCR
> ./t1server                % Programm ohne Optionen starten
> ./t1server 8888 3000 3000 0 % Port, Timeouts, Debug
```

Da *t1server* als Netzwerk-Serverprozess dient, der von den *MCCApplets* aus angesprochen wird, ist kein eigenes User-Interface erforderlich. Nach erfolgreichem Start schreibt *t1server* lediglich einige Zeilen mit Lizenzinformationen und den verwendeten Parametern in die Standardausgabe:

```

t1server (C) 2004-2005 F.N.Hendrich, University of Hamburg
port=8888
timeout=1800 secs.
idletimeout= 300 secs.
NOTE: requests are allowed from localhost only.
Example: http://localhost:8888/YXNjaWltYXQ=
Use http://localhost:8888/exit to stop the server
Use http://localhost:8888/status for server status report

```

Dabei sollte das Programm auch unter Windows aus einer Shell (cmd.exe) heraus gestartet werden, da einige Funktionen ihre Ausgaben direkt in die Standardausgabe schreiben; dies gilt insbesondere auch für eventuelle Fehlermeldungen. Falls das Programm nicht startet, bitte die Einstellungen auf falsche oder unvollständige Pfade überprüfen und gegebenenfalls korrigieren. Der allererste Programmstart von *t1server* kann durchaus mehrere Sekunden dauern, da die Matlab Component Runtime in diesem Fall zunächst einmal das CTF-Archiv auspacken und entschlüsseln muss, wobei automatisch die benötigten Unterverzeichnisse angelegt werden. Falls *t1server* nach eventuellen Updates nicht mehr startet, muss dieses Verzeichnis eventuell von Hand gelöscht werden, um eine Aktualisierung der ausgepackten Dateien zu erzwingen.

Der Serverprozess kann auf mehrere Arten wieder beendet werden:

Server beenden

- Abbruch durch den Benutzer über die Shell (CNTL-C) oder den Taskmanager.
- Abbruch durch Anfrage der URL `http://localhost:8888/exit`, beispielsweise durch Eintippen der URL in einem Webbrowser.
- Automatischer Abbruch bei Erreichen der über den zweiten Parameter übergebenen Gesamt-Laufzeit des Prozesses. Damit wird sichergestellt, dass der Prozess nicht unbeabsichtigt auf Dauer weiterläuft. Die Voreinstellung für den Timeout-Wert ist 1800 Sekunden bzw. 30 Minuten. Mit einem Parameterwert 0 kann der Auto-Timeout abgeschaltet werden.
- Automatischer Abbruch bei Erreichen des über den dritten Parameter übergebenen Idle-Timeouts. Damit wird sichergestellt, dass der Prozess nicht unbeabsichtigt auf Dauer weiterläuft. Die Voreinstellung für den Idle-Timeout-Wert sind 300 Sekunden bzw. 5 Minuten. Der zugehörige Zähler wird nach jeder neuen Anfrage zurückgesetzt.

Zum Testen des Servers einfach die URL `http://localhost:8888/status` in einem Webbrowser eingeben. Der Server antwortet mit seinen Parametern und den aktuellen Timeout-Werten.

Server testen

Nach dem Start des Servers kann das interaktive Skript wie gewohnt benutzt werden. Einfach die entsprechenden Webseiten öffnen und ein *MCCApplet* doppelklicken, um den zugehörigen Matlab-Code über den *t1server* auszuführen. (Hinweis: aus bisher nicht geklärten Gründen ignoriert der Server manchmal den allerersten Zugriff. In diesem Fall das jeweilige Applet einfach erneut anklicken.)

Selbstverständlich ist es auch möglich, die http-Anfragen an den *t1server* ohne die *MCC-Applets* zu erstellen, etwa durch Eintippen eines Base64-kodierten Befehls im Eingabefeld des Browsers, etwa `http://localhost:8888/YXNjaWltYXQ=` zum Aufruf der Funktion `asciimat`. Auf diese Weise können Anfragen an den *t1server* auch (ohne Applets) als Hyperlinks in Webseiten eingebaut werden; allerdings können solche Links dann nicht mehr vom Anwender editiert werden.

3.9 Erzeugen der Funktionsliste für *t1shell* und *t1server*

Abhängigkeitsanalyse

Anders als im Matlab-Workspace lässt sich der Funktionsumfang einer vorcompilierten Matlab-Applikation nicht mehr nachträglich erweitern. Vielmehr müssen alle benötigten Funktionen von vornherein bekannt sein und während der Abhängigkeitsanalyse durch den Matlab-Compiler erkannt werden. Für Funktionen, die nur implizit über callbacks oder *eval()* aufgerufen werden, sind für den Compiler entweder zusätzliche sogenannte *Pragma*-Anweisungen oder eine explizite Liste der Funktionen erforderlich.

Angesichts des Gesamtumfangs von ca. 500 Funktionen alleine für die T1-Vorlesung liegt es nahe, das Aufstellen dieser Funktionsliste zumindest teilweise zu automatisieren. Die folgende Aufzählung beschreibt die zum Erstellen der Funktionsliste notwendigen Schritte, die sich nach entsprechender Anpassung auch für weitere Vorlesungen verallgemeinern lassen:

MakeFunctionList

1. Aufruf des Java-Programms *MakeFunctionList* zur Erstellung einer ersten Rohfassung der Funktionsliste aus den vorhandenen *mscriptview*-Skriptdateien. (Dieser Schritt könnte ebensogut mit einer Matlab-Funktion erledigt werden, aber die Implementierung in Java war etwas einfacher).

Das Programm liest alle via Kommandozeile übergebenen *.m*-Dateien zeilenweise ein und sortiert alle passiven Elemente (Kommentarzeilen) sowie alle aktiven Elemente (Matlab-Code) aus, die offenbar den Matlab-Workspace benötigen. Bei den übrigen Zeilen handelt es sich um Matlab-Funktionsaufrufe, die sich prinzipiell mit dem Matlab-Compiler in eigenständige Applikationen übersetzen lassen. Diese Zeilen werden in die Ausgabedatei *mfl_database.txt* geschrieben.

Zum Beispiel übergibt der folgende Funktionsaufruf auf Unix-Systemen alle *t1*-Skriptdateien an *MakeFunctionList*. Auf Windows-Systemen hängt die Wirkung von der verwendeten Shell ab; im Notfall können alle gewünschten Dateien einzeln auf der Befehlszeile angegeben werden:

```
setenv TOOLBOXDIR ~/matlab7           % bitte anpassen,
cd $TOOLBOXDIR/t1                     % z.B. ~/matlab7/t1
java de.mmkh.tams.MakeFunctionList t1_*.m
edit mfl_database.txt                 % bei Bedarf
```

Da die Heuristen in *MakeFunctionList* nicht alle Spezialfälle abdecken, empfiehlt es sich, die erzeugte Ausgabedatei in einem Texteditor zu überprüfen. Offensichtlich nicht sinnvoll compilierbare Codezeilen sollten dann gelöscht werden. Natürlich wäre es noch besser, solche Zeilen in den originalen *mscriptview*-Skriptdateien zu modifizieren und *MakeFunctionList* danach erneut zu starten.

Hilfedatei

2. Anschließend muss Matlab gestartet werden, um die *t1shell.mat* Datei mit den Metadaten und Hilfetexten für die einzelnen Funktionen zu erzeugen. Dies erfolgt durch Einlesen der im vorherigen Schritt erzeugten Datei *mfl_database.txt* mit der Funktion *read_mfl_database*:

```
cd TOOLBOXDIR/work                     % Matlab work Verzeichnis
matlab7                                 % Matlab starten
> cd TOOLBOXDIR/t1
> info = read_mfl_database( 'mfl_database.txt' )
> save 't1shell.mat' info
```


3. Im nächsten Schritt wird durch Aufruf der Funktion *maket1evalinc* die Matlab-Funktion *t1evalinc.m* erstellt, die aus den vorher erstellten Einträgen in *t1shell.mat* und einer zusätzlichen hardcodierten Liste von besonderen Funktionen eine Liste aller von den compilierten Programmen aufrufbaren Matlab-Funktionen enthält:

```
> cd TOOLBOXDIR/t1
> maket1evalinc           % erzeugt t1evalinc.m
> edit t1evalinc.m       % bei Bedarf editieren
```

4. Jetzt kann der Matlab-Compiler gestartet werden, um die Programme *t1shell* (siehe Abschnitt 3.7) und *t1server* (siehe Abschnitt 3.8) zu erzeugen. In beiden Fällen ergibt sich die Liste der eingebundenen Funktionen aus der Abhängigkeitsanalyse des Matlab-Compilers zusammen mit den explizit in *t1evalinc.m* aufgelisteten Funktionen:

```
> mcc -m -d /temp/mcc t1shell.m
> mcc -m -d /temp/mcc t1server.m
> cp t1shell.mat /temp/mcc
> dir /temp/mcc           % Dateiliste anzeigen
t1shell                   % Executable
t1shell.ctf              % Matlab-Code als CTF
t1shell.mat              % Hilfedatei
t1server                 % Executable
t1server.ctf            % Matlab-Code als CTF
```

Unter Windows muss eventuell ein Laufwerksbuchstabe angegeben werden, etwa `mcc -m -d c:/temp/mcc`, und die erzeugten Programme haben die Endung `.exe`. Das Ausgabeverzeichnis kann frei gewählt werden, muss aber bereits vor Aufruf des Compilers existieren.

5. Zum Deployment der Applikationen einfach die benötigten Dateien an die Anwender verteilen. Für *t1shell* sind dies:

```
t1shell(.exe), t1shell.ctf, t1shell.mat
```

und für *t1server* entsprechend:

```
t1server(.exe), t1server.ctf
```

6. Nach einem Wechsel der Matlab-Version muss eventuell auch die Bibliothek *pnet.dll* (bzw. *pnet.so*) mit den TCP/IP-Netzwerkfunktionen für ein *t1server* neu erzeugt werden. Dies erfolgt mit dem Matlab MEX-Compiler:

```
> cd TOOLBOXDIR/tcp_udp_ip
> mex pnet.c
```

Anschließend erneut den Matlab-Compiler aufrufen, um das *t1server*-Programm neu zu erzeugen (Schritt 4).

3.10 MScript2HtmlConverter

automatische Konvertierung

Wie in Abschnitt 2.1 bereits angedeutet wurde, ist die Content-Erstellung für interaktive Skripte im HTML-Format gegenüber dem *mscriptview*-Konzept aufwendiger. Zwar unterstützt HTML alle für die Skripte notwendigen Textattribute sowie Graphiken und Hyperlinks; die aktiven Elemente müssen jedoch separat als Applets eingebunden werden. Als Lösung für dieses Problem wurde im Rahmen einer Diplomarbeit [37] ein Konverter entwickelt, der die bestehenden *mscriptview*-Skripte automatisch nach HTML umsetzt. Damit kann die Content-Erstellung weiterhin auf ASCII-Basis mit jedem Texteditor erfolgen. Im einzelnen übernimmt der Konverter die folgenden Funktionen:

1. Einlesen der angegebenen Quelldateien und optional aller über Querverweise referenzierten Dateien, so dass die vollständige Umsetzung eines Skripts mit einem Aufruf möglich ist.
2. Analyse der Textstruktur und Trennung der aktiven Elemente (eingebetteter Matlab-Skriptcode) vom statischen Text.
3. Erzeugen von HTML-Code für Java-Applets für alle bei der Analyse erkannten aktiven Skripte. Der Skriptcode wird als Parameter für die Applets übergeben und von diesen angezeigt.
4. Umsetzung der statischen Texte nach HTML mit den passenden Formatierungen wie Formeln, Tabellen, Blocksatz, Fettschrift. Da die Formatierungen in den Matlab-Skripten nur teilweise explizit markiert sind, wird eine Heuristik zur Erkennung von Paragraphen und Tabellen eingesetzt.
5. Formeln werden ebenfalls nach HTML konvertiert, soweit die benötigten Symbole zur Verfügung stehen. Leider implementieren die verschiedenen Browser (z.B. Internet Explorer, Mozilla, Opera, Konqueror) jeweils nur einen Subset der vom W3C für HTML definierten Symbole. Komplexere Formeln können alternativ als Java-Applets oder als vorberechnete (z.B. via LaTeX erzeugte) Abbildungen eingebunden werden.
6. Erkennen von in den Skriptseiten eingebauten Hyperlinks und direkte Umsetzung in HTML-Links. Parallel dazu wird eine interne Liste aller entsprechenden Skriptseiten und Querverweise aufgebaut und zur Erzeugung von zusätzlichen Hyperlinks zur Navigation eingesetzt. Schließlich wird eine Indexseite aller Skriptseiten erzeugt, sofern diese nicht bereits im ursprünglichen Skript vorhanden war.

Aufruf Der Konverter ist in Java geschrieben und kann wahlweise über die Kommandozeile oder mit einer Benutzeroberfläche gestartet werden. Die eingebauten Heuristiken wurden ursprünglich nur für die T1-Vorlesung entwickelt, mittlerweile aber mit allen vorhandenen Matlab-Dateien getestet. Sie erlauben die vollautomatische Umsetzung des Contents von Matlab nach HTML. Bis auf eine Handvoll Ausnahmen, aufgrund mehrdeutiger Formulierung oder besonders aufwendiger Formeln, sind keine manuellen Nacharbeiten notwendig. Damit ist auch die Integration unserer Materialien in die gängigen E-Learning Plattformen problemlos möglich. Derzeit stehen drei vollständige Vorlesungen (*Technische Informatik 1*, *Digitale Signalverarbeitung*, *Nachrichtentechnik und Datenübertragung*) zusätzlich zur Matlab-Version auch als HTML-Version zur Verfügung.

Das folgende Beispiel zeigt einen typischen Aufruf des Konverters auf der Kommandozeile. Mit den angegebenen Optionen werden die zu übersetzende Datei (t1.m) und das Zielverzeichnis für die erzeugten HTML-Dateien ausgewählt:

```
java uni_hh.elearning.util.Mscript2HtmlConverter -b \  
-m -f /tmp/matlab7/toolbox/t1/t1.m -d /tmp/appletscripts/t1
```

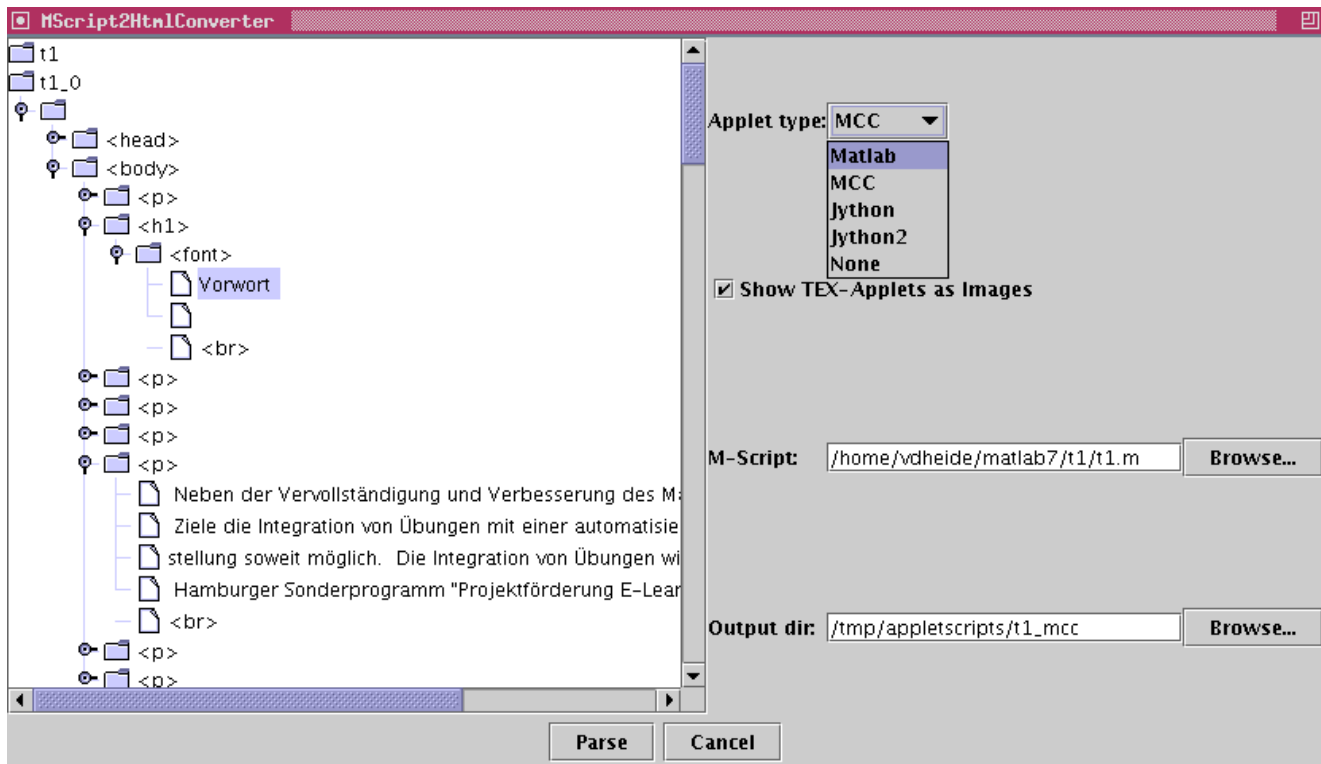


Abbildung 13: Das User-Interface von MScript2HtmlConverter. Die Baumansicht präsentiert die Dokumentenstruktur der ausgewählten msv-Skriptdateien.

Im einzelnen unterstützt MScript2HtmlConverter die folgenden Parameter:

```
tams20> java uni_hh.elearning.util.MScript2HtmlConverter -help
Usage:
  MScript2HtmlConverter -b [-j|-j2|-n] [-i] -f M-File -d OutputDir
  MScript2HtmlConverter [-j|-j2|-n] [-i] [-f M-File] [-d OutputDir]
  MScript2HtmlConverter -help
Options: -b      Batchmode, no GUI
         -s      Scriptmode with input/output pipes
         -j      Use JythonApplets instead of MatlabApplets for Code
         -j2     Use JythonApplets instead of MatlabApplets for Code
         -m      Use MCCApplets instead of MatlabApplets for Code
         -n      No special Applets for Code
         -i      Use images instead of TEX-applets
         -f      M-Script, that will to be converted
                Linked files will also be converted
         -d      Directory, where HTML-Files will be saved
                Must be a subdirectory of 'appletscripsts'
```

Im interaktiven Modus können die Optionen für Applet-Typ, Umsetzung von Formeln mit TeX-Applets oder als Abbildungen, Pfad der Eingabedatei(en) und Ausgabeverzeichnis nacheinander über das User-Interface gewählt werden. Nach Anklicken von *Parse* werden die Eingabedateien eingelesen und die Dokumentenstruktur als Baum angezeigt, siehe Abbildung 13. Weiteres Anklicken von *Save* schreibt die HTML-Dateien.

Aus technischen Gründen (Ermittlung der Codebase-Parameter für die Applets) muss der Pfad zum Zielverzeichnis ein Unterverzeichnis *appletscripsts* enthalten und kann erst nach der Konvertierung umbenannt werden.

3.11 TeXtoPNGConverter

- Formeln* Natürlich bilden mathematische Formeln eine wichtige Komponente unserer Skripte zur technischen Informatik. Im *mscriptview*-Browser werden Formeln mit Matlab-Funktionen dargestellt, wobei die üblichen mathematischen Symbole, Operatoren und die griechischen Buchstaben unterstützt werden. Die Eingabe der Formeln folgt dabei dem bewährten Konzept von TeX, bleibt aber in Bezug auf Schriftauswahl, Formatierungen und Layoutmöglichkeiten weit hinter dem Vorbild zurück. Erst in der aktuellen Version (Matlab 7) wurde der Funktionsumfang der Formeldarstellung drastisch erweitert.
- MathML, Applets* Leider zeigten schon unsere ersten Experimente mit der Umsetzung von Matlab nach HTML, dass viele Webbrowser die gängigen Standards (HTML-Symbole bzw. MathML-Erweiterung) entweder gar nicht oder nur unzureichend unterstützten. Insbesondere musste auf die eigentlich wünschenswerte und gleichzeitig recht geradlinige Umsetzung der TeX-Formeln nach MathML verzichtet werden. Wegen massiver Performanceprobleme konnte auch die Alternative, alle Formeln mit (bereits vorhandenen) Applets darzustellen, nicht realisiert werden. Bereits bei Einbettung einiger Dutzend mathematischer Symbole als Applets ergaben sich teilweise indiskutable Ladezeiten der jeweiligen Webseiten.
- Abbildungen* Deshalb verwenden unsere aktuellen HTML-Skripte statische Abbildungen im PNG-Dateiformat für alle Formeln, die nicht mit Standard HTML-Konstrukten umgesetzt werden können. Eine aufwendige Heuristik in *MScript2HtmlConverter* analysiert die einzelnen Formeln und zerlegt diese bei Bedarf in die „HTML-kompatiblen“ und „komplexen“ Anteile. Die Menge der HTML-kompatiblen Symbole ergibt sich dabei als der gemeinsame Funktionsumfang aller von uns getesteten gängigen Webbrowser. Alle übrigen, komplexen Anteile der Formeln werden schließlich wieder in TeX-Syntax an die Java-Klasse *TeXtoPNGConverter* übergeben, die die zugehörige Abbildung berechnet. Der Funktionsumfang ähnelt dem Funktionsumfang von Matlab bis Version 6.5; zusätzlich können aber die originalen Computer-Modern Schriftarten verwendet werden. Abbildung 14 zeigt einige Beispiele für Formeln in TeX-Syntax und die Darstellung von TeXtoPNGConverter.
- Funktionsumfang* Sowohl *TeXtoPNGConverter* als auch unsere *TeXApplets* verwenden intern die Hilfsklassen aus *jfig.utils.LP2* zur Formatierung ihrer TeX-Eingaben. Dieser Parser unterstützt derzeit die folgenden TeX-Makros:
- Gruppierung von Objekten via { und }.
 - Sub- und Superskripte in Formeln via _ und ^; allerdings müssen diese immer geklammert sein. Also bitte $x^{\{2\}}$ statt x^2 schreiben!
 - Fontauswahl via `\rm`, `\sf`, `\em`, `\it`, `\sl` `\tt`, sowie `\itt`, `\btt` für kursive bzw. fette Varianten von `\tt`.
 - Umschalten zwischen Text- und Mathematikmodus mit `$` bzw. `$$` sowie `\mathrm`.
 - Farbauswahl via `\black`, `\red`, `\green`, `\blue`, `\cyan`, `\magenta`, `\yellow`, `\white`, sowie die (xfig) Farben `\darkblue`, `\lightblue`, `\darkgreen`, `\darkred`, `\gold`.
 - Auswahl der absoluten Schriftgröße (in pt) via `\fivept`, `\sixpt`, ... `\thirtypt`, `\thirtyfivept`, `\fourtypt`, ... sowie `\xpt`, `\xipt`, `\xiipt`, ... `\xxvpt`, `\xxxpt`.
 - Auswahl der relativen Schriftgröße via `\tiny`, `\scriptsize`, `\footnotesize`, `\small`, `\normalsize`, `\large`, `\Large`, `\LARGE`, `\Huge`, `\HUGE`.

- Einzelne ASCII-Symbole: `\hashsign`, `\atsign`, `\lbracket`, `\rbracket`, `\lbrace`, `\rbrace`, `\backslash`, `\verticalbar`, `\tildechar`, `\lesschar`, `\equalchar`, `\greaterchar`.
- Mathematische Funktionen (TeXbook Kapitel 18.2): `\arccos`, `\arcsin`, `\arctan`, `\arg`, `\atan`, `\cos`, ... `\tanh`.
- Griechische Buchstaben (TeXbook Anhang F.1): `\alpha`, ... `\omega` sowie `\varepsilon`, `\vartheta`, `\varpi`, `\varrho`, `\varsigma`, `\varphi`.
- Griechische Buchstaben (TeXbook Anhang F.2): `\Alpha`, ... `\Omega`.
- Kalligraphische Symbole (TeXbook Anhang F.3): `\calA`, ... `\calZ`.
- Verschiedene Symbole (TeXbook Anhang F.4): `\aleph`, `\hbar`, ... `\spacedsuit`.
- Die `\oldstyle`-Variante der Ziffern (TeXbook Anhang F.5) wird nicht unterstützt.
- Große Operatoren (TeXbook Anhang F.6): `\sum`, `\prod`, ... `\biguplus`. Leider wird die Spezialbehandlung der Sub- und Superskripte bisher nicht unterstützt; dafür gibt es die Varianten `\Sum`, `\Prod` und `\Int`, also zum Beispiel `\Sum_{i=0}^N` statt `\sum_{i=0}^N` schreiben.
- Binäre Operatoren (TeXbook Anhang F.7): `\minus`, `\slash`, `\pm` ... , `\odot` sowie `\dagger`, `\ddagger`, `\amalg`.
- Relationen (TeXbook Anhang F.8): `\lessthan`, ... `\perp`.
- Negierte Relationen (TeXbook Anhang F.9): `\not`, `\notin`.
- Pfeile (TeXbook Anhang F.10): `\leftarrow`, `\Leftarrow`, ... , `\nrightarrow` außer `\rightleftharpoons` und `\buildrel`.
- Weitere Pfeile (LaTeX Companion, Tabelle 8.6): `\looparrowleft`, ... `\nrightarrow`.
- Öffnende und schließende Symbole (TeXbook Anhang F.11 und F.12): `\lbrack`, `\lfloor`, ... , `\rceil`. Einige Symbole werden nicht unterstützt.
- Satzzeichen (TeXbook Anhang F.13): `\colon`, `\ldots`, `\cdots`.
- Akzente (LaTeX Companion, Tabelle 8.2): `\hat`, `\check`, `\acute`, `\grave`, `\bar`, `\vec`, `\dot`, `\ddot`, `\tilde`, `\dq`. Achtung: bitte Klammern verwenden, also z.B. `\hat{X}` statt `\hat X`.
- AMS (LaTeX Companion, Tabelle 8.13): `\digamma`, `\beth`, `\daleth`, `\gimel`, `\lll`, `\ggg`, `\percent`, `\promille`, `\copyright`, `\paragraph`, `\degrees`, `\square`.
- `\overline`, `\underline`
- `\sqrt`
- Brüche mittels `\frac{}{}`
- links, zentriert oder rechts ausgerichtete *Stapel* mit jeweils drei Gruppen: `\lstack`, `\cstack`, `\rstack`, also zum Beispiel `\lstack{a}{b}{c}`.

$a^2 + b^2 = c^2$	<code>\$a^{2} + b^{2} = c^{2}\$</code>
$\cos^2 x + \sin^2 x = 1$	<code>\$\$\cos^{2}x + \sin^{2}x = 1\$</code>
$e^{i\pi} + 1 = 0$	<code>\$e^{i\pi} + 1 = 0\$</code>
$\ln a \cdot b = \ln a + \ln b$	<code>\$\$\ln\ a\cdot b = \ln\ a + \ln\ b\$</code>
$\sum_{i=0}^N i = n(n+1)/2$	<code>\$\$\Sum{i=0}{N} i = n (n+1)/2\$</code>
$S_i(t+1) = \text{sgn} [\sum_j J_{ij} \cdot S_j(t)]$	<code>\$\$S_{i}(t+1) = \text{sgn}[\,\sum_{j} J_{ij}\cdot S_{j}(t)]\$</code>
$J_{ij} = N^{-1/2} \sum_{\mu} \xi_i^{\mu} \xi_j^{\mu}$	<code>\$\$J_{ij} = N^{-1/2} \sum_{\mu} \xi_{i}^{\mu} \xi_{j}^{\mu}\$</code>

Abbildung 14: Beispiele für die Umsetzung von TeX-formatierten Formeln in Abbildungen via TeXtoPNGConverter. Der Funktionsumfang umfasst alle einzelnen mathematischen Operatoren und Symbole, die griechischen und kalligraphischen Buchstaben, Sub- und Superskripte sowie Fontauswahl, aber nur wenige Funktionen zur relativen Platzierung von Symbolen. Als Schriftarten können alternativ die Java Systemschriften oder die Computer-Modern Fonts benutzt werden.

Aufruf Der Aufruf von *TeXtoPNGConverter* kann sowohl interaktiv auf der Kommandozeile als auch von anderen Java-Programmen aus erfolgen. Während der darzustellende Text in letzterem Fall einfach direkt als Java String-Objekt übergeben werden kann, ist dies beim Aufruf auf der Kommandozeile wegen der vielen Sonderzeichen problematisch. Sowohl das Dollarzeichen als auch die Klammern (`$\()` `[]` `{}`) haben in den meisten Shells ihre eigene Bedeutung und müssten mühsam als inaktiv markiert werden. Deshalb erwartet TeXtoPNG-Converter seine interaktive Eingabe in einer Datei:

```
edit formel.tex
type formel.tex
$$\Sum{i=0}{N} i = \frac{n(n+1)}{2}$
java jfig.utils.TeXtoPNGConverter formel.tex formel.png
```


3.12 Simulator T1-Hades

Als Werkzeug für die Aufgaben zum Entwurf digitaler Schaltungen stellen wir unseren Studierenden seit mehreren Jahren das selbstentwickelte Framework *Hades* (Hamburg Design System) zur Verfügung [9, 10]. Die gesamte Software ist in Java implementiert und kann wahlweise als Applikation oder als Applet genutzt werden. Im Rahmen des Projekts wurde eine speziell an die aktuelle Vorlesung und Übungen angepasste Version des Editors und Simulators erstellt.

Interaktive Simulation

Wesentlicher Vorteil von Hades gegenüber vielen herkömmlichen Simulatoren ist das Prinzip der interaktiven Simulation — eine Schaltung kann bereits simuliert werden, während sie noch aufgebaut wird. Alle Änderungen der Schaltung werden sofort in die Simulation übernommen. Der traditionelle Entwicklungszyklus mit seinen separaten Phasen (Edit-Compile-Simulate-Analyse) ist daher überflüssig. Durch Visualisierung der Logikpegel auf allen Leitungen (*glow-mode*) ergibt sich ein intuitiver Überblick über das Verhalten der Schaltung; Kurzschlüsse oder offene Leitungen sind durch entsprechende Farbgebung auf einen Blick zu erkennen.

T1-Hades

Dieser Simulator wird von uns seit mehreren Jahren in Praktika und für Übungen eingesetzt und hat sich dabei bewährt. Im einzelnen weist die an die Vorlesung angepasste Version *T1-Hades* die folgenden Änderungen auf:

- Integration der im zweiten Projektbericht [12] beschriebenen Algorithmen zur automatischen Überprüfung von Schaltungen durch Simulation und Signaturanalyse. Die dafür neu entwickelten bzw. erweiterten Java-Klassen stehen natürlich auch in der Vollversion zur Verfügung.
- Integration der Schaltungsbeispiele zur Vorlesung sowie aller Vorlagen und Testbenches für die Übungen direkt in die *hades-ws2004.jar*-Archivdatei. Diese Schaltungen können daher ohne weitere Installationsschritte sofort geladen werden.
- Neues Menü zum Direktzugriff auf die Schaltungsbeispiele zur Vorlesung, z.B. die Demos zum Hamming-Code oder der diversen Addierer.
- Neues Menü zum Direktzugriff auf die Vorlagen für die Übungsaufgaben und der zugehörigen automatischen Überprüfung.
- Angepasstes Popup-Menü mit Beschränkung der für die Übungsaufgaben benötigten Gatter und Flipflops, dabei Verwendung der deutschen DIN-Symbole für die Gatter.
- Bereitstellen eines Java-Webstart Installers auf unseren Webseiten; dadurch besonders einfacher Download und automatische Installation.
- Reduzierung der Codegröße von über 5 MByte (*hades.jar*) auf ca 1.2 MByte (*hades-ws2004.jar*) durch Weglassen nicht benötigter Simulationsmodelle und Dokumentation. Entsprechend kürzere Downloadzeiten beim Zugriff über Java-Webstart oder beim direkten Download.
- Zusätzliche Kurzdokumentation mit Hinweisen zum Webstart-Download und zur Bedienung des Simulators.

Die vorhandene Dokumentation inklusive der Kurzreferenzkarte und dem umfangreichen Hades-Tutorial [8] kann natürlich auch für die T1-Version von Hades übernommen werden.

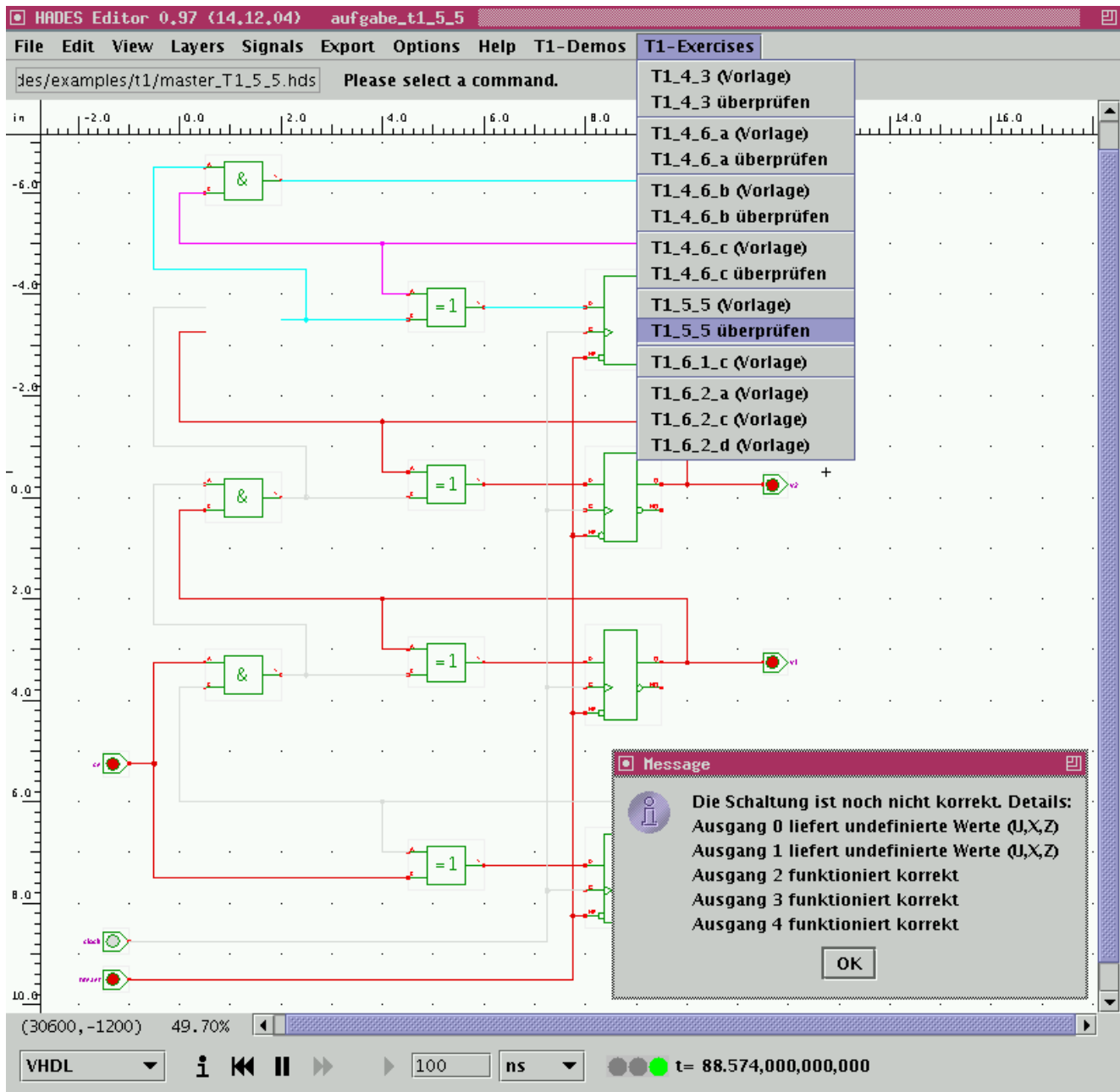


Abbildung 15: Beispiel zur Überprüfung einer digitalen Schaltung mit der speziell an die Vorlesung T1 angepassten Version T1-Hades unseres Hades-Simulators. In der Abbildung fehlt noch ein UND-Gatter des zu entwerfenden 4-bit Binärzählers, so dass zwei Ausgänge falsche Werte liefern. Man beachte, dass die undefinierten Leitungen im Glow-Mode auf einen Blick durch ihre Farbe zu erkennen sind.

Der T1-Hades Simulator steht auf den Webseiten der T1-Übungen unter der URL <http://tams-www.informatik.uni-hamburg.de/lehre/ws2004/t1uebung/applets/> zum Download zur Verfügung. Dort entweder den `hades.jnlp` Webstart-Download anklicken oder die `hades-ws2004.jar` Archivdatei direkt herunterladen und den Simulator anschließend in einer Shell starten:

Aufruf

```
java -jar hades-ws2004.jar
java -jar hades-ws2004.jar -file dateiname.hds
```

3.13 PRIMA-Simulator

In vielen Lehrbüchern und Vorlesungsskripten zur Thematik der digitalen Systeme wird das Thema Rechnerarchitektur und speicherprogrammierbarer Rechner („von-Neumann Rechner“) mit einem *bottom-up*-Ansatz eingeführt. Aufbauend auf der Boole’schen Algebra werden zunächst die Grundkomponenten eines Rechners (Multiplexer, Rechenwerke, Register, etc.) vorgestellt. Dann wird aus diesen Grundkomponenten ein einfacher Mikrorechner aufgebaut und dessen Funktionsweise anhand der Ausführung einzelner Befehle erläutert. Anschließend werden einfache Maschinen-Programme für diesen Rechner vorgeführt und die Grundprinzipien der Assembler-Programmierung erläutert.

Die PRIMA Dieses bewährte Vorgehen wird auch in unserer Vorlesung eingesetzt [4, 5]. Als Grundlage dient die *Primitive Maschine* (PRIMA), eine sehr einfache Akkumulatormaschine mit lediglich vier Registern und 256 Worten Hauptspeicher bei 8-bit Wortbreite. Der Befehlssatz umfasst ausschließlich die grundlegenden Rechenoperationen, Lade- und Speicherbefehle mit direkter Adressierung sowie einige Sprungbefehle. Durch die horizontale Befehlskodierung reichen wenige Gatter zur Realisierung des Rechners aus, die gesamte Schaltung lässt sich bequem auf einem Blatt Papier skizzieren. Der einfache Befehlszyklus der Maschine umfasst jeweils drei Takte (Befehl holen, Adresse holen, Befehl ausführen). Lediglich das Inkrementieren des Befehlszählers erfolgt implizit parallel zu den übrigen Operationen, was die Beobachtung der Zeitabläufe stark erleichtert. Da komplexere Adressierungsarten fehlen, müssen viele Operationen wie z.B. Arrayzugriffe mittels selbstmodifizierendem Code realisiert werden.

Der besonders einfache Aufbau der PRIMA erleichtert nicht nur das Verständnis für die Studierenden, sondern auch die Simulation. Derzeit verfügen wir über mehrere Simulatoren der PRIMA, sowohl als einfache standalone-Applikationen auf der Befehlsebene bis hinunter zu einer Simulation auf Gatterebene. Leider konnte ein bereits 1996 realisiertes, stark graphisch orientiertes Applet [42] didaktisch nicht überzeugen. Die in diesem Applet eingesetzten Animationen zur Darstellung der einzelnen Datentransfers sehen zwar hübsch aus, lenken aber eher vom Verständnis der Vorgänge ab als dieses zu unterstützen.

Simulator Abbildung 16 zeigt den im Rahmen dieses Projekts neu entwickelten Simulator, der als kleines Java-Applet realisiert ist und ohne weitere Installation benutzt werden kann. Die graphische Oberfläche stellt die Register- und Speicherinhalte des Rechners dar; diese können alternativ als dezimale oder hexadezimale Werte dargestellt werden und lassen sich direkt editieren. Neben der Ausführung einzelner oder mehrerer Befehle lassen sich auch die einzelnen Taktschritte simulieren, wobei die zugehörigen Operationen auf Wunsch im mittleren Textfeld protokolliert werden, um die Zeitabläufe genau mitzuverfolgen. Als weitere Hilfestellung werden die zuletzt zugegriffenen Speicherstellen farblich hervorgehoben, so dass Lese- und Schreibzugriffe während der Simulation unmittelbar deutlich werden. Der Simulator kann alternativ als eigenständiges Java-Programm oder als Applet gestartet werden:

```
java de.mmkh.tams.Prima           % Applikation
appletviewer prima.html          % Applet
```

Die Applet-Version des Simulators kann auch unter der folgenden URL aufgerufen werden: <http://tams-www.informatik.uni-hamburg.de/applets/jython/prima.html>

Assembler Zusammen mit dem neuen Simulator wurde auch ein einfacher Assembler erstellt, der die Programmentwicklung für die PRIMA gegenüber der reinen Maschinensprache erheblich erleichtert. Da die Studierenden aber auch die Programmierung auf der reinen Maschinensprache kennenlernen sollen, ist noch zu entscheiden, ob und wann dieser den Studierenden zur Verfügung gestellt wird.

The screenshot shows the PRIMA-Simulator interface. The main window displays a memory/register table with columns for PC, BR, AR, Akku, MemOut, and AluOut. The table contains 19 rows of data, with the last row (18) highlighted in yellow. The control panel on the left includes fields for PC (d9), AR (fd), BR (09), AKKU (09), OV (00), state (01), SW (checkbox), trace (checkbox), hex (checkbox), and disassemble (checkbox). The bottom of the interface has a 'Cmd>' prompt and several buttons: Takt, Befehl, 5 Befehle, Reset, RAM löschen, Laden..., and Sichern...

Abbildung 16: Simulator für die primitive Maschine (PRIMA) als Java-Applet. Neben Register- und Speicherinhalten wird auf Wunsch ein Ablaufprotokoll dargestellt; auch eine Disassemblierung der Speicherinhalte ist möglich. Die zuletzt vom Programm zugewiesenen Speicheradressen werden farblich hervorgehoben.

Als reines Kommandozeilenprogramm liest *PrimaAssembler* eine Eingabedatei (*filename.asm*) mit PRIMA-Assemblercode und generiert daraus eine Datei (*filename.rom.txt*) mit dem zugehörigen Maschinencode und ein Cross-Reference-Listing mit den verwendeten Symbolen für Adressen und Konstanten:

```
java PrimaAssembler t1_6_2b.asm      % Assemblercode-Eingabedatei
edit t1_6_2b.rom.txt                % erzeugter Maschinencode
edit t1_6_2b.symbols.txt            % zugehörige XREF-Tabelle
```

3.14 Yield-Demonstration

Parallel zur Entwicklung der Algorithmen zur Überprüfung von Übungsaufgaben entstanden im Rahmen des Projekts auch einige kleinere Applets zum Themengebiet der Technischen Informatik. Ein Beispiel dafür bietet die Java-Klasse *YieldDemo* mit den zugehörigen Klassen *YieldGUI* und *YieldApplet*.

Konzept Die Klassen demonstrieren die Ausbeute (*Yield*) bei der Halbleiterfertigung von hochintegrierten *VLSI*-Schaltkreisen, d.h. den Anteil der funktionsfähigen Schaltungen aus der Gesamtzahl der produzierten Schaltungen. Da einmal produzierte Mikrochips nicht mehr repariert werden können, reicht bereits ein einziger kleiner Defekt aus, um den ganzen Schaltkreis unbrauchbar zu machen.

Typische Fehlerursachen sind zum einen Staubkörnchen während der Belichtungs- und Ätzprozesse, die die gewünschten Strukturen stören, und zum anderen winzige Materialdefekte in den Siliziumdioxid-Isolationsschichten der verwendeten MOS-Transistoren. In beiden Fällen lässt sich die Verteilung der Fehler über die gesamte Siliziumscheibe (*Wafer*) durch eine zufällige aber gleichmäßige Verteilung von Punktdefekten schon gut annähern. Das Applet benutzt genau dieses Fehlermodell, um die typische Verteilung von Fehlern zu visualisieren.

Zur Einbettung der Yield-Demonstration in eine Webseite dient die Java-Klasse *YieldApplet*, während die Klasse *YieldGUI* auch als eigenständiges Hauptprogramm aufgerufen werden kann und die Schieberegler und Textfelder zur Parametereinstellung enthält. In beiden Fällen dient die Klasse *YieldDemo* zur Anzeige des Wafers mit den Chips und Defekten:

```
java de.mmkh.tams.YieldGUI           % Applikation
appletviewer yield.html              % Applet
```

Derzeit können beim Aufruf des Programms keine Parameter für Chipgröße oder Defektwahrscheinlichkeit übergeben werden, sondern die Werte müssen interaktiv eingestellt werden. Die Nachrüstung solcher Parameter für den Programmaufruf wäre aber bei Bedarf leicht möglich.

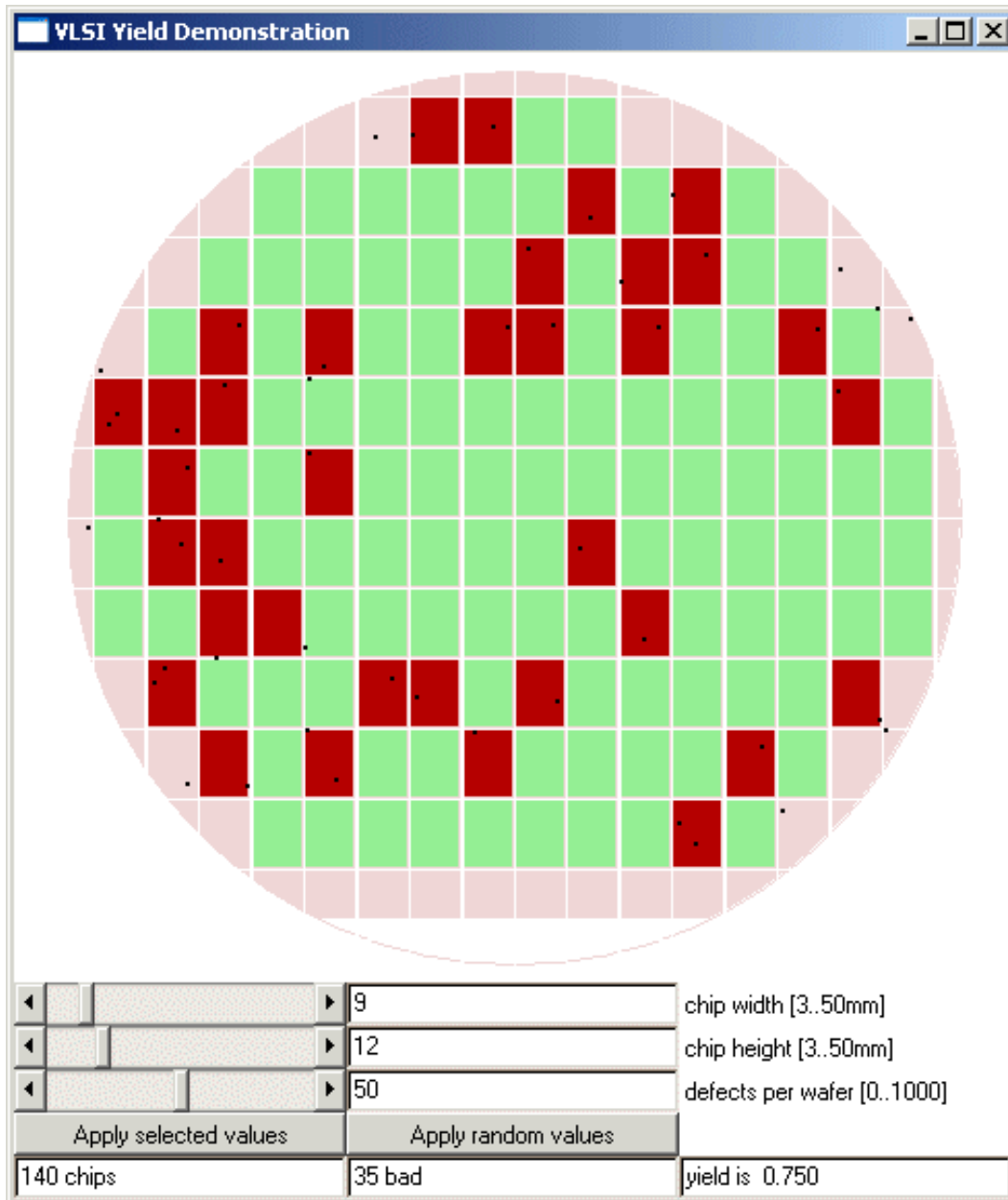


Abbildung 17: Applets zur Demonstration der Ausbeute (Yield) bei der VLSI-Chipfertigung. Über die Schieberegler können die Chipgröße sowie die mittlere Defektwahrscheinlichkeit interaktiv eingestellt werden. Das Programm berechnet dann die zugehörige neue Anordnung des Chips auf dem Wafer, erzeugt eine zufällige Menge von Defekten und berechnet daraus die jeweilige Ausbeute.

Literatur

- [1] R. E. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Transactions on Computers, C-35(8):677–691, 1986
- [2] R. E. Bryant, D. R. O’Hallaron, *Computer Systems, A Programmer’s Perspective*, Prentice Hall, 2003, ISBN 0-13-034074-X
- [3] Fachbereich Informatik der Universität Hamburg, *Studienführer Informatik 2002/2003*, www.informatik.uni-hamburg.de/
- [4] K. v. d. Heide, *Vorlesung Technische Informatik 1*, Universität Hamburg, FB Informatik, WS2002, tams-www.informatik.uni-hamburg.de/lehre/ws2002/t1Vor/
- [5] K. v. d. Heide, *Vorlesung Technische Informatik 2*, Universität Hamburg, FB Informatik, SS2003, tams-www.informatik.uni-hamburg.de/lehre/ss2003/vorlesungen/T2/
- [6] 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/
- [7] 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/
- [8] N. Hendrich, *Hades Tutorial*, tams-www.informatik.uni-hamburg.de/applets/hades/archive/tutorial.pdf
- [9] 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
- [10] N. Hendrich, *Automatic checking of students’ designs using built-in selftest methods*, Proc. 3rd European Workshop on Microelectronics Education, EWME-2002, Baiona, 2002
- [11] N. Hendrich, K. v.d.Heide, *Automatische Überprüfung und Hilfestellung zu Vorlesungs-begleitenden Übungsaufgaben*, Projektbericht, Multimedia-Kontor Hamburg, 2003
- [12] N. Hendrich, K. v.d.Heide, *Automatische Überprüfung und Hilfestellung zu Vorlesungs-begleitenden Übungsaufgaben*, Zweiter Projektbericht, Multimedia-Kontor Hamburg, 2004
- [13] N. Hendrich, K. v.d.Heide, *Automatische Überprüfung und Hilfestellung zu Vorlesungs-begleitenden Übungsaufgaben*, Dritter Projektbericht, Multimedia-Kontor Hamburg, 2004
- [14] N. Hendrich, *Check*Applets*, Software-Handbuch und Tutorial, Multimedia-Kontor Hamburg, 2005
- [15] M. Mayer, *Konzeption und Umsetzung eines Java-Applets zur Logikminimierung mit KV-Diagrammen*, Studienarbeit, Fachbereich Informatik, 1998
- [16] J. L. Hennessy, D. A. Patterson, *Computer organization and design: the hardware/software interface*, Morgan Kaufmann, 1998, ISBN 1-558-60491-X

-
- [17] J. Hugunin, *Python and Java — The Best of Both Worlds*, Proc. 6th International Python Conference, San Jose, 1997,
- [18] D. Jansen (Hrsg.), *Handbuch der Electronic Design Automation* Hanser, 2001 ISBN 3-446-21288-4
- [19] *Jython Environment for Students*, Georgia Institute of Technology, 2002 <http://coweb.cc.gatech.edu/cs1315/814>
- [20] Jython project homepage, www.jython.org
- [21] J.W. Eaton and others, *GNU Octave Project*, www.octave.org
- [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] A. S. Tanenbaum, *Structured Computer Organization, 4th. Edition*, Prentice Hall, 1999 ISBN 0-13-020435-8
- [30] imc information multimedia communication AG, *Clix 4 Campus* Lernplattform, http://www.im-c.de/homepage/clix_campus.htm
- [31] World wide web consortium, *Extensible Markup Language (XML)*, <http://www.w3.org/XML/>
- [32] P. Hubwieser, *Didaktik der Informatik*, Springer 2000, ISBN 3-540-43510-7
- [33] H. Wojtkowiak, *Test und Testbarkeit digitaler Schaltungen*, Teubner 1988, ISBN 3-519-02263-X
- [34] N. Freed, N. Borenstein, *Multipurpose Internet Mail Extensions — MIME, Part One: Format of Internet Message Bodies*, Network Working Group, RFC 2045, www.ietf.org/rfc/rfc2045.txt
- [35] D. Raggett, A. Le Hors, I. Jacobs, Eds. *World-wide web consortium, HTML 4.01 Specification*, W3C Recommendation 24 December 1999, <http://www.w3.org/TR/html401>
- [36] S. Emmerson, *Java Specification Requests, JSR 108: Units Specification*, <http://www.jcp.org/en/jsr/detail?id=108>, <http://jade.dautelle.com>
- [37] A. Ruge, *Ein HTML-Browser für interaktive Lehrbücher*, Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 2004

-
- [38] K. Schneider, *Verificatin of reactive systems: formal methods and algorithms*, Springer, 2004
 - [39] B. Steffen, G. Levi (eds.), *Verification, Model Checking, and Abstract Interpretation*, Proc. VMCAI 2004, LNCS 2937, Springer 2004, ISBN 3-540-20803-8
 - [40] W. Menzel, *INCOM – Inputkorrektur durch Constraints und Markups*, E-Learning Consortium Hamburg Projekt, Fachbereich Informatik, Universität Hamburg, 2003
 - [41] M. Sommer, *Inside CPU*, E-Learning Consortium Hamburg Projekt, Hamburger Universität für Wirtschaft und Politik, 1997
 - [42] Y. Nahapetian, *Simulation eines von-Neumann-Rechners in der Programmiersprache Java*, Studienarbeit, Fachbereich Informatik, Universität Hamburg, 1996
 - [43] Lattice Semiconductor Corp., *GAL 16V8 datasheet*, Hillsboro, Oregon, 2004
 - [44] T. Lindholm, F. Yellin, *Java Virtual Machine Specification, 2nd Edition*, Addison-Wesley, 1999
 - [45] C. Shannon, *The synthesis of two-terminal switching circuits*, Bell Syst.Techn.J.28, 59-98 (1949)
 - [46] I. Wegener, *The Complexity of Boolean Functions*, Wiley-Teubner, 1987, <http://ls2-www.cs.uni-dortmund.de/monographs/bluebook>
 - [47] Arash Vahidi, *JDD, a Java Binary Decision Diagram Library*, <http://javaddlib.sourceforge.net/jdd/intro.html>
 - [48] *Java compiler compiler - the Java parser generator*, <https://javacc.dev.java.net/>