Automatische Überprüfung und Hilfestellung zu Vorlesungs-begleitenden Übungen

< HTML-Integration und Applets >

Norman Hendrich

Universität Hamburg, Fachbereich Informatik

Vogt-Kölln-Str. 30, D 22527 Hamburg

hendrich@informatik.uni-hamburg.de

`tams-www.informatik.uni-hamburg.de/forschung/interaktives-skript/`

Übersicht

Einführung

- Das interaktive Skript
- Integrierte Übungsaufgaben
- Beispiele, State of the art

Interaktive Skripte in HTML

- Konzept und Varianten
- mscript2html-Konverter
- Matlab für jedermann

Die Check*Applets

- Konzept
- Beispiele

Zusammenfassung, Diskussion

Was bisher geschah...

bisherige Vorträge zur Thematik:

○ "Das interaktive Lehrbuch" (Matlab)	K.v.d. Heide	WS'2003
○ "Projektvorschau: Überprüfung..."	N. Hendrich	WS'2003
○ "Ein HTML-Browser für interaktive..."	A. Ruge	SS'2004

behandelte Themen:

- Konzept des interaktiven Lehrbuchs und integrierter Übungen
- Klassifikation der T-Übungsaufgaben

- Behandlung von Multiple-Choice- und Zahlenwertaufgaben
- Überprüfung digitaler Schaltungen via Simulation und BIST-Analyse
- Klartext-Problem: der Anwender hat alle Quelltexte

Das interaktive Skript

bzw. "interaktives Lehrbuch"

- erläuternde Texte, eingebettete Formeln
- eingebettete Medien: Graphiken, Animationen, Audio, Video
- Hyperlinks: im Skript, auf Webseiten, auf externe Programme

- eingebettete aktive Applets (mit GUI)
- eingebetteter Skriptcode
- eingebettete Übungen (mit sofortiger) Überprüfung

- jederzeit erweiterbar (auch von den Studierenden)
- und später im Berufsleben produktiv nutzbar

- einfache Content-Erstellung

Ziele des ELCH-Projekts

- Übungsaufgaben im Skript integriert:

Unterstützung der Studierenden:

- geringere Hemmschwelle zur Bearbeitung der Aufgaben
- automatische Überprüfung der Lösungen
- sofortiger Feedback (nicht erst eine Woche später)

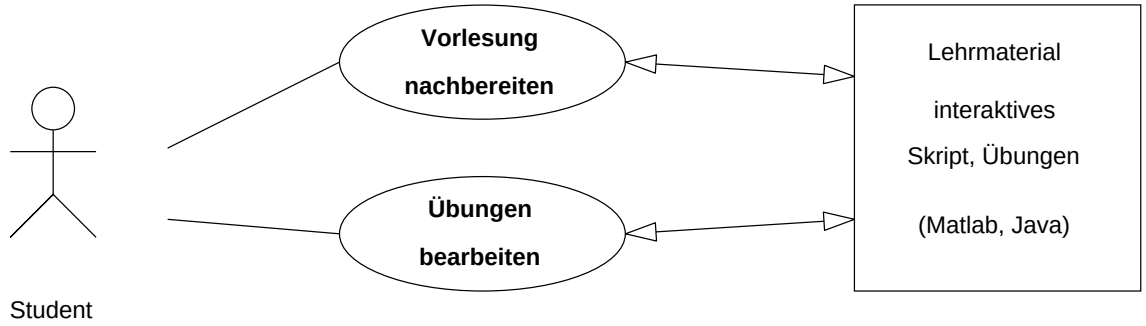
- kontextabhängige Hilfestellungen
- gezielte Gegenbeispiele helfen bei der Fehlersuche

Unterstützung der Übungsgruppenleiter

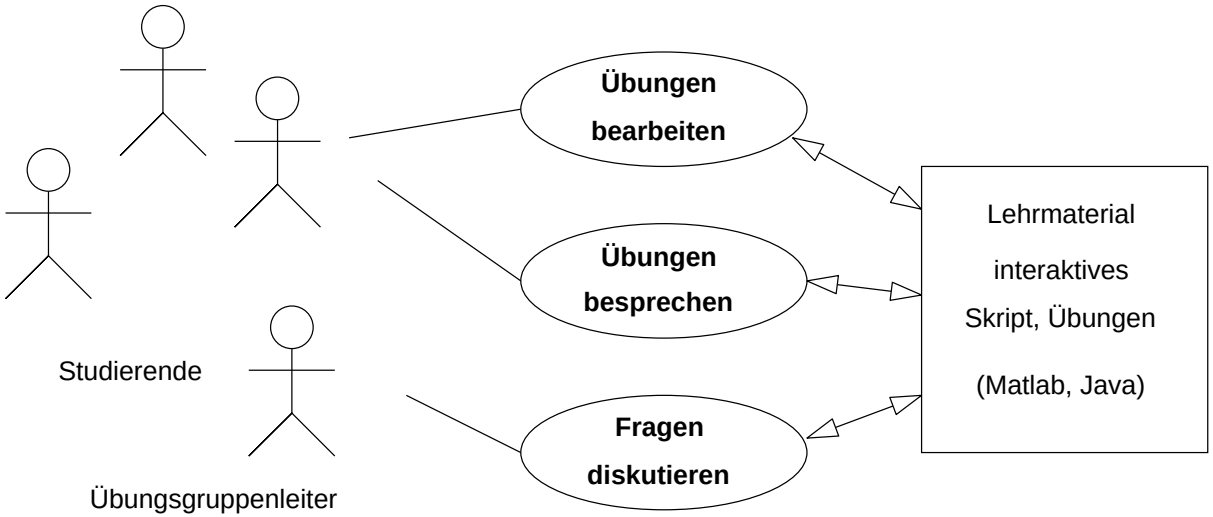
- erleichtert das "Ausprobieren" während der Übungsstunden
- automatische (Vor-) Korrektur vieler Aufgaben

Use-Cases

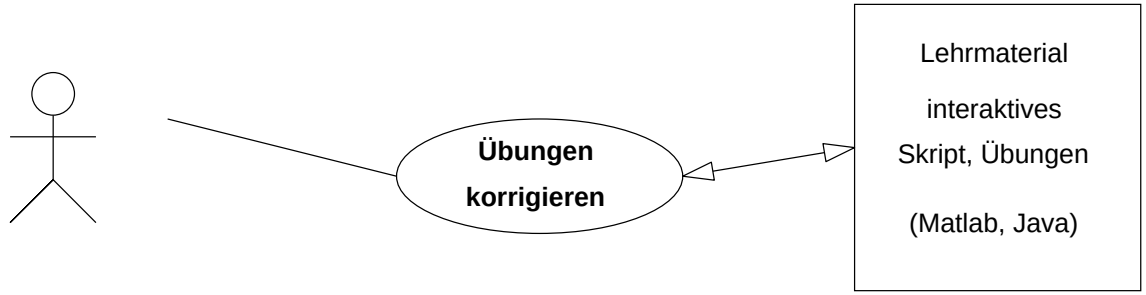
a)



b)



c)



Was ist mit den E-Learning Plattformen ?

State of the Art: diverse HTML "E-Learning Plattformen":

- statische HTML-Texte, zugehörige "content editors"
- alternativ auch als Windows (oder Mac) Applikationen
- lediglich minimale Interaktions/Selbsttestmöglichkeiten:

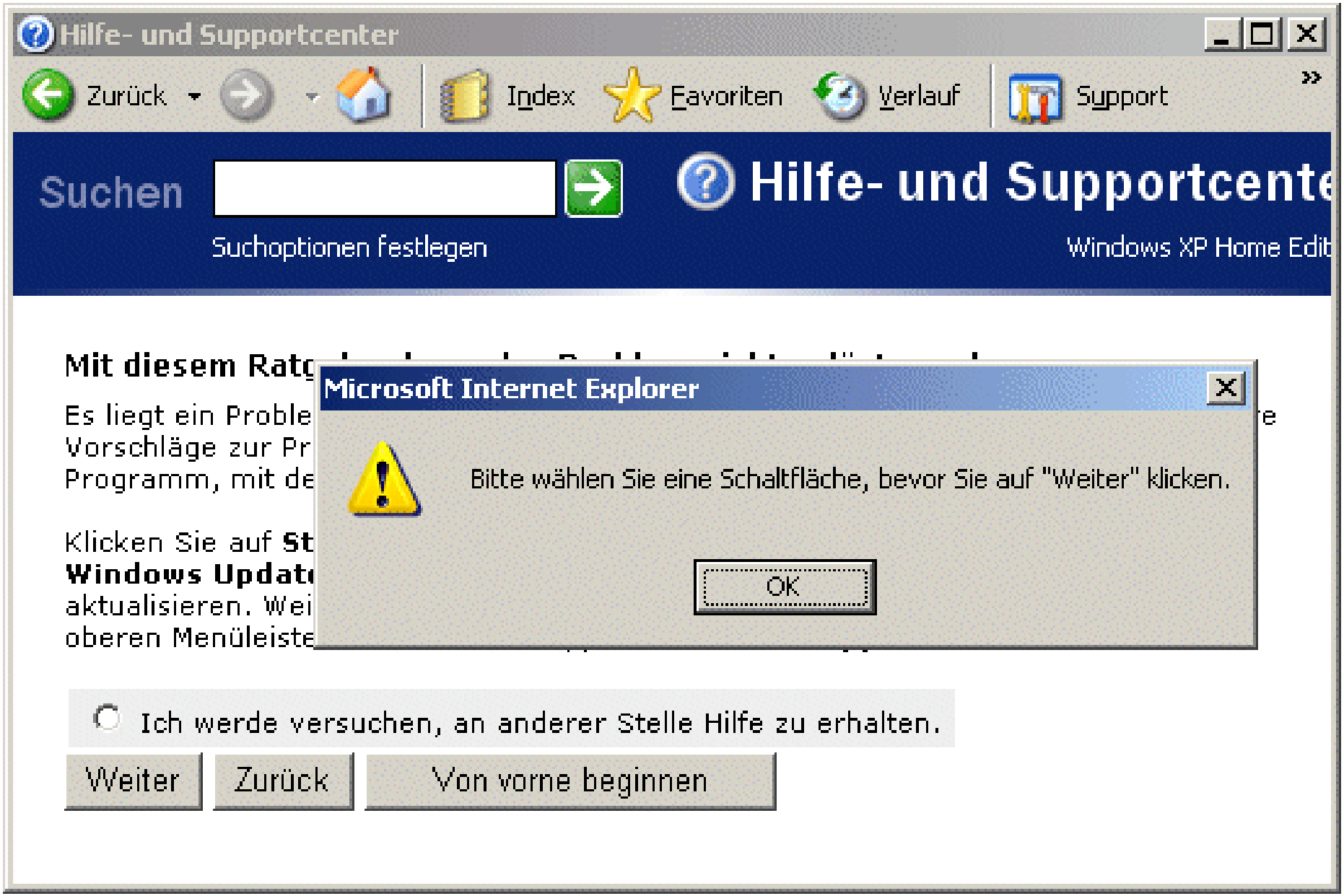
  - multiple-choice (via HTML-Forms, server-basiert)
  - numerische Werte (via HTML-Forms, server-basiert)
  - Texteingaben (Auswertung durch Betreuer)

=> unzureichend für die technische Informatik

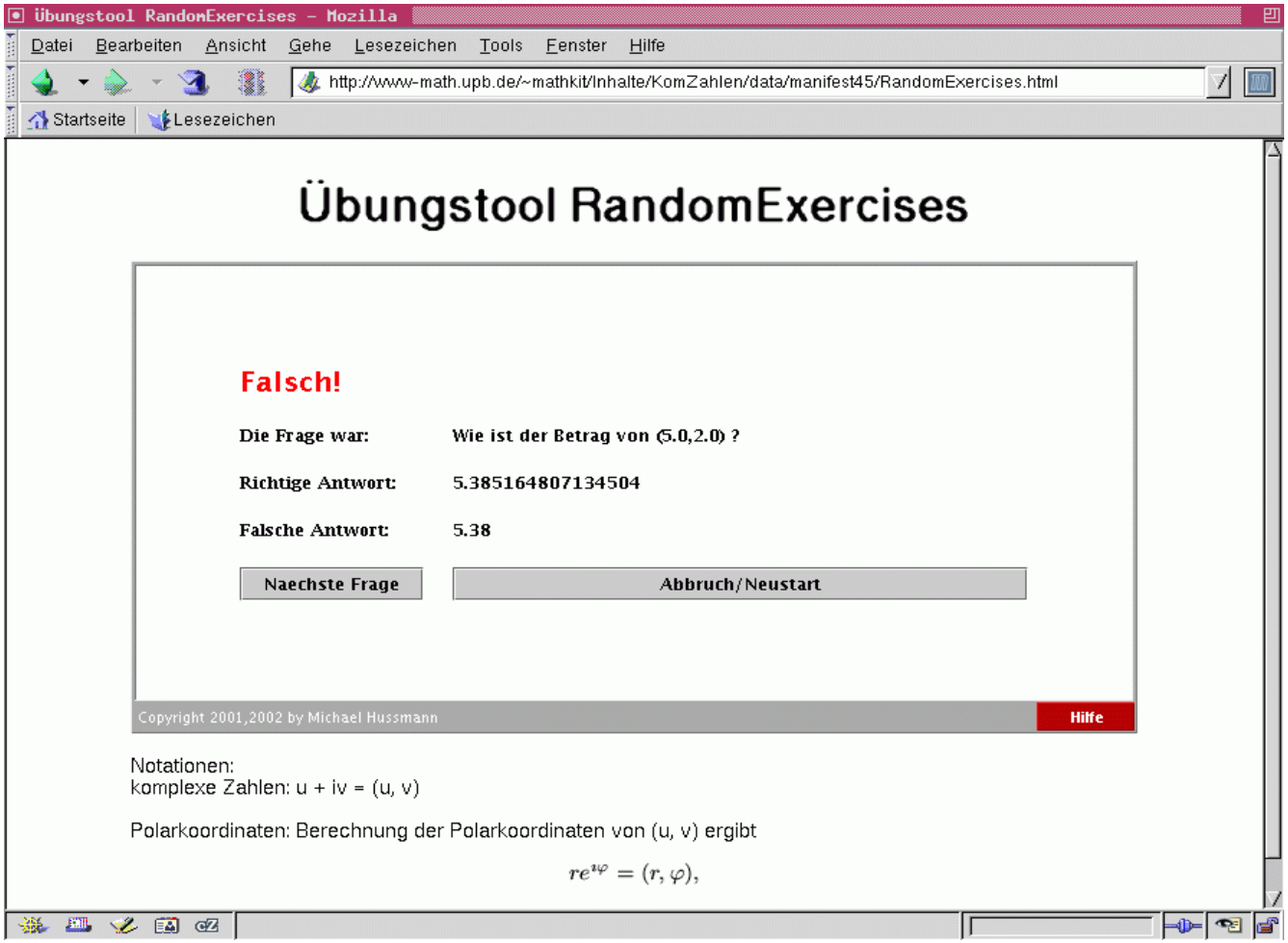
positive Ausnahmen:

  - Sprachkurse (Aussprachetests, usw.)
  - Programmieraufgaben (Compiler/Interpreter/Testläufe)

Beispiel: Multiple-Choice mit Hilfe

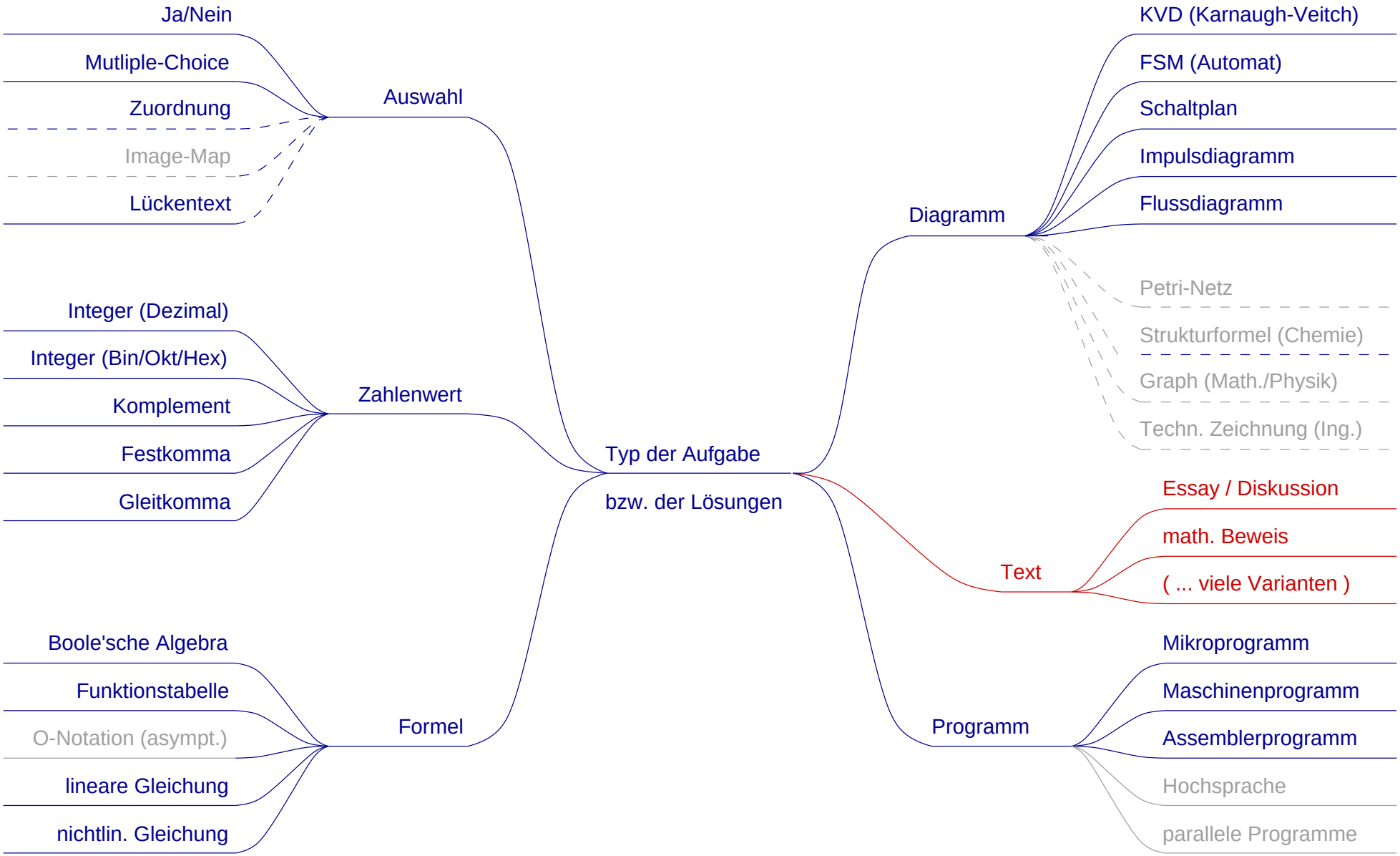


Beispiel: Zahlenwerte



- Math-Kit Demo, zufällig gestellte Aufgaben
- externer Taschenrechner nötig
- erwartet (implizit) > 5 Stellen Genauigkeit

T-Übungsaufgaben: Klassifikation



Interaktive Skripte in HTML

Einführung

- Das interaktive Skript
- Integrierte Übungsaufgaben
- Beispiele, State of the art

Interaktive Skripte in HTML

- Konzept und Varianten
- mscript2html-Konverter
- Matlab für jedermann

Die Check*Applets

- Konzept
- Beispiele

Zusammenfassung, Diskussion

Interaktive Skripte in HTML

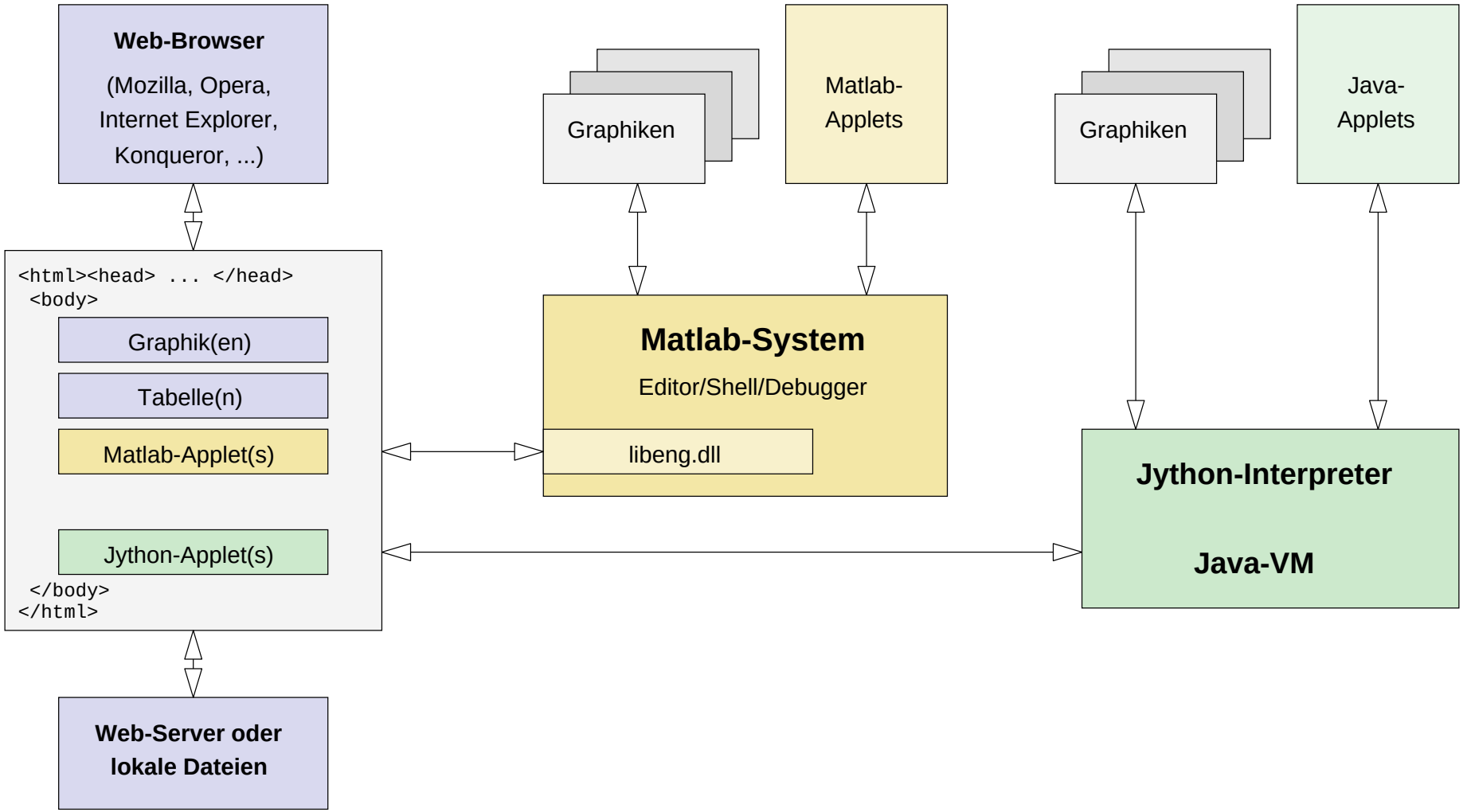
- formatierter Text, Hyperlinks, eingebettete Graphiken
- Formeln über HTML-Symbole oder MathML
- aktive Elemente über JavaScript, Java, Flash

- Nutzer findet die gewohnte Umgebung vor (z.B. MSIE, Mozilla)
- Text ist auch ohne aktive Elemente lesbar
- Komfortfunktionen bereits vorhanden: Bookmarks, Drucken, ...
- Hyperlinks zu externen Webseiten, Suchmaschinen, etc.
- einfache Integration in Web-/Elearning-Plattformen

Nachteile und Probleme:

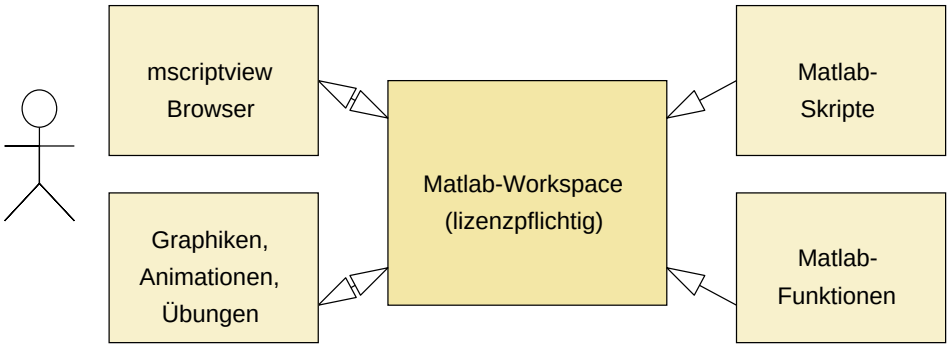
- Kompatibilität/Performance/Stabilität?
- Applet-Restriktionen (Sandbox-Konzept)
- Anbindung an externe Applikationen, insb. Matlab?
- Content-Erstellung?

Interaktive Skripte in HTML: Prinzip

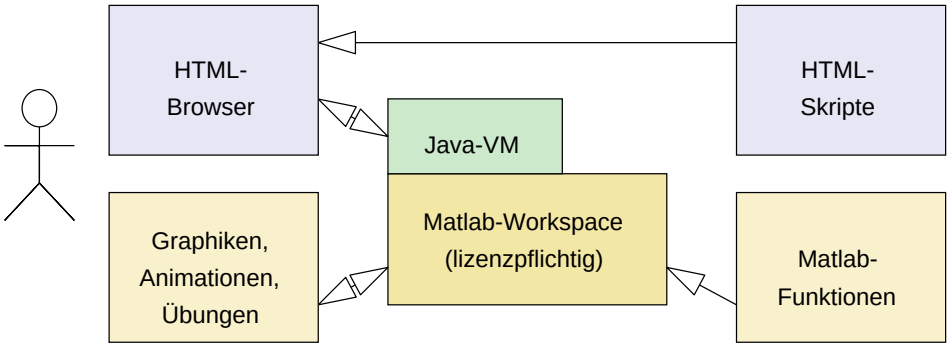


Architekturvarianten

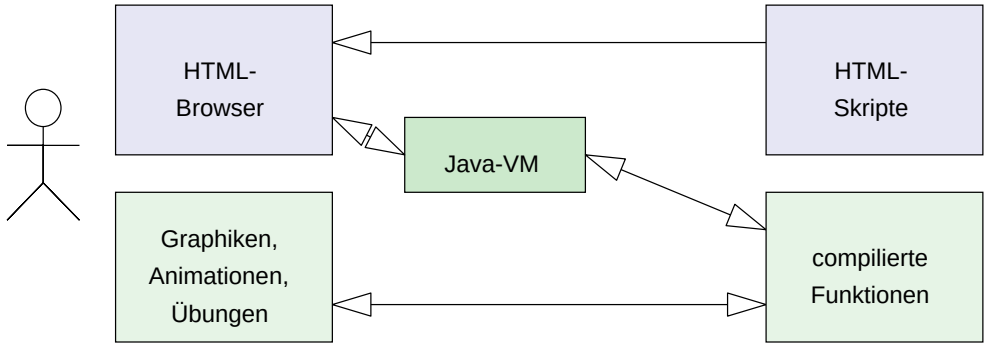
a) mscriptview + Matlab



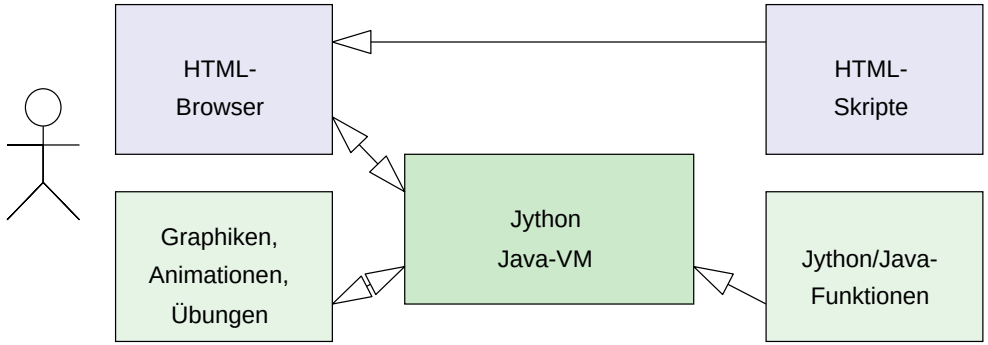
b) Webbrowser + Matlab



c) Webbrowser + externe Applikationen (z.B. MCR)



d) Webbrowser + Jython/Java



mscript2html-Konverter

- Einlesen der (.m) Quelldateien
- Heuristik zur Analyse der Textstruktur, insbesondere zur
- Trennung aktiver Elemente (Matlab-Code) von statischem Text

- Erzeugen von Java-Applets für alle aktiven Elemente
- Übergabe des eigentlichen Codes als Applet-Parameter

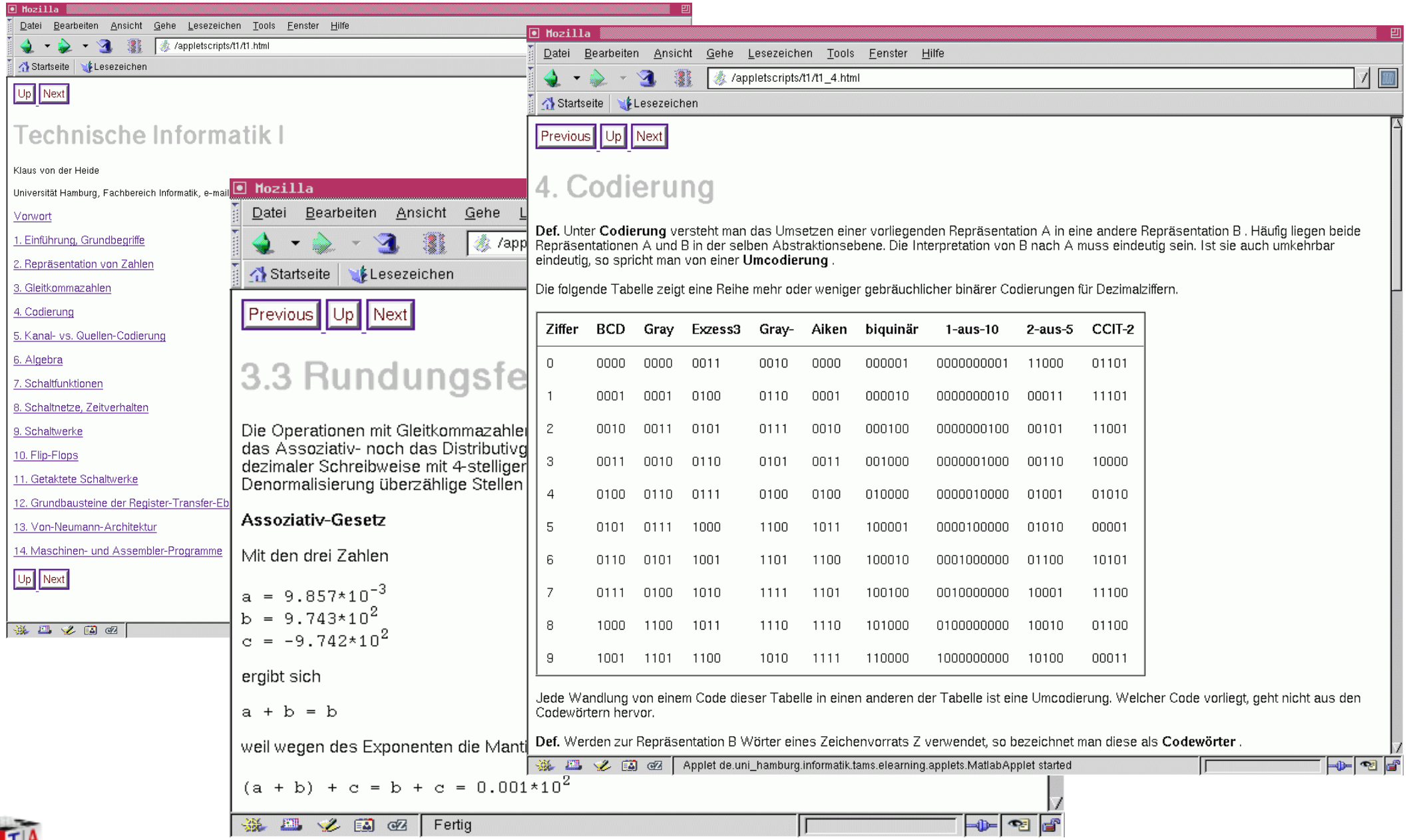
- Umsetzung der statischen Elemente nach HTML, inkl.
- Fontauswahl, Formatierungen, Tabellen, Blocksatz

- aufwendige Heuristik zur Umsetzung von Formeln
- drei Varianten: HTML, "TeX"-Applets, eingebettete GIF-Bilder

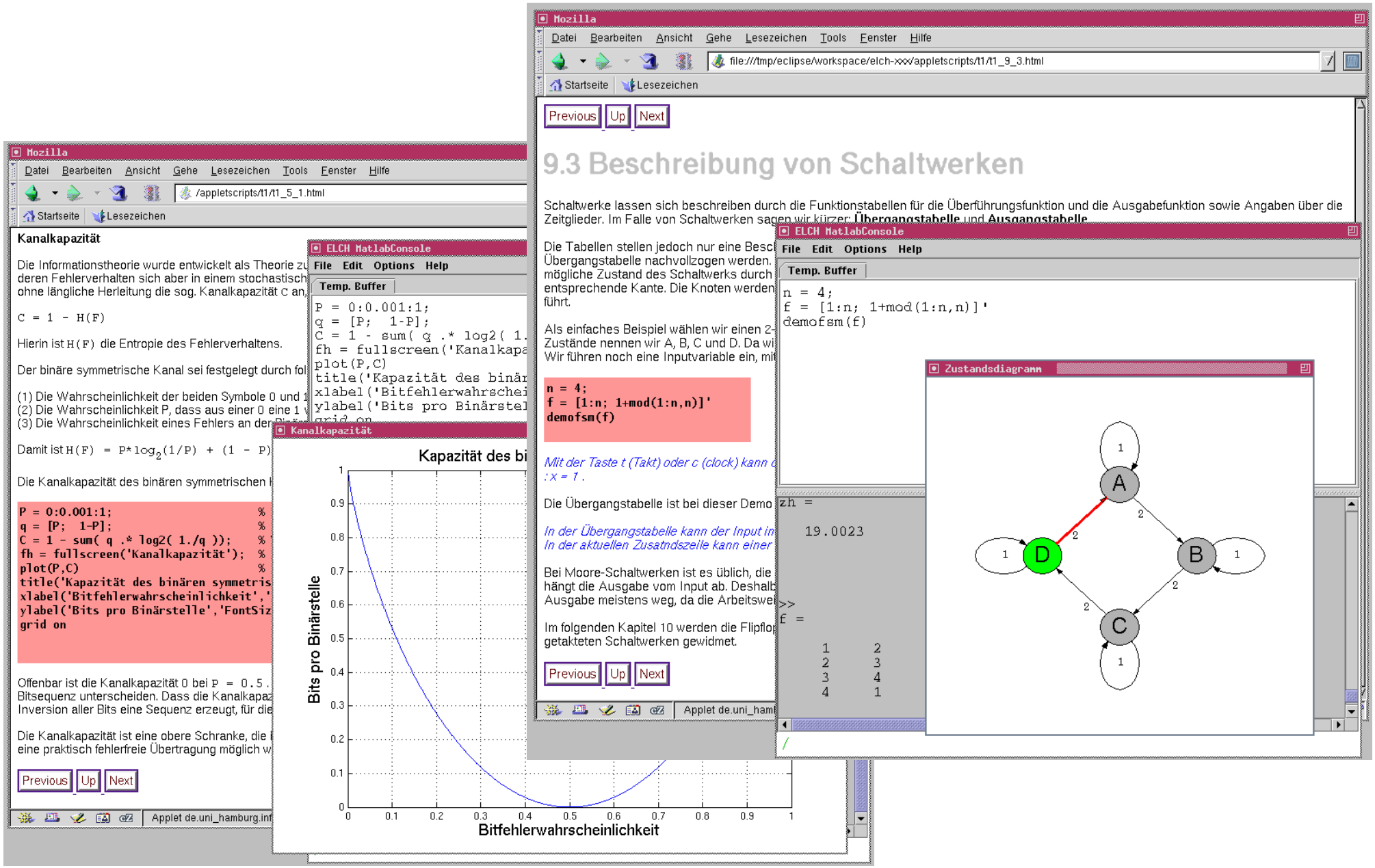
- Erkennung von Links und Umsetzung nach HTML
- Erzeugen von Indexseite und Navigations-Links

- T1/T2/DSP/NT-Skripte konvertiert (Demo)

mscript2html: Beispiele



mscript2html: Beispiele



MatlabApplet

Wie kann Matlab von Applets aus aufgerufen werden?

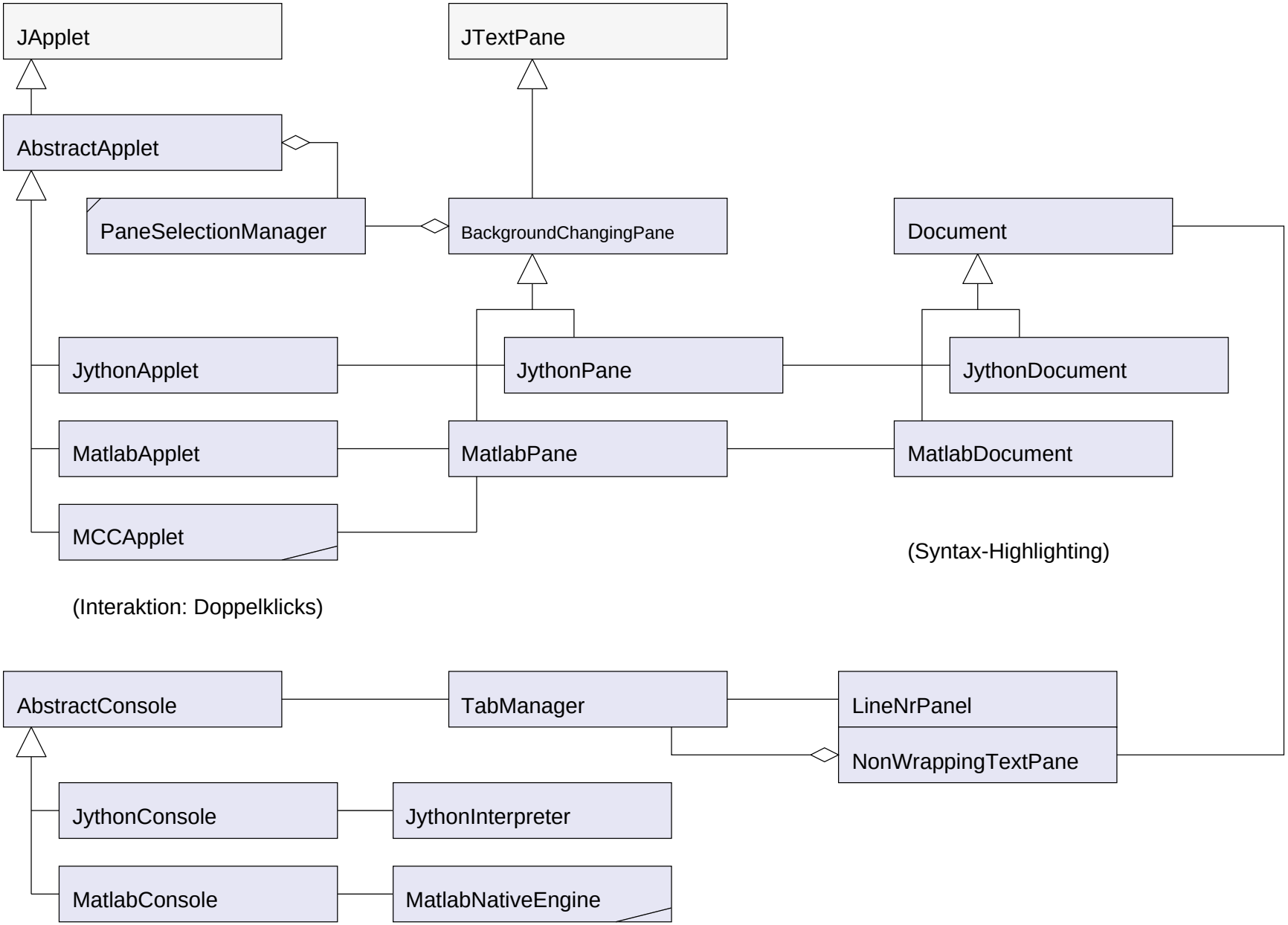
- Aufruf von Applikationen via System.exec(program, args[])
- aber Parameterübergabe nur beim ersten Aufruf

- Zugriff auf Matlab-Kern via C-Interface ("libeng.dll"/"libeng.so")
- `open() / close() / evalString() / getOutputString()`
- Realisierung mit separatem Serverprozess und tcp-Protokoll
- Doppelklick im Applet <-> Server <-> JNI <-> Matlab

- separate GUI-Klassen für Applets und Console
- jeweils Matlab- und Jython-Version mit Syntaxhighlighting

- entsprechend komplexe Klassenhierarchie...

Console/Applets: Klassenhierarchie



Matlab-Compiler

- Matlab R14 benutzt JIT-Compiler für m-Skripte/m-Funktionen
- bisheriger Compiler (.m -> .c -> .exe) überflüssig
- durch komplette Neuentwicklung ersetzt

aktueller Matlab-Compiler:

- Erzeugen von standalone Applikationen aus m-Funktionen
- lizenzfrei (gegenüber Mathworks)
- Intellectual Property Schutz (Binärcode statt m-Code)

- Unterstützung des gesamten Sprachumfangs, inkl. eval()
- Unterstützung aller Rechenfunktionen
- Unterstützung von handle-graphics und GUIs
- Unterstützung fast aller Toolboxes (außer symbolic math)

Matlab-Compiler: Prinzip

- Abhängigkeitsgraph der User-Funktion(en)
- Verschlüsseln aller benötigten Funktionen (3DES?)
- Abspeichern als .ctf-Datei, "component technology file"
- Erzeugen eines .exe-Wrappers zum Entschlüsseln des CTF

- zugehörige MCR, "Matlab component runtime"
  - plattformabhängiges ZIP-Archiv (bzw. Windows-Installer)
  - alle benötigten DLLs für den Matlab-Kern / die Toolboxes
  - alle benötigten m-Funktionen (verschlüsselt)
  - inklusive Interpreter/JIT-Compiler (eval)
  - aber ohne Workspace, Editor, Debugger, Help

- MCR einmal auf Rechner des Endanwenders installieren
- Paar aus .exe/.ctf liefern und ausführen

mccwrapper / result2fig

- compilierte Funktionen laufen ohne Workspace
- Parameterübergabe?
- Behandlung / Anzeige der Rückgabewerte?

- Mathworks dokumentiert C-Wrapper / aber extrem aufwendig
- Umschreiben aller Funktionen kommt nicht in Frage

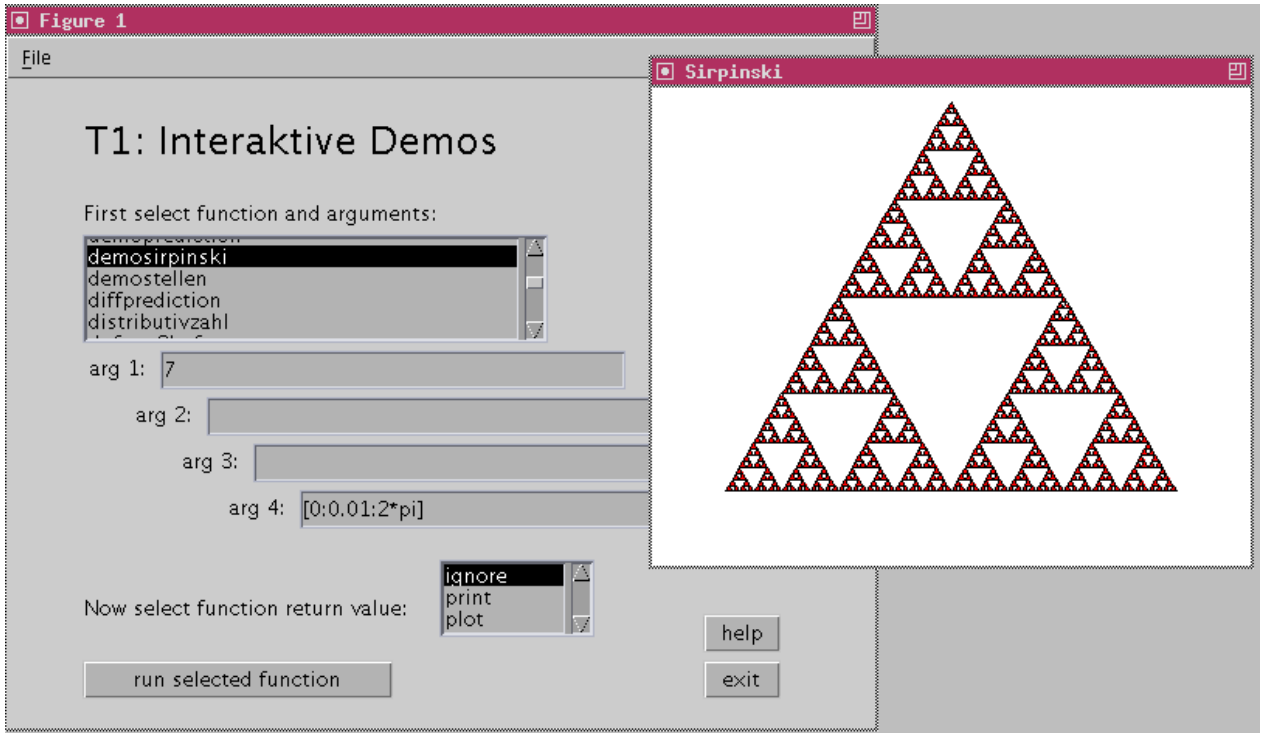
=> mccwrapper.m / result2fig.m:

- Matlab-Wrapper-Funktion, referenziert alle relavanten Funktionen
- Übergabe von "Modus", "Funktionsname", "Parametern" als Strings
- Konvertieren der Strings in Matlab-Objekte via eval()
- Aufruf der gewählten Funktion
- Anzeige des Rückgabewerts als Textfenster bzw. via result2fig

```
mccwrapper.exe print demofsm "[1:4; 1+mod(1:4,4)]'"
```

t1runner

- Parameterübergabe an compilierte Funktionen?
- Schachtelung von Funktionsaufrufen?
- Startup-Delay bei Aufruf jeder neuen Funktion?
- Hilfe zu den Funktionen?



=> t1runner.exe:

- Matlab-GUI zum Aufruf der T1-Funktionen
- Auswahl der Funktion, dann (bis zu) vier Argumente
- Eingabe der Argumente als Strings in Matlab-Syntax, Arrays möglich
- Ausführen der Funktion, Resultat als Graphik oder Textfenster
- Wiederholung der Schritte: "workspace light"

t1server

- Applet-Anbindung der compilierten Funktionen?

einfachste Variante:

- Aufruf der Funktion via System.exec("funcname.exe", [args])
- leicht zu implementieren
- aber Startup-Delay bei jedem Funktionsaufruf (neuer MCR-Prozess)

einzige Alternative:

- Starten eines Serverprozess via System.exec("t1server.exe", [args])
- anschließend Netzwerkzugriff auf den (laufenden) Server
- Übergabe von Funktionsnamen und Argumenten vom Applet
- erfolgreicher Test mit tcp_udp_ip-Toolbox und mcc-Demoversion
- Lizenzfragen noch nicht geklärt

*Die Check*Applets*

Einführung

- Das interaktive Skript
- Integrierte Übungsaufgaben
- Beispiele, State of the art

Interaktive Skripte in HTML

- Konzept und Varianten
- mscript2html-Konverter
- Matlab für jedermann

Die Check*Applets

- Konzept
- Beispiele

Zusammenfassung, Diskussion

*Check*Applets: Konzept*

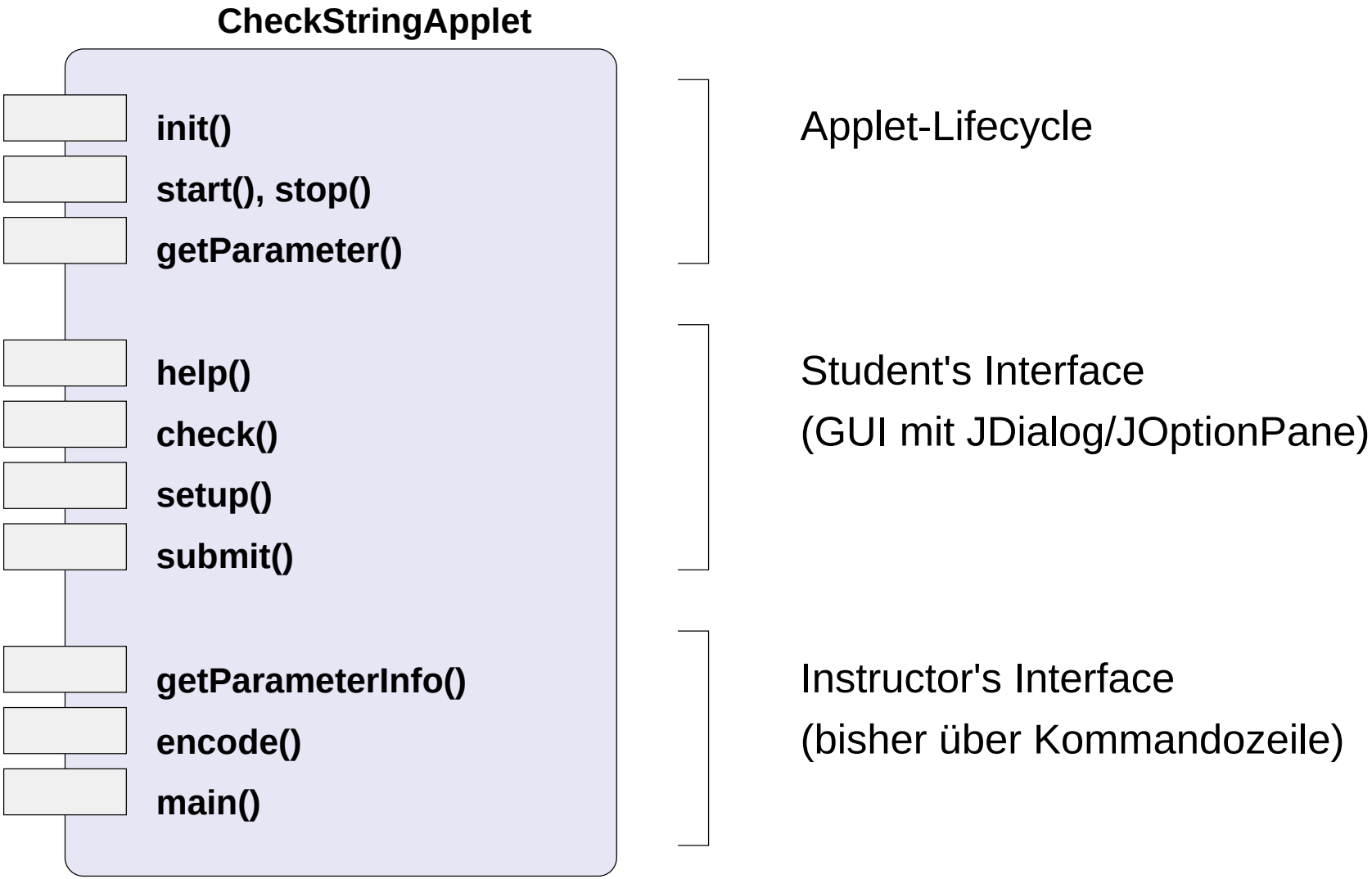
- Einsatz des Projekts im T-Zyklus ab WS'2004 fest vorgesehen
- aber Applet/Console-Software noch nicht "foolproof"
- Matlab-Lizenzkosten ein echtes Problem (Grundstudium)
- Matlab-Compiler nicht rechtzeitig verfügbar
- Bedenken wegen Downloadgröße (MCR 80MB, Jython 2MB)
- Recycling der T1-Aufgaben, keine Anpassung an Überprüfbarkeit

=> last-minute Änderung: zusätzliche "low cost" Applets

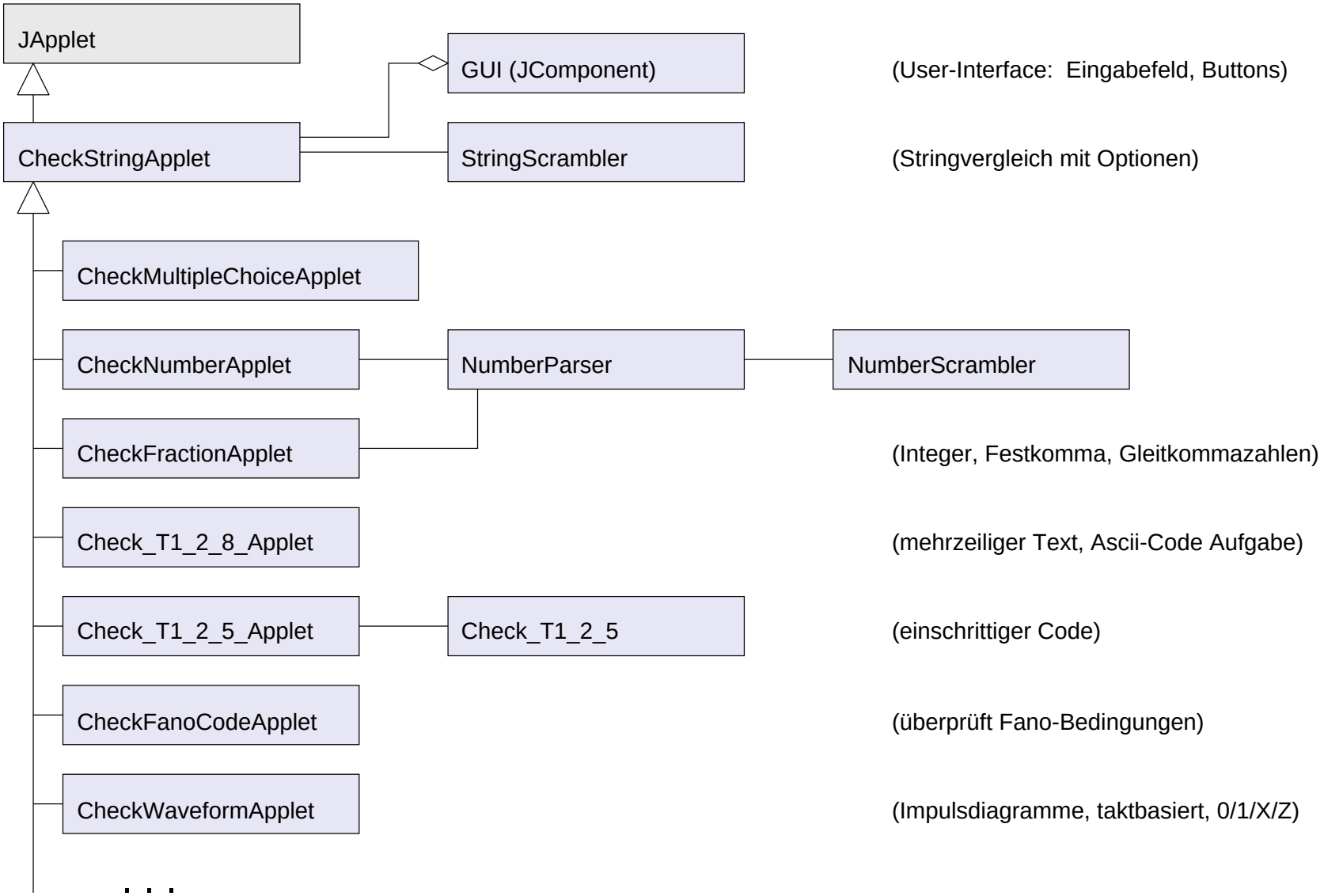
- einfache Java-Applets, möglichst kompakter Download
- Applet zur 1:1 Stringüberprüfung als Notnagel
- separate Applets für die diversen Aufgabentypen

- aber keine interaktive Umgebung

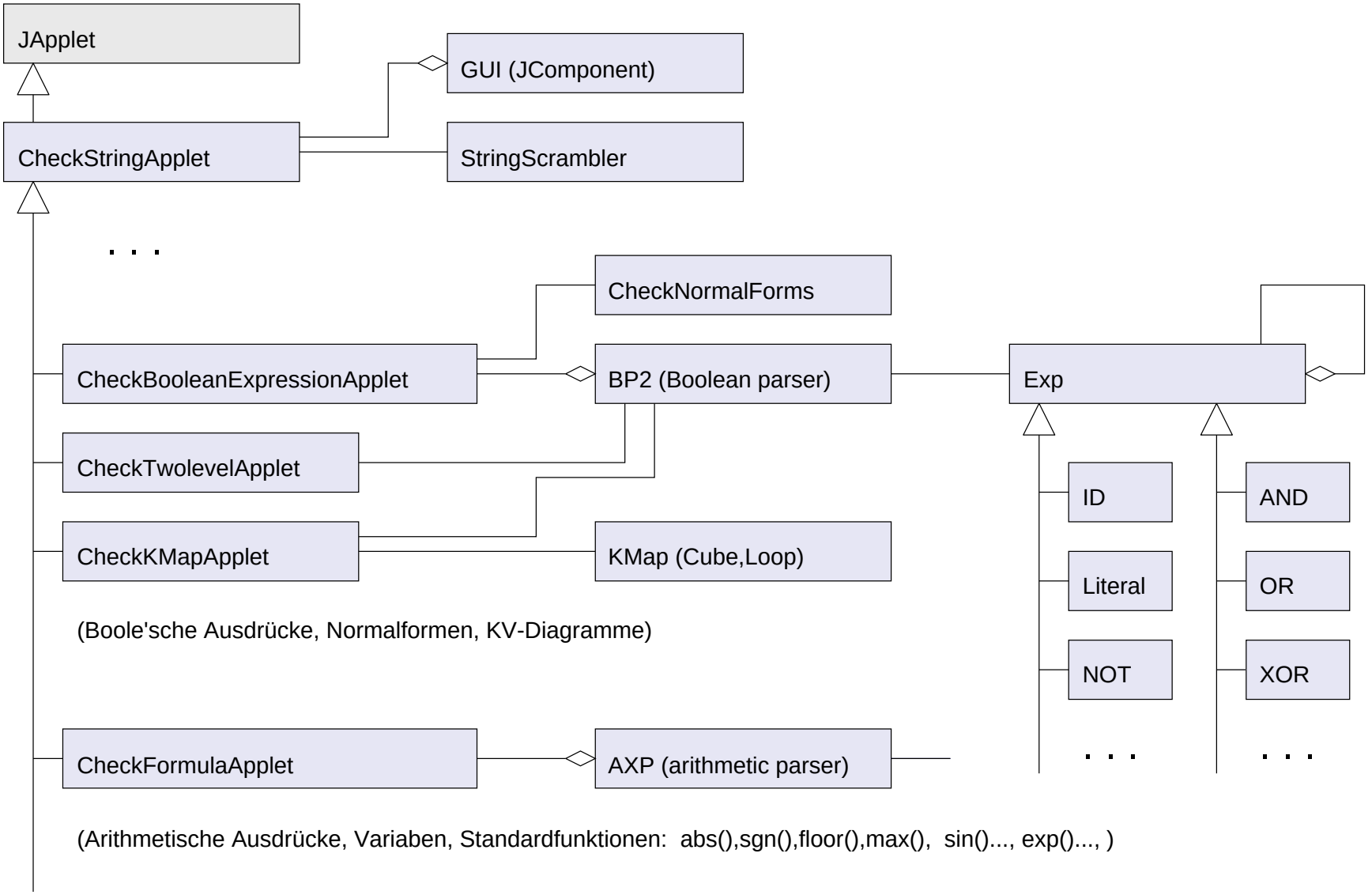
*Check*Applet API*



*Check*Applets: Klassenhierarchie (1)*



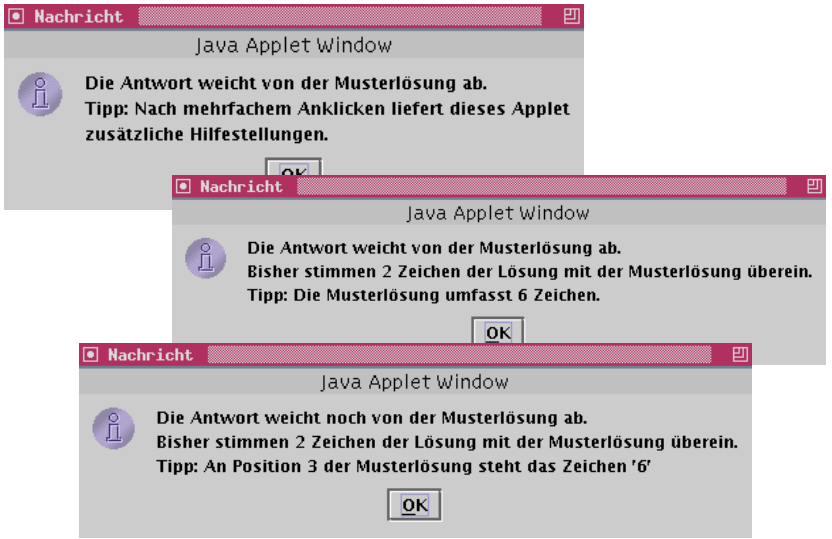
*Check*Applets: Klassenhierarchie (2)*



CheckStringApplet

- Basisklasse für die übrigen Applets
- Applet-Lifecycle, GUI, Hilfsfunktionen
- eingeschränkte I18N Unterstützung für die GUI

- Vergleich der Eingabe mit vorgegebener Musterlösung
- Optionen: ignore-case, strip-spaces, purge-spaces
- einzeilige oder mehrzeilige Eingabe, Unicode möglich
- Hilfestellung als "overlap" von Eingabe und Musterlösung



StringScrambler

- Studenten haben Zugang zum HTML/Skript-Quellcode
- Applet-Parameter sind im Klartext zugänglich

=> ganz so einfach sollte man nicht an die Lösung gelangen

- Erschwerung mit obfuscated Methoden
- Variante der Base64-Kodierung
- Ausgabe enthält nur [0-9a-zA-z+/=]
- gut für Applet-Parameter / XML-Kodierung / etc. geeignet
- analog NumberScrambler für Integer/Long/Double-Werte

```
Viel\nSpaß\nbeim\nLösen\nder\nAufgaben!
k100qRQirVWuXVY0plDvieoColHvoVkBXTUEo1swn1k9Xf==

00 01 100 101 110 1110 1111
bSPvbSTvbiP/XST/bhPAbiPvbiTAbRPAbiTA
```

*Check*Applets: der allererste Feedback...*

```
Date: Thu, 28 Oct 2004 13:32:25 +0200
From: S... H...
To:   Norman Hendrich <hendrich@informatik.uni-hamburg.de>
Subject: T1 Übungen

Hallo Norman,
...
Bezüglich der, zur Selbstkontrolle eingesetzen, Applets habe ich
allerdings meine Bedenken: Es kostet nur wenige Augenblicke das
JAR-Applet zu dekompilieren und die decode(..)-Methode in der
StringScrambler Klasse zu finden (wo doch der applet-Parameter
für den codierten Lösungswert auch noch "scrambled" heißt).

Ich würde vorschlagen, in den check-Methoden der Appletklassen
jeweils die kodierten Werte anstatt der dekodierten zu vergleichen.
Dadurch  wird die decode-Methode im StringScrambler überflüssig.
...
```

- Tradeoff: Sicherheit vs. Content-Erstellung vs. Hilfestellung
- Vergleich der kodierten Strings ist jetzt implementiert...
- bei Bedarf: SH4/MD5/etc. - aber dann keine Hilfestellung

*Check*Applets: Hinter den Kulissen*

```
<html><head><title>Übungen zu T1, Aufgabenblatt 3</title></head>
<body>

<h3>Aufgabe 3.2 (Informationstheorie)</h3>
<h4>3.2.a</h4>
Berechnen Sie den Entscheidungsgehalt (in bit,
auf zwei Nachkommastellen genau):
<br>
<applet code=de.mmkh.tams.CheckNumberApplet.class codebase="."
 archive="checkapplets.jar" width="600" height="70"
>
<param name="default"   value="x.yy" />
<param name="format"    value="double" />
<param name="tolerance" value="0.01" />
<param name="value"     value="4" />
<param name="scrambled" value="NC4wMCA="/ >
</applet>
<p>

...
</body>
</html>
```

Defaultwert für das Eingabefeld

Parameter für den Zahlenwertparser

"verschlüsselter" Wert der Lösung

*Check*Applets: Content Creation*

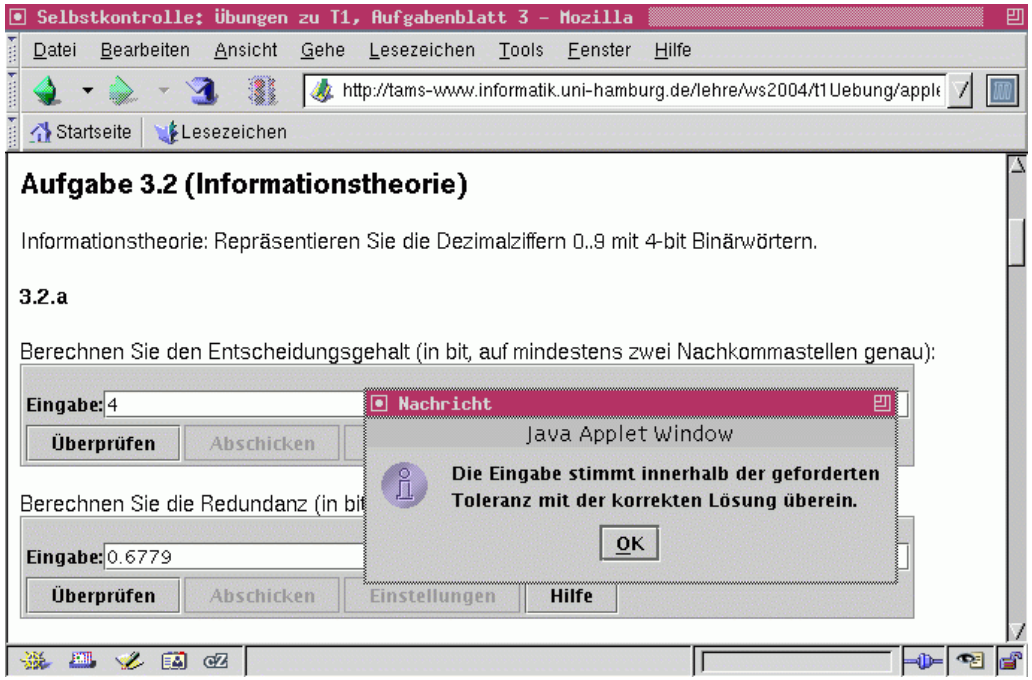
- Applets verfügen über main() und encode() Methoden
- evtl. zusätzliche Parameter, abhängig vom Aufgabentyp:

```
java CheckStringApplet -encode "musterlösung"
java CheckStringApplet -oneway "musterlösung"
```

- bisher keine weitere Unterstützung
- passender Editor wäre aber leicht möglich:
  - Auslesen der Parameter über getParameterInfo()
  - Erzeugen des zugehörigen HTML <applet>-Blocks
  - Berechnung der erforderlichen Größe width x height
  - Eingabe der Musterlösung und Aufruf von -encode
- evtl. Integration in geeignetes HTML-Framework

CheckNumberApplet

- Überprüfung von Zahlenwerten
- Gleitkomma oder Integerwerte mit Zahlenbasis 2..36
- CheckFractionApplet für Festkommazahlen
- Vorgabe der Zahlenbasis, Stellenzahl (inkl. führende Nullen)
- Hilfestellungen: korrekt / falsches Format / zu klein / ...
- I18N-Unterstützung: Meldungen über ResourceBundles



Formeln

$$y(t) = b(t - 2\tau) \ \vee \neg b(t - 3\tau)$$

$$x + 1 = 1,\ y + z = 0,\ x + y + z = 1$$

$$H = \Sigma_{i=1}^{N} -p(x_i)\ \log_2\ p(x_i) \ \approx 3.32 \text{ bit}$$

$$R_{23} = \frac{R_2 \ \cdot R_3}{R_2 + R_3}$$

$$D_3 = C_e\ z_0\ z_1\ z_2\ \oplus z_3$$

$$\ldots$$

$$u_c(t) = U_0\ (1 - e^{-t/\tau})$$

- Eingabeformat: ASCII bzw. Matlab/Java-Syntax
- aber keine Scans von Handskizzen
- auch Word/OOo-Formeleditor nicht möglich

- Überprüfung?
- Hilfestellungen??

Symbolic Math Toolbox

- Eingabe der Lösung f_{user} als Matlab-String
- vorgegebene Musterlösung (Matlab-String, .mat-File, ...)

Überprüfung durch Bildung der Differenz und Vereinfachung:

- Prüfe, ob $|f_{user} - f_{master}| \equiv 0$ bzw. $< \Delta$

- Bei Bedarf Iteration der Algorithmen (simple(),simplify(),factor(),...)
- zusätzlich "Kurvendiskussion": Asymptoten, Minima, Maxima, usw.

- liefert vollständige Überprüfung
- aber keine Garantie, dass die Vereinfachung klappt
- bei Bedarf Ergänzung um numerische Auswertung

- Student kann Musterlösung direkt zugreifen und auslesen

Numerische Auswertung:

- Eingabe der Lösung f_{user} als Matlab-String
- Musterlösung als Ausdruck oder Matrix vorberechneter Werte

Überprüfung durch Auswertung an vorgegebenen Stützstellen

- Berücksichtigung von Rundungsfehlern
- Vorgabe absoluter oder relativer Toleranzen
- Vorgabe fester (äquidistant, adaptiv) oder zufälliger Stützstellen

- Hilfestellung durch Protokollierung der maximalen Abweichung
- Ausgabe der Argumente für abweichende Funktionswerte

- CheckFormulaApplet setzt auf eigenen Parser
- Gleitkommawerte, Variablen, Arithmetik, Standardfunktionen

Boole'sche Ausdrücke

- ca. 50% aller Aufgaben in T1
- diverse Varianten und Nebenbedingungen:
  - Infix-Notation: (b0 & b1) | (b2 + ~b3)
  - funktionale Notation: XOR(AND(b0,b1), b3)
  - disjunktive, konjunktive, Reed-Muller Form
  - Beschränkung auf bestimmte Gatter (z.B. NAND2)
  - Beschränkung der Realisierungskosten
  - KV-Diagramme
  - PAL-Schaltpläne (zweistufige Logik, AND-OR bzw. OR-AND)
- diskrete Werte und Argumente: vollständige Enumeration möglich
- eindeutige Normalformen (Funktionstabelle, DNF, KNF, RMF, RoBDD)
- aber Komplexitätsproblem: (fast) alle Operationen O(2**n)

Boole'sche Ausdrücke

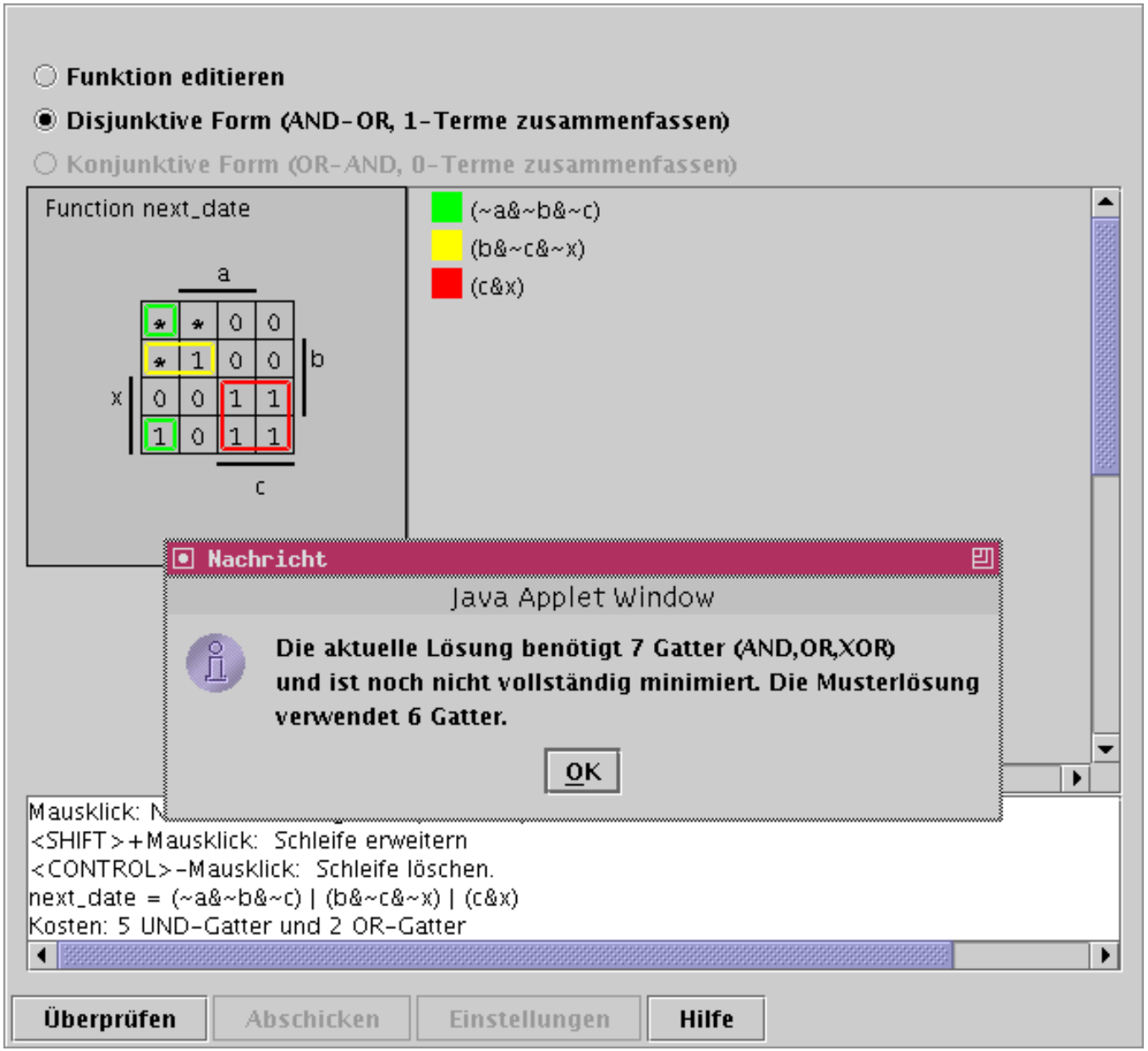
Ansatz zur Überprüfung:

- Eingabe als ASCII-Text - oder graphisch über geeignete GUI
- Parsen des Ausdrucks
- Umwandlung in Normalform oder Funktionstabelle
- Vergleich mit Musterlösung (als Ausdruck oder Funktionstabelle)
- liefert gleichzeitig Gegenbeispiele als Hilfestellung
- kein Komplexitätsproblem für die Übungen: n_vars < 10

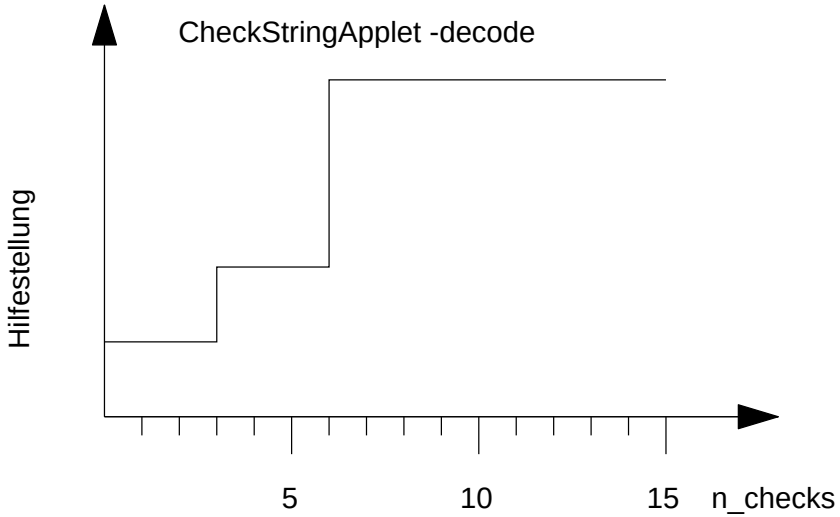
zwei (offensichtliche) Varianten:

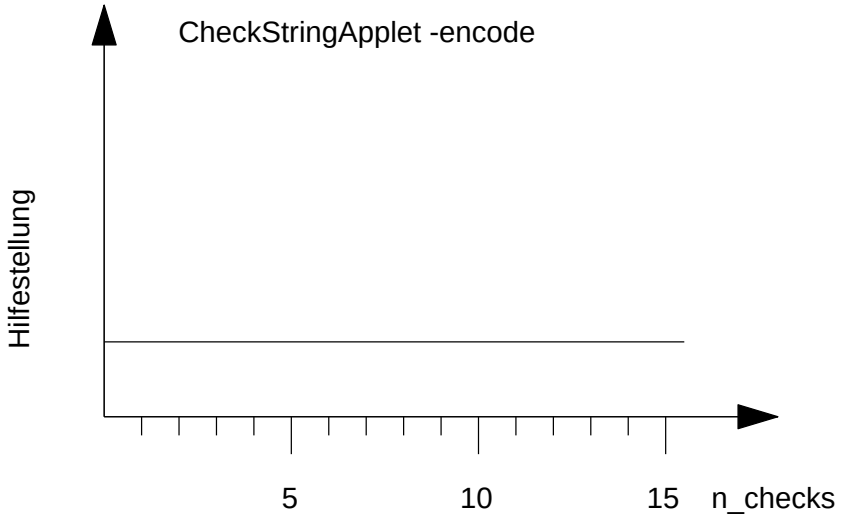
- Nutzen des vorhandenen Parsers der Plattform, z.B. Matlab, Jython
  - Auswertung via eval() o.ä., daraus Funktionstabelle
  - Stringmanipulation zur Prüfung von Nebenbedingungen
- "custom"-Parser mit Unterstützung der Nebenbedingungen

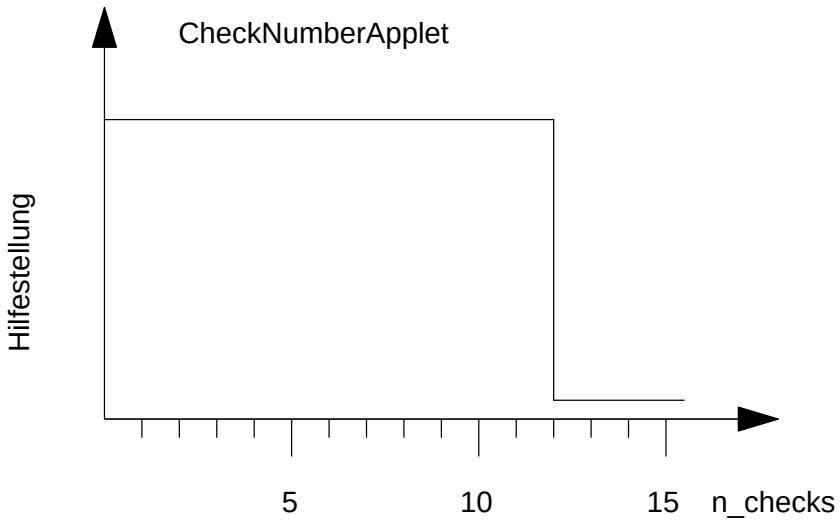
CheckKMapApplet



Zeitabhängigkeit der Hilfestellung ?!

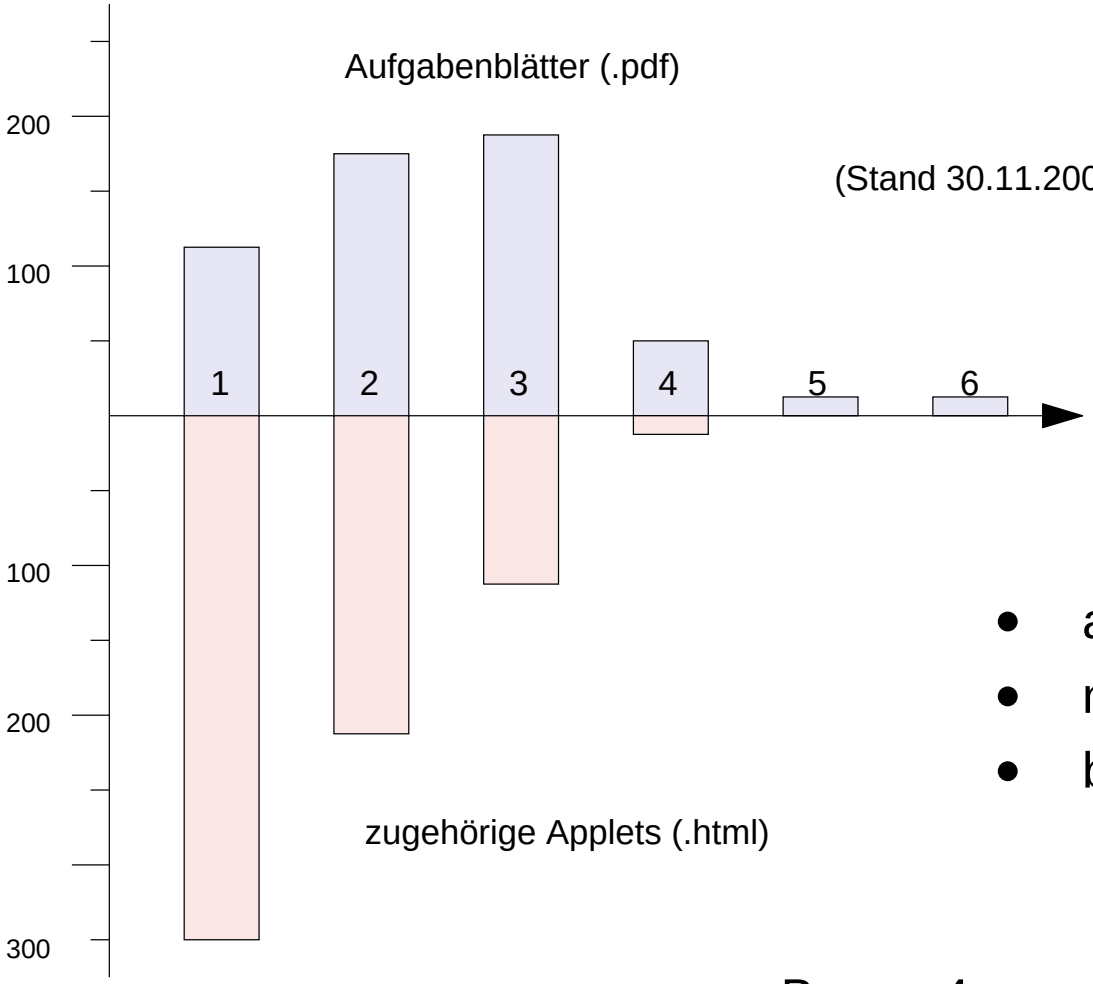






- Meinungen !?

*Check*Applets: Statistik*



- aus Apache access_log
- nur vollständige Downloads
- bisher keine weitere Auswertung

- Bogen 4 gerade erst freigeschaltet
- Applets werden erst ab Bogen 4 interessant
- Evaluation (Nutzerbefragung) später

Status

- Analyse und Klassifikation der "T-Aufgaben"
- Konzepte und Algorithmen für die wichtigsten Aufgabentypen

- Infrastruktur für interaktive Skripte in HTML: Matlab / Jython
- lizenzfreies Matlab-Skript via Matlab-Compiler vorbereitet

- Klassenhierarchie zusätzlicher "einfacher" Check*Applets
- Unterstützung für alle wichtigen Aufgaben fertig implementiert
  M.-Choice, Textvergleich, Zahlen, Formeln, Boole'sche Algebra
- diverse Spezialapplets, einfache Implementierung via Vererbung
- kompakter Code, einfache Benutzung, auch ohne interaktive Platform

- derzeit im "Feldtest": Übungen zur T1-Vorlesung

  `tams-www.informatik.uni-hamburg.de/forschung/interaktives-skript/`
  `tams-www.informatik.uni-hamburg.de/lehre/ws2004/t1Uebung/applets/ 2085336456/`

TODO

- Vorbereiten weiterer T1/T2-Übungsaufgaben
- Evaluierung / Befragung der Studierenden und Gruppenleiter

- Dokumentation der Check*Applets
- Editor für die Applets bzw. Erweiterung eines HTML-Editor
- Marketing: Anwendung auf weitere Übungen/Vorlesungen

- Erstellen des "compilierten" Matlab-Skripts, mögliche Varianten: einzelne Funktionen / t1runner / t1server

- . . .
- . . .

Zusammenfassung

- Konzept des "interaktiven Lehrbuchs"
- Varianten: Matlab/mscriptview, Matlab/Applets/HTML, ...
- Softwarearchitektur dafür, mscript2html-Konverter
- Konzept des lizenzfreien interaktiven Matlab-Skripts

- Klassifikation der Typen von Übungsaufgaben
- Algorithmen zur Überprüfung und Hilfestellung
- Integration in die Matlab-/Jython-Skripte
- oder als standalone-Variante mit den Check*Applets

- praktischer Einsatz läuft, erste Evaluation in Kürze
- . . .

Diskussion

- Fragen, Ideen, Hinweise zum Konzept?
- weitere Typen von Übungsaufgaben?
- entsprechende Formulierung von Übungsaufgaben?
- Integration in den neuen BSc./MS. Studiengang?

Backup-Folien

Ausgangssituation

- geringes Interesse vieler Studenten
- T-Lehrstoff gilt als schwer

"Augen zu und durch"-Ansatz:

- Vorlesung anhören, aber kaum nachbereiten
- sehr schlechte aktive Beteiligung an den Übungen (T1/T2)
- Praktikum als Nachhilfekurs nutzen
- Klausur/Prüfung versuchen (man darf ja mehrmals)

- Korrelation mit Scheinpflicht: bessere Situation für T3/T4 (!)
- Rolle der OE ?

=> Übungen direkt in die Vorlesung und das Skript integrieren
=> und zwar mit interaktiven Hilfsmitteln ("Applets")
=> sofortige automatische Überprüfung der Lösungen
=> Hilfestellungen zu Lösungsansätzen, soweit möglich

Lehrstoff im T-Zyklus:

T1

- Information, Repräsentation
- Zahlensystem, Arithmetik
- Schaltnetze, Schaltwerke
- Schaltungsentwurf
- von-Neumann Rechner

T2

- Lineare Netze
- Bauelemente, Logik-Glieder
- Halbleiterspeicher
- Programierbare Bausteiene
- Mikroelektronik, Prozesse

T3

- Rechnerarchitektur
- Pipelining, Caches
- Ein-/Ausgabe, Interrupts
- Betriebssysteme
- Scheduling
- Schutzmeachnismen

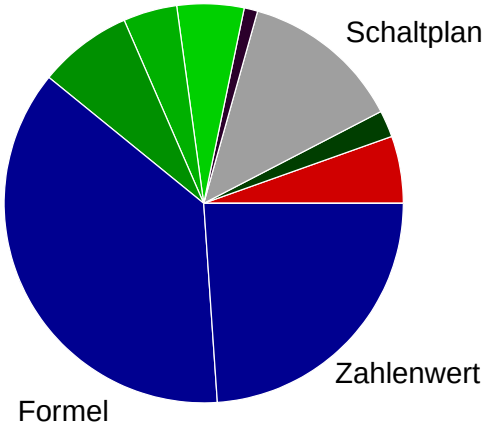
T4

- Informationstheorie
- Parallelverarbeitung
- Datenübertragung
- Vermittlungsnetze
- Lokale Rechnernetze
- Modellierung, Analyse, Entwurf

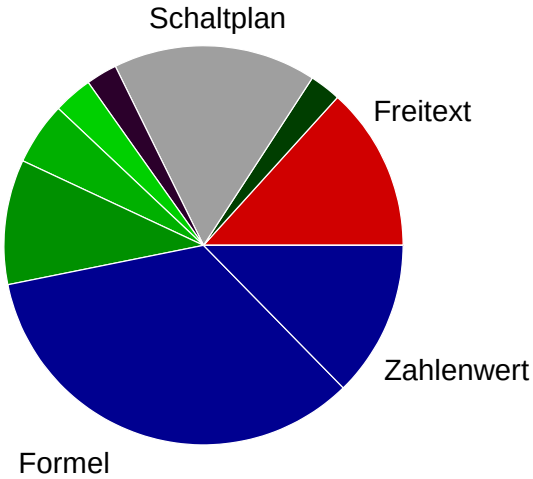
(Hinweis: demnächst Umschichtung nach Einführung des BSc./MS.)

T-Übungsaufgaben: Häufigkeiten

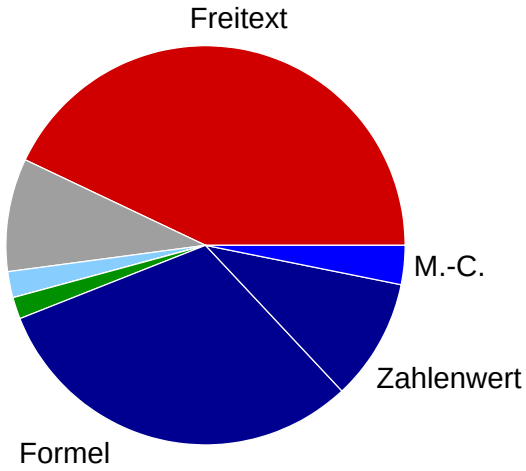
von der Heide (T1/T2)

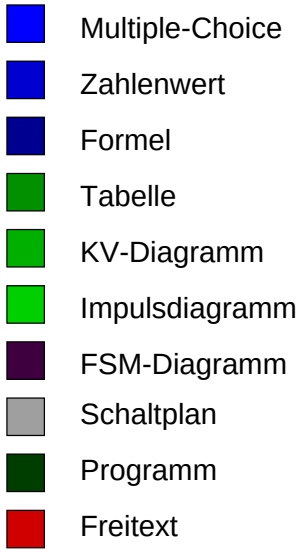


Schiffmann/Schmitz (T1/T2)



Tanenbaum (T1/T3)





- (fast) keine Auswahl-Aufgaben
- diverse Repräsentationen für Zahlenwerte
- logische Ausdrücke unter Formeln eingeordnet
- Funktionstabellen, Schaltpläne, FSMs

- T4 nicht berücksichtigt (eigenes ELCH-Projekt)

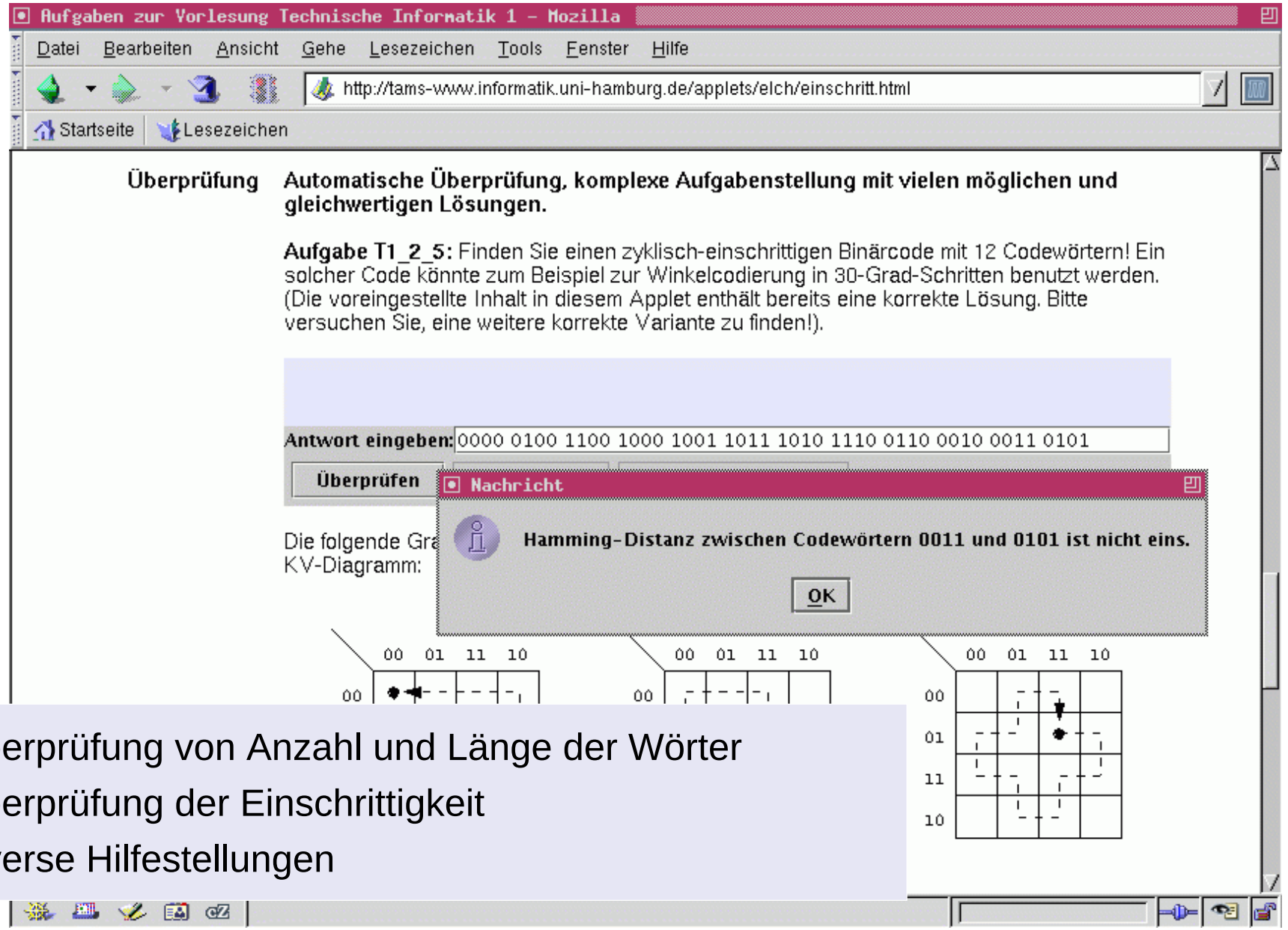
Ansätze zur Überprüfung

Art der Aufgabe:	Ansatz zur Überprüfung
Auswahl	Vergleich mit Musterlösung
Zahlenwert	Werte- und Einheitenvergleich
Formel	symbolische und numerische Auswertung, Plausibilität
Tabellen	wie Zahlenwerte und Formeln
Diagramm	anwendungsspezifisch
KV-Diagramm	vollständige Auswertung, evtl. Normalform
FSM	Simulation und Test der Struktur
Schaltplan	Simulation (vollständig, pseudozufällig)
Programm	Ausführung, Vergleich der Ausgaben
Text	nicht unterstützt

Ansätze zur Hilfestellung

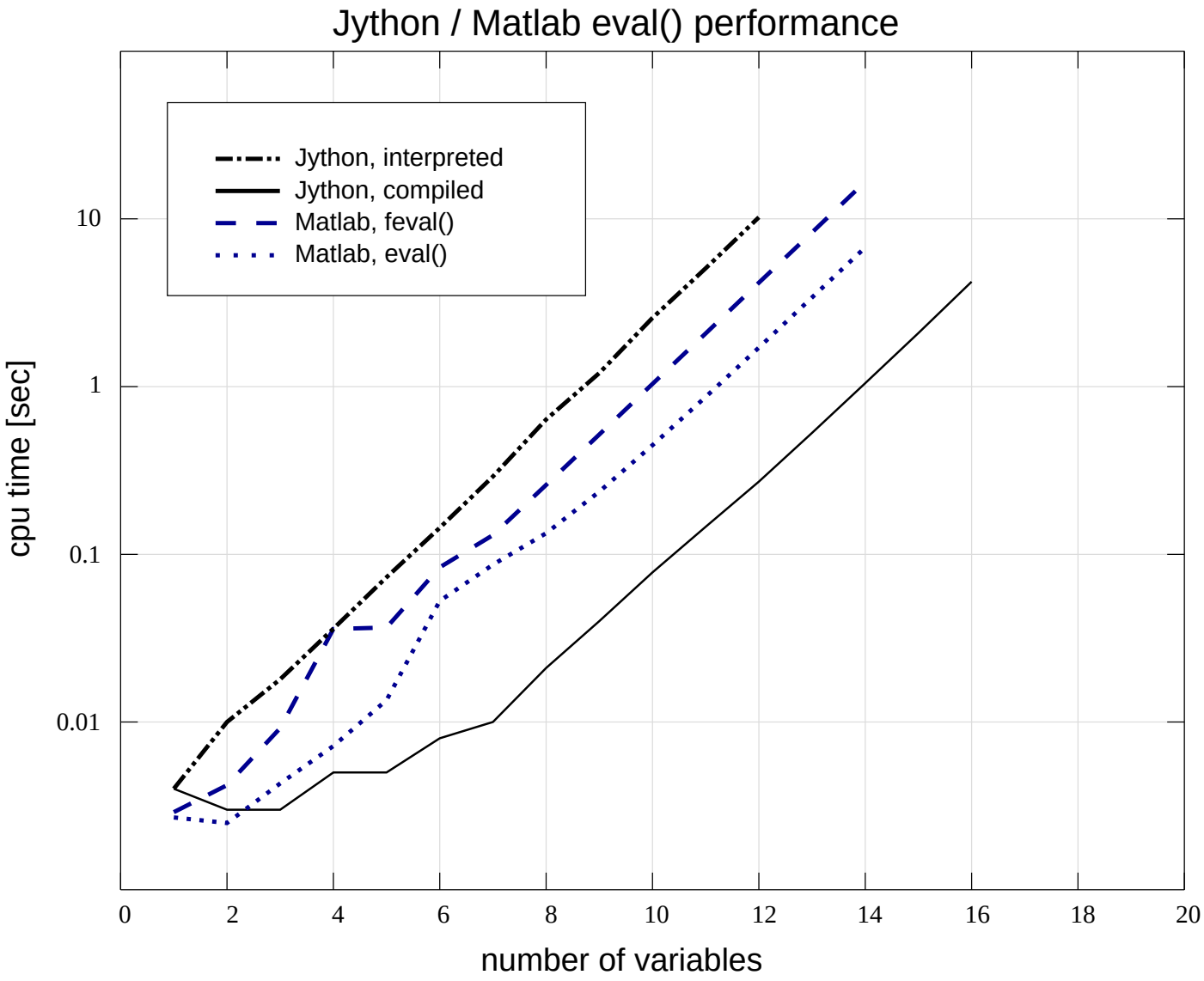
Art der Aufgabe:	Ansatz zur Überprüfung
Auswahl	nicht möglich / anwendungsspezifisch
Zahlenwert	Plausibilität, Wertebereich, Dreher, Einheiten
Formel	Gegenbeispiele aus vollständiger Auswertung Plausibilität
Tabellen	wie Zahlenwerte und Formeln
Diagramm	anwendungsspezifisch
KV-Diagramm	Gegenbeispiele, Hinweise auf Probleme
FSM	Test der Struktur, Gegenbeispiele
Schaltplan	Test der Struktur, Gegenbeispiele
Programm	anwendungsspezifisch
Text	nicht unterstützt

Check_T1_2_5_Applet



- Überprüfung von Anzahl und Länge der Wörter
- Überprüfung der Einschrittigkeit
- diverse Hilfestellungen

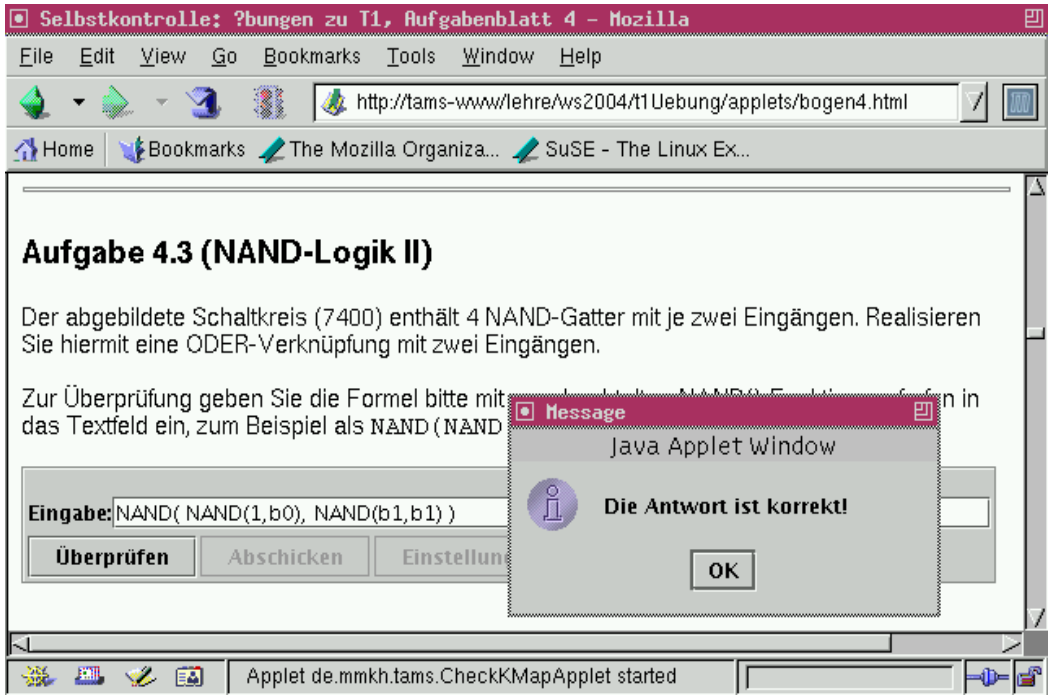
Boole'sche Ausdrücke: Performance



Aufstellung der Funktionstabelle von (a&b)|(c&d)|(e&f&g&g)|(l&m)

BP2: Boolean Parser v.2

- selbstentwickelter Parser (mit javacc Parser-Generator)
- je eine innere Klasse pro Verknüpfung
- Literale (0/1), Variablen, Infix-Notation, Funktionen
- derzeit ca. 50 KByte Classcode (Applet-Download!)
- eval() deutlich schneller als Matlab/Jython
- bei Bedarf leicht erweiterbar



Sackgasse: XHTML-Browser

- mscriptview: ASCII + Fontauswahl + TeX-Formeln
- JythonConsole: bisher nur reiner ASCII-Text
- HTML wäre nett, aber normale Browser sind nicht erweiterbar

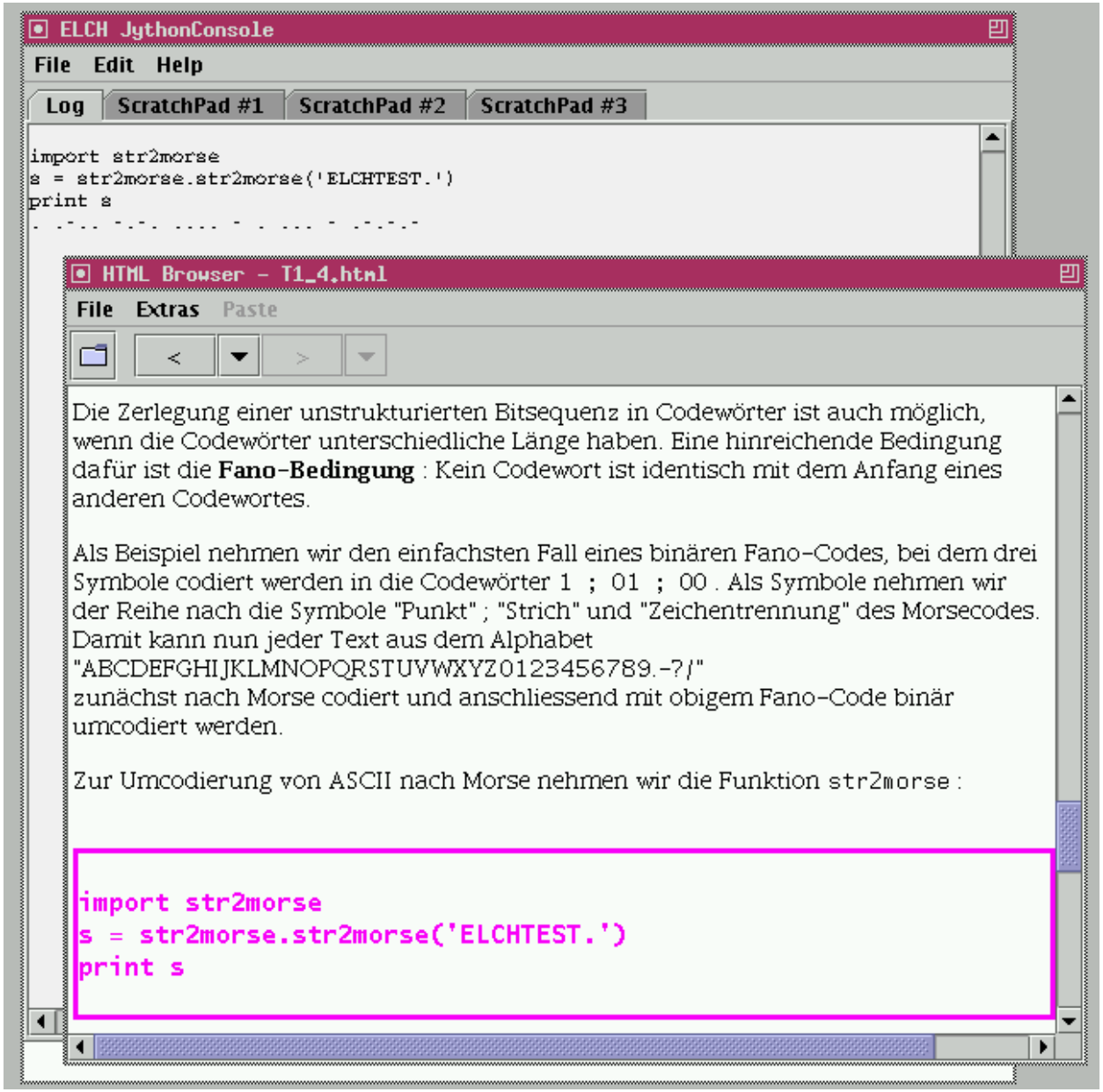
=> eigener, XML/XHTML-basierter Browser (?!)

- auf Grundlage von javax.swing.text.html (HTML 3.2)
- Erweiterungen über neue HTML-Tags integrierbar:

```
<jython>for i in range(1,10): print i</jython>
<matlab>f = sin( 100*[0:0.01:2*pi] )</matlab>
<tex>$e^{i\pi} + 1 = 0$</tex>
```

- leider nicht praxistauglich (diverse Swing-Probleme)
- keine Zusatzfunktionen (Preferences, Bookmarks, ...)
- Konzept und Test in der Diplomarbeit von Andreas Ruge

XHTML-Browser: Screenshot



Jython-Console

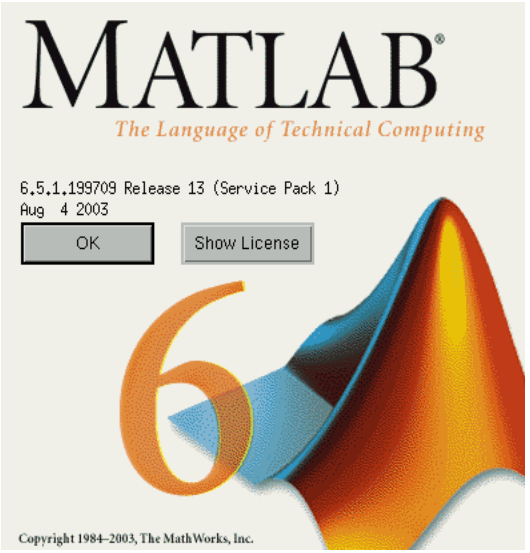
- Editor
- Interpreter

XHTML-Browser

- HTML 3.2
  <jython>
  <matlab>
  <tex>
  <dvi>
- div. Probleme
- keine Doku

Matlab:

"Matrix Laboratory"



- System für rechnergestützte Mathematik
- entwickelt seit ~1984, www.mathworks.de
- Schwerpunkt auf technischen Anwendungen

- einfache Syntax für Vektor-/Matrixoperationen
- numerische und symbolische Algorithmen

• "workspace"	IDE mit Editor und Debugger
• "handle-graphics"	objektorientierte Graphik/Plots
• "toolboxes"	anwendungsspezifische Erweiterungen
• "simulink"	graphische Modellierung
• "system-level design"	Codeerzeugung für DSPs / FPGAs
• "compiler"	brauchbar seit Version 7.0.1 (R14.1)

Matlab: Codebeispiele (1)

```
x = 0 : pi/100 : 2*pi;    % Vektor mit 200 Elementen
y = sin( x );             % ditto
plot( x, y );             % 2D-Funktionsplot, autoscale
xlabel( '0 : 2\pi' );     % x-Label, TeX-Annotation
...

% Samplerate, Tonfrequenz, Bits pro Minute
fs = 4000; frequenz = 800; bpm = 160; string = 'ELCHTEST';
morse = morsecode( string, round(fs/10*bpm/60)*[1 3 1 3 4]));
envel = filter( fir1( 60, bpm/f2 ), 1, morse );
tone  = envel .* sin(2*pi*frequenz*(1:length(envel))/fs);
sound( tone, fs );
```

- Demo(s)

Matlab: Codebeispiele (2)

```
A = [1,1,1; 1,2,3; 1,3,6]; % Pascal(3) Matrix
b = [3; 1; 4];             % Spaltenvektor
x = A \ b;                 % Lösung des Gleichungssystems Ax=b
...

s = '((x-2).^2 - 5)';        % String
f = inline( s );             % Inline-Funktion
v = feval( f, 0.3 );         % Auswertung f(0.3)
z = fzero( f, 0.1 );         % sucht Nullstelle nahe x=0.1
m = fminbnd( f, 0, 4 );      % Minimum in [0..4]
fplot( f, [0, 6] );          % Funktionsplot
...
```

- alle gängigen Matrixoperationen verfügbar

Matlab: "Applets"

These: Verständnis erfordert "Spielen" mit dem Stoff

- Umgang mit Matlab kann nicht immer vorausgesetzt werden
- vereinfachter Zugang wünschenswert

=> zusätzliche interaktive "Applets"

- im Skript eingebettete kleine Applikationen
- eigenes, möglichst eingängiges User-Interface
- einfachere Bedienung als über die Kommandozeile
- auch standalone einsetzbar und erweiterbar
- handle-graphics unterstützt alle üblichen GUI-Komponenten

- Demo

Matlab: Java-Interface

seit Matlab 5.3 auch Zugriff auf Java-Objekte:

```
java on;                          % für Matlab 5.3

generator = java.lang.Random % Konstruktor
x = generator.nextDouble     % Methodenaufruf

editor = hades.gui.Editor    % Hades-Editor
editor.doOpenDesign( 'test.hds', 0 );
editor.getSimulator.runFor( 3.0 ); % simuliert bis t=3.0 sec.
...
```

- läuft auch mit 5.3 Student Edition (einige Einschränkungen)
- Konfiguration über CLASSPATH

Eingebettete Skripte:

- Beispiel-Code im Matlab-Browser: **t1_3_1.m**

```
% Die folgende Funktion bietet eine interaktive Demo für die
% drei Formate nach IEEE-754:
demoieee754
% Werden Operationen durchgeführt mit der Repräsenation für
% {\fontname{Courier}}NaN}, so ist das Resultat immer ...
% ...
inf/(-1+1)   % liefert das Resultat +inf
inf/-(1+1)   % liefert das Resultat -inf
```

- markierter Code kann in den Editor kopiert werden
- Experimente mit anderen Parametern
- Erweiterung der bestehenden Funktionen

- einfache Content-Erstellung durch "Kommentar-Trick"

Das interaktive Skript: mscriptview

t4_4

```
randb(Anzahl,[-1,+1])
```

Achten Sie darauf, dass alle Muster nur die Symbole {-1,+1} benutzen.
Setzen Sie folgende Parameter:

```
neutral = 1 ;   tol = 0.1;
```

```
schrittsync( 100, 102.3, randb(20,[-1 +1]), 0.4, 1, 0.1);
```

Versuch 4.3: Wiederholen Sie Versuch 4.2 unter Einsatz des Bitstuffing.
Verwenden Sie hierbei die Funktion bitstuffing oder/und geeignete Bitmuster wie z.B.

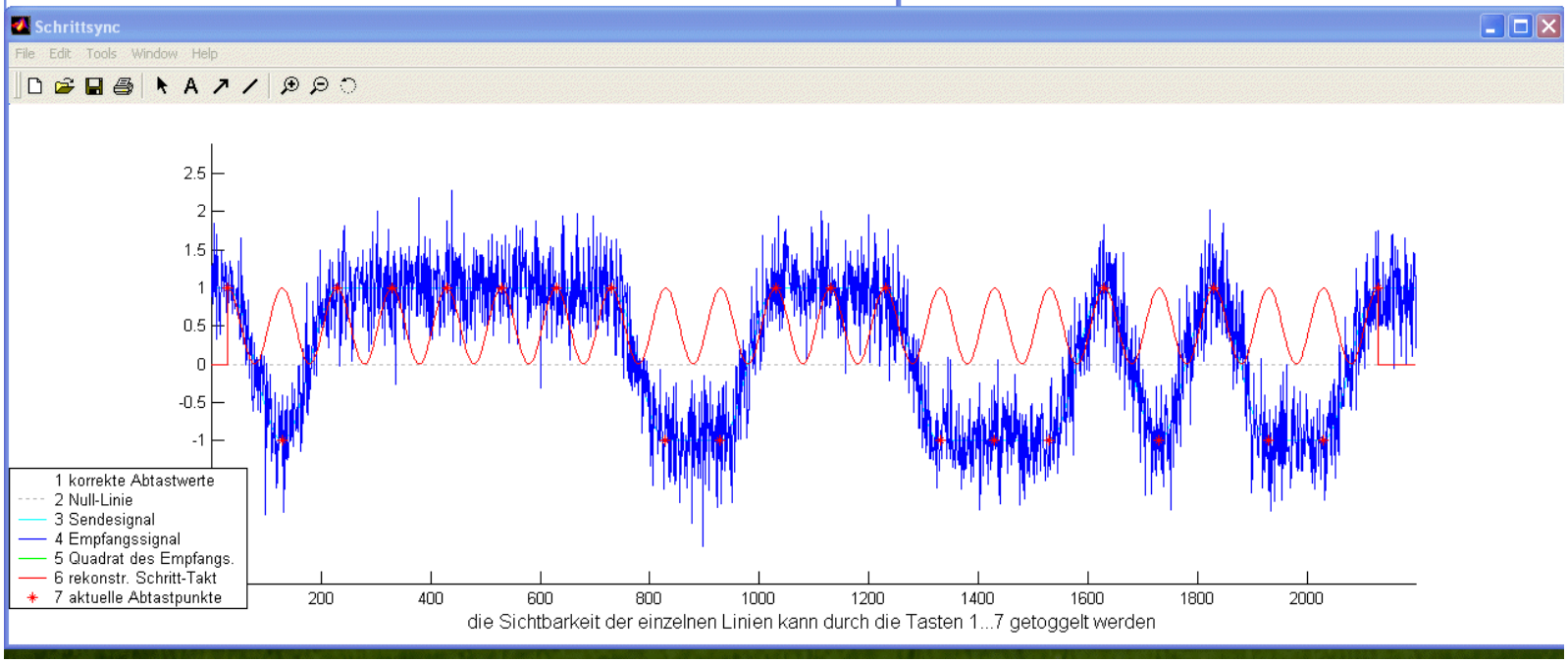
```
2*[1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1] - 1
```

Folgende Fragen sind zu beantworten:
c. Wird das Maximum des Spektrums an der Schrittfrequenz deutlicher?
d. Werden andere Maxima geschwächt oder gar kräftiger?

t4

T4-Praktikum

1. Zeitbereich und Frequenzbereich
2. Abtastung
3. Datenübertragung im Basisband
4. Schrittsynchronisation
5. Modulation



Jython

Python / Jython:

- objekt-orientierte Skriptsprache
- Guido van Rossum, CWI Amsterdam, 1991
- derzeit eine der "modernsten" Skriptsprachen (?!)

- Objekte, Module, Klassen, Mehrfachvererbung
- dynamische Typisierung, keine primitiven Datentypen
- einfache Syntax, Blockbildung über Einrückung
- umfangreiche Klassenbibliotheken

- Jython: Re-Implementierung in Java, B. Hugunin, 1999
- volle Integration in Java-API (Jython)
- praktisch gleiche Performance wie C-Python

Jython: Beispiel

```
import java

def findObject( editor, comment ):
  iterator = editor.getObjects()
  while iterator.hasMoreElements():
    obj = iterator.nextElement()
    tmp = obj.getComment()
    if (tmp == comment):
      return obj
  return None

def transistorDemo( editor ):
  tran = findObject( editor, "T1-channel" )
  if (tran == None):
    print "Object 'T1-channel' not found, sorry!"
    return

  for i in range(0,10):
    tran.setLineColor( java.awt.Color.blue )
    editor.doRedraw()
    java.lang.Thread.currentThread().sleep( 500 )

    tran.setLineColor( java.awt.Color.red )
    editor.doRedraw()
    java.lang.Thread.currentThread().sleep( 500 )
  return

if __name__ == "__main__":
  import jfig
  editor = jfig.gui.JModularEditor()
  editor.doParseFile( "transistor.fig", 0 ) # don't merge
  java.lang.Thread.currentThread().sleep( 2000 )
  transistorDemo( editor )
```

Digitale Schaltungen: Codebeispiel

```
def check_t1_5_5( editor, n_cycles=2000, seed=0xcafebabe, dialog=1 ):

  dm   = editor.getDesignManager()
  baos = ByteArrayOutputStream()
  editor.doSave( baos )
  dm.saveAsVirtualFile( "aufgabe_t1_5_5.hds", baos.toString() )

  testbench = dm.getDesign( None, 'testbench_t1_5_5.hds', 0 )
  simulator = Class.forName( 'VhdlBatchSimKernel' ).newInstance()

  # create simulator, setup and run the simulation
  #
  simulator.setDesign( testbench )
  analyzer0 = testbench.getComponent( 'analyzer0' )
  analyzer0.setSeed( seed )
  ...
  simulator.runFor( 20.0 )

  # read and check signatures
  #
  expected = [ 0x9020266e, 0xa459395a, 0x44b80bd9, ... ]
  signatures = []
  signatures.append( analyzer0.getValue() )
  ...
  for i in range(len(signatures)):
    if (signatures[i] == expected[i]):
      result = 'Ausgang Y' + str(i) + ' funktioniert korrekt'
    else: ...

  if (dialog): JOptionPane.showMessageDialog( None, result )
  return signatures
```

Digitale Schaltungen: