Automatische Überprüfung und Hilfestellung zu Vorlesungs-begleitenden Übungen

(Stand des Projekts im Juni 2004)

Norman Hendrich, Klaus von der Heide

Universität Hamburg, Fachbereich Informatik (TAMS)

Vogt-Kölln-Str. 30, D 22527 Hamburg

hendrich@informatik.uni-hamburg.de

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

Übersicht

- Das interaktive Skript
- Überprüfung und Hilfestellung zu Übungsaufgaben

Darstellung der interaktiven Skripte?

- proprietärer Viewer (mscriptview)
- XHMTL-Viewer (ElearningEditorKit)
- HTML+Applet+Server Konzept

Diskussion

- coming soon: Überprüfung von mathematischen Formeln

Ausgangssituation

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

"Augen zu und durch"-Ansatz:

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

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

Ziele

- Übungsaufgaben im Skript integriert:

Unterstützung der Studierenden:

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

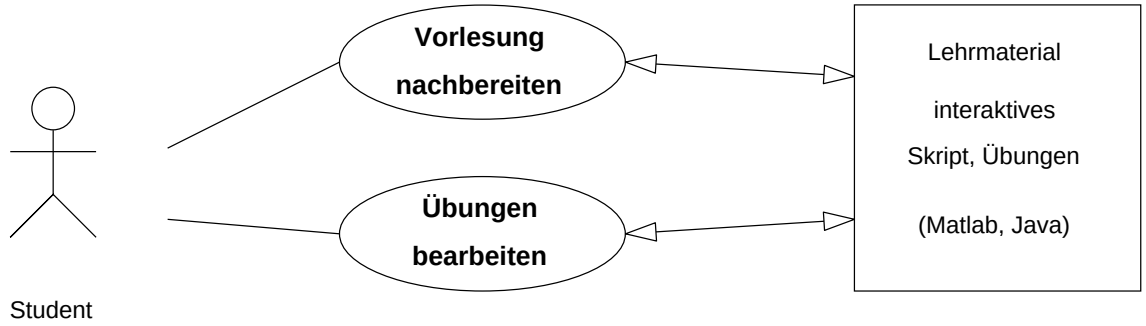
- kontextabhängige Hilfestellungen
- gezielte Gegenbeispiele helfen bei der Fehlersuche

Unterstützung der Übungsgruppenleiter:

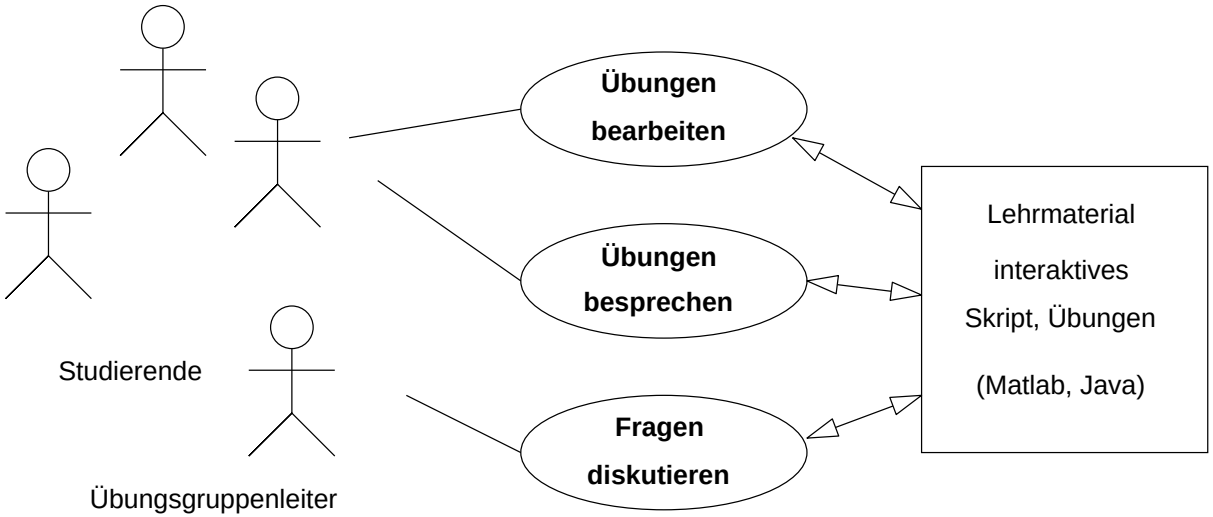
- erleichtert das "Ausprobieren" während der Übungsstunden
- automatische (Vor-) Korrektur vieler Aufgaben

Use-Cases

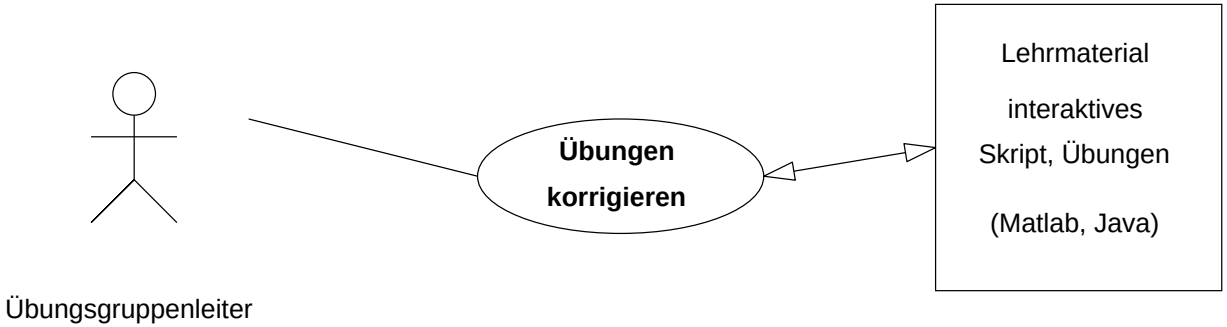
a)



b)



c)



Das interaktive Skript

bzw. "interaktives Lehrbuch"

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

- eingebettete aktive Elemente (z.B. Matlab/Jython-Funktionen)
- eingebettete aktive Applets (mit eigener GUI)
- eingebettete Übungen (mit sofortiger Überprüfung)

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

- einfache Content-Erstellung

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"

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

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 (morsesound aus t1_4.m)

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
- numerische und symbolische Operationen

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

Matlab: Akzeptanz-Problem?

bisherige Version der interaktiven Skripte:

- vollständig in Matlab implementiert
- proprietärer Browser zur Darstellung erforderlich (mscriptview)
- ansonsten Skripte nur mühsam lesbar

- Arbeiten mit dem Material erfordert Matlab und Lizenz
- nicht gegügend Lizenzen (floating-licence) für alle Studierenden
- unterschiedliche Ausgangslage für Grund-/Hauptstudium

=> Aufbereitung des Materials für "Standard"-Viewer ?!

=> Erstellung neuen Materials auf Basis "freier" Plattformen ?!

=> Umsetzung der Skripte nach HTML

Drei Alternativen:

proprietärer, selbstentwickelter Viewer (mscriptview)

- volle Integration in die Plattform (z.B. Matlab)
- Standardfunktionen müssen bereitgestellt werden (z.B. Drucken)

Standard-Web-Browser (Internet Explorer, Mozilla, ...)

- Einbindung der Skripte über Applets / Plugins (Java, Flash)
- Anwender haben ihre gewohnte Umgebung (z.B. Bookmarks)
- Vernetzung mit externen Webseiten / Plattformen (z.B WebCT)
- Dokumentenformat liegt fest (HTML)

modifizierter Web- oder XML-Browser (Swing HTMLEditorKit)

- Erweiterungen des Dokumentenformats möglich (z.B. <matlab>)

EditorKit: Features

- Java Swing (JDK 1.2+) umfaßt Texteditor-Framework
- javax.swing.text.*
- Packages und Klassen für Unicode / RTF / HTML

- Parser ist HTML 3.2 kompatibel
- auch als Editor nutzbar (HTMLWriter)
- kein JavaScript
- keine Applets oder Plugins

- über Subklassen flexibel erweiterbar
- Einbettung vorhandener JComponents in die Darstellung

- Doku: "Swing in a nutshell" (extra chapter on website)
- Sun: "API is expected to change"

ElearningEditorKit

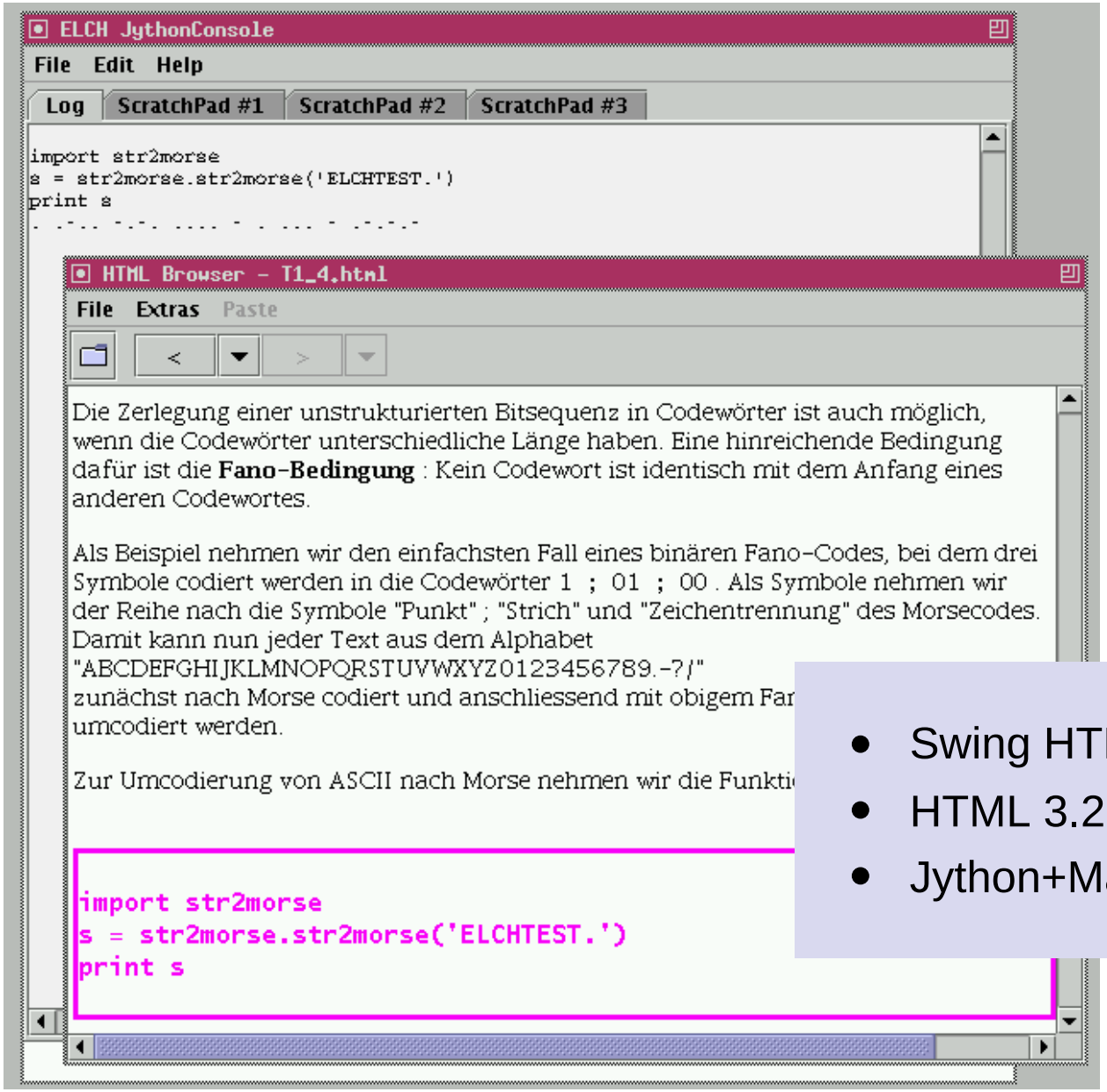
OO-Konzept erlaubt Erweiterungen von EditorKit:

- abgeleitete, erweitere Klassen schreiben
- und über factory-Mechanismus instantiieren
- Beispiel für zusätzliche HTML-Tags

```
<applet src="applet-url" ...> <param /> ... </applet>
<tex>Euler-Formel: $e^{i\pi} + 1 = 0$</tex>
<fig src="figure-url" width=600 height=400> </fig>
<jython> ... </jython>
<matlab> ... </matlab>
```

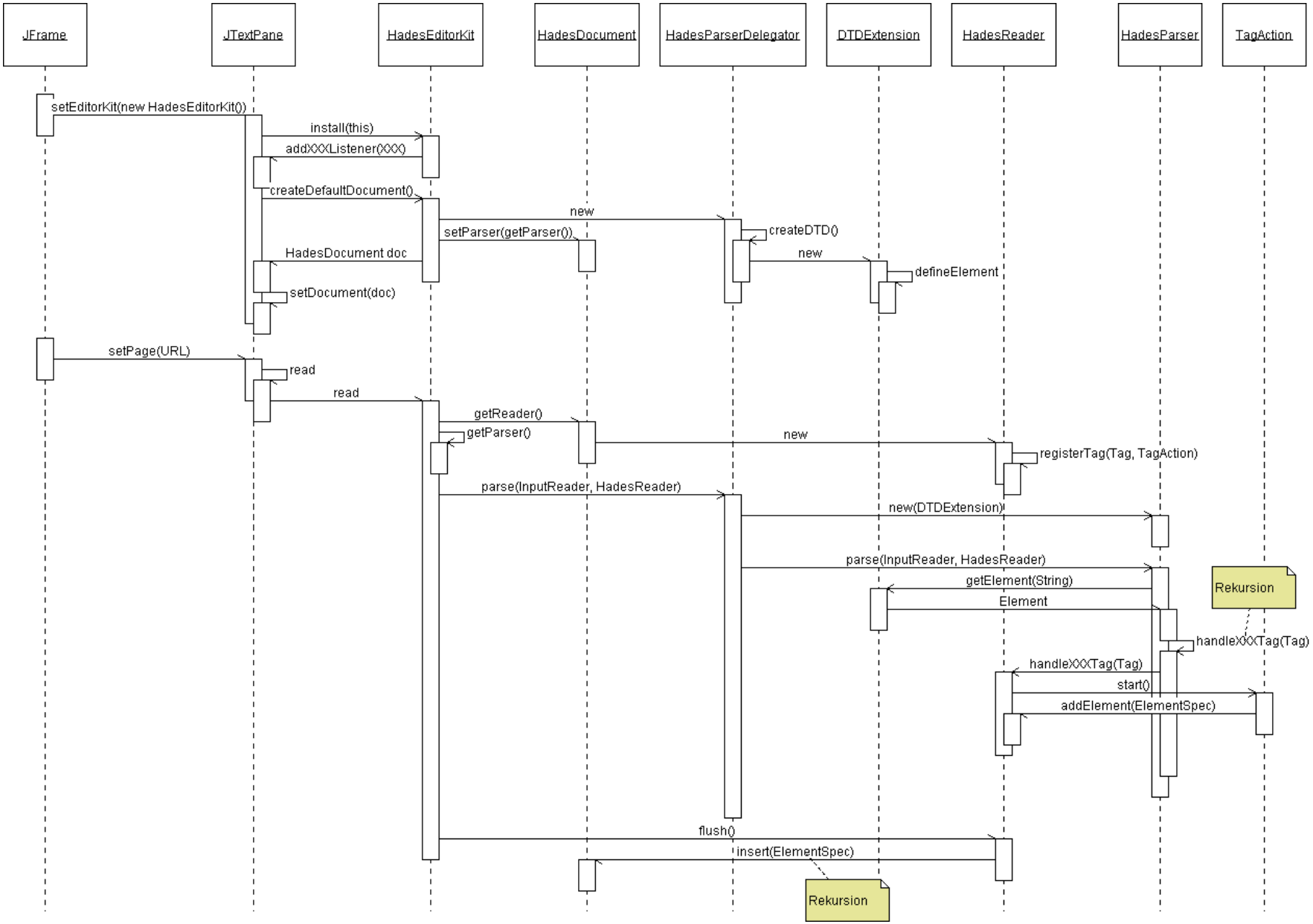
- Prototyp implementiert und getestet
- enorme Komplexität / aber keine brauchbare Doku
- diverse Ärgernisse (Drucken, kein xhtml, kein Javascript, ...)
- Beispiel:

ElearningEditorKit: Prototyp

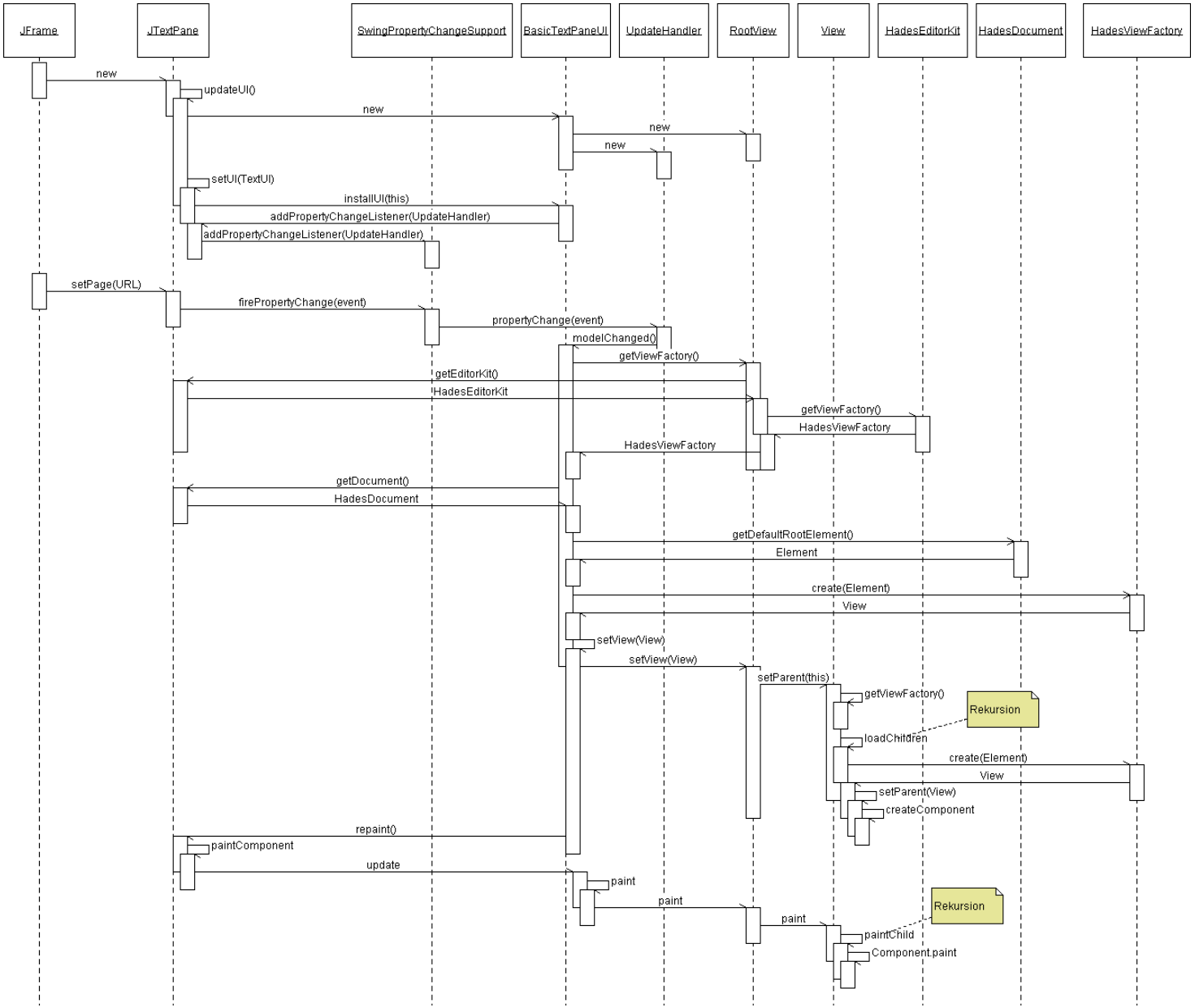


- Swing HTMLEditorKit
- HTML 3.2
- Jython+Matlab+FIG+TeX

ElearningEditorKit: Parsing ...



ElearningEditorKit: ... und Darstellung



ElearningEditorKit: Status und Fazit

Swing-Bibliothek enthält HTML-Komponente (HTMLEditorKit)

Vorteile:

- HTML 3.2 kompatibel, auch als Editor
- flexibel erweiterbar (aber schlecht dokumentiert)
- unverselle Lösung für alle Plattformen (write once: PDAs?)
- Prototyp unterstützt Applets, Jython und Matlab-Skripte

Nachteile:

- Startup langsam (Seiten werden komplett geparst)
- Drucken nur eingeschränkt (z.B. keine Seitenumbrüche)
- Probleme mit existierenden Seiten (kein XHTML, kein Javascript)
- wenig fehlertolerant (bei fehlerhaften Seiten)
- keine GUI und Utilities (z.B. Bookmark-Verwaltung)

=> Ansatz (erstmal) nicht weiter verfolgt

HTML+Applets+Server: Idee

- Skript-Texte nach HTML umsetzen
- interaktive Codeblöcke als eingebettete Java-Applets
- Ausführung über zusätzlichen Serverprozess ("Console")

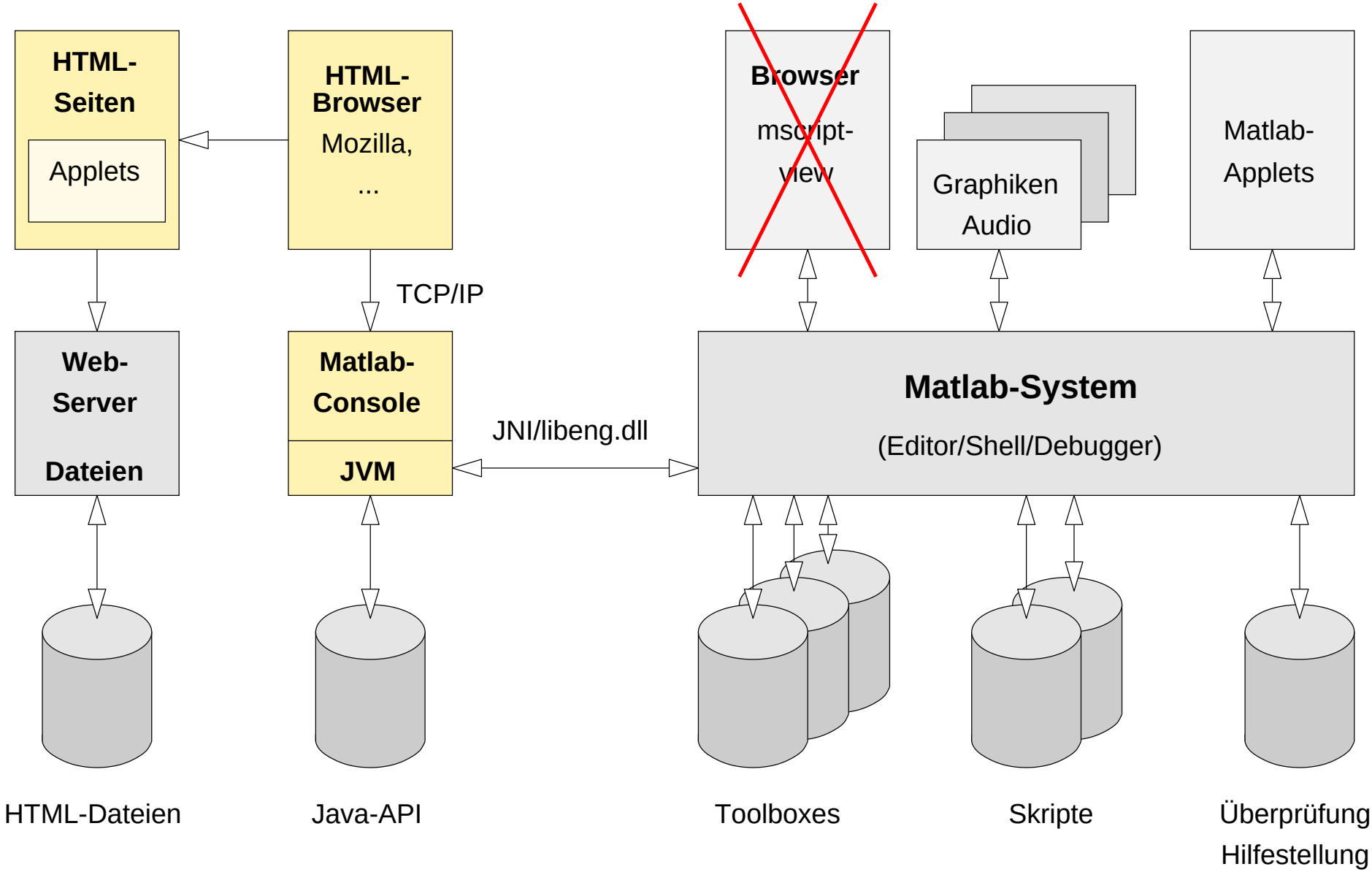
- wichtig: Texte sind auch ohne Matlab gut lesbar

Aufgaben der Applets:

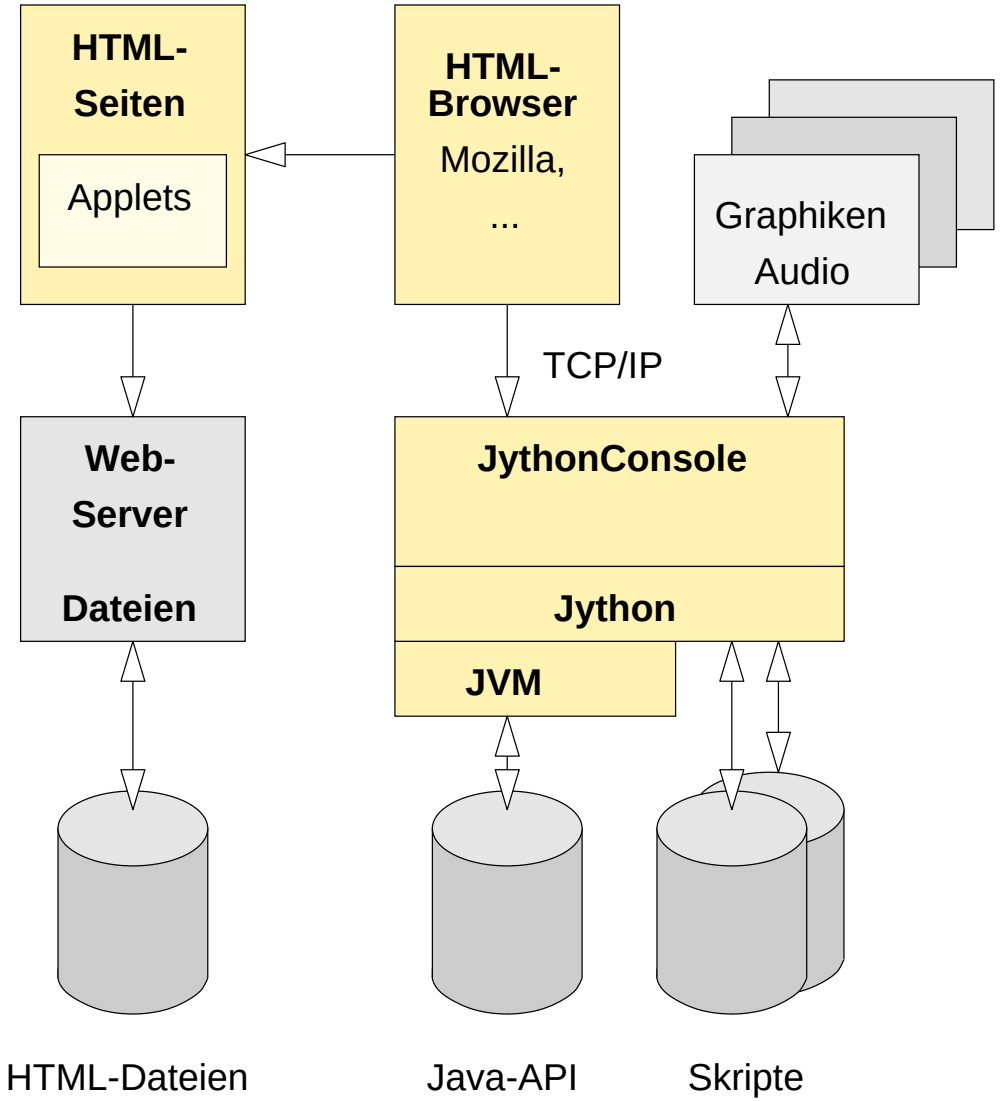
- Darstellung des ausführbaren Codes
- Ereignisbehandlung: Hervorheben, Ausführen, Editieren
- Übergabe des Codes an die jeweilige Console
- Applet benötigt nur Netzwerkzugriff auf localhost

- Serverprozess läuft als Applikation mit vollen Rechten
- Matlab-"Engine", Jython+JVM, Webserver, Datenbanken, ...

HTML+Applets+Server: Matlab



HTML+Applets+Server: Jython



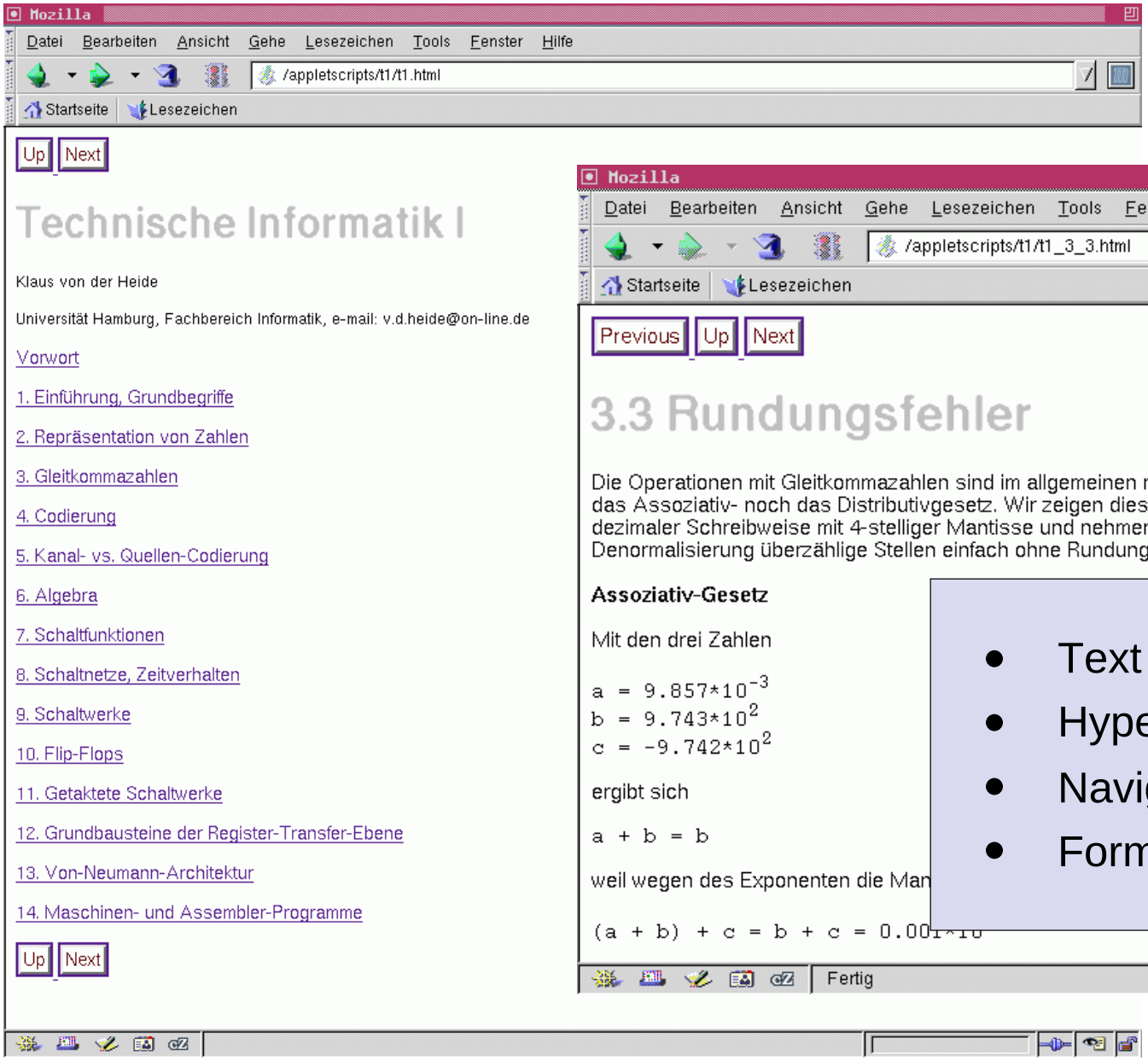
- Codeblöcke als Jython-Skripte und Funktionen
- voller Zugriff auf Java-API inklusive Audio-/Video
- Java-Variante der Matlab "HandleGraphics" 'Funktionen
- bisher nur wenig Content

HTML: Codebeispiel

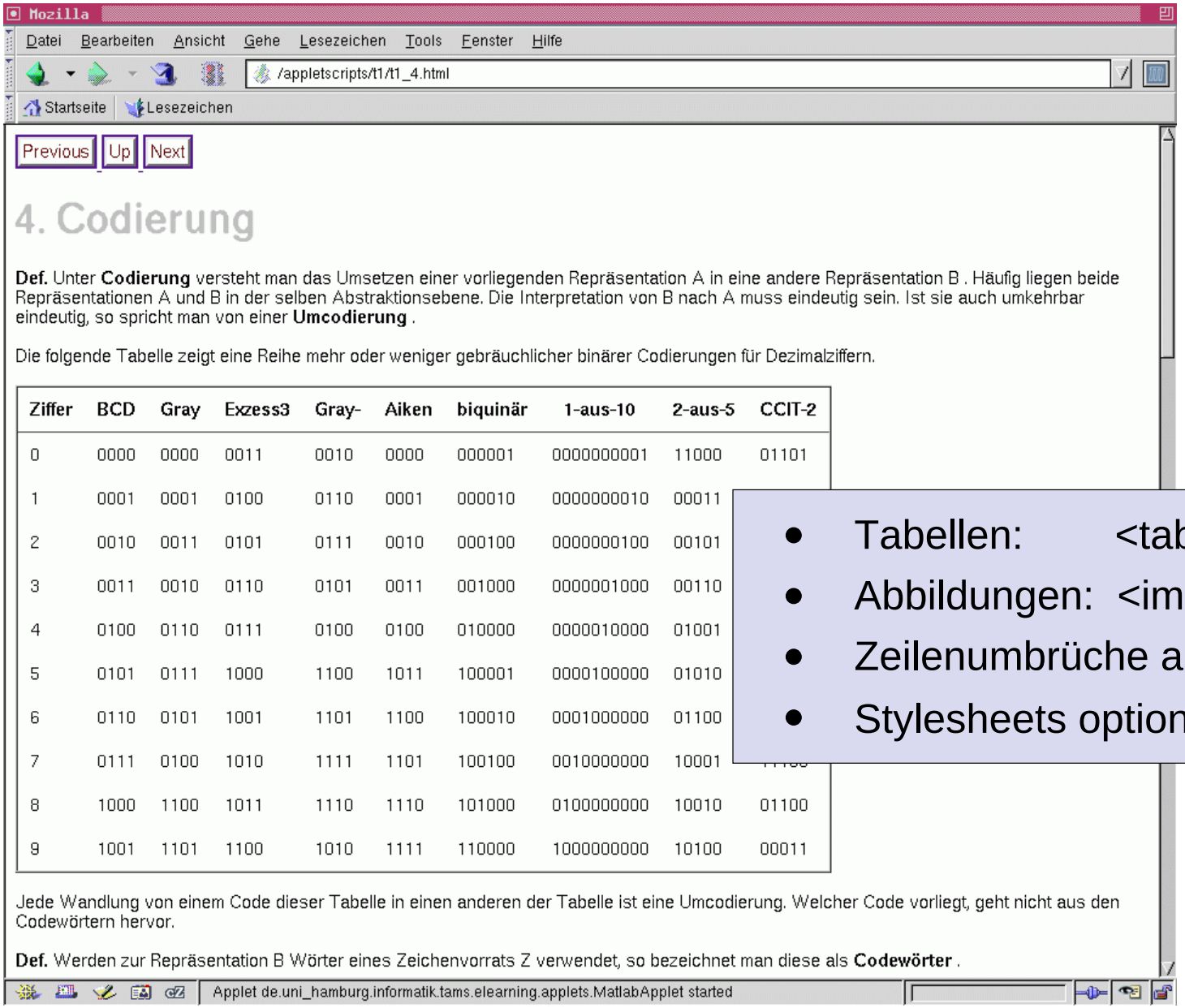
```
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<a href="t1_3_3.html"><img src="../icons/prev.png"></a> ...

<h1><font color="Silver">
4.  Codierung
</font></h1>

<p>
<b>Def.</b>   Unter  <b>Codierung</b>  versteht man das Umsetzen
einer vorliegenden Repräsentation  A  in eine andere Repräsentation  B .
Die Interpretation von  B  nach  A  muss eindeutig sein.
Ist sie auch umkehrbar eindeutig, so spricht man von
einer  <b>Umcodierung</b> .
</p>

<p>
Ein einschrittiger Code mit 12 Wörtern ist z.B. der folgende
</p>
<p>
<applet code="de.uni_hamburg.informatik.tams.elearning.applets.MatlabApplet" width="510" ...>
<param value="n = 12;" name="line1">
<param value="binmatout(num2binmat(einschritt(n)',ceil(log2(n))))" name="line2">
<param value="2" name="nlines">
</applet>
</p>

</body>
</html>
```

```
n = 12;
binmatout(num2binmat(einschritt(n)',ceil(log2(n))))
```

T1-Skript in HTML (1)

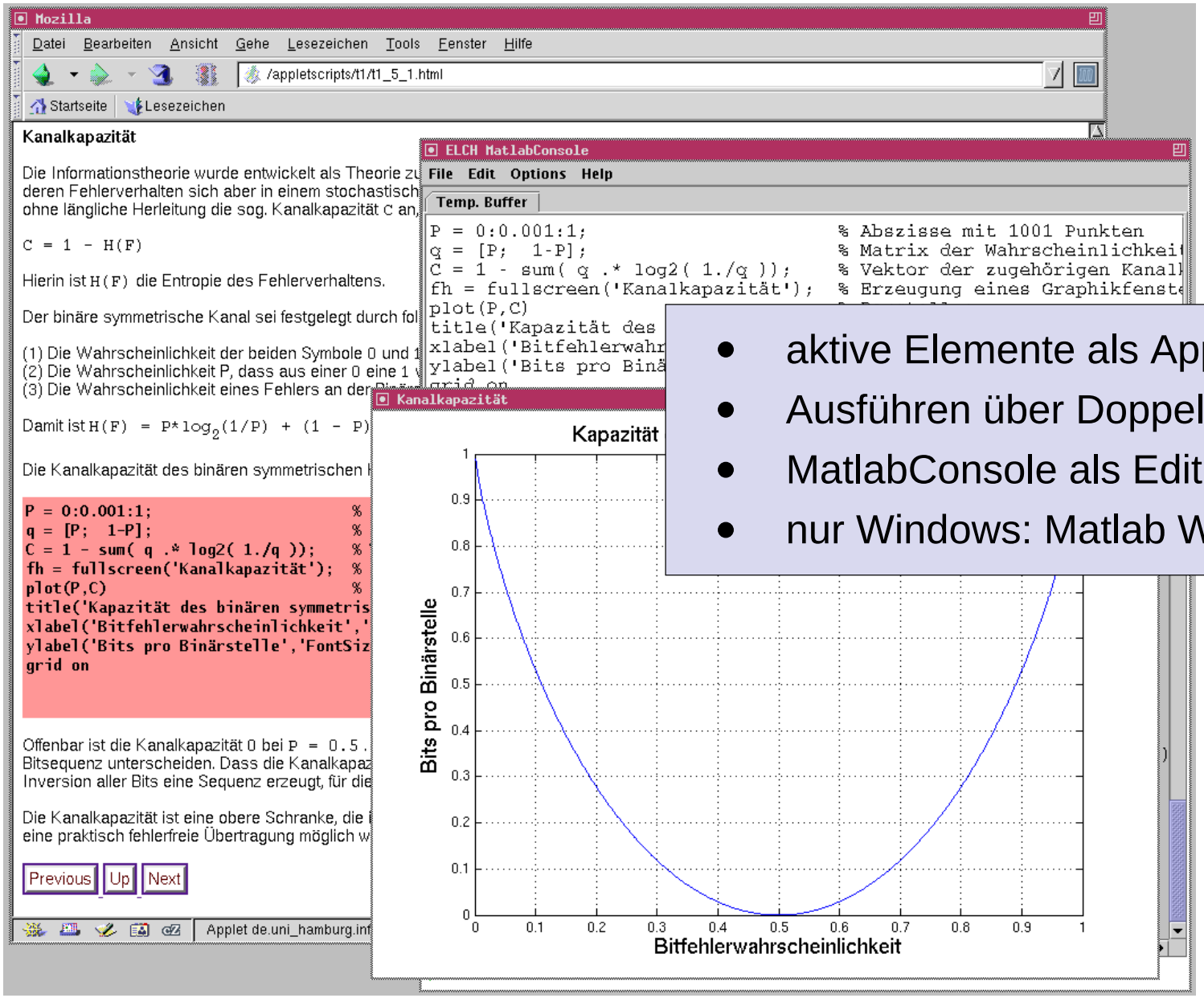


T1-Skript in HTML (2)



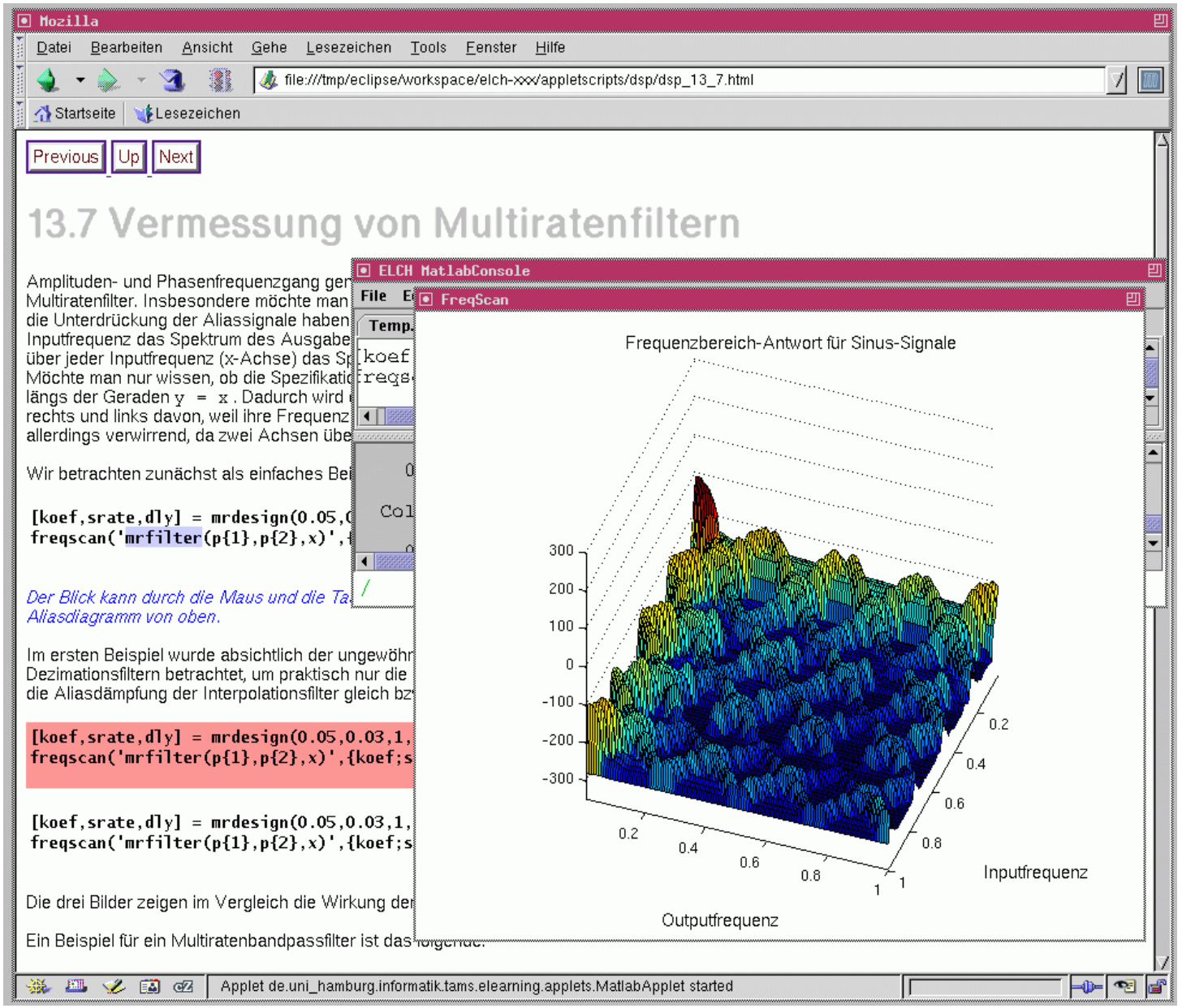
- Tabellen: <table>
- Abbildungen: <img>
- Zeilenumbrüche automatisch
- Stylesheets optional

T1-Skript in HTML (3)



- aktive Elemente als Applets
- Ausführen über Doppelklick
- MatlabConsole als Editor
- nur Windows: Matlab Workspace

DSP-Skript in HTML (4)



Woher kommen die HTML-Seiten?

- Skripte liegen als Matlab-Dateien vor
- Vorlesungen DSP, NT, T1, T2, (Teile) Praktikum T4

=> Konverterprogramm erzeugt daraus HTML-Dateien:

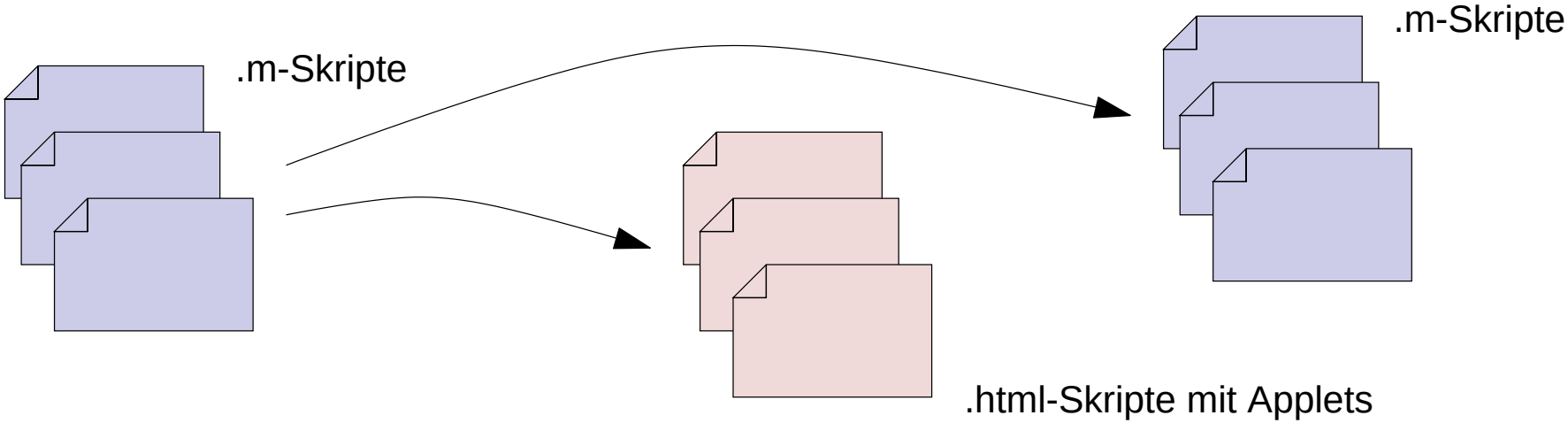
Text mit Formatierungen: (Schriftart, Schriftgröße, Farben)

- Formeln, Sub/Superskripte, Symbole (LaTeX-Notation)
- Navigation, Hyperlinks (Inhalt, prev/next/up)
- Einbettung von Medien: Bilder, Videos, Animationen

Java-Applets als interaktive Elemente:

- Ausführung über MatlabConsole und Workspace
- andere Skriptsprachen möglich
- keine besonderen Sicherheitseinstellungen nötig

Matlab2HTML Konverter:



- je eine HTML-Datei pro Matlab Skriptdatei
- je ein Applet pro Matlab-Codeblock
- Hyperlinks zur Navigation
- Dateibaum mit .m-Skripten, Matlab Pfad-Initialisierung
- Heuristiken zur Erkennung und Umsetzung von Tabellen sowie Formeln als HTML oder Applets

- Details: Vortrag von A.Ruge (OS TAMS, 22.06.2004, 16:15, F334)

HTML Content-Erstellung

vorhanden: Konverter von Matlab nach HTML (s.u.)
aber: Applet+Server-Konzept ist universell

jeder andere HTML-Editor reicht aus, um Skripte zu erstellen:

- geeignete Texteditoren
- HTML-Editoren (emacs, vi, nedit)
- Content-Management-Systeme (Mozilla, HoTMetaL)
- . . .

weitere / vorhandene Konverter ebenfalls nutzbar:

- Beispiel: DA Y.Sünneli (OS TAMS, 29.06.2004, 16:15, F334)
- LaTeX -> Docbook -> HTML+Applets

Zusammenfassung:

- Konzept für HTML-Version der interaktiven Skripte
- Zwei Architekturen als Prototypen erprobt:
  - ElearningEditorKit auf Basis von Java2/Swing
  - Standard-HTML mit eingebetteten Applets
- HTML+Applet+Server-Konzept funktioniert gut
- auch für andere Plattformen als Matlab (Jython, PHP, ...)
- Integration in E-Learning Portale möglich (WebCT, ...)

- Konverter erlaubt vollautomatische Umsetzung
- bisher konvertiert: Vorlesungen T1, DSP, NT
- Installer in Vorbereitung (install4j)

- Fragen, Wünsche, Anregungen?