Diplomarbeit

Werkzeug zur Unterstützung der Erstellung von Lerneinheiten im Internet

Claus Horst

Fachhochschule Lübeck

Alle Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt und sind möglicherweise eingetragene Warenzeichen.

Einige der verwendeten Warenzeichen und Warennamen sind im Anhang aufgeführt.

Diplomarbeit, Lübeck November 2000

Inhalt

Einleitung

Diese Ausarbeitung behandelt meine Diplomarbeit und beschäftigt sich mit dem Thema „Werkzeug zur Unterstützung der Erstellung von Lerneinheiten im Internet".

Die Schwerpunkte stellen einerseits das Anfertigen einer Arbeitsumgebung dar, in der gelernt werden soll, und andererseits die Erstellung eines Generators, mit dessen Hilfe die Lerninhalte einfach, mit einem einheitlichen Layout, effektiv erstellt werden können. Bei den Lerneinheiten handelt es sich um Weiterbildungsseminare im Bereich Gastromarketing, welche die Firmen Apollinaris und Schweppes im Internet presentieren wollen.

Die Inhalte existieren zum Teil bereits als Begleitmaterial zu Seminaren und sollen über einen längeren Zeitraum internetgerecht aufgearbeitet und umgesetzt werden.

Der Entwurf, die Modellierung und die Implementierung erfolgte mit Unterstützung der Multimedia-Agentur kiwi interaktive medien in Hamburg.

Die Verwendung des Zeichens → weist darauf hin, das der Verwendete Begriff im Glossar erklärt wird.

Danke

Mein Dank richtet sich an die Firma *kiwi interaktive medien,* die mich bei der Erstellung dieser Arbeit sowohl fachlich als auch technisch unterstützt hat. Gleichermaßen möchte ich Astrid Trost danken, welche diese Arbeit Korrekturgelesen hat.

In diesem Zusammenhang möchte ich auch nicht unerwät lassen, dass es für mich nur deshalb möglich war diese Diplomarbeit zu erstellen, weil ich von meinen Professoren so ausgiebig auf umfangreiche Aufgaben vorbereitet wurde.

KAPITEL 1

Aufgabenstellung

Die Gestaltung und Erstellung von Lerneinheiten, die online über das Internet angeboten werden, ist ein wichtiges und aktuelles Gebiet der Informatik.

Die Aufgabe der Diplomarbeit besteht darin, ein Werkzeug zur Unterstützung der Erstellung von Lerneinheiten im Internet zu entwickeln und den Einsatz des Werkzeuges bei der Erstellung einer Lerneinheit zu unterstützen und zu evaluieren. Die Lerneinheiten sollen zum Lernen über das Internet geeignet sein und die multimedialen und interaktiven Möglichkeiten dieses Mediums effektiv ausnutzen.

Das Werkzeug ist ein Generator, dessen Einsatz das Anfertigen der Lerneinheiten vereinfachen, beschleunigen und ein einheitliches Aussehen innerhalb dieser sicherstellen soll. Es stellt des Weiteren eine Bibliothek zum einfachen Abfragen des Wissensstandes, zum Einbinden von Sound-, Flash-, Realplayer- und Quicktimeplayern, sowie einige gestalterische Erweiterungen zur Verfügung. Es wird u.a. die Erstellung der Navigation durch die Lerneinheiten und des Inhalts- und Stichwortverzeichnisses vereinfacht.

Es wird die Erstellung einer Lerneinheit mit der entwickelten Unterstützung betreut und evaluiert. Diese Lerneinheit soll exemplarisch den Einsatz und die Vorteile des Werkzeuges demonstrieren. Sie ist Teil des Gastronomiehandbuches, welches von den Firmen Apollinaris und Schweppes im Internet angeboten werden soll.

Die Lerneinheit umfasst das Vermitteln des Stoffes, welcher durch Beispiele und Übungen verdeutlicht wird. Hierzu gehört auch das Abfragen des Gelernten zu Selbstkontrolle.

Aufteilung

Die Diplomarbeit gliedert sich in drei Teile:

- Generator
- Arbeitsumgebung
- Erstellung

Jedes Teil wird einzeln behandelt und im Folgenden näher beschrieben. Auch wenn jedes Teil für sich betrachtet wird und auch für sich eingesetzt werden kann, entfaltet sich erst durch die Verbindung aller Teile ihre volle Leistungsfähigkeit.

Im Detail

Auf den folgenden Seiten ist die Aufgabenstellung im Detail aufgeführt, und zwar in form eines Lasten- und eines Pflichtenheftes.

Lastenheft

1 Zielsetzung

Das Produkt soll Lerneinheiten zum Gastronomiehandbuch realisieren.

Ein Schwerpunkt liegt in der Erstellung eines *Generators*, der den Aufbau der Lerneinheiten vereinfachen und beschleunigen soll. Der Generator soll auch ein einheitliches Aussehen innerhalb der Lerneinheiten sicherstellen. Ebenso umfasst der Generator die Erstellung einer Modulsammlung, auf die von Ihm zurückgegriffen wird.

Einen weiteren Schwerpunkt stellt die der Erstellung der *Arbeitsumgebung* für die Lerneinheiten dar. Hiermit ist die Navigation durch die Lerneinheiten, das Inhalts- und Stichwortverzeichnis sowie die Anzeige weiterer Informationen gemeint. Ebenso umfasst dies die Betreuung bei der Erstellung einer Lerneinheit, welche exemplarisch den Einsatz des Generators und dessen Vorteile aufzeigt.

2 Produkteinsatz

2.1 Anwendungsbereiche

Die Arbeitsumgebung soll sowohl zum Lernen geeignet sein als auch zum Nachschlagen.

Der Generator soll beim Entwickeln weiterer Lerneinheiten helfen, diese einfach und schnell zu erstellen und belibig zu erweitern.

2.2 Zielgruppen

Mit der Arbeitsumgebung sollen Gastronomen erreicht werden die sich fortbilden wollen.

Der Generator ist für Entwickler weiterer Lerneinheiten gedacht. Voraussetzung für die Handhabung sind Grundkenntnisse in der Erstellung von Html Dokumenten.

3 Produktfunktionen

Arbeitsumgebung

- / LF10A / Navigation durch die Lerneinheiten
- / LF20A / Drucken einzelner Kapitel
- / LF30A / Statusanzeige
- / LF40A / Nachschlagewerk
- / LF50A / Inhaltsverzeichnis
- / LF60A / Notizen
- / LF70A / Lesezeichen
- / LF80A / Kapitelbezogenes Diskussionsforum
- / LF90A / Hilfe zur Bedienung der Websites

Generator

/ LF10G / Module zur Einbindung von multimedialen und interaktiven Mitteln
/ LF20G / Beispiele zu den Modulen
/ LF30G / Generieren von Lerneinheiten
/ LF40G / Einbindung der Module in die Lerneinheiten
/ LF50G / Einbindung der Lerneinheiten in die Arbeitsumgebung

4 Produktdaten

Arbeitsumgebung

/ LD10A / Fortschritt des Lernenden speichern
/ LD20A / Zuletzt besuchtes Kapitel speichern
/ LD30A / Notizen zu den Kapiteln speichern
/ LD40A / Lesezeichen speichern
/ LD50A / Diskussionsbeiträge zentral speichern

5 Produktleistungen

Arbeitsumgebung

/ LL10A / Gut strukturierter Aufbau
/ LL20A / Pädagogisch durchdacht
/ LL30A / Übersichtliche Gestaltung und Bedienung
/ LL40A / Optisch ansprechendes Design
/ LL50A / Leicht bedienbar
/ LL60A / Einheitliches Erscheinungsbild
/ LL70A / Das Produkt soll in einem Netscape und Microsoft Browser ab Version 3.0 mit eventuellen Plug-Ins laufen
/ LL80A / Die Rechenleistung eines z.Z. üblichen PC's soll ausreichen (P200)
/ LL90A / Der Einsatzort sind Einzelplatzrechner mit Internetanbindung

Generator

/ LL10G / Ausführen der Möglichkeiten (Konzept, Beispiel, Einsatzmöglichkeit)
/ LL20G / Gut strukturierter Aufbau
/ LL30G / Übersichtliche Gestaltung und Bedienung
/ LL40G / Leicht bedienbar
/ LL50G / Leicht verständlich
/ LL60G / Das Produkt soll unter Windows 9x oder NT eingesetzt werden
/ LL70G / Die Rechenleistung eines z.Z. üblichen PC's soll ausreichen (P200)
/ LL80G / Der Einsatzort sind Einzelplatzrechner

6 Qualitätsanforderungen

Produktqualität	sehr gut	gut	normal	nicht relevant
Funktionalität			X	
Zuverlässigkeit			X	
Benutzbarkeit	X			
Effizienz		X		
Änderbarkeit			X	
Strukturierung	X			
Übersichtlichkeit	X			

Pflichtenheft

1 Zielsetzung

Das Produkt soll Lerneinheiten zum Gastronomiehandbuch realisieren.

Ein Schwerpunkt liegt in der Erstellung eines *Generators*, der den Aufbau der Lerneinheiten vereinfachen und beschleunigen soll. Der Generator soll auch ein einheitliches Aussehen innerhalb der Lerneinheiten sicherstellen. Ebenso umfasst der Generator die Erstellung einer Modulsammlung, auf die von Ihm zurückgegriffen wird.

Einen weiteren Schwerpunkt stellt die der Erstellung der *Arbeitsumgebung* für die Lerneinheiten dar. Hiermit ist die Navigation durch die Lerneinheiten, das Inhalts- und Stichwortverzeichnis sowie die Anzeige weiterer Informationen gemeint. Ebenso umfasst dies die Betreuung bei der Erstellung einer Lerneinheit, welche exemplarisch den Einsatz des Generators und dessen Vorteile aufzeigt.

1.1 Musskriterien

Arbeitsumgebung

- Navigation durch Lerneinheiten
- Drucken einzelner Kapitel
- Statusanzeige (Fortschritt, besuchte Kapitel, Verständnis)
- Nachschlagewerk (Stichwortverzeichnis)
- Inhaltsverzeichnis
- Hilfe zur Bedienung

Generator

- Module zur Einbindung von multimedialen und interaktiven Mitteln
- Generieren von Lerneinheiten
- Einbindung der Module in die Lerneinheiten
- Einbindung der Lerneinheiten in die Arbeitsumgebung

1.2 Wunschkriterien

Arbeitsumgebung

- Nachschlagewerk zu allen Lerneinheiten (Stichwortsuche, Volltextsuche)
- Diskussionsforum zu den einzelnen Kapiteln
- Notizen zu den einzelnen Absätzen
- Lesezeichen

Generator

- Beispiele zu den Modulen

1.3 Abgrenzungskriterien

Arbeitsumgebung

Der Inhalt der Lerneinheit wird bereitgestellt und nicht im Rahmen dieses Projektes zusammengetragen. Es werden lediglich Hilfestellungen bei der Erstellung einer Lerneinheit gegeben.

Generator

Zu generierende Lerneinheiten müssen in einem im Rahmen dieses Projektes definierten Format vorbereitet sein.

2 Produkteinsatz

2.1 Anwendungsbereiche

Arbeitsumgebung

Die Arbeitsumgebung soll sowohl zum Lernen geeignet sein als auch zum Nachschlagen.

Generator

Der Generator soll beim Entwickeln weiterer Lerneinheiten helfen, diese einfach und schnell zu erstellen und belibig zu erweitern.

2.2 Zielgruppen

Arbeitsumgebung

Mit der Arbeitsumgebung sollen Gastronomen erreicht werden die sich fortbilden wollen.

Generator

Der Generator ist für Entwickler weiterer Lerneinheiten gedacht. Voraussetzung für die Handhabung sind Grundkenntnisse in der Erstellung von Html Dokumenten.

2.3 Betriebsbedingungen

- Einzelplatzrechner
- Ungestörter Arbeitsplatz

3 Produktumgebung

3.1 Software

Arbeitsumgebung

- Betriebssysteme: Windows 9x, Windows NT, Mac OS
- Netscape, Microsoft Browser ab Version 3.0 mit evtl. PlugIns.

Generator

- Betriebssysteme: Windows 9x, Windows NT
- Java Runtime Envioment

3.2 Hardware

Ein Computer, der die geforderte Software unterstützt und über die Rechenleistung eines z.Z. üblichem PCs (P200) verfügt.

3.3 Orgware

Es muss der Zugriff über ein Netzwerk auf Ressourcen des Lernprogramms bereitstehen.

3.4 Produktschnittstellen

4 Produktfunktionen

Arbeitsumgebung

/ PF10A / Navigation durch die Lerneinheiten
/ PF11A / Anzeige Nächste / Vorherige Kapitel
/ PF12A / Direkte Anwahl der Kapitel
/ PF20A / Drucken einzelner Kapitel
/ PF30A / Statusanzeige
/ PF31A / Anzeige, ob Seiten besucht wurden
/ PF32A / Anzeige, ob Tests bestanden wurden
/ PF33A / Anzeige des aktuellen Kapitels
/ PF33A / Anzeige des Lernerfolges (Text, Grafik)
/ PF40A / Nachschlagewerk
/ PF41A / Index aus vorgegebenen Stichworten
/ PF42A / Suche im Index
/ PF43A / Volltextsuche in allen Inhalten
/ PF50A / Inhaltsverzeichnis
/ PF60A / Notizen
/ PF61A / Notizen des Benutzers am Seitenrand
/ PF70A / Lesezeichen
/ PF80A / Kapitelbezogenes Diskussionsforum
/ PF81A / Es können Beiträge zu einzelnen Abschnitten gebracht werden
/ PF82A / Abschnitte mit Beiträgen werden markiert
/ PF90A / Hilfe zur Bedienung

Generator

/ PF10G / Module zur Einbindung von multimedialen und interaktiven Mitteln
/ PF11G / Wissensabfrage
/ PF12G / Sound (Hintergrundmusik, Effekte, Vorlesen,...)
/ PF13G / Animationen (Flash,...)
/ PF14G / Filme (Realplayer,Quicktime,...)
/ PF15G / Grafiken
/ PF20G / Beispiele zu den Modulen
/ PF21G / Webseiten, auf denen die multimedialen und interaktiven Möglichkeiten kurz und einfach präsentiert werden
/ PF30G / Generierung von Lerneinheiten
/ PF31G / Textinhalt in einen einheitlichen Stil bringen
/ PF32G / Grafiken einheitlich einbinden
/ PF33G / Links standardisieren
/ PF40G / Einbindung der Module in die Lerneinheiten
/ PF41G / Erweiterbar um neue Module
/ PF50G / Einbindung der Lerneinheiten in die Arbeitsumgebung
/ PF51G / Generierung von Index aus Stichworten
/ PF52G / Generierung des Inhaltsverzeichnis
/ PF53G / Verlinkung generieren

5 Produktdaten

Arbeitsumgebung

/ PD10A / Speicherung des Fortschritts des Lernenden
/ PD20A / Speicherung des zuletzt besuchtes Kapitels
/ PD30A / Speicherung der Notizen zu den Kapiteln
/ PD40A / Speicherung der Lesezeichen
/ PD50A / Speicherung der Diskussionsbeiträge (zentral)

6 Produktleistungen

Arbeitsumgebung

/ PL10A / Gut strukturierter Aufbau
/ PL20A / Pädagogisch durchdacht
/ PL21A / Nicht überladen durch zu viele Informationen oder Grafiken
/ PL22A / Download-Zeiten berücksichtigen
/ PL30A / Übersichtliche Gestaltung und Bedienung
/ PL40A / Optisch ansprechendes Design
/ PL50A / Leicht bedienbar
/ PL51A / Intuitive Bedienung
/ PL60A / Einheitliches Erscheinungsbild
/ PL61A / Einheitliches Layout
/ PL70A / Das Produkt soll in einem Netscape und Microsoft Browser ab Version 3.0 mit evtl. PlugIns laufen

/ PL71A / Fehlende Plug-Ins werden gemeldet und optional installiert

/ PL72A / Fehlende Plug-Ins stören nicht die Funktion und das Gesamterscheinungsbild

/ PL80A / Die Rechenleistung eines z.Z. üblichen PC's soll ausreichen (P200)

/ PL90A / Der Einsatzort ist an Einzelplatzrechnern mit Internetanbindung

Generator

/ PL10G / Ausführen der Möglichkeiten (Konzept, Beispiel, Einsatzmöglichkeit)

/ PL20G / Gut strukturierter Aufbau

/ PL30G / Übersichtliche Gestaltung und Bedienung

/ PL40G / Leicht bedienbar

/ PL41H / Hilfe mit Benutzungsbeispiel

/ PL50G / Leicht verständliche Bedienung

/ PL60G / Das Produkt soll unter Windows 9x oder NT eingesetzt werden

/ PL70G / Die Rechenleistung eines z.Z. üblichen PC's soll ausreichen (P200)

/ PL80G / Der Einsatzort ist an Einzelplatzrechnern mit Internetanbindung

7 Benutzungsoberfläche

Die Oberfläche soll ein einheitliches Bild vermitteln und somit eine einfache intuitive Bedienung ermöglichen.

7.1 Bildschirmlayout

Arbeitsumgebung

/ PB10A / Einheitliches Erscheinungsbild über die Gesamte Lerneinheit

/ PB20A / Einheitliche intuitive Navigation

Generator

/ PB10G / Einheitliches Erscheinungsbild über alle Beispiele

7.2 Drucklayout

Arbeitsumgebung

/ PB30A / Die Lerneinheit soll nicht im Ganzen gedruckt werden

/ PB31A / Das Drucken einzelner Abschnitte soll möglich sein

7.3 Tastaturbelegung

Arbeitsumgebung

/ PB50A / Die im jeweiligen Browser übliche

Generator

/ PB20G / Die unter dem jeweiligen Betriebssystem übliche

7.4 Dialogstrukturen

Arbeitsumgebung

/ PB60A / Einheitliches Erscheinungsbild

/ PB70A / Eingaben zum Ermitteln des erreichten Verständnisses

Generator

/ PB30G / Filedialoge zur Auswahl von Quell- und Zieldateien sowie Pfad

8 Qualitäts-Zielbestimmung

Produktqualität	sehr gut	gut	normal	nicht relevant
Funktionalität				
Angemessenheit			X	
Richtigkeit		X		
Interoperabilität			X	
Ordnungsmäßigkeit		X		
Sicherheit			X	
Zuverlässigkeit				
Reife			X	
Fehlertoleranz		X		
Wiederherstellbarkeit		X		
Benutzbarkeit				
Verständlichkeit		X		
Erlernbarkeit	X			
Bedienbarkeit	X			
Effizienz				
Zeitverhalten			X	
Verbrauchsverhalten			X	
Änderbarkeit				
Analysierbarkeit			X	
Modifizierbarkeit			X	
Stabilität			X	
Prüfbarkeit			X	
Übertragbarkeit				
Anpaßbarkeit		X		
Installierbarkeit			X	
Konformität			X	
Austauschbarkeit			X	

9 Globale Testfälle

Arbeitsumgebung

/ T10L / Die Lerneinheiten sind linear und zufällig durchzugehen

Generator

/ T10L / Der Generator ist mit der Erstellung der Lerneinheit zu testen

10 Entwicklungsumgebung

10.1 Software

Frei wählbar

10.2 Hardware

Frei wählbar

10.3 Orgware

Frei wählbar

10.4 Entwicklungsschnittstelle

Frei wählbar

KAPITEL 2

Arbeitsumgebung

Bei der Arbeitsumgebung handelt es sich um Websites zum Lernen im Internet. Es ist jedoch nicht der eigentlich zu erlernende Inhalt gemeint, sondern die Umgebung in welcher der Inhalt presentiert wird. Hierzu zählen unter anderem das globale optische Erscheinungsbild, die Navigation durch die Inhalte, das Inhalts- und Stichwortverzeichnis und die Bildschirmaufteilung. Das Einpflegen der Inhalte in die Arbeitsumgebung wird mit Hilfe des → Generators geschehen. Durch den Einsatz des Generators wird auch sichergestellt, dass alle Inhalte ein einheitliches Layout haben.

Rahmenbedingungen

Zielgruppe

Die Arbeitsumgebung soll von Gastronomen genutzt werden, die mit der grundsätzlichen Bedienung von Internetseiten vertraut sind. Die Inhalte sollen Gastronomen ansprechen, die sich im Internet fortbilden wollen. Da der Schwerpunkt der Inhalte im Bereitstellen von Lernunterlagen liegt, sollte die Umgebung nicht verspielt sein, sondern den Anwender mit grundlegenden Funktionen versorgen.

Plattform

Um einen einfachen und unkomplizierten Einsatz gewährleisten zu können, muss die Arbeitsumgebung sowohl unter Netscape und Microsoft Browsern ab Version 3.0 lauffähig sein. Als Betriebsysteme sollen Windows 9x, Windows NT und Mac OS unterstützt werden. Die Inhalte werden im Internet angeboten.

Funktionen

Die Funktionen der Arbeitsumgebung erstrecken sich über die bereits genannten Aufgaben wie die Navigation durch die Inhalte und das Inhalts- und Stichwortverzeichnis. Hierzu kommt noch, die Möglichkeit einzelne Seiten zu drucken. Ebenso eine Statusanzeige mit Informationen zum Lernfortschritt, besuchte Kapitel und dem aktuellen Kapitel. Wichtig ist auch eine Hilfefunktion zur Bedienung der Arbeitsumgebung. Wünschenswert wäre, dass zusätzlich noch Funktionen zur Verfügung ste-

hen, die Möglichkeiten bieten wie Suchen im Stichwortregister oder eine Volltextsuche, Erstellen von Notizen am Seitenrand, kapitelbezogenes Diskussionsforum und ein setzen von Lesezeichen.

Da die Browser zur Darstellung von Inhalten, die über Text und Bild hinausgehen, Plug-Ins benötigen, muss sichergestellt werden das fehlende Plug-Ins das Erscheinungsbild nicht stören.

Folgerungen

Zusammenfassend ergibt sich aus den beschriebenen Kriterien Folgendes:

Um eine solche Fülle von Informationen und Möglichkeiten übersichtlich darzustellen, ist es erforderlich, einen gut strukturierten Aufbau sicherzustellen. Es muss sparsam mit Grafiken umgegangen werden, um einem überladenen, unübersichtlichen und umständlich bedienbaren Eindruck entgegen zu wirken.

Die Arbeitsumgebung muss intuitiv bedienbar sein, damit der Lernende seine volle Aufmerksamkeit auf das zu Lernende lenken kann.

Funktionen wie ein Index oder ein Stichwortverzeichnis müssen nicht ständig sichtbar sein, da diese während des normalen Lernvorganges nicht gebraucht werden. Solche Funktionen werden nur zum Nachschlagen benötigt.

Sollen einzelne Abschnitte gedruckt werden, so wird nur der Inhalt gedruckt, nicht aber die umliegende Navigation.

Die Programmierung der Arbeitsumgebung muss sich auf Funktionen beschränken, die sowohl der MSIE als auch Netscape verstehen. Um auch die Möglichkeiten der neueren Browsergenerationen nutzen zu können, müssen Alternativen für ältere Browser eingebaut werden.

Da die Arbeitsumgebung und deren Inhalte im Internet bereitgestellt werden, ist es wichtig, die Größe der Arbeitsumgebung niedrig zu halten, um unangenehm lange Ladezeiten zu vermeiden. Dies muss während der gesamten Entwicklung berücksichtigt werden.

In Bezug auf Plug-Ins ist zu bemerken, dass für fehlende Plug-Ins Alternativen einspringen. Dies könnte bei einem Film zum Beispiel ein Bild sein.

Entwurf

Die Erstellung der Arbeitsumgebung gliedert sich in zwei Teile:

- Das Layout
- Die Funktionalität

Obwohl beide Teile fest miteinander verbunden sind, so werden sie doch unabhängig voneinander entworfen.

Der getrennte Entwurf wird jedoch dadurch eingeschränkt, dass eine relativ umfangreiche Schnittstelle zwischen beiden Teilen nötig ist. Da diese Schnittstelle überwiegend durch das Layout und der damit verbundenen Aufteilung in einzelne Bereiche bestimmt wird, kann dieses auch nur in einem sehr engen Rahmen nachträglich noch geändert werden.

Layout

Das Layout ist der Teil der Arbeitsumgebung, mit dem der Anwender in Berührung kommt. Daher muss es übersichtlich und ansprechend sein und eine intuitive Bedienung ermöglichen. Es soll nur wirklich benötigte und sinnvolle Bedienelemente zur Verfügung stellen, um dies zu gewährleisten.

Weder dürfen wichtige oder benötigte Bedienungselemente fehlen, noch dürfen zu viele Bedienungselemente vorhanden sein. Der Anwender soll nicht dazu animiert werden, mit der Arbeitsumgebung herumzuspielen, und doch muss diese über ein ansprechendes Design verfügen, um einen seriösen Eindruck zu vermitteln und den Anwender nicht abzuschrecken.

Das Layout bestimmt jedoch nicht nur die Arbeitsumgebung und die Bildschirmaufteilung. Auch die Inhalte die mit Hilfe der Arbeitsumgebung vermittelt werden sollen, werden vom Layout mitbestimmt. Schließlich sollen sie ja im Rahmen der Arbeitsumgebung leicht lesbar und angenehm zu betrachten sein. Das Layout bestimmt also auch die grundlegende Darstellung des Inhaltes.

Ein letzter Punkt, der durch das Layout bestimmt wird ist die Farbgebung. Auch dieser Teil fließt sowohl in die Arbeitsumgebung als auch in die Inhalte ein.

Bedienelemente

Aus den Anforderungen ergibt sich, dass die Arbeitsumgebung über folgende Bedienungselemente verfügen muss:

- „Weiter“- Button
- „Zurück“- Button
- „Hilfe“- Button
- „Drucken“- Button
- „Inhalt“- Button (da das Inhaltsverzeichnis nicht ständig sichtbar sein soll)
- „Index“- Button (da das Indexverzeichnis nicht ständig sichtbar sein soll)
- „Suchen“- Button mit Eingabefeld für Suchbegriff

Hinzu kommen noch folgende Anzeigen:

- Besuchte Seiten
- Bestandener Wissenstest

Beim Erstellen des Layouts ist also zu beachten, dass die genannten Elemente sinnvoll plaziert werden.

Zusätzliche Fenster

Immer wenn ein zusätzliches Fenster geöffnet werden muss, sollte dieses einem einheitlichen Erscheinungsbild entsprechen. Zusätzliche Fenster werden z.B. für das Inhaltsverzeichnis, das Index oder die Hilfeseite benötigt.

Es muss also auch ein Layout für zusätzlich geöffnete Fenster festgelegt werden.

Funktionalität

Dieser Teil beschreibt, welche Funktionalität die Arbeitsumgebung bieten sollte.

Navigation

Einen wesentlichen Bestandteil der Funktionalität macht die Navigation aus. Es muss eine Funktion zur direkten Anwahl eines beliebigen Kapitels zur Verfügung stehen sowie eine Funktion zum sequenziellen Durchlaufen der Inhalte. Natürlich muss auch vom Ergebnis der Suchfunktion aus direkt in des entsprechende Kapitel gesprungen werden können.

Um das Nachschlagen zu vereinfachen, sollten bei einem Rollover über ein Kapitelllink Informationen zu diesem angezeigt werden. Dies könnte durch eine Kurzbeschreibung zu den Kapiteln sowie durch ein beschreibendes Bild geschehen.

Um die Navigation zu erleichtern, sollte kenntlich gemacht sein, welche Kapitel zu einem früheren Zeitpunkt bereits besucht wurden.

Inhalt

Das Inhaltsverzeichnis sollte in einem eigenen Fenster dargestellt werden, da dieses während des normalen Lernens nicht benötigt wird, jedoch auch parallel zum eigentlichen Inhalt dargestellt werden soll.

Es ist sinnvoll, das Inhaltsverzeichnis nicht einfach als eine lange Liste darzustellen, da geplant ist umfangreiche Lerninhalte einzubinden. Eine lange Liste wäre dann sehr unübersichtlich. Besser ist in diesem Fall, wenn beim ersten Aufruf nur die Hauptthemen dargestellt werden und erst nach einem Mausklick weitere Unterkapitel sichtbar werden.

Index

Es wehre gut das Index ebenfalls in einem eigenem Fenster darzustellen. Auch dieses wird nur selten benötigt und sollte parallel zu dem eigentlichen Inhalt sichtbar sein. Es sollte wie für ein Index üblich alphabetisch sortiert sein und über Sprungmarken zu den einzelnen Buchstaben verfügen.

Lesezeichen

Lesezeichen brauchen nur im Index sichtbar sein, um unnötig viele Informationen zu vermeiden. Sie sollten lokal gespeichert werden. Es sollten mehrere Lesezeichen gesetzt werden können. Es wäre sinnvoll, wenn das zuletzt besuchte Kapitel anhand eines Lesezeichens erkennbar wäre.

Notizen

Es muss auch die Möglichkeit bestehen Notizen einzugeben, welche am rechten Bildschirmrand angezeigt werden. Solche Notizen sollen lokal gespeichert werden und bei einer späteren Betrachtung des Kapitels wieder angezeigt werden.

Wissenstest

Die erreichten Ergebnisse bei Wissenstests müssen lokal gespeichert werden. Eine Auswertung sollte möglichst zu jeder einzelnen Frage möglich sein. Fehlerhaft beantwortete Fragen sollten Verweise auf die Textstellen anbieten, wo der entsprechende Stoff noch einmal nachgelesen werden kann.

Der Lernende sollte ständig einen Überblick darüber haben, in wieweit er den zu lernenden Stoff bereits verstanden hat und in welchen Kapiteln noch Lücken sind.

Drucken

Es ist sinnvoll, die Druckfunktion so zu gestalten, dass sie es erlaubt den Inhalt einzelner Kapitel auszudrucken. Die Navigationselemente sollen jedoch nicht mit gedruckt werden.

Hilfe

Die Hilfeseite sollte Informationen zur Bedienung der Arbeitsumgebung leicht verständlich und übersichtlich zur Verfügung stellen. Hinzu kommen Informationen zum Herausgeber.

Plug-Ins

Es ist wichtig sicherzustellen, dass für alle Inhalte, die über einfachen Text oder Bilder hinaus gehen die benötigten Plug-Ins existieren. Fehlt ein Plug-In so muss ein alternatives Element einspringen, bzw. dieses Element muss ganz weggelassen werden. Es soll nicht vorkommen, dass fehlende oder zu alte Plug-Ins zu Fehlermeldungen führen oder ständig Meldungen erscheinen, die den Benutzer auffordern, das entsprechende Plug-In zu installieren.

Umsetzung

Nachdem nun die Rahmenbedingungen und der Entwurf abgehandelt wurden, geht es auch an die Umsetzung der Arbeitsumgebung. Der folgende Teil beschreibt die Feinheiten und die technischen Details.

Umgebung

Bevor ich mich jedoch intensiv mit der Arbeitsumgebung auseinander setze und Entscheidungen zur Umsetzung treffe, gilt es, die Sprachen, in welchen die Arbeitsumgebung erstellt wird, festzulegen. Erst danach macht es Sinn, sich mit den Feinheiten zu beschäftigen.

Sprachen

Dies scheint auf den ersten Blick widersprüchlich. Es kommt jedoch nicht nur Html zur Erstellung von Websites in Betracht. Genauso könnte ich ja auch Xml oder Xhtml verwenden. Hierzu kommt noch die Möglichkeit der serverseitigen Unterstützung der Websites, welche wieder eine Fülle von weiteren Sprachen wie PHP oder Perl zur Verfügung stellt.

Ich habe mich entschieden, soweit wie möglich auf serverseitige Lösungen zu verzichten. Damit möchte ich die Möglichkeit offen halten, dass die Arbeitsumgebung auch offline ohne eine Verbindung zu einem Webserver lauffähig ist. Es ist in Deutschland immer noch üblich, dass Benutzer des Internets für die Zeit bezahlen müssen, die sie online sind. Es ist unwahrscheinlich, dass jemand entspannt lernt, wenn er immer im Hinterkopf die Kosten sieht. Ist die Website in ihren Grundfunktionen schon auf einen Webserver angewiesen, würde dies ein Offline Arbeiten von vornherein ausschließen.

Die Arbeitsumgebung soll also die grundlegende Funktionalität auch ohne serverseitige Unterstützung gewährleisten. Diese Einschränkung lässt weiterhin offen, dass

für Funktionen die während des normalen Lernens nicht benötigt werden, z.B. eine Volltextsuche, eine serverseitige Unterstützung erforderlich ist.

Die Arbeitsumgebung soll in Html erstellt werden. Durch diese Entscheidung stelle ich sicher, dass auch ältere Browser sie darstellen können. Selbst der Einsatz von Dhtml würde Browser der Dreier-Generation ausschließen. Einer Erweiterung der Möglichkeiten durch den Einsatz von JavaScript spricht nichts entgegen, jedoch darf auch hier nur der Befehlssatz benutzt werden, der auch von älteren Browsern unterstützt wird. Auch Java-Applets dürfen zum Einsatz kommen, sollten jedoch nur wenn nötig eingesetzt werden, da dies die Ladezeiten steigern würde.

Serverseitig sollte soweit möglich nur Perl zum Einsatz kommen. Perl wird von fast allen Webspace Providern unterstützt und führt von daher kaum zu Schwierigkeiten, wenn die Webseite auf einen anderen Server portiert werden soll. Die Portierbarkeit auf andere Server als den, den ich zum Testen einsetze, ist erforderlich, da es nicht sicher ist, dass der Kunde, für den die Webseite erstellt wird, dauerhaft bei dem selben Webspace Provider bleibt.

Layout

Auch das Layout ist nicht völlig sprachenunabhängig. Würde beispielsweise die Arbeitsumgebung teilweise mit Flash realisiert werden, so könnten schon beim Layout wesentlich umfangreichere dynamische Elemente eingeplant werden als bei dem einfachen Einsatz von Html.

Globales Erscheinungsbild

Da die Arbeitsumgebung ein Teil von Gastromarketing ist, soll sie auch als ein solcher erkannt werden. Dies erreiche ich durch eine Farbgebung, welche sich an die der bestehenden Gastromarketing-Seiten orientiert.

Die Weiterbildungsseiten, die in der Arbeitsumgebungenthalten sind, werden über einen Link von Gastromarketing erreicht werden. Die Arbeitsumgebung soll in einem eigenen Fenster laufen. Um anzudeuten das es sich nur um einen Teil von Gastromarketing handelt, soll das Fenster der Arbeitsumgebung kleiner als das von Gastromarketing sein.

Aufteilung

Die Aufteilung beschreibt, wo welche Bedienelemente plaziert werden. Hierzu habe ich mehrere Entwürfe (Skizzen) gemacht und diese mit dem Creative Director vom Kiwi besprochen. Zu dem Entwurf, für den wir uns entschieden haben, habe ich dann die in der Abb. 1 und Abb. 2 dargestellten Vorlagen vom Screendesigner bekommen.

Abb. 1: Das Layout der Arbeitsumgebung

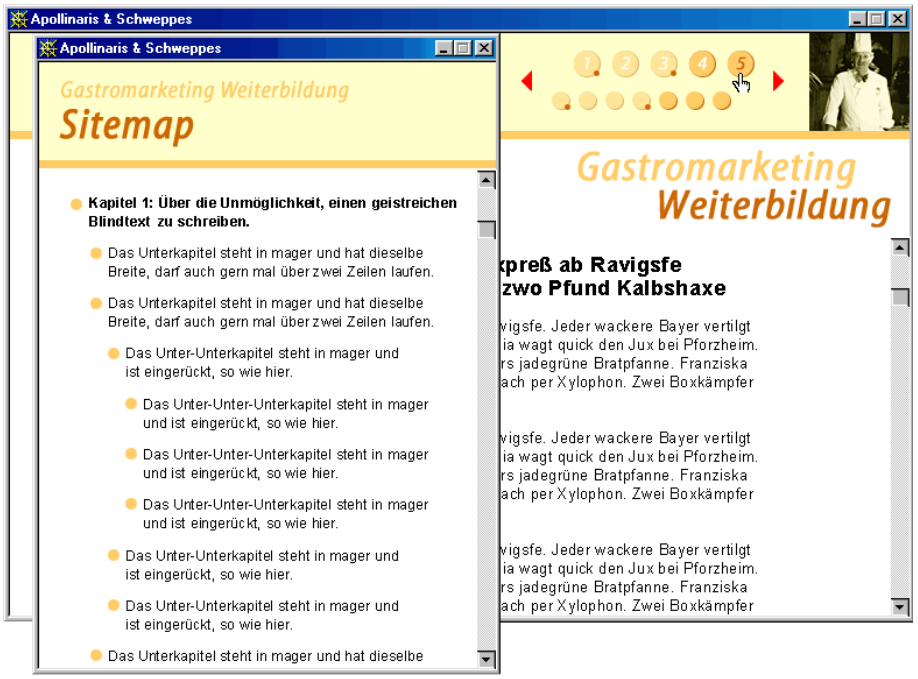


Abb. 2: Das Layout für Fenster

Diese Vorlagen wurden vom Designer in Photoshop erstellt und dienen bei der Implementierung als Richtlinie für die Farbwahl und die Positionierung der einzelnen Elemente.

Beschreibung

Links oben befinden sich die grundlegenden Bedienungselemente. Rechts daneben wird eine Übersicht über die zur aktuellen Lerneinheit existierenden Kapitel dargestellt. Über diese Buttons ist auch zu erkennen, welche Kapitel / Unterkapitel bereits besucht wurden und wo noch Übungen zu absolvieren sind. Ganz rechts wird ein Bild gezeigt, welches das aktuelle Kapitel repräsentiert. Links in der Mitte steht ein kurzer beschreibender Text zu dem aktuellen Kapitel. Rechts in der Mitte steht das Logo von Gastromarketing.

Im unteren Teil befindet sich der eigentliche Inhalt. Dieser Teil soll sich in drei Spalten aufteilen. In der linken Spalte sollen den Abschnitt beschreibende Grafiken stehen, in der Mittleren Spalte der Text und andere vermittelnde Inhalte, am rechten Rand stehen Bemerkungen des Anwenders.

Um eine übersichtliche und gut lesbare Form des Inhaltes sicherzustellen, soll das Fenster die dargestellte Breite nicht übersteigen.

Bei einem Rollover über einen der Button rechts oben soll sowohl das Bild als auch der beschreibende Text zum jeweiligen Kapitel angezeigt werden. Dies ermöglicht eine einfache Suche nach Inhalten.

Zusätzliche Fenster wie in Abb. 2 dargestellt teilen sich lediglich in zwei Teile. Im oberen Teil steht der Titel des Fensters, im unteren Teil der Inhalt. Die zusätzlichen Fenster sollen ungefähr die in Abb. 2 gezeigte Breite haben.

Funktionalität

Grundsätzlich gibt es drei Möglichkeiten, die geforderte Funktionalität zu realisieren.

Die einfachste Lösung scheint auf den ersten Blick die meistens übliche Methode, bei welcher auf jeder Seite Links zum nächsten und vorherigen Kapitel stehen. In diesem Fall müsste eine zusätzliche Seite erstellt werden, welche das Inhaltsverzeichnis enthält. Eine Anzeige im Inhaltsverzeichnis, welches Kapitel gerade besucht wird, ist so nicht möglich.

Eine andere Möglichkeit wäre, dass jede Seite zusätzlich zu den Links zur vorherigen und nächsten Seite noch per Funktionsaufruf bei dem Inhaltsverzeichnis bekanntgibt, dass sie gerade angezeigt wird. Im Inhaltsverzeichnis könnte dann alles weitere erledigt werden, damit angezeigt wird, welche Seite aktiv ist.

Die von mir gewählte Methode funktioniert auf andere Weise. Die einzelnen Seiten kennen nicht mehr die vorherige und nächste Seite. Stattdessen werden diese Informationen in einer separaten Datei gespeichert. Hierdurch steht ständig der gesamte Navigationsbaum zur Verfügung. Es müssen auch nicht zwei separate Navigationslisten erstellt werden, wie es in der zuvor vorgestellten Methode der Fall ist.

Kapitelliste

Sämtliche Informationen über den Navigationsbaum werden in einer Kapitelliste gespeichert. Es wird zu jeder Lerneinheit eine Kapitelliste erstellt. In dieser Liste stehen folgende Informationen zu jedem Kapitel:

- Name des Kapitels
- Dateiname der Website (Adresse)
- Kurze Beschreibung zu dem Kapitel
- Stichworte zu diesem Kapitel
- Beschreibendes Bild zu dem Kapitel

Die Bearbeitungsreihenfolge und der Zusammenhang zwischen Kapitel und Unterkapitel wird durch die Reihenfolge in der Liste und einem zusätzlichen Parameter bestimmt.

Dadurch, dass für jede Lerneinheit eine eigene Kapitelliste besteht, können Lerneinheiten einfach zu den Gastromarketing-Seiten hinzugefügt und auch wieder entfernt werden.

Navigation

Die gesamte Navigation läuft nur noch über JavaScript Funktionen, die das Austauschen der Inhalte übernehmen. Es kann auch leicht das aktuelle Kapitel angezeigt werden, da die Aktualisierung der Anzeige von diesen Funktionen gleich miterledigt wird.

Notizen

Notizen werden in einem separaten Fenster eingegeben, welches beim Anklicken einer Notiz geöffnet wird. Es kann zu jedem Absatz eine Notiz erstellt werden. Die Stellen, an denen noch keine Notizen eingegeben sind, jedoch zur Eingabe von Notizen zur Verfügung stehen, sind dezent markiert.

Wissenstest

Um eine sofortige Kontrolle der eingegebenen Antworten zu ermöglichen, soll zu jeder Frage ein Button zum Überprüfen der Antwort bestehen. Das Ergebnis der Antwortauswertung wird direkt bei der Antwort angezeigt. Zusätzlich wird in der Übersicht angezeigt, ob alle Fragen zu einem Kapitel und zu den Unterkapiteln richtig beantwortet wurden. So lässt sich leicht erkennen, in welchen Kapiteln noch Fragen unbeantwortet sind.

Implementierung

In den vorherigen Abschnitten habe ich bereits die Voraussetzungen und die Struktur der Website beschrieben. Im letzten Abschnitt bin ich dann auf die Besonderheiten der Umsetzung eingegangen. In diesem Abschnitt nun beschreibe ich die programmiertechnische Umsetzung.

Layout

Die Arbeitsumgebung habe ich in die in der Grafik (Abb. 3) dargestellten Frames unterteilt.

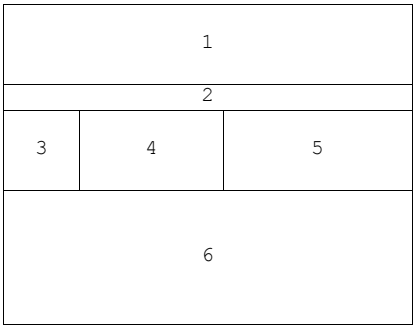


Abb. 3: Aufteilung in Frames

Im ersten Frame befindet sich die gesamte Navigation, also alle Buttons der Arbeitsumgebung. Hierzu kommt noch das Vorschaubild.

Der zweite Frame enthält lediglich den Trennbalken als optisches Element. Im unsichtbaren Bereich findet das Abspielen von Sounds statt.

Der dritte Frame bleibt leer. Jedoch findet auch hier im unsichtbaren Bereich etwas statt. In diesem Frame wird geprüft welche Plug-Ins bereitstehen.

Im vierten Frame wird eine kurze Beschreibung zu dem aktuellen Kapitel angezeigt.

Der fünfte Frame enthält lediglich das „Gastromarketing Weiterbildung“-Logo.

Im sechsten Frame schließlich steht der zu lernende Inhalt. Die einzelnen Spalten dieses Frames sind über Tabellen realisiert.

Die Unterteilung der einzelnen Bereiche in Frames hat den Vorteil, dass sich die einzelnen Inhalte auch ohne den Einsatz von Dhtml ändern lassen.

Funktionalität

In diesem Abschnitt werde ich noch einmal vertieft auf die Bereiche eingehen, die nicht einfach durch Betrachten des Sourcecodes zu verstehen sind.

Verzeichnisstruktur

Die Arbeitsumgebung hat die folgende Verzeichnisstruktur:

- / - im Rootverzeichnis befindet sich die Einstiegsseite *index.htm*, welche das Fenster öffnet in dem die Arbeitsumgebung dargestellt wird. Weiterhin steht hier noch die Seite *weiterbildung.html*, welche die Aufteilung in die einzelne Frames übernimmt.
- */scripts* - hier befinden sich alle Skripts zur Arbeitsumgebung.
- */images* - hier sind alle Bilder zu finden, welche global benötigt werden.
- */images/navi* - hier stehen alle Bilder, welche für die Navigation benötigt werden.
- */htm* - hier befindet sich die Willkommensseite und alle weiteren Seiten, die direkt zur Arbeitsumgebung gehören. Hier ist auch die Datei style.css zu finden, welche das globale Aussehen bestimmt.

- */htm/brandl_barkunde* - in diesem Verzeichnis steht die erste umgesetzte Lerneinheit.
- */htm/beispiel* - in diesem Verzeichnis befinden sich Beispiele zu den Funktionen des Generators.
- */htm/brandl_barkunde/images* - hier befinden sich Bilder zur Lerneinheit
- */htm/brandl_barkunde/src* - hier befinden sich die Quellen zur Lerneinheit, welche der Generator für die Erstellung der Lerneinheit benutzt.

Jede Lerneinheit wird in einem eigenem Verzeichnis erstellt, welches unterhalb von /htm anzusiedeln ist.

Die Verzeichnisse *.../src* werden nicht auf den Webserver gestellt, sie werden nur zur Generierung der Lerneinheiten benötigt.

Kapitelliste

Die Kapitelliste ist eine verkettete Liste. Jeder Eintrag kennt das nächste Kapitel und das nächste Unterkapitel. Die Abb. 4 zeigt ein Beispiel einer Verkettung.

```
K0 - K1 - K2 - K3
     |          |
     K1.1       K3.1 - K3.2 - K3.3
                |              |
                K3.1.1         K3.3.1 - K3.3.2
```

Abb. 4: Beispiel zur Verkettung von Kapiteln

Erstellt werden Einträge in die Kapitelliste, indem ein neues Objekt vom Type 'Kapitel' angelegt wird. Diese Klasse ist in der Datei kapitel.js definiert. Der Konstruktor von 'Kapitel' übernimmt das Einketten in die Liste. In der Datei kapitelliste.js steht ein JavaScript-Programm, welches lediglich neue Objekte von Kapitel erzeugt. Das hat dann folgendes Aussehen:

```
new Kapitel( 1, name, adr, beschr, stichworte, bild );    // Buch 1
  new Kapitel( 2, name, adr, beschr, stichworte, bild );  // Kapitel 1
  new Kapitel( 2, name, adr, beschr, stichworte, bild );  // Kapitel 2
    new Kapitel( 3, name, adr, beschr, stichworte, bild );// Kapitel 2.1
  new Kapitel( 2, name, adr, beschr, stichworte, bild );  // Kapitel 3

new Kapitel( 1, name, adr, beschr, stichworte, bild );    // Buch 2
```

Der erste Parameter gibt lediglich an, ob es sich bei dem Eintrag um ein Buch/Lerneinheit (1), ein Kapitel (2), ein Unterkapitel (3) oder ein Unterunterkapitel (4) handelt. Durchnummeriert werden die Kapitel dann selbstständig.

Die Kapitelliste ist in der Datei *kapitelliste.js* gespeichert. Eine solche Datei befindet sich in dem Hauptverzeichnis zu jeder Lerneinheit.

Welche Lerneinheiten und somit auch Kapitellisten bereitstehen, ist in der Datei *bücherliste.js* im Verzeichnis */scripts* eingetragen. Über diese Datei werden Lerneinheiten also zur Arbeitsumgebung hinzugefügt oder entfernt.

Navigation

Alle zur Navigation benötigten Funktionen befinden sich in der Datei *navibar.js*. In dieser Datei ist zu jedem Button eine gleichnamige Funktion definiert, welche beim Klick auf einen Button aufgerufen wird.

Zusätzlich zu diesen Funktionen sind in dieser Datei noch Funktionen definiert, die zur Darstellung der Vorschau und der Statusinformationen benötigt werden.

Der Vorschautext wird direkt in den Frame geschrieben. Hierfür wird also keine neue Seite geladen, sondern eine generiert. Das hat den Vorteil, dass es erheblich schneller geht als das Laden einer neuen Seite. Der darzustellende Text ist in der Klasse 'Kapitel' gespeichert.

Inhalt

Die Funktionen, die zur Darstellung des Inhaltsverzeichnisses benötigt werden sind in der Datei sitemap.js definiert.

Um nicht lediglich eine starre Liste anzuzeigen wird das Inhaltsverzeichnis beim Öffnen des Fensters generiert. Beim Anklicken von Kapiteln wird die Anzeige mit aus- oder eingeklappten Unterkapiteln erneut dargestellt. Die benötigten Informationen zur Darstellung des Inhaltsverzeichnisses holt sich die Anzeigefunktion aus der Kapitelliste.

Index

Alle Funktionen zur Darstellung eines Indexverzeichnisses sind in *content.js* definiert. Die Suche im Index ist jedoch in *navibar.js* gespeichert, da diese Funktion ohne Aufruf einer neuen Seite funktioniert.

Das Index wird erst beim Öffnen des Fensters generiert. Auch hier stammen die benötigten Informationen aus der Kapitelliste.

Utilities

Zu dieser Rubrik zählen alle Funktionen, die nicht direkt zur Navigation gehören oder zur Darstellung des Inhaltes benötigt werden. Sie sind in der Datei *utility.js* gespeichert.

Besonders interessant sind hier die Funktionen zur Überprüfung von Antworten auf Wissensfragen. Diese Funktionen überprüfen, ob Antworten richtig sind, speichern die Antwort in Cookies und zeigen an ob die Antwort richtig war. Dieser Teil muss mit Sicherheit noch erweitert werden, damit eine umfassende und komfortable Abfrage von Gelerntem möglich ist. Bisher sind nur grundlegende Abfragemethoden umgesetzt.

Ebenso interessant sind die Funktionen zur Eingabe und Darstellung von Notizen. Auch diese Informationen werden in Cookies gespeichert.

Zusätzlich sind in dieser Datei noch die Funktionen zum Vorladen von Bildern, Öffnen von Fenstern, speichern von Cookies und Abspielen von Sound realisiert.

Drucken

Um lediglich den Inhalt drucken zu können ist es beim MSIE erforderlich ein VBScript auf jeder Inhaltsseite einzufügen, welches das Drucken eines einzelnen Frames ermöglicht.

Plug-Ins

Die Abfrage der installierten Plug-Ins geschieht bereits beim Starten der Arbeitsumgebung in einem eigenen Frame. Zu jedem abgefragten Plug-In wird eine globale Variable gesetzt, welche speichert, ob das entsprechende Plug-In vorhanden ist.

Auf diese Weise ist beim Anzeigen einer Seite bereits bekannt, ob alle benötigten Plug-Ins vorhanden sind. Es muss nicht immer wieder auf das selbe Plug-In getestet

werden. Informationen über Plug-Ins, bei denen der Test, ob es existiert, länger dauert, stehen ebenfalls zum benötigten Zeitpunkt bereit.

Test

Die Arbeitsumgebung wurde in drei Phasen getestet.

In der ersten Phase habe ich jede Funktionen einzeln und im Zusammenspiel mit der Arbeitsumgebung getestet. Hierdurch konnte ich sicherstellen, dass jede Funktion auch so reagiert wie ich es erwarte. Dieser Teil fand während der Erstellung der Arbeitsumgebung statt.

In der zweiten Phase habe ich das Zusammenspiel der Funktionen untereinander und mit der Arbeitsumgebung getestet. Hierzu habe ich die Beispielseiten verwendet. Getestet habe ich unter anderem das sequentielle Navigieren innerhalb der Lerneinheit und das zufällige Navigieren über das Inhaltsverzeichnis und Stichwortverzeichnis. Hinzu kam das willkürliche richtige und falsche Beantworten von Fragen, das Setzen von Notizen und das Drucken von einzelnen Seiten.

Diese Tests habe ich unter verschiedenen Betriebsystemen, unterschiedlichen Browsern und Browser-Versionen wiederholt.

In der dritten Phase wurde die Arbeitsumgebung mit der „Lerneinheit“ ’Brandels Barkunde’ verbunden. Die Tests fanden überwiegend durch Mitarbeiter der Firma Kiwi statt. Es wurde nicht mehr explizit nach Fehlern gesucht, sondern die Arbeitsumgebung im Alltagsgebrauch getestet.

Testergebnis

Die Tests zeigten, dass die Arbeitsumgebung entsprechend der Aufgabenstellung arbeitet.

Alle während der Tests noch auftretenden Fehler konnten behoben werden. Zusätzlich zu der Behebung der Fehler wurde die Arbeitsumgebung in Bezug auf die Benutzerfreundlichkeit noch in einigen Feinheiten optimiert.

Zu diesen Feinheiten gehört z.B. die Art, wie die Notizen dargestellt werden und die Anpassung von StyleSheet und Fenstergrößen an weitere Browser.

Es hat sich auch gezeigt, dass es vorteilhaft wäre, wenn die Möglichkeit besteht, mehrere Benutzerprofile abzuspeichern. Es müssten die bereits besuchten Seiten und die beantworteten Fragen benutzerabhängig gespeichert werden. Dies würde die Möglichkeit bieten, dass mehrere Anwender an einem PC arbeiten.

KAPITEL 3

Generator

Bei dem Generator handelt es sich um einen Compiler zur Übersetzung von Seiten mit erweitertem Befehlssatz in normales Html und zur Einbindung dieser Html- Seiten in die → Arbeitsumgebung. Durch den Einsatz des Generators wird die Erstellung der Lerninhalte erheblich erleichtert und beschleunigt. Dies bewerkstelligt der Generator, indem er die Inhalte von einer einfach herzustellenden und übersichtlichen Form, die unabhängig von dem Generator erstellt wird, in eine an die Arbeitsumgebung angepasste Form bringt. Außerdem generiert er das Inhalts- und Stichwortverzeichnis. Ein weiteres Feature ist, dass der Generator selbst definierte Erweiterungen in Html übersetzt und somit die Gestaltung der Lerninhalte flexibel an sich ändernde Bedürfnisse angepasst wird.

Rahmenbedingungen

Zielgruppe

Der Generator soll von Anwendern bedient werden, die nur grundlegende Kenntnisse im Umgang mit Computern und MS-Windows haben. Die Benutzer haben auch grundlegende Kenntnisse in der Erstellung von Html Dokumenten mit Hilfe eines Html Editors. Der Schwerpunkt der Arbeit soll in der Erstellung neuer Inhalte liegen und nicht in der Gestaltung und dem Design.

Plattform

Um einen einfachen und unkomplizierten Einsatz gewährleisten zu können, muss der Generator unter einer grafischen Oberfläche laufen. Da die Übersetzung von umfangreichen Inhalten sehr rechenintensiv ist, ist der Einsatz eines leistungsstarken PC's zu empfehlen. Sobald mehrere Anwender parallel an den Inhalten arbeiten, ist eine Netzwerkanbindung von Vorteil. Dadurch kann auf ein zentral definiertes Layout zurückgegriffen werden, und eventuelle Änderungen fließen sofort in alle Entwicklungen ein.

Funktionen

Die Funktionen des Generators gliedern sich in zwei Teile:

- Übersetzen von Erweiterungen in Html
- Einbinden der Inhalte in die Arbeitsumgebung

Die Übersetzung von Erweiterungen in Html ist erforderlich, da die Inhalte in einem einfachen übersichtlichen Stil geschrieben werden sollen. Trotzdem sollen beim Layout die gestalterischen Möglichkeiten von Html zur Verfügung stehen. Es muss also zusätzliche, einfach zu lernende Elemente geben, welche die Gestaltung beeinflussen. Diese Elemente müssen jedoch in standart-Html übersetzt werden, damit die Inhalte von normalen Browsern angezeigt werden können. Die Übersetzung hat außerdem den Vorteil, dass z.B. standard-Kopf- und Fußzeilen eingesetzt werden können, ohne dass dies bei der Erstellung beachtet werden muss.

Das Einbinden der Inhalte in die Arbeitsumgebung ist die zweite Aufgabe. Diese Funktion erstellt das Inhalts- und Stichwortverzeichnis, generiert die → Verlinkung zu den anderen Seiten und passt das Layout an die Arbeitsumgebung an.

Folgerungen

Zusammenfassend ergibt sich aus den beschriebenen Kriterien Folgendes:

Um eine Einarbeitungszeit bei der Benutzung des Generators zu vermeiden, sollten die Erweiterungen Html-Syntax haben. Außerdem sollte der Generator über ein grafisches Benutzerinterface verfügen und nur mit grundlegenden Einstelloptionen ausgestattet sein. Da die Anwender unter MS-Windows arbeiten und sich mit dieser Oberfläche auskennen, sollte auch der Generator unter MS-Windows laufen. Die Erstellung der Inhalte sollte weiterhin unter Zuhilfenahme eines Html-Editors möglich sein und wenig von der bisherigen Erstellung von Webseiten abweichen. Abweichungen von der üblichen Erstellung von Webseiten sollten sich weitgehend auf Erleichterungen beschränken und den Anwender nicht mit zusätzlichen Regeln oder Einschränkungen belasten. Es sollte möglich sein, die erstellten Inhalte auch schon vor der Übersetzung zu betrachten um eine grobe Vorstellung von dem endgültigen Ergebnis zu erhalten, was psychologisch wichtig ist.

Die Positionierung und Farbgebung der Inhalte sollte so weit wie möglich automatisch geschehen und –sofern überhaupt nötig– einfach durch Schlüsselworte beeinflusst werden. Da es denkbar ist, dass das Design der Seiten geändert werden soll, bietet es sich, an die Erweiterung des Html-Befehlssatzes nicht statisch in den Generator einzubinden, sondern dynamisch, um eine nachträgliche Änderung zu ermöglichen. Eine dynamische Einbindung würde auch die Erstellung neuer Erweiterungen vereinfachen.

Es wäre auch vorteilhaft, wenn die Einbindung der Inhalte in die Arbeitsumgebung über Html-erweiterungen gesteuert werden könnte, was den Generator simpel halten und universell einsetzbar machen würde. Dadurch könnte der Generator leicht an andere Projekte angepasst werden.

Entwurf

Die Erstellung des Generators gliedert sich in drei Teile:

- Die Benutzerschnittstelle
- Der Übersetzer
- Der erweiterte Befehlssatz

Alle Teile sollten eigenständig sein, um eine leichte Anpassung an geänderte Bedürfnisse zu gewährleisten. Zu beachten ist das die Benutzerschnittstelle über eine festzulegende Schnittstelle mit dem Übersetzer verbunden werden muss. Ebenso ist die Form der Erweiterung (Syntax) festzulegen. Das Folgende Diagramm (Abb. 5) zeigt das Zusammenspiel der einzelteile.

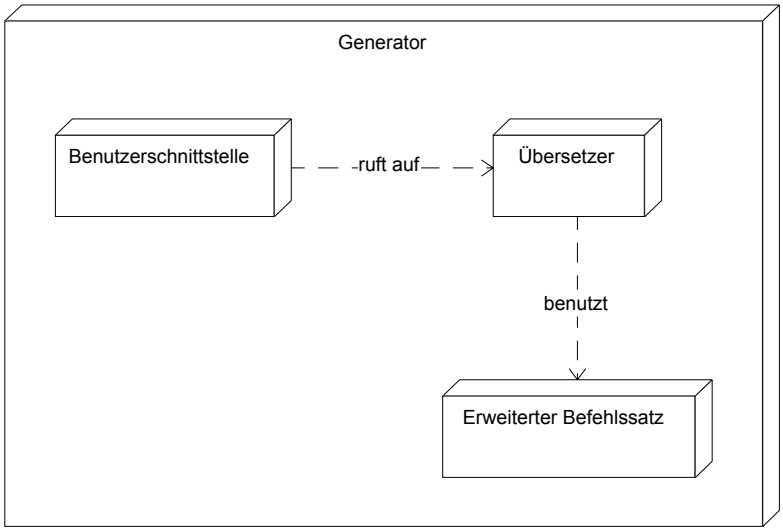


Abb. 5: Der Generator und seine Elemente

Benutzerschnittstelle

Die Benutzerschnittstelle ist der Teil des Generators mit dem der Anwender in Berührung kommt. Daher muss sie übersichtlich und leicht bedienbar sein. Sie muss über ein grafisches Benutzerinterface verfügen, soll nur grundlegende Einstelloptionen bieten und Statusmeldungen zum Fortschritt des Übersetzungsvorganges ausgeben.

Bedienelemente

Aus den Anforderungen ergibt sich, dass die Benutzerschnittstelle über folgende Elemente verfügen muss:

- Anzeige und Auswahl der zu übersetzenden Datei
- Ausgabe des Übersetzungsfortschritts
- Ausgabe von Statusmeldungen (Warnungen, Fehler)
- Button, der den Übersetzungsvorgang einleitet / stoppt

Die genannten Elemente werden während der Arbeit mit dem Generator immer wieder benötigt und sollten daher auch ständig sichtbar bzw. einfach erreichbar sein.

Zusätzlich wären noch folgende Bedienelemente wünschenswert:

- Festlegen der Arbeitsverzeichnisse
- Laden und Speichern der aktuellen Einstellungen
- Einstellen des Debuglevels

Da die zuletzt genannten Elemente jedoch nur gelegentlich benötigt werden, ist es sinnvoll, dass diese nicht ständig sichtbar sind.

Erweiterter Befehlssatz

Dieser Teil beschreibt in welcher Art Erweiterungen in die Dokumente einfließen und wie diese realisiert werden.

Syntax

Wie bereits festgestellt sollen sich die Erweiterungen aus der Sicht des Entwicklers möglichst nahtlos in die gewohnte Erstellung von Web-Sites integrieren. Um dies zu gewährleisten, ist es sinnvoll, dass Erweiterungen die Syntax von Html verwenden. Erweiterungen sind also Tags, deren Bedeutung selbst festgelegt wird. Wenn Erweiterungen syntaktisch identisch mit der übrigen Website sind, hat das auch noch den Vorteil, dass sich auch Tags, die in Html bekannt sind, mit einer neuen oder erweiterten Bedeutung belegen lassen. Das ist dann sinnvoll, wenn z.B. der Body-Tag grundsätzlich eine bestimmte Hintergrundfarbe erhalten soll, ohne dass dieses explizit angegeben wird. Wenn nun also im Quellcode der Tag: `<body>` auftaucht, wird dieser vom Generator durch den Tag `<body bgcolor="#00FF00">` ersetzt.

Erstellung

Zu bedenken ist, dass die Parameter der Tags bei der Definition von Erweiterungen zur Verfügung stehen müssen. Zusätzlich sollen auch Fallunterscheidungen auf Grund der Parameter möglich sein. Wenn beispielsweise bei einem Tag ein bestimmter Parameter angegeben ist, ein soll zusätzlicher Html-Code eingefügt werden. Als Beispiel könnte bei dem Image-Tag ein weiteres Bild angegeben werden, welches bei einem Rollover angezeigt werden soll. Ist dieser zusätzliche Parameter nicht angegeben, soll auch zu dem Image-Tag kein zusätzlicher Code für den Rollover eingebunden werden.

Häufig ist es auch erforderlich, Zustände zu speichern oder einfache Rechnungen durchzuführen, die schon während des Übersetzungsvorganges gebraucht werden. Auch das Ausgeben von Statusinformationen während des Übersetzungsvorganges gehört zu den interessanten Hilfsmitteln.

Die in diesem Abschnitt geforderten Eigenschaften der Erweiterungen gehen deutlich über ein einfaches Austauschen von Tags gegen Html-Code hinaus. Vielmehr erinnern sie an die Eigenschaften von JavaScript, nur mit der Einschränkung, dass diese Funktionalität schon beim Übersetzen benötigt wird und nicht erst beim Betrachten der eigentlichen Seite. Es wird also, neben der einfachen Benutzung von Html, eine Makrosprache benötigt, die Einfluss auf den Übersetzungsvorgang nehmen kann.

Makrosprache

Makros, die in Erweiterungen eingesetzt werden, müssen eine Syntax haben, die leicht zu erlernen ist. Unter Berücksichtigung des Einsatzortes bietet es sich an, die Syntax soweit möglich an der von JavaScript zu orientieren. Dies beinhaltet die arith-

metischen Operationen sowie die Ablaufkontrolle. Da es sich bei den Makros voraussichtlich um kleine Aufgaben handelt, ist es nicht erforderlich, die Definition eigener Funktionen zu unterstützen.

Der Geltungsbereich von Variablen muss an die Aufgabenstellung angepasst sein. Im Detail bedeutet dies, dass Parameter der Tags als Variablen zur Verfügung stehen müssen. Zusätzlich müssen Variablen ebenso wie die Parameter, im Teil der Erweiterungen, der durch Html beschrieben wird, zur Verfügung stehen. Variablen müssen während des gesamten Übersetzungsvorganges ihren Wert behalten, damit sie auch für Makros anderer Erweiterungen zur Verfügung stehen. Variablen müssen auch dann noch Ihren Wert behalten, wenn weitere Dokumente übersetzt werden.

Speicherung

Nun bleibt noch die Frage zu klären, wie die Erweiterungen gespeichert werden. Eine Möglichkeit wäre, die Erweiterungen fest in den Übersetzer einzubauen. Dies scheidet allerdings von vornherein aus, da es zu unflexibel ist. Eine bessere Möglichkeit besteht darin, alle Definitionen in einer Datei abzulegen, in der die einzelnen Erweiterungen durch Schlüsselworte eingeleitet und benannt werden. Als drittes bleibt noch die Möglichkeit, für jeden Tag eine eigene Datei anzulegen. Ich habe mich für die letzte Variante aus folgenden Gründen entschieden:

- Es ist leicht, einen Überblick über die bereits existierenden Erweiterungen zu behalten
- Einzelne Erweiterungen lassen sich ohne großen Aufwand aktivieren und deaktivieren
- Es ist einfach, eine individuelle Liste von Erweiterungen zu erstellen
- Ein Fehler in einer Erweiterung hat keine Auswirkung auf andere Erweiterungen
- Fehler bei der Erstellung von Erweiterungen lassen sich leicht eingrenzen

Übersetzer

Dieser Teil des Generators soll die eigentliche Übersetzungsaufgabe erledigen. Dies geschiet, indem er Erweiterungen gegen deren entsprechenden Html-Code austauscht. Der Benutzer kommt mit dem Übersetzer nicht direkt in Kontakt, sondern bedient diesen lediglich über die Benutzerschnittstelle.

Da der Übersetzer jedoch ein eigenständiges Modul sein soll, ist es vorteilhaft wenn dieses auch aus der Kommandozeile heraus aufgerufen werden kann um ihn besser testen zu können.

Funktionen

Aus den Anforderungen an den Generator und insbesondere an die Erweiterungen ergeben sich folgende Funktionen für den Übersetzer:

- Erkennen von Tags, die nicht zum normalen Html-Befehlssatz gehören
- Erkennen von Tags, die eine neue Bedeutung erhalten haben
- Austauschen von Tags gegen deren Definition
- Auswerten der Tagdefinitionen (Erweiterungen)
- Austauschen der in den Tagdefinitionen enthaltenen Variablen gegen deren Wert
- Auswerten der in den Tagdefinition enthaltenen Makros
- Austauschen der Makros gegen deren Ergebnis

All dies sind Aufgaben, wie sie von einem Compiler bzw. einem Interpreter erledigt werden. Und doch sind sie so verschieden, dass es ungeschickt wäre, alle Aufgaben mit einem einzigen Compiler durchführen zu wollen. Besser ist es, dies in Teilbereiche zu unterteilen und jedes für sich zu kompilieren. Dabei ergeben sich folgende Bereiche:

- Die Dokumente
- Die Erweiterungen
- Die Makros

So übernimmt der erste Compiler die Aufgabe, die Dokumente nach Tags zu durchsuchen, die ersetzt werden sollen. Diese Tags werden durch das Ergebnis, das der Erweiterungscompiler liefert, ausgetauscht.

Der zweite Compiler durchsucht die Erweiterungen nach Variabeln und Makros. Die Variablen werden durch ihren Wert ersetzt und die Makros durch das Ergebnis, das der Makrocompiler liefert.

Der dritte Compiler oder besser Interpreter übersetzt die in den Erweiterungen enthaltenen Makros und gibt deren Ergebnis zurück.

Syntax im Detail

Nun ist es als nächstes wichtig, die Syntax der einzelnen Module exakt festzulegen. Hiermit sind die Dokumente gemeint, in denen zusätzliche Tags verwendet werden. Es geht weiterhin um die Syntax der Erweiterungen und der Makros.

Dokumente

Dieser Teil ist der einfachste dieses Kapitels, da hier keine neue Syntax erstellt wird. Alle Erweiterungen werden ausschließlich über die in Html übliche Syntax in die Dokumente integriert. Im Einzelnen heißt das, dass sich jede Erweiterung nach außen hin wie jeder andere Tag auch darstellt.

Das sieht dann so aus:

```
<Tagname  Bezeichner1='Wert1' Bezeichner2='Wert2'> Text </Tagname>
```

Da es auch möglich sein soll, die Bedeutung von bereits existierenden Tags zu „überladen“, ist es sinnvoll auch eine Möglichkeit zu bieten auf das „Original“ zuzugreifen. Um dies zu erreichen wird am Ende des Tags vor der schließenden spitzen Klammer ein Ausrufungszeichen „!“ angegeben, korrespondierend zu dem Ausrufungszeichen, das einen Kommentar einleitet.

Das sieht dann so aus:

```
<Tagname Bezeichner1='Wert1'!>
```

Ein ebenfalls nicht unwichtiger Punkt ist die Tatsache, dass im Html die Groß- und Kleinschreibung nicht relevant ist. Dies soll auch bei Erweiterungen der Fall sein damit es nicht zu unnötigen Fehlern kommt.

Erweiterungen

Erweiterungen sollen eine Syntax haben, die sich kaum von der Syntax der normalen Html-Dokumente unterscheidet. Die wesentlichen Unterschiede sind, dass Erweiterungen keine einleitenden Tags wie z.B. `<html>` brauchen, dass sie die Variablen erkennen und dass es möglich ist, Makros einzufügen.

Eine Erweiterung enthält in seiner einfachsten Form nur den Html-Code, der anstelle des Tags in das Dokument eingesetzt wird.

Sollen nun Variablen in einer Erweiterung eingefügt werden, so weicht dies von der üblichen Html-Syntax ab. Variablen werden benötigt, wenn z.B. einer der übergebenen Parameter benutzt werden soll. Diese Parameter werden nämlich in Variablen gespeichert. Wie im einzelnen Variablen definiert sind und welchen Geltungsbereich sie haben wird im nächsten Abschnitt erläutert. Vorab nur soviel: Variablen zeichnen sich durch ein führendes Dollarzeichen „$“ aus. Durch dieses werden sie auch innerhalb der Erweiterungen angesprochen. Taucht an beliebiger Stelle in einer Erweiterung ein Dollarzeichen gefolgt von einem Bezeichner (der Name einer Variable) auf, so wird dieser durch den Wert der Variable ersetzt. Das Ende eines Variablenbezeichners wird durch ein Zeichen angezeigt, das für diesen nicht zulässig ist.

Die Verwendung von Variablen könnte dann folgendermaßen aussehen:

```
<a href="$adresse">$text</a>
```

Zum Übersetzungszeitpunkt werden die Variablen `$adresse` und `$text` durch deren Inhalt ersetzt.

Neben den Variablen gibt es noch einen weiteren Zusatz, der vom üblichen Html abweicht. Gemeint ist die Verwendung von Makros. Wie Makros definiert sind, wird im folgenden Abschnitt erklärt. Eingefügt werden sie durch das einleitende Tag `<Makro>` und abgeschlossen durch das Tag `</Makro>`. Alles was zwischen diesen beiden Tags steht, ist ein Makro und wird besonders behandelt. Auch hierzu ein Beispiel:

```
<makro>
// Dieses Makro enthält nur diesen Kommentar
</makro>
```

Makros tauchen im fertig übersetzten Dokument nicht mehr auf. Von ihnen bleibt nur das übrig, was innerhalb der Makros explizit als Ausgabe angegeben ist.

Makros

Makros stellen eine eigenständige Programmiersprache dar. Diese orientiert sich stark an JavaScript; das bedeutet, die Syntax ist bei beiden Sprachen gleich, jedoch nicht deren Befehlssatz. Makros kennen Variablen, bei denen der Datentyp automatisch festgelegt wird. Auch müssen Variablen nicht definiert werden, sondern können einfach benutzt werden. Makros kennen einfache arithmetische Operationen und bedingte Ausführung von Code. Des Weiteren kennen Makros Schleifen und einige vorgegebene Funktionen. Jede Anweisung muss durch ein Semikolon abgeschlossen

werden. Es ist auch möglich, Anweisungen zu einem Block zusammen zu fassen; das wird realisier indem man mehrere Anweisungen in eine geschweifte Klammer setzt.

Variablen

Einen besonderen Status besitzen die Variablen. Ihr Gültigkeitsbereich soll nicht auf die Ausführung eines Makros beschränkt sein, jedenfalls nicht der Gültigkeitsbereich aller Variablen. Um die Handhabung der Variablen möglichst einfach zu gestalten, ist es erforderlich, Variablen mit unterschiedlichen Gültigkeitsbereich einzuführen.

Um den Grund zu verdeutlichen, bringe ich ein Beispiel. Angenommen, alle Variablen würden während des gesamten Übersetzungsvorganges ihre Gültigkeit behalten. Würde nun in einem Dokument ein neu definiertes Tag aufgerufen, so wären von diesem Moment an die Variablen, die den Parametern des Tags entsprechen, mit Werten belegt. Nehmen wir weiter an, der selbe Tag wird ein weiteres Mal verwendet, diesmal allerdings ohne Angabe von Parametern. Da die Parametervariablen jedoch bereits belegt wurden, würde diese Erweiterung mit den Parametern des ersten Aufrufs übersetzt, was sicher nicht im Interesse des Autors des Dokumentes wäre.

Es wäre auch nicht sinnvoll alle Variablen grundsätzlich nach Beenden eines Tags zu löschen. Dann wäre es beispielsweise nicht möglich mitzuzählen, wie oft ein Tag bereits verwendet wurde oder Werte bis zum Aufruf eines Weiteren Tags zu speichern.

Aus diesen und ähnlichen Gründen brauchen Variablen unterschiedliche Gültigkeitsbereiche. Die unterschiedlichen Gültigkeitsbereiche werden durch unterschiedliche Präfixe der Variablennamen gekennzeichnet. Folgende Einteilung der Gültigkeitsbereiche erweist sich als gut.

- Wärend des gesamten Übersetzungsvorganges (Präfix $$$)
- Wärend der Übersetzung eines Dokumentes (Präfix $$)
- Wärend der Übersetzung einer Erweiterung (Präfix $)
- Wärend der Übersetzung eines Makros (keine Präfix)

Wichtig ist noch, dass Variablen, die noch nicht mit einem Wert belegt sind, trotzdem verwendet werden dürfen, ohne dass dieses einen Fehler hervorruft. Praktisch ist z.B. eine automatische Vorbelegung mit Null oder einem leerem String.

Arithmetik

Makros sollen alle üblichen arithmetischen Operationen kennen. Ausgenommen sind lediglich die Bitoperationen, da diese nicht erforderlich sind. Es ist jedoch vorgesehen, diese Operationen bei Bedarf ohne großen Aufwand nachzurüsten.

Bedingte Ausführung

Um nicht lediglich einem starren Ablauf folgen zu müssen, sollten Makros auch eine bedingte Ausführung kennen. Hierzu gibt es die in JavaScript übliche if-Anweisung.

Schleifen

Letztlich gibt es noch die Möglichkeit, eine Anweisung innerhalb einer Schleife wiederkehrend auszuführen, um ein Mittel gegen den starr linearen Ablauf zu haben. Als Anweisungen die eine Schleife einleiten, stehen die while-Anweisung und die for-Anweisung zur Verfügung. Die Syntax ist auch hier die selbe wie in JavaScript.

Funktionen

Funktionen können nicht selbst definiert werden, was auch nicht wichtig ist, da Makros lediglich kleine Aufgaben erledigen, die selten über einige wenige Zeilen hinausgehen. Es muss jedoch einige vordefinierte Funktionen geben. Hierzu zählen z.B. eine Funktion zur Rückgabe des Ergebnisses. Dies könnte wie üblich mit Hilfe der return-Funktion geschehen, es erscheint mir jedoch sinnvoller, eine Art print-Funktion einzuführen, die ähnlich wie in Perl den Rückgabewert direkt an die Return-Variable anhängt. Dies erspart dem Nutzer selbst eine solche Variable einzuführen. Außerdem sind alle Rückgabewerte ja nichts anderes als Ausgaben in die Zieldatei.

Des weiteren werden Funktionen zum Aufruf eines bestimmten Teiles des Generators benötigt. Es müssen auch Funktionen zur File-Ein- und Ausgabe zur Verfügung stehen. Vergessen werden dürfen auch nicht Funktionen zur Stringmanipulation.

Welche Funktionen im Einzelnen benötigt werden, und wie diese benannt werden, wird sich während der Erstellung von Erweiterungen zeigen. Aus diesem Grund muss auch eine Möglichkeit gefunden werden, den Befehlssatz an Funktionen einfach aufzustocken, ohne sich jedes Mal von neuem mit der Materie des Compilers auseinandersetzen zu müssen.

Es soll ferner nicht ausgeschlossen sein, den Compiler um eine Möglichkeit zu erweitern, eigene Funktionen innerhalb der Makros zu definieren.

Umsetzung

Nachdem nun die Rahmenbedingungen und der Entwurf abgehandelt wurden, geht es an die Umsetzung. Dieser Teil beschreibt die Feinheiten und die technischen Details.

Umgebung

Bevor ich mich mit dem Generator ausgiebig beschäftige und Entscheidungen zur Umsetzung treffe, gilt es, die Sprache und Arbeitsumgebung, sowie die Werkzeuge mit dessen Hilfe der Generator erstellt wird, festzulegen. Erst danach macht es Sinn, sich um Feinheiten zu kümmern.

Sprache

Als Sprache für die Erstellung des Generators kommen erst einmal alle Sprachen in Betracht, die eine Erstellung von MS-Windows Programmen unterstützen. Ein weiteres wichtiges Kriterium ist die Unterstützung bei der Erstellung von Compilern.

Werkzeuge zur Erstellung von Compilern sind mir vor allem für C++ bekannt, dies allerdings nur unter Linux. Für MS-Windows war es mir nicht möglich, ein solches Werkzeug ausfindig zu machen. Auch ist es problematisch, einen C++-Compiler für Windows zu finden, der als freie Software erhältlich ist. Dies wäre wünschendswert, da die Firma, in dessen Auftrag ich diesen Generator erstelle, über keinen kommerziellen C++-Compiler verfügt, und sich die Anschaffung für eine einmalige Aufgabe nicht rechnet. Alternativ wäre es möglich, den Generator unter Linux zu erstellen und auf einem Linuxrechner, der als Server fungiert, laufen zu lassen. Dies würde allerdings zusätzlichen Hardwareaufwand bedeuten, wäre jedoch durchaus denkbar.

Neben den Werkzeugen zur Erstellung von Compilern in C++ existieren nahezu gleichwertige Werkzeuge für Java. Der Einsatz von Java ist in Bezug auf die Kompatibilität zu MS-Windows unproblematisch, da der Java-Compiler als freie Software für alle gängigen Plattformen erhältlich ist. Vorteilhaft ist dass es ebenfalls freie Entwicklungsplattformen für Java gibt. Zusätzlich wird auch die Erstellung von grafischen Benutzerinterfaces von Java unterstützt.

Da ich mit der Erstellung von Javaprogrammen vertraut bin und alle wesentlichen Bedingungen von dieser Sprache erfüllt werden, habe ich mich auf Java festgelegt. Wesentlich zu diese Entscheidung beigetragen hat auch, dass sich bei der Entwicklung unter Java, die geringsten Schwierigkeiten abzeichnen. Ich werde zur Unterstützung meiner Entwicklung die IDE von Borland, den → J-Builder, einsetzen, welcher als Freeware erhältlich ist.

Aussichten

Mit der Entscheidung für Java ergeben sich gleich mehrere neue Möglichkeiten. Neben dem Vorteil, dass der Generator dadurch plattformunabhängig ist, und dem Nachteil, dass er etwas langsamer arbeitet, als wenn er unter C++ erstellt würde, gilt es noch Folgendes zu bedenken: Wenn der Generator unter Java erstellt wird, besteht die Möglichkeit, ihn als Kommandozeilenapplikation zu erstellen, er kann ein grafisches Benutzerinterface erhalten und als Applet in einem Browser laufen.

Die Kommandozeilenvariante fällt heraus, da schon vorab festgestellt wurde, dass ein grafisches Benutzerinterface existieren muss.

Die Variante als Applet hat den Vorteil, dass sich der Generator direkt mit der Dokumentation verbinden lässt. Die übersetzten Seiten könnten direkt in einer Vorschau betrachtet werden; auch ohne sie zuvor zu speichern. Auch wäre es so möglich, von jedem Arbeitsplatz aus direkt auf den Generator zuzugreifen, ohne zuvor Java auf dem betreffenden Rechner einzurichten. Die ganze Sache hat jedoch einen Haken. Das Sicherheitskonzept von Java erlaubt es einem Applet nicht, auf lokale Dateien zuzugreifen, jedenfalls nicht im Normalfall. Möchte man dies umgehen, würde es bedeuten, dass im einfachsten Fall auch anderen Programmen der Zugriff auf lokale Dateien gestattet würde.

Bleibt noch die Variante als Applikation mit grafischem Benutzerinterface. Diese Variante hat den Nachteil, dass eine Vorschau der übersetzten Dateien nur sehr aufwendig zu realisieren ist. Hinzu kommt, dass ein → JRE auf den Rechnern, auf denen der Generator laufen soll, eingerichtet werden muss. Das Einrichten des JRE bedeutet nur einen geringen Aufwand, und der Generator läuft danach wie jede andere Applikation, auch ohne Sicherheitsprobleme.

Ich habe mich entschieden, sowohl die Applet- als auch die Applikationsvariante umzusetzen.

Die Applet Variante ist für Anwender, die den Komfort lieben und keine Bedenken wegen der Sicherheitsrisiken haben.

Die Applikation-Variante ist Vorzuziehen bei Anwendern, die auf Nummer sicher gehen wollen und daher lieber auf die Vorschau verzichten.

Der Compiler Compiler (Werkzeug zur Compilererstellung)

Wie ich bereits festgestellt habe benötige ich ein Werkzeug zur Erstellung der verschiedenen Compiler, die erforderlich sind. Unter Linux für C++ stehen für solche Aufgaben der Lex-Scannergenerator und der Yacc-Parsergenerator zur Verfügung. Auch für Java gibt es inzwischen schon mehrere Implementierungen solcher Werk-

zeuge. Es ist festzustellen, dass die Benutzung dieser Werkzeuge jedoch von der unter Linux für C++ bekannten abweicht. Nachdem ich mehrere Java-Implementierungen von Lex und Yacc verglichen habe, habe ich mich für die Lösung von C. Scott Ananian[1] entschieden, der bereits existierende Werkzeuge weiterentwickelt hat.

Die von C. Scott Ananian vorgestellten und von mir verwendeten Werkzeuge tragen die Bezeichnungen → JLex und → CUP. JLex ist das Java-Äquivalent zu dem bekannten Scannergenerator Lex. Bei CUP handelt es sich um das Java-Äquivalent zu dem bekannten LR(1) Parsergenerator Yacc.

Die ausschlaggebenden Vorteile dieser Java Implementierungen sind:

- Große Ähnlichkeit mit deren C++ Vorbildern
- Ein einfach gehaltenes Beispiel zum Einsatz
- Eine gute und umfassende Dokumentation

Java

Java ist nicht gleich Java. Inzwischen gibt es viele Versionen von Java, wobei jede ihre Vorteile und Nachteile hat. Entscheide ich mich für Java 1.0x, stelle ich damit sicher, dass es auf jedem Rechner läuft. Diese Version fällt jedoch schon daher heraus, da die Werkzeuge JLex und CUP ein Java 1.1x voraussetzen. Der Einsatz des J-Builders übertrifft das ganze noch, da dieser auf Java 1.2x aufbaut. Um zukunftsorientiert zu arbeiten, bestünde noch die Möglichkeit, Java 1.3x einzusetzen. Und es gibt noch eine Versionsvorgabe. Da ich beabsichtige unter anderem eine Applet-Variante zu erstellen, sollte höchstens Java 1.1x zum Einsatz kommen, da die aktuellen Browser lediglich diese Variante umgesetzt haben.

Konflikt

Trotz dieses Versions-Getümmels ist doch zu erkennen, das hier Widersprüche auftauchen. Zum einen soll höchstens Java 1.1x zum Einsatz kommen, um kompatibel zu den aktuellen Browsern zu bleiben, zum anderen kommt Java 1.2x zum Einsatz, sobald ich den J-Builder einsetze.

Lösung

Nun gibt es zwei Möglichkeiten, diesen Konflikt aus dem Weg zu räumen. Es ist möglich, einem Browser eine höhere Java-Version als Plug-In zur Verfügung zu stellen. Damit käme der Browser auch mit denen vom J-Builder erstellten Programmen zurecht, hätte aber den Nachteil, dass die Geschwindigkeitsoptimierung, welche die fest eingebaute → JVM integriert hat, wegfallen würde.

Die andere Möglichkeit besteht in der Flexibilität von Java. Auch wenn ein Programm mit einem Java 1.2x Compiler übersetzt wurde, so kann es doch mit einer JVM 1.1x ausgeführt werden, sofern lediglich Klassen und Funktionen zum Einsatz kommen, die bereits im Befehlssatz von Java 1.1x enthalten sind.

Wenn also sichergestellt ist, dass der Teil des Generators, der im Webinterface zum Einsatz kommt, mit dem Befehlssatz von Java 1.1x auskommt, so läuft der Generator ohne Update auch im Browser. Bei der Erstellung des grafischen Interfaces der Applikation kann hingegen der volle Umfang von Java 1.2x zum Einsatz kommen, also der J-Builder ohne Einschränkungen benutzt werden.

1. C. Scott Ananian, 1998

Übersetzer

Einen Teil des Generators stellt der Übersetzer dar. Der Übersetzer leistet die eigentliche Arbeit, das Austauschen der Erweiterungen gegen Html. Er ist nicht selbständig lauffähig, sondern braucht noch ein Ein-/Ausgabe Interface.

Der Übersetzer teilt sich wie bereits beschrieben in drei grundlegende Teile auf, von denen jedes einen eigenständigen Compiler darstellt.

- Der Dokument-Compiler
- Der Erweiterung-Compiler
- Der Makro-Compiler

Im Folgenden werden zunächst die Gemeinsamkeiten der drei Teile beschrieben und anschließend wird auf jedes Teil im Einzelnen eingegangen.

Gemeinsamkeiten

Die Compiler bestehen im Wesentlichen aus einer Scanner-Klasse, die mit JLex erstellt wird, einer Parser-Klasse, die mit CUP erstellt wird und einer Auswertungs-Klasse, welche die Steuerung übernimmt.

Die Steuerung hat die Aufgabe einen Quelltext und die Umgebungsvariablen bereitzustellen sowie die Initialisierung des Compilers vorzunehmen. Der Quelltext wird an den Scanner weitergereicht, der den Text in Tokens zerlegt. Ein Objekt dieses Scanners wird an den Parser übergeben, welcher die einzelnen Tokens vom Scanner abruft. Der Parser hat dann die Aufgabe, die Syntax zu erkennen, zu überprüfen und der Syntax entsprechende Schritte einzuleiten. Das Ergebnis des Parsers wird zum Schluss an die aufrufende Steuerklasse zurückgeliefert, welche dieses wiederum als Ergebnis bereit stellt. Bei dem Parser handelt es sich um einen LR(1) Parser.

Die folgende Grafik (Abb. 6) stellt das Zusammenspiel der bisher erwähnten Komponenten dar.

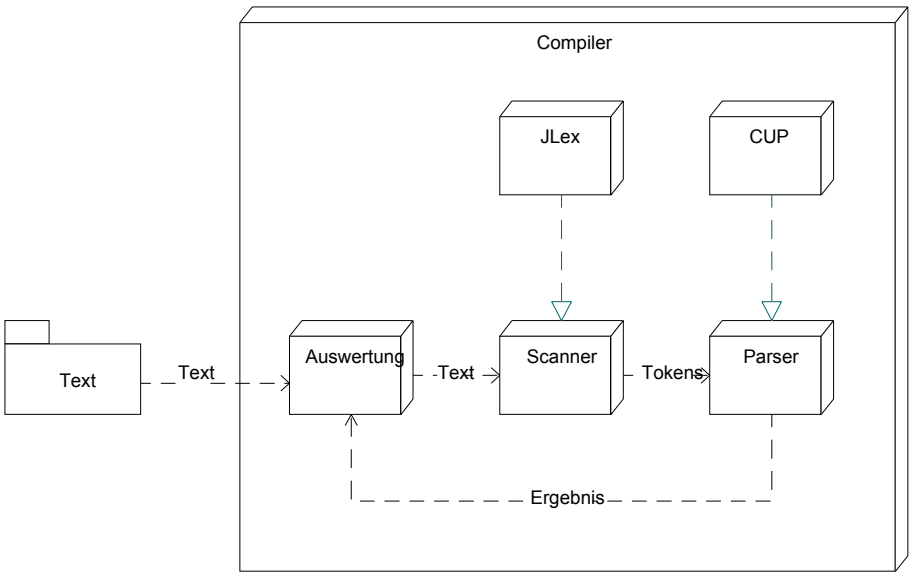


Abb. 6: Arbeitsweise der Compiler

Dokument-Compiler

Der Dokument-Compiler hat die Aufgabe, Html-Dokumente nach Tags zu durchsuchen und die Tags mit ihren Parametern an den Erweiterung-Compiler weiterzureichen. Im Html-Dokument enthaltene Kommentare werden gelöscht. Alles was kein Tag ist, bleibt unverändert. Die Tags werden durch das Ergebnis des Erweiterung-Compilers ersetzt.

Die nachfolgende Abb. 7 zeigt exemplarisch die Arbeitsweise des Document-Compilers anhand eines Beispieldokumentes.

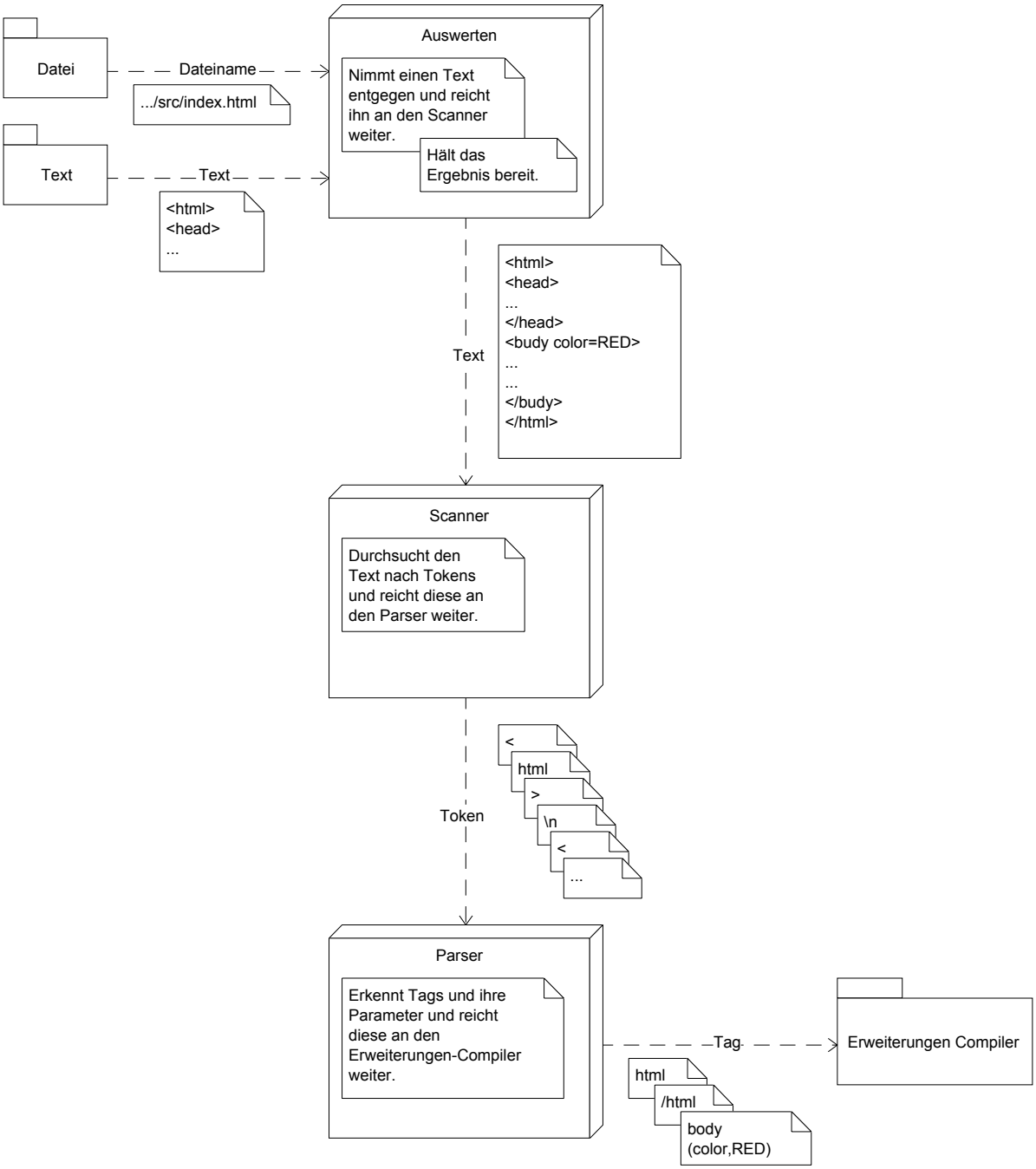


Abb. 7: Arbeitsweise des Dokument-Compilers

Nachdem ein Dokument an die Auswertung des Compilers übergeben wurde, wird der Text an den Scanner weitergereicht. Der Scanner zerlegt den Text in Tokens und übergibt diese an den Parser, welcher dann die Aufgabe hat, Tags und ihre Parameter zu erkennen und an den Erweiterung Compiler weiterzuleiten.

Während der Initialisierung wird auch der Makrovariablen-Speicher aufbereitet. Es bleiben also nur Projekt-Variablen erhalten. Zuvor definierte Dokument-, Tag- oder Makrovariablen verlassen an dieser Stelle ihren Geltungsbereich.

Erweiterung-Compiler

Der Makro-Compiler hat die Aufgabe, zu überprüfen, ob für einen Tag eine neue Definition vorliegt, und diese gegebenenfalls aufzubereiten. Existiert keine neue Definition, so wird der Tag unverändert zurückgegeben. In der neuen Definition existierende Beschreibungen werden gelöscht. Variablen werden durch ihren Wert ersetzt. Makros werden an den Makro-Compiler weitergereicht und durch dessen Ergebnis ersetzt.

Die nachfolgende Abb. 8 zeigt exemplarisch den Übersetzungsvorgang eines Erweiterungs-Dokumentes.

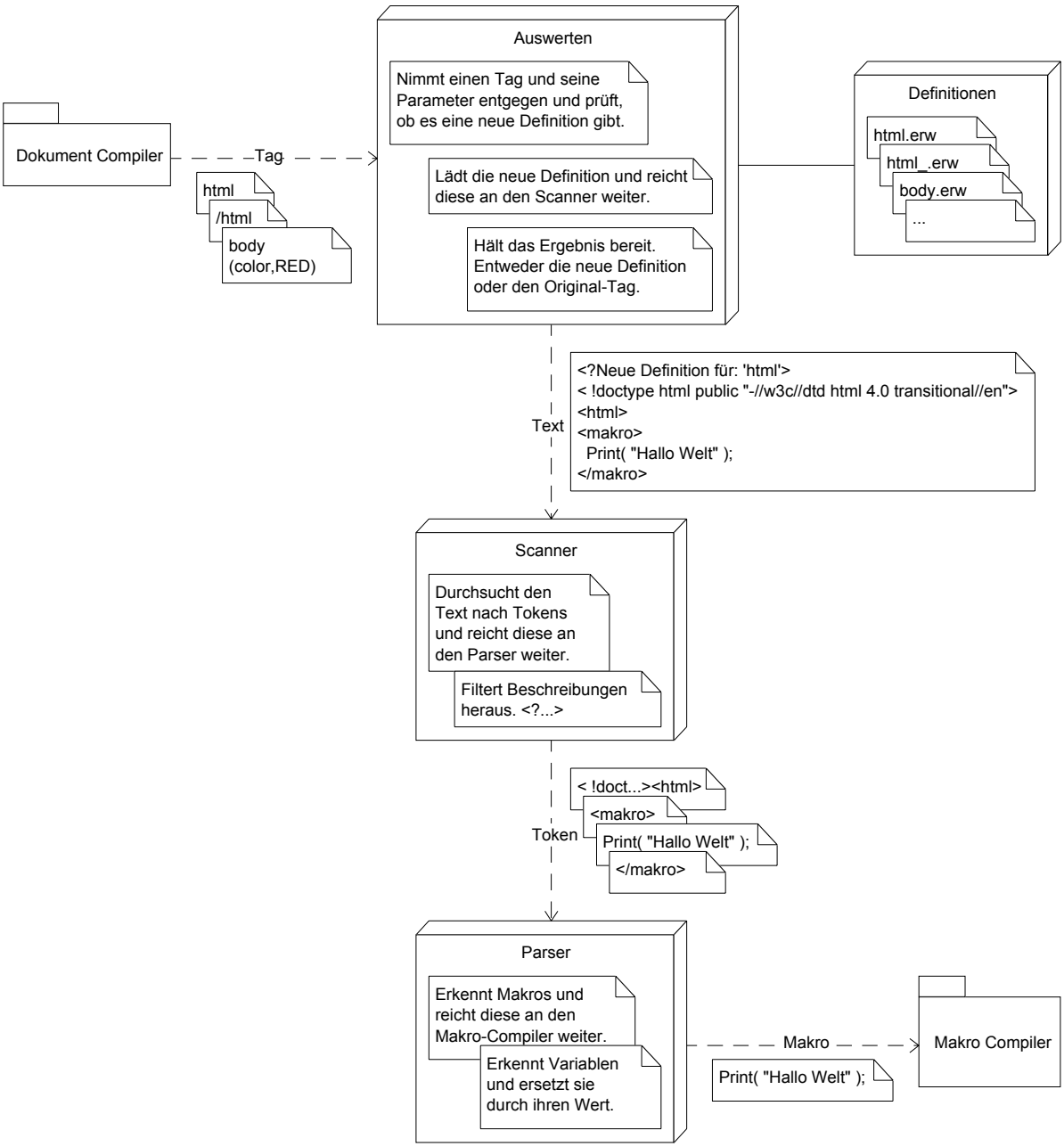


Abb. 8: Arbeitsweise des Erweiterung-Compilers

Nachdem ein Tag an die Auswertung des Compilers übergeben wurde, überprüft sie ob zu diesem Tag eine neue Definition existiert. Die neue Definition wird an den Scanner weitergereicht. Der Scanner zerlegt den Text in Tokens und reicht diese an den Parser weiter, welcher dann die Aufgabe hat, Variablen und Makros zu erkennen. Variablen werden durch ihren Wert ersetzt, Makros werden an den Makro-Compiler weitergeben.

Während der Initialisierung wird auch der Makrovariablen Speicher aufbereitet. Es bleiben also nur Projekt- und Dokument-Variablen erhalten. Zuvor definierte Tag- oder Makro-Variablen verlassen an dieser Stelle ihren Geltungsbereich.

Makro-Compiler

Der Makro-Compiler hat die Aufgabe, Makros auszuführen. Es ist also eigentlich kein Compiler, sondern eher ein Interpreter. Das Makro wird ausgeführt und die Print-Ausgaben werden zurückgegeben.

Die folgende Abb. 9 zeigt die Arbeitsweise des Makro-Compilers anhand eines Beispielmakros.

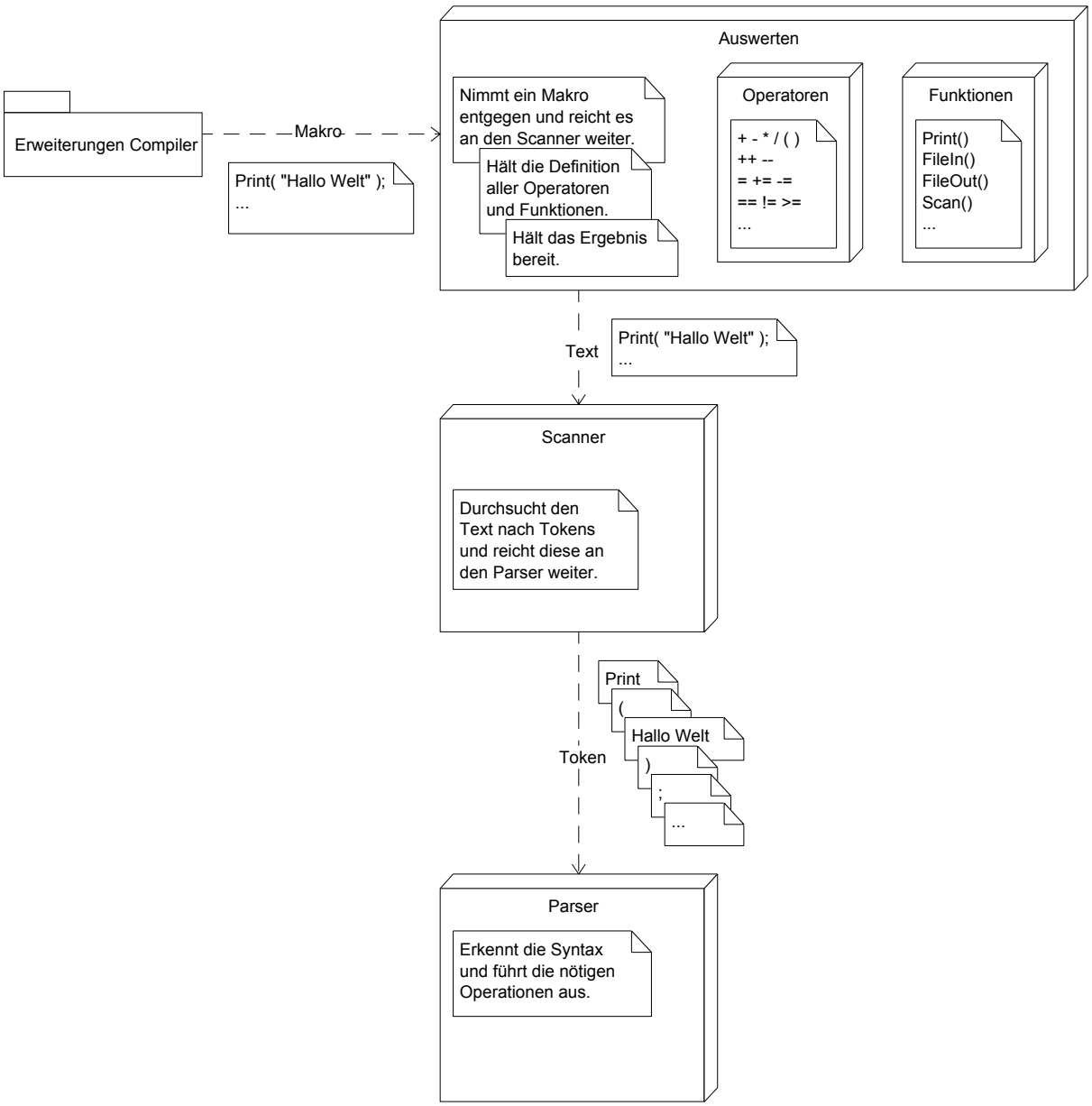


Abb. 9: Arbeitsweise des Makro-Compilers

Nachdem ein Makro an die Auswertung des Compilers übergeben wurde, wird der Text an den Scanner weitergereicht. Der Scanner zerlegt den Text in Tokens und reicht diese an den Parser weiter, welcher dann die Syntax der Makros zu erkennen und die Kommandos auszuführen hat.

Die in den Makros vorkommenden Funktionen und Operationen sind in der Auswerten-Klasse definiert.

Während der Initialisierung wird auch der Makrovariablen-Speicher aufbereitet. Zuvor definierte Makrovariablen verlassen an dieser Stelle ihren Geltungsbereich.

Der Interpreter

Die Realisierung des Makro-Interpreters weicht ein wenig von der Realisierung eines Compilers ab. Der Grund ist, das ein Compiler für jede Anweisung, die im Quelltext vorkommt, ein Ergebnis erzeugt, also vereinfacht gesagt Quellcode gegen Zielcode austauscht. Eine solche Arbeitsweise ist für einen Interpreter nicht geeignet.

Wenn in einem Interpreter eine bedingte Ausführung von Code gefordert wird, muss die Bedingung sofort geprüft und nicht übersetzt werden. Der folgende Code darf nur bei einer wahren Bedingung übersetzt werden. Ein Compiler würde die Bedingung übersetzen und den dazugehörigen Code auch, ohne die Bedingung zu prüfen.

Das gleiche Problem tritt ein, wenn eine Schleife gefordert ist. Ein Code, der in einer Schleife steht, muss mehrmals übersetzt werden, solange die Bedingung der Schleife erfüllt ist. Der Schleifenkonstrukt jedoch, muss lediglich ausgeführt werden.

Ein weiteres Problem ist, dass bedingte Ausführungen oder Schleifen ineinander verschachtelt sein können. Innerhalb einer Schleife können weitere Schleifen auftreten.

Realisiert habe ich dies, indem der Interpreter einen Code nur dann ausführt, wenn er nicht innerhalb einer bedingten Ausführung steht. Bedingter Code wird erst dann ausgeführt, wenn feststeht, dass er ausgeführt werden soll. Die Ausführung des Codes geschieht dann, indem der bedingte Code erneut dem Makro-Compiler übergeben wird. Bei Schleifen wird der bedingte Code mehrmals übersetzt bzw. ausgeführt.

Grafisches Interface

Wie bereits erwähnt wird es zwei →Gui geben. Eines, das in Java realisiert ist, damit der Generator als eigenständiges Programm laufen kann, das andere als Website, um eine einfache Bedienung mit Vorschau anbieten zu können.

Da das Gui Funktionen zur Auswahl der Arbeitspfade zur Verfügung stellt, macht es Sinn, auch das Speichern der vorgenommenen Einstellungen an dieser Stelle zu realisieren.

Damit der Übersetzer nicht das Gui blockiert, während er arbeitet, wird er als Thread gestartet. Um trotzdem Rückmeldung über den Fortgang der Übersetzung zu erhalten, gehen beide Varianten unterschiedliche Wege. Gemeinsam ist beiden Varianten, dass alle Statusmeldungen an eine Funktion übergeben werden, die jedes Gui zur Verfügung stellen muss.

Java-Umgebung

Sobald der Compiler gestartet wurde, meldet er durch Aufruf einer Funktion des Gui, dass er nun arbeitet. Nach Abschluss des Übersetzungsvorganges meldet er wiederum, dass er fertig ist.

Die Statusmeldungen werden von der Statusfunktion direkt in das Statusfenster weitergelenkt.

Html-Umgebung

Bei dieser Variante gestaltet sich das Ganze ein wenig umständlicher. Da das Javaprogramm nicht direkt auf die Webseite und ihre Elemente zugreifen kann (ich konnte keine Möglichkeit in Erfahrung bringen), fragt ein JavaScript-Programm in regelmäßigen Abständen nach, ob neue Informationen vorliegen. Liegen neue Informationen vor so werden sie dargestellt.

Hierfür stellt der Generator drei Funktionen zur Verfügung. Die erste speichert den aktuellen Zustand des Übersetzers, also ob er noch arbeitet oder fertig ist. Die zweite speichert alle eingehenden Statusmeldungen, bis sie abgerufen wurden. Die Dritte speichert den Pfad zu der Seite, die gerade übersetzt wurde, bzw. das Übersetzungsergebnis dieser Seite.

Die Information über die aktuell übersetzte Seite dient der Vorschau der Seiten. Sobald diese Information bereitsteht, wird die Seite angezeigt.

Implementierung

In den vorherigen Abschnitten habe ich bereits die Voraussetzungen und die Struktur des Generators beschrieben. Im letzten Abschnitt bin ich dann auf die Besonderheiten der Umsetzung eingegangen. In diesem Abschnitt soll es um die programmiertechnische Umsetzung gehen.

Entwicklungsumgebung

Den Generator habe ich mit dem → Java™ 2 SDK V1.2.2Win32 erstellt. Der Übersetzer benutzt lediglich den Befehlssatz der → JVM 1.1 um kompatibel zu Browsern zu bleiben. Als Hilfsmittel zur Erstellung des Java-Gui habe ich den Borland J-Builder V3.5 verwendet. Das Webinterface habe ich für den → MSIE 5.0 erstellt.

Eine Einschränkung für das Webinterface auf den MSIE ist sinnvoll, da sich bei diesem die Sicherheitseinschränkungen in Bezug auf Java einfach abstellen lassen. Die Einschränkung auf die Version 5.x ist erforderlich, da die Einstellungen bezüglich Java-Applets von älteren Versionen ignoriert werden oder überhaupt nicht vorgenommen werden können. Es ist unproblematisch, das Webinterface nur für den MSIE zu erstellen, da dieser auf allen Rechnern installiert ist, an denen der Generator benötigt wird.

Grafisches Interface

Html Gui

Die einzigen nennenswerten Besonderheiten dieses Teils sind die Speicherung der Einstellungen, die Kommunikation mit dem Übersetzer und die Darstellung der Vorschau. Der Rest ist einfachste Html Programmierung, deren Arbeitsweise direkt aus dem mitgelieferten Sourcecode ersichtlich ist.

Das Speichern der vorgenommenen Einstellungen wie Pfade und Debuglevel habe ich über Cookies realisiert. Dadurch ist gewährleistet, dass jeder Benutzer seine eigenen Einstellungen vornehmen kann. Durch Angabe eines Verfallsdatums für das Cookie habe ich gleichzeitig sichergestellt, das ein Benutzer, der den Generator lange Zeit nicht benutzt hat, die Standard-Einstellungen vorfindet.

Die Kommunikation mit dem Übersetzer übernimmt ein JavaScript-Programm. Mit dem Start eines Übersetzungsvorganges beginnt das JavaScript-Programm in Abständen von 100ms zu überprüfen, ob neue Statusmeldungen anstehen und ob bereits eine Seite fertig übersetzt wurde.

Die Statusmeldungen werden dann in einem Textfeld ausgegeben.

Fertig übersetzte Seiten werden in einem eigenem Fenster angezeigt. Hierfür gibt es zwei verschiedene Varianten.

In der einfachsten liefert der Übersetzer die Adresse, wohin die fertig übersetzte Seite gespeichert wurde. Diese Adresse wird dann angezeigt:

```
fenster.location.href = result;
```

Als zweites gibt es noch die Möglichkeit, dass lediglich eine Vorschau stattfinden soll. In diesem Fall wird die übersetzte Seite nicht abgespeichert. In diesem Fall liefert der Übersetzer den Quelltext der fertig übersetzten Seite zurück. Diese wird dann direkt in das Fenster geschrieben:

```
fenster.document.open();
fenster.document.write( result );
fenster.document.close();
```

Das Resultat der Implementierung hat dann das in Abb. 10 dargestellte Aussehen:

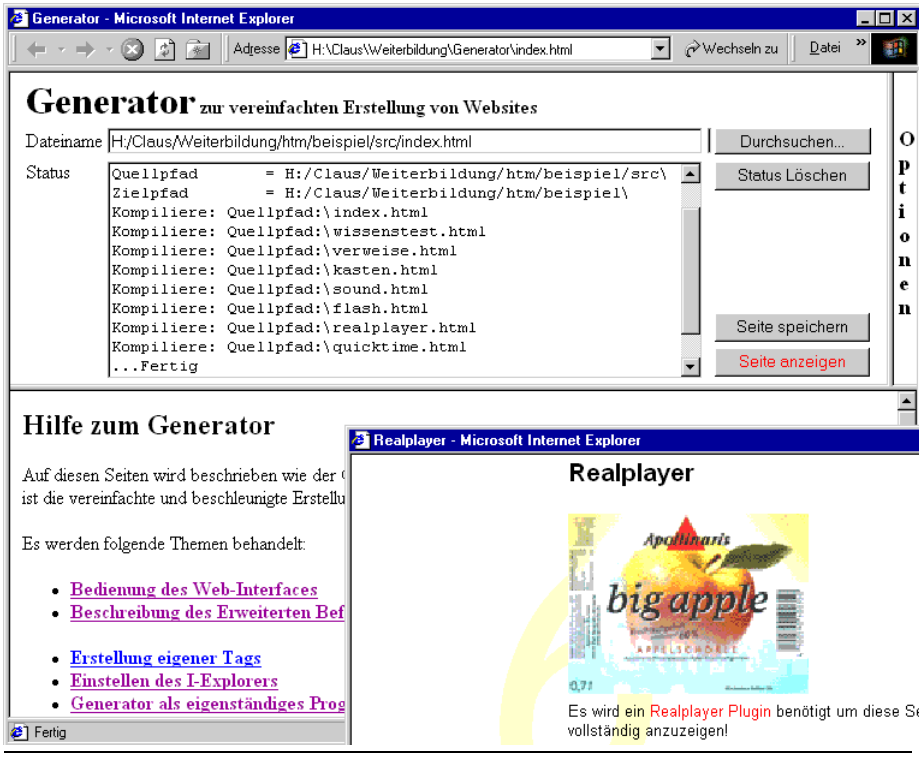


Abb. 10: Html Gui - Schnapshot

Das folgende Klassendiagramm (Abb. 11) zeigt die Schnittstellenklasse und die Module des Generators.

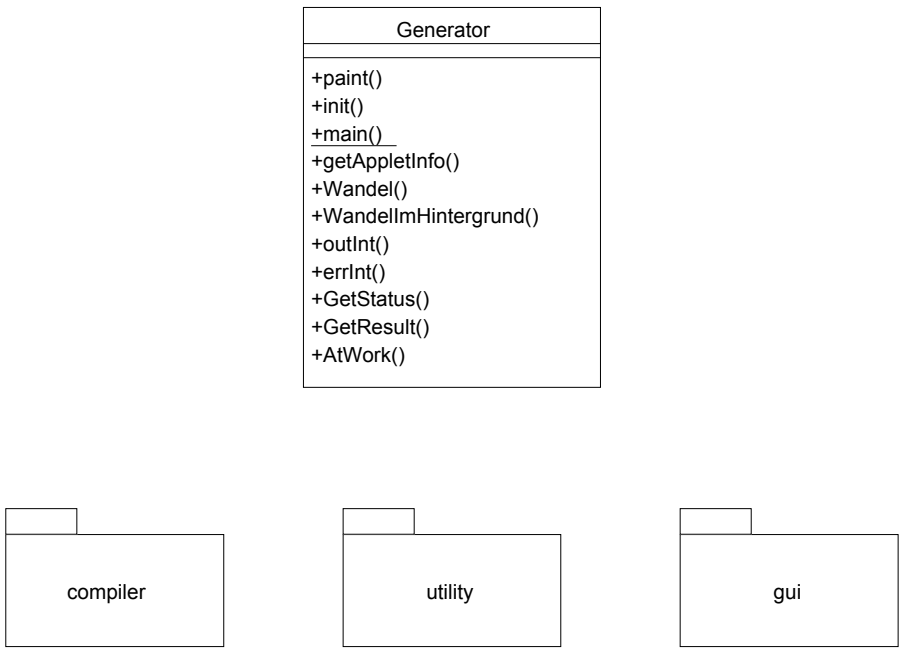


Abb. 11: Generator Klasse und seine Module

Java Gui

Dieses Gui hat nur wenig Besonderheiten.

Erwähnenswert ist, dass Benutzereinstellungen in einer lokalen Datei gespeichert werden, und zwar in Textform in dem Format:

```
Parameter = Wert
```

So lassen sich diese Einstellungen einfach mit einem Texteditor überprüfen und bearbeiten.

Die Kommunikation mit dem Übersetzer ist einfach gehalten. Das Gui ruft zum Start den Übersetzer auf, welcher als eigenständiger Thread läuft. Die Rückmeldungen des Übersetzers geschehen über in einem Interface definierte Funktionen.

Eine Möglichkeit der automatischen Vorschau während des Übersetzungsvorganges oder gar eine Vorschau ohne vorheriges Speichern der übersetzten Seiten ist nicht gegeben.

Das Folgende Klassendiagramm (Abb. 13) zeigt das Gui Paket.

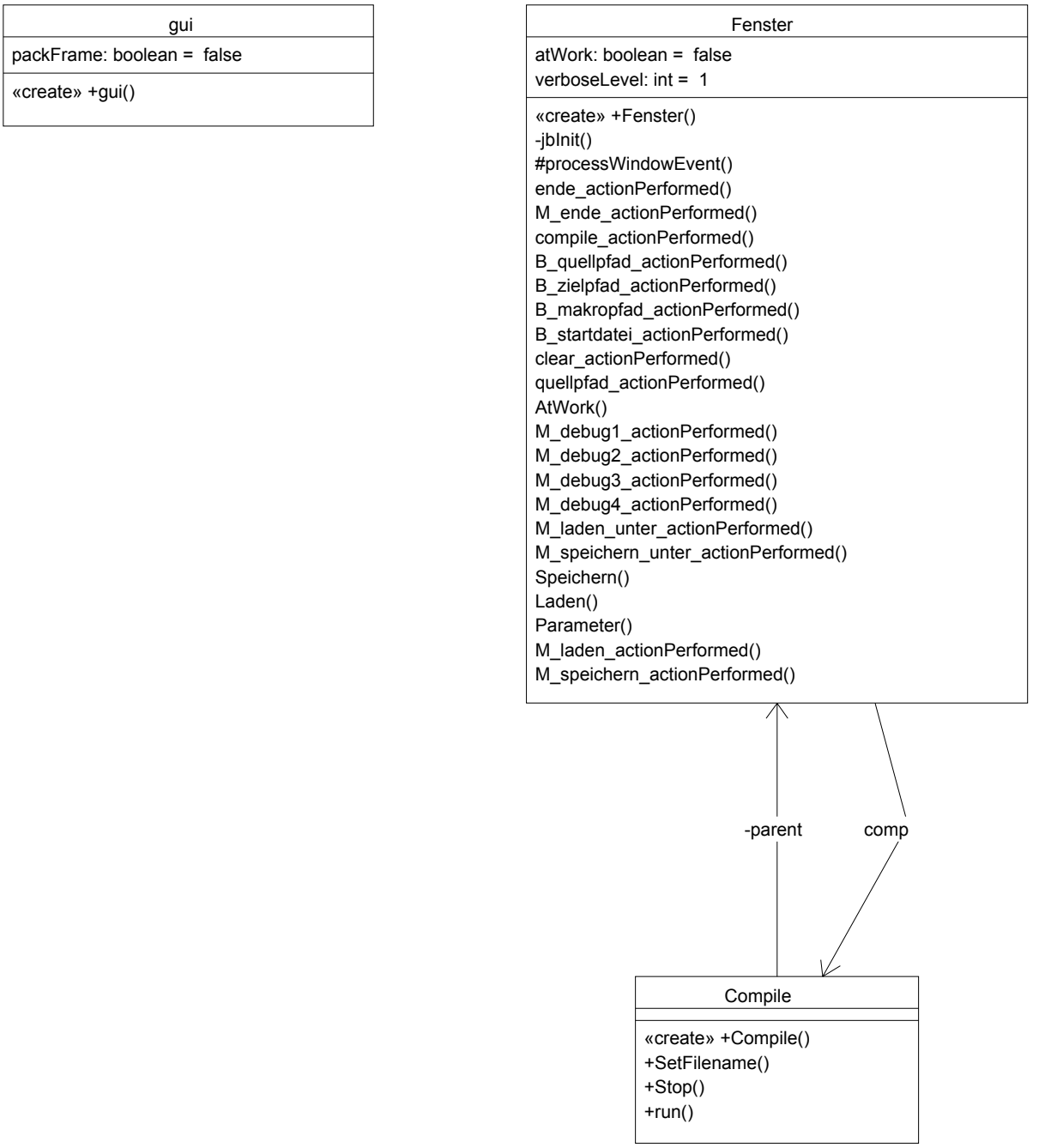


Abb. 12: Gui Klasse

Die Funktionsweise des Gui ist leicht aus dem mitgelieferten Sourcecode ersichtlich.

Das Resultat der Implementierung hat dann das in Abb. 13 dargestellte Aussehen:

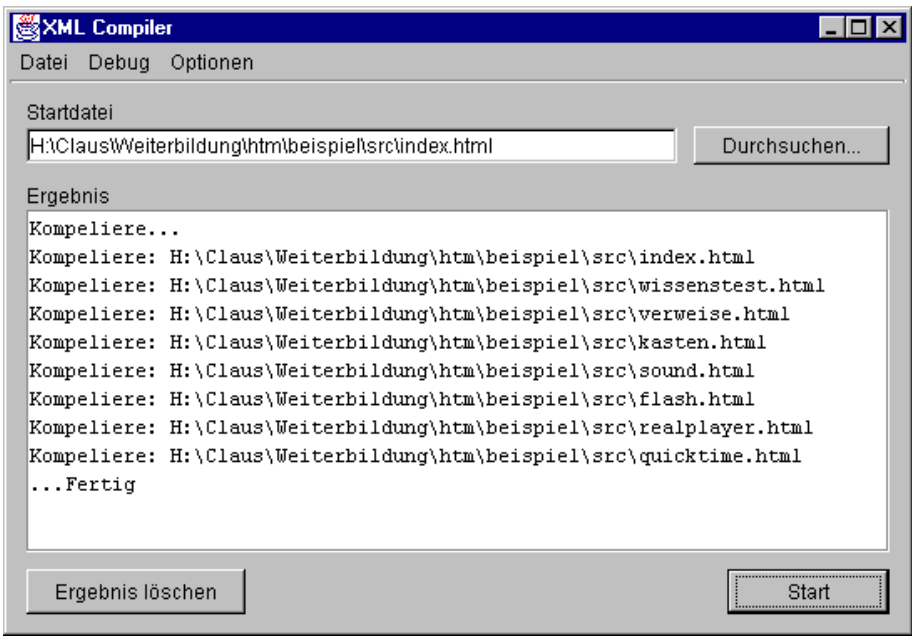


Abb. 13: Java Gui - Schnapshot

Generator

Über die Generator-Klasse wird der Generator gestartet. Die Generator-Klasse übernimmt dabei die Aufgabe, den Generator zu initialisieren und zu entscheiden, ob eventuell mitgelieferte Parameter ausreichen, um direkt den Übersetzer zu starten, oder ob das Java-Gui gestartet werden muss. Im Fall eines Applets übernimmt diese Klasse auch die Kommunikation mit dem Webinterface.

Übersetzer

Obwohl der Übersetzer den umfassensten Teil des Generators ausmacht, beschreibe ich in diesem Abschnitt nur auf einige wenige Teile noch detailierter. Die meisten Teile sind schon anhand der bisherigen Beschreibungen und des mitgelieferten Sourcecode leicht zu verstehen, weshalb ich auf diese nicht weiter eingehe.

Vorab beschreibe ich jedoch noch kurz die Aufgabe einiger Klassen:

- Die Auswerten-Klasse ist die Basisklasse für alle drei Compiler. In ihr sind diegGlobalen Variablen, die jeder Compiler benötigt, abgelegt. Hinzu kommt noch ein Objekt der ErrorHandling-Klasse und die Verwaltung des Übersetzungsergebnisses.
- Die Compile-Klasse ist die Schnittstelle, um den Übersetzer als Thread laufen zu lassen.
- Die Variablen-Klasse übernimmt die Verwaltung der Variablen. In ihr können Variablen gespeichert und der zu einer Variable gehörige Wert ausgelesen werden.
- Die ErrorHandling-Klasse verwaltet Fehlermeldungen. Sie wird ständig über die gerade übersetzten Zeichen informiert und speichert Fehlermeldungen und deren genaue Auftrittposition.

Makro-Parser

Auf die Arbeitsweise des Makro Parsers möchte ich noch einmal gesondert eingehen, weil dieser eine besondere Arbeitsweise aufweist. Er muss einen Code, der nur bedingt ausgeführt wird oder innerhalb von Schleifen steht, besonders behandeln. Dieser Code darf nur übersetzt werden, wenn die Bedingung wahr ist. Andernfalls muss er ignoriert werden. Um dies zu realisieren, zählt der Parser eine Variable hoch, sobald er erkennt, dass ein bedingt ausgeführter Code folgen wird. Jedes Mal, wenn der Parser auf ein IF, ein WHILE oder ein FOR trifft, zählt er diese Variable hoch. Mit dem Verlassen des bedingten Bereiches wird die Variable wieder heruntergezählt.

Solange diese Variable nicht den Wert Null hat wird kein Code mehr ausgeführt. Statt dessen wird der Code selbst zurückgeliefert. Die Bedingung muss nun entscheiden, ob der Code übergangen werden soll (Bedingung ist nicht erfüllt) oder ob der Code erneut einem Makro-Compiler übergeben wird (Bedingung ist erfüllt).

Hierzu ein Beispiel, bei dem der folgende Code dem Makro-Compiler übergeben wird.

```
if( true )
  Print( "Bedingung ist wahr" );
```

Hierauf reagiert der Parser wie folgt:

[if] --> *bedingung erkannt* --> **bedingt++;**

[(true)] --> *Ausdruck 1 merken*

[Print(„Bedingung ist wahr“)] --> *Ausdruck 2 nicht interpretieren sondern als Ergebnis zurückgeben, da* **bedingt>0**

[;] --> *Bedingung zu Ende* --> **bedingt--;** *und ersten Ausdruck auswerten. Da dieser wahr ist, wird der zweite Ausdruck an den Makro Compiler übergeben und der gesamte Ausdruck durch dieses Ergebnis ersetzt.*

Die gleiche Arbeitsweise funktioniert auch bei Schleifen, nur dass in einem solchen Fall der zweite Ausdruck so oft ausgeführt wird, bis der erste Ausdruck nicht mehr wahr ist.

Dass die Variable „**bedingt“** als Zähler realisiert ist, ist daher wichtig, da sich innerhalb der Bedingung noch weitere Bedingungen befinden können, die beim Verlassen der Bedingung ebenfalls die Variable zurücksetzen.

Abgesehen von dieser besonderen Behandlung arbeitet der Interpreter wie ein ganz normaler Compiler. Alles weitere ist daher auch leicht anhand des mitgelieferten Sourcecodes zu verstehen.

Variablen

Wie ich bereits beschrieben habe, gibt es Variablen mit unterschiedlichen Geltungsbereichen. Diese Variablen unterscheiden sich anhand der Anzahl der vorangestellten Dollarzeichen ’$’. Diese Art der Definition birgt einen großen Vorteil. Ist beispielsweise die Ausführung eines Makros beendet, werden alle Variablen gelöscht, die kein führendes Dollar haben. Alle übrigen Variablen werden von der aufrufenden Funktion übernommen. Wird nun die Definition einer Erweiterung verlassen, werden alle Variablen, die nicht mit mindestens zwei Dollarzeichen beginnen, verworfen und nur der Rest wird übernommen. Das gleiche gilt sinngemäß, wenn die Übersetzung eines gesamten Dokumentes abgeschlossen ist.

Syntax der Compiler

Dokumente

Die Syntax der Dokumente, also der Dateien, die Erweiterungen enthalten, weicht nicht von der Syntax normaler Html-Dokumente ab.

Erweiterungen werden in Form von Tags in die Dokumente integriert. Jede Erweiterung ist also ein eigenes Tag, welches genauso wie jedes andere Tag Parameter haben kann.

Soll ein Tag nicht durch seine neue Definition (Erweiterung) ersetzt werden, so genügt es, dem Tag vor der schließenden spitzen Klammer ein Ausrufezeichen hinzuzufügen.

Welche zusätzlichen Tags (Erweiterungen) verwendet werden können, hängt davon ab, welche definiert wurden und zum Zeitpunkt der Übersetzung zur Verfügung stehen, sich also im Erweiterungspfad befinden.

Erweiterung

Erweiterungen bestehen aus zwei Arten von Code.

Zum einen bestehen sie aus Html-Code die sich nur dadurch vom normalen Html unterscheiden, dass sie auch Variablen enthalten können, die während der Übersetzung durch ihren derzeitigen Wert ersetzt werden. Variablen sind durch ein führendes Dollarzeichen und einem darauf folgenden alphanumerischen Bezeichner gekennzeichnet. Sie können an jeder beliebigen Position im Html Code vorkommen. Das Dollarzeichen selbst wird durch einen vorangestellten Backslash '\' dargestellt.

Die zweite Art von Codes sind Makros. Sie werden durch die Schlüsselworte `<makro>` und `</makro>` eingeschlossen. Dieser Teil des Codes wird durch die Ausgaben des Makros ersetzt.

Makro

Die Syntax ist dieselbe wie in JavaScript, jedoch ohne Berücksichtigung der groß- und Kleinschreibung.

Variablen

Variablen haben verschiedene Geltungsbereiche, die wie folgt gekennzeichnet sind:

- *name* - Variable innerhalb des Makros
- *$name* - Variable innerhalb der Erweiterung
- *$$name* - Variable innerhalb des Dokumentes
- *$$$name* - Variable innerhalb des Projektes

Wobei „*name*“ der Name der Variable ist.

Mit „Projekt“ ist der gesamte Übersetzungsvorgang gemeint. Ein Projekt kann auch mehrere Dokumente beinhalten, wenn ein Dokument die Übersetzung eines weiteren Dokumentes veranlasst.

Innerhalb von Makros definierte Variablen verlassen ihren Geltungsbereich erst mit dem Beenden des Makros.

Funktionen

Es sind bisher folgende Funktionen realisiert:

- *Print*(wert) - fügt „wert“ in das Html Dokument ein
- *System*(wert) - gibt „wert“ als Statusanzeige aus
- *Message*(wert) - gibt „wert“ als Message (mit Zeilenangabe) aus
- *Set*(wert1, wert2) - weist der Variable die durch „wert1“ beschrieben wird, den „wert2“ zu
- *GetVariablenliste*(filter) - listet alle Variablen auf, die mit „filter“ beginnen (var1=„wert1“ var2=„wert2“ ...)
- *Scan*(filename) - übersetzt die durch „filename“ bezeichnete Datei
- *ScanText*(text) - übersetzt den in „text“ übergebenen Inhalt als Dokument
- *Exec*(text) - übersetzt den in „text“ übergebenen Inhalt als Makro
- *FileOut*(filename, text) - Speichert „text“ in der durch „filename“ beschriebenen Datei
- *FileAdd*(filename, text) - Hängt „text“ an der durch „filename“ beschriebenen Datei an
- *FileIn*(filename) - Lädt den Inhalt der durch „filename“ beschriebenen Datei
- *SubString*(begin, text, end) - Gibt einen SubString von „text“ zurück, der nach dem erstem auftreten von „begin“ anfängt und vor dem letztem Auftreten von „end“ endet
- *SubString*(n1, text, n2) - Gibt einen SubString von „text“ zurück, der an der n1’sten Position anfängt und an der n2’sten Position von hinten endet
- *SubStr*(n1, text, n2) - Gibt einen SubString von „text“ zurück, der an der n1’sten Position anfängt und n2 Stellen lang ist
- *ReplaceString*(text, alt, neu) - Tauscht jedes auftreten des Strings „alt“ in „text“ gegen den String „neu“ aus
- *ReplaceChar*(text, alt, neu) - Tauscht jedes auftreten des Zeichens „alt“ in „text“ gegen das Zeichen „neu“ aus

Konstrukte

Es gibt folgende Konstrukte:

- *IF*(bedingung) anweisung
- *IF*(bedingung) anweisung1 *ELSE* anweisung2
- *WHILE*(bedingung) anweisung
- *FOR*(aktion ; bedingung ; aktion) anweisung

bedingung - Ein Vergleich bzw. „*true*“ oder „*false*“

Vorbelegte Variablen

Volgende Variablen sind vorbelegt:

- *true* - Der Wert „true“
- *false* - Der Wert „false“
- *$filename* - Name der Makrodatei
- *$$filename* - Name des Dokumentes
- *$$$quellpfad* - Der beim Aufruf übergebene Quellpfad
- *$$$zielpfad* - Der beim Aufruf übergebene Zielpfad

Vorbelegte Dateien

Folgende Dateien sind vorbelegt:

- *erweiterung/_init.erw* - Diese Datei wird zu Beginn des Übersetzungsvorganges eines Projektes ausgeführt
- *erweiterung/_compile.erw* - Diese Datei wird zum Übersetzen jeder Html-Datei aufgerufen. Ihr wird in dem Parameter filename die zu übersetzende Datei übergeben

Test

Der Generator wurde in drei Phasen getestet.

In der ersten Phase habe ich den Generator mit fehlerhaften Dateien versorgt. Dadurch lässt sich feststellen wie, fehlertolerant der Generator ist, und ob die entstehenden Fehlermeldungen ausreichend präzise sind.

In der zweiten Phase habe ich den Generator mit selbst erstellten Dokumenten gefüttert. Diese Dokumente enthalten zusammen alle bisher erstellten Erweiterungen, wodurch sichergestellt ist, dass diese die gewünschten Ergebnisse bringen. Es handelt sich bei den Dokumenten um die Beispielseiten.

In der dritten Phase musste der Generator Dokumente übersetzen, die im Rahmen des Weiterbildungs projektes erstellt wurden. Diese Dokumente wurden nach einer grundlegenden Einweisung mit Hilfe der beiliegenden Dokumentation erstellt. Durch diese Dokumente ist sichergestellt, dass der Generator auch alltags tauglich ist und nicht nur mit optimal erstellten Dokumenten arbeitet. Hinzu kommt, dass diese Dokumente typische Anwenderfehler enthalten, wodurch auch die Fehlertoleranz und die Qualität der Fehlermeldungen getestet werden kann.

Um den Generator nicht nur anhand von gelieferten Ergebnissen prüfen zu können, verfügt er über die Möglichkeit, in einen Debug-Modus versetzt zu werden. In diesem Zustand liefert er abhängig vom Debuglevel unterschiedlich ausführliche Statusinformationen, anhand derer die korrekte Arbeitsweise leicht überprüft werden kann.

Performancetest

Zusätzlich zu den Funktionstests habe ich den Generator noch einem Performance-Test unterzogen. Hierbei habe ich die beiden Versionen (Applet und Applikation) verglichen. Zu diesem Test gehört eine Untersuchung des benötigten Speichers sowie eine Überprüfung der Arbeitsgeschwindigkeit. Zum testen habe ich den Generator

die Lerneinheit „Brandels Barkunde“ übersetzen lassen. Als Testsystem diente mir ein PentiumIII 600 PC mit 128MB Ram unter Win NT4.

Testergebnis

Die Tests ergaben, dass der Generator entsprechend der Aufgabenstellung arbeitet.

Alle während der Tests noch auftretenden Fehler hatten nur geringe Ausmaße und konnten behoben werden. Zusätzlich zur Fehlerbehebung wurde der Generator in Bezug auf die Benutzerfreundlichkeit noch in einigen Feinheiten optimiert.

Dazu gehört z.B. die Ausgabe einer Warnung, wenn der Pfad der Quelldatei nicht der übliche ist.

Es hat sich gezeigt, dass die Fehlermeldungen in einigen Situationen noch präziser sein könnten. In welchen Situationen dies jedoch im einzelnen vorkommt, wird sich erst im Laufe der Benutzung herausstellen und kann daher auch jetzt noch nicht optimiert werden.

Der Performance-Test hat folgende Maximalwerte ergeben:

	Applet	**Applikation**
Speicherbedarf Programm	13 MB	11 MB
Speicherbedarf Übersetzung	68 MB	46 MB
Zeit Übersetzung	9:20 Min.	2:18 Min.

Die folgenden Diagramme zeigen noch einmal den Speicherbedarf während der Übersetzung. Jeder Teilstrich in x-Richtung sind 6 Sekunden.

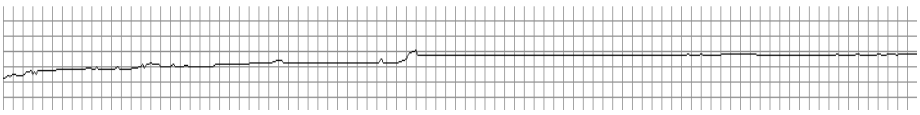


Abb. 14: Speicherbedarf Applet

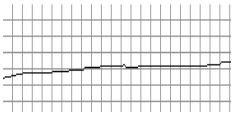
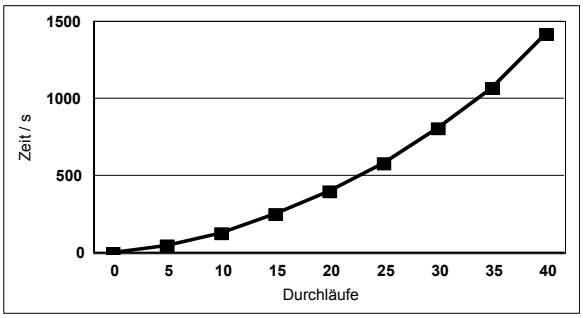


Abb. 15: Speicherbedarf Applikation

Es ist leicht zu erkennen, dass die Applikation dem Applet in Bezug auf die Performance deutlich überlegen ist. Die Applikation benötigt nur 67% des Speichers und sogar nur 25% der Zeit.

Es war auffällig, dass der Generator während der Übersetzung größerer Projekte im Verlauf immer langsamer wird, also für die Übersetzung einzelner Seiten immer länger braucht. Um dieses Verhalten näher zu untersuchen, habe ich eine Testseite rekursiv übersetzt. Die folgenden Diagramme (Abb. 16, Abb. 17) zeigen das Verhalten des Generators bei dieser Übersetzung. Anhand dieses Versuches lässt sich ebenfalls das Verhalten bei speicheraufwendigen Übersetzungen gut untersuchen.



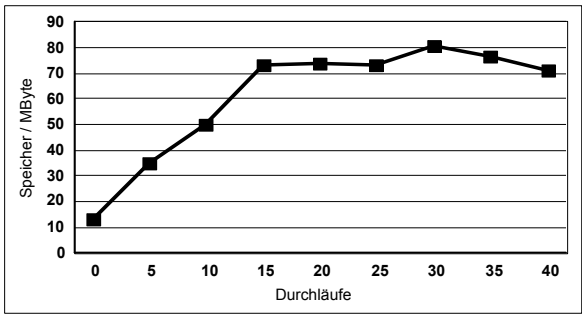
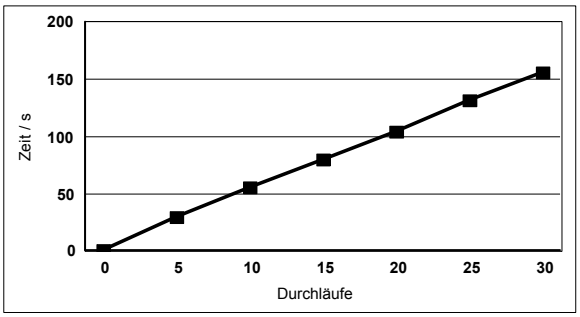


Abb. 16: Rekursiver Aufruf beim Applet



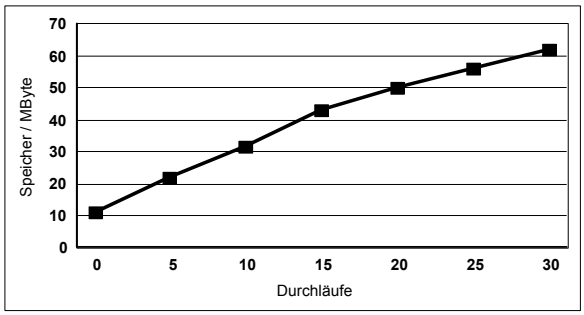


Abb. 17: Rekursiver Aufruf bei der Applikation

Auffällig ist an diesen Diagrammen, dass lediglich das Applet, mit zunehmender Rekursionstiefe, langsamer wird. Die Applikation zeigt ein durchweg lineares Verhalten. Der Speicherbedarf des Applets ist auch erheblich höher als der Bedarf der Applikation. Es werden jedoch nicht mehr als 80 MByte belegt, jedenfalls scheinbar. Zwar steigt nun nicht mehr der verwendete Speicher des Applets, jedoch der belegte Speicher von Windows steigt weiter. Bei der Applikation ist schon früher Schluss. Diese stoppt einfach die Ausführung, sobald der verwendete Speicher 64 MByte erreicht hat. Dieser Effekt ist jedoch im regulären Betrieb nicht relevant, da selbst große Projekte keine solch starke Verschachtelung aufweisen, wie in diesem Versuch und damit auch nicht einen solch großen Speicherbedarf an den Tag legen. Der Grund für diesen Effekt ist vermutlich in dem verwendetem JRE zu suchen.

KAPITEL 4

Erstellung

In diesem Abschnitt geht es um die Erstellung der Inhalte die im Umfeld der → Arbeitsumgebung gelernt werden sollen. Die Inhalte werden mit Unterstützung des → Generators von Mitarbeitern der Firma Kiwi erstellt. Meine Aufgabe besteht in der Einweisung und Betreuung der Mitarbeiter.

Rahmenbedingungen

Zielgruppe

Die Inhalte sollen von Anwendern erstellt werden, die nur grundlegende Kenntnisse im Umgang mit Computern und MS-Windows haben. Die Anwender haben auch grundlegende Kenntnisse in der Erstellung von Html Dokumenten mit Hilfe eines Html-Editors.

Plattform

Um eine einfache und unkomplizierte Erstellung gewährleisten zu können, werden die Inhalte unter Zuhilfenahme von HomeSite erstellt. Die Übersetzung der Inhalte wird unter beiden gui's durchgeführt. Als Betriebssystem soll Windows 9x oder Windows NT zum Einsatz kommen.

Aufgaben

Meine Aufgabe besteht darin, die Anwender in die Erstellung von Inhalten einzuweisen. Dies beinhaltet eine Einweisung in die Besonderheiten, die bei der Erstellung der Inhalte beachtet werden müssen und in die zusätzlichen Tags, welche die Erstellung erleichtern. Hierzu gehört auch eine Einweisung in die Handhabung des Generators und eine grundlegende Erklärung der Funktionsweise.

Eine weitere Aufgabe besteht in der Betreuung während der Erstellung. Die erstellten Inhalte müssen auf korrekte Erstellung überprüft werden, die Anwender müssen korrigiert werden und ich muss bei Fragen zur Verfügung stehen und Hilfestellungen geben.

Folgerungen

Um eine korrekte und schnelle Erstellung von Inhalten zu gewährleisten, ist eine gründliche Einführung in die Erstellung der Inhalte nötig. Es ist wichtig, dass die Anwender einen Überblick über die zur Verfügung stehenden Erweiterungen haben, damit diese auch eingesetzt werden. Die Anwender müssen wissen, wo sie schnell nachschlagen können, welche Erweiterungen zur Verfügung stehen. Witerhin müssen dass die Anwender die Aufgabe des Generators kennen, damit dieser korrekt eingesetzt werden kann. Es ist auch wichtig, dass ich als Ansprechpartner vor ort bin, damit aufkommende Fragen gleich beantwortet werden können. Zusätzlich bekomme ich nur dadurch, dass ich als Ansprechpartner ständig zur Verfügung stehe, Rückmeldung über alle auftretenden Schwierigkeiten, was mir hilft, deren Ursache zu beheben.

Vorgehensweise

Um Erfahrungen mit dem Einsatz des Generators sammeln und auf Änderungsvorschläge schnell und flexibel reagieren zu können, ist es sinnvoll, zu Anfang nur einen Anwender in die Erstellung von Lerninhalten einzuführen. So bleibt mir genug Zeit, neben erforderlichen Änderungen und Verbesserungen die Erstellung der Inhalte zu verfolgen.

Arbeitsumgebung

Als Einstig sollte die Arbeitsumgebung mit den Beispielinhalten vorgestellt werden. So bekommt der Anwender einen Einblick, in welches Umfeld die zu erstellenden Inhalte passen sollen und wie diese dargestellt werden.

Erweiterungen

Als nächstes sollte dem Anwender gezeigt werden, welche zusätzlichen Elemente zur Verfügung stehen und wie diese einzusetzen sind. Dies lässt sich ebenfalls gut anhand der Beispielseiten durchführen.

Besonderheiten

Erst nach dieser grundlegenden Einführung ist es sinnvoll die Besonderheiten zu erklären, die bei der Erstellung von Inhalten zu beachten sind. Dies sind im Wesentlichen die erweiterte Funktionalität des Paragrafen und die zusätzlichen Meta-Tags, mit denen die Verlinkung der einzelnen Seiten geschieht.

Generator

Als nächstes sollte der Generator und deren Arbeitsweise vorgestellt werden. Hierzu könnte eine einfache ‘Hallo Welt’ Seite erstellt werden, welche nur die wichtigsten Elemente enthält. Zur Demonstration sollte die übersetzte Seite und deren Quelltext gezeigt werden, damit die Anwender sehen, auf welchen Umfang eine solch einfache Seite nach der Übersetzung anwächst, und das, obwohl keine überflüssigen Elemente enthalten sind.

Umsetzung

In diesem Abschnitt beschreibe ich die Erfahrungen, die ich während der Erstellung von Inhalten sammeln konnte.

Erfahrungen

Es zeigt sich, dass es den Anwendern nicht schwer fällt, die erweiterten Möglichkeiten einzusetzen, sofern ihnen Beispiele als Vorlage zur Verfügung stehen. Gewöhnungsbedürftig ist die Einschränkung, dass sämtliche Elemente in Paragrafen eingefasst sein sollten.

Auf positive Resonanz stößt durchweg die übersichtliche und vereinfachte Erstellung, bei der wenig Zeit auf das Layout verwendet werden muss.

Nur schwer verständlich ist die Vorgehensweise, die zur Verkettung von Kapiteln notwendig ist. Dieser Punkt muss noch entscheidend geändert werden oder ich muss eine wesentlich verbesserte Beschreibung zur Verfügung stellen. Hier entstanden mit Abstand die meisten Fehler.

Beim Einsatz des Generators zeigte sich, dass das Webinterface wesentlich mehr Zuspruch findet als die Applikation. Der Hauptgrund ist die Möglichkeit der Vorschau von Seiten. Entscheidend ist jedoch auch, dass beim Webinterface alle Beschreibungen gleich mit eingebaut sind.

Ergebnis

Es empfiehlt sich, den Generator auf dem PC des Anwenders vollständig zu installieren und auf das zu bearbeitende Projekt einzustellen und dies nicht dem Anwender zu überlassen, da die Installation nicht unbedingt unproblematisch abläuft. Ein Anwender, der mit dieser Aufgabe alleine gelassen wird, würde den Generator vieleicht ablehnen. Es sollte jedoch für einen Systemadministrator einfach sein, diesen einzurichten. Ebenfalls sollte ein Link zum Generator auf den Desktop oder in das Startmenü gelegt werden.

Während der Übersetzung zeigten sich mehrere kleine Fehler in der Implementierung des Generators und der Arbeitsumgebung, welche jedoch leicht behoben werden konnten.

Als entscheidendste Unzulänglichkeit des Generators stellte sich dessen Eigenschaft heraus, Sonderzeichen nicht verarbeiten zu können. Doch auch dies ließ sich leicht abstellen.

KAPITEL 5

Fazit

In diesem Kapitel fasse ich die Ergebnisse und Erfahrungen dieser Diplomarbeit zusammen.

Zusammenfassung

Mit der Erstellung dieser Diplomarbeit verfolgte ich das Ziel, eine Möglichkeit zu finden, Internetseiten schnell und einfach zu erstellen. Zusätzlich sollten diese erstellten Seiten designerisch hochwertig und ansprechend sein. Im speziellen ging es um die Gestaltung von Lerneinheiten. Diese zeichnen sich dadurch aus, dass große Mengen von Internetseiten in einem einheitlichen Erscheinungsbild erstellt werden müssen.

Als erstes begann ich einen Rahmen zu schaffen, in dem die Lerneinheiten präsentiert werden. Hierbei habe ich gleichzeitig das Erscheinungsbild der Inhalte festgelegt. Orientiert habe ich mich dabei an der Gestaltung bereits existierender Internetseiten zum gleichen Thema.

Erheblich schwieriger und komplexer war die Umsetzung der Funktionalität. Hier musste ich einen Weg finden, der es ermöglicht, ohne großen Aufwand die Internetpräsenz dieser Seiten leicht und unkompliziert um Inhalte zu erweitern. Den Schwerpunkt habe ich hierbei darauf gelegt, dass die Lerneinheiten nach der Einbindung in den Internetauftritt sicher funktionieren und keine aufwendigen Tests vonnöten sind. Als besonders arbeitsintensiv hat sich die Forderung nach einer Kompatibilität zu älteren Browsern der Dreier-Generation herausgestellt. Um die Möglichkeiten der neueren Browser zu nutzen und trotzdem nicht alle Seiten doppelt erstellen zu müssen, waren umfangreiche und komplexe Tests und Fehlerbehandlungen nötig.

Letztendlich hat sich das gewählte Konzept bewährt. Die Websites genügen den geforderten Ansprüchen, sind kompatibel zu alten Browsern und nutzen die aktuellen Möglichkeiten. Was nun noch blieb, war eine Möglichkeit zu finden, die Inhalte auf eine einfache Art erstellen zu können.

Bei diesem Punkt habe ich mich entschieden einen neuen Weg zu gehen. Wenn bisher mehrere Websites mit gleichem Layout erstellt werden sollen, werden Vorlagen erstellt, die alle einheitliche Elemente enthalten. Kommen innerhalb dieser Seiten wiederkehrend ähnliche Elemente vor, so werden diese als Module angelegt und an entsprechender Stelle eingesetzt. Auf diese Art ist es zwar einfach, Seiten zu erstel-

len, sollen jedoch Änderungen vorgenommen werden, so muss sich der Anwender durch große Mengen schwer lesbaren Html-Code durcharbeiten. Die Wahrscheinlichkeit, dass hierbei Fehler entstehen, ist recht groß.

Mit dem von mir beschrittenen Weg versuche ich einen Ausweg aus diesem Problem zu finden. Meine Idee war, neue Tags einzuführen, welche spezielle an die Begebenheit angepasste Aufgaben erfüllen. Zusätzlich sollten Parameter, die immer wiederkehrend einheitlich sind, nicht mehr angegeben werden müssen. Wenn eine Seite fertig erstellt wurde, wird aus dieser die entsprechende Html-Seite generiert, die wieder von jedem Browser angezeigt werden kann. Sollen nun Änderungen vorgenommen werden, so kann dies weiterhin in der leicht lesbaren Seite geschehen.

Die Aufgabe bestand nun darin, einen Generator zu entwickeln, der die benötigte Übersetzung durchführt. Die Schwierigkeit bestand nicht in der Programmierung, sondern in dem Entwurf. Es galt festzustellen, welche Anforderungen im einzelnen zu erfüllen sind, und ein Konzept zu entwerfen, welches diesen Anforderungen genügt. Die Schwierigkeit war nun, keine wichtigen Anforderungen zu übersehen und alle gut strukturiert zu verbinden.

So blieb es nicht aus, dass ich während der Umsetzung immer wieder auf Schwachstellen im Konzept stieß, die ich überarbeiten musste. Insgesamt gesehen ist es mir gelungen, einen Generator zu entwerfen, der nicht nur den geforderten Ansprüchen genügt, sondern darüber hinaus noch so flexibel geworden ist, dass er sich leicht an neue Aufgaben anpassen lässt.

Die Erstellung der Lerneinheiten mit Hilfe des Generators war unproblematisch. Die Anwender, die mit Hilfe des Generators Inhalte zu den Lerneinheiten erstellten, begrüßten durchweg die vereinfachte Erstellung. Es hat sich jedoch gezeigt, dass der Generator nicht geeignet ist, lediglich einzelne Tags zu vereinfachen. Für solche Aufgaben ist die Geschwindigkeit des Generators ungenügend, was den Zeitgewinn bei der Erstellung zunichte macht.

Beurteilung

Die Idee, die Arbeitsumgebung von den Inhalten zu trennen, hat sich als gut erwiesen. Dadurch ist es einfach, die Weiterbildungsseiten mit neuen Inhalten zu versehen, ohne dass jedes Mal komplexe und zeitaufwendige Tests anfallen. Ist die Funktionalität der Arbeitsumgebung einmal sichergestellt, so wird sie auch bei neuen Inhalten zuverlässig sein.

Das Konzept, mit dem die einzelnen Seiten einer Lerneinheit verknüpft werden, hat sich als für den Anwender unfreundlich und schwer verständlich erwiesen. Dieses Konzept ist lediglich für Programmierer leicht durchschaubar, für die der Umgang mit Datenbäumen nichts Besonderes ist. Sollten Inhalte in größerem Maßstab als bisher geplant erstellt werden, wäre es sinnvoll, ein anderes Konzept zu entwickeln und zu verwenden.

Der Einsatz des Generators hat sich als enorme Erleichterung bei der Erstellung von Inhalten herausgestellt. Hierdurch ist es den Anwendern möglich, sich auf das Zusammentragen und Erstellen von Inhalten zu konzentrieren. Die Zeit, in der zusätzliche Inhalte erstellt werden, ist deutlich niedriger als bei der herkömmlichen Erstellung von Webseiten.

Das Erstellen von Erweiterungen erweist sich als einfach und schnell. Mit Hilfe der Makrosprache lassen sich selbst Erweiterungen mit komplexen Aufgaben realisieren.

Der Performancetest hat ergeben, dass der Generator als Applet erheblich langsamer arbeitet als die Applikation. Während der Übersetzung einer einzelnen Seite ist dies jedoch nicht von Bedeutung. Hier überwiegen die Vorteile, die das Webinterface mit sich bringt. Sollen jedoch ganze Projekte übersetzt werden, macht es Sinn, die Applikation zu benutzen. Grundsätzlich lässt sich jedoch sagen, dass die Arbeitsgeschwindigkeit des Generators für dessen Aufgabe völlig ausreichend ist.

Der bisherige Einsatz des Generators in Verbindung mit der Arbeitsumgebung hat gezeigt, dass bei der Erstellung von Internetseiten mit einheitlichem Layout viel Zeit gespart werden kann.

Ausblick

Die Arbeitsumgebung wurde für eine spezielle Problemstellung entworfen und zur Lösung dieser optimiert. Durch die hohe Spezialisierung ist es nicht möglich, diese Lösung an andere Bedürfnisse anzupassen. Da die Arbeitsumgebung modular aufgebaut ist, können jedoch einzelne Teile der Arbeitsumgebung auch für spätere Projekte verwendet werden.

Der Generator wurde so entworfen, dass er möglichst einfach an erweiterte Aufgabenstellungen angepasst werden kann. Wegen dieser guten Anpassungsmöglichkeiten wird der Generator schon jetzt auch in anderen Projekten eingesetzt. Er findet überall dort Einsatz, wo wiederkehrend einheitliche Elemente in Websites gefordert sind. Zusätzlich wird der Generator als Toolsammlung für Lösungen im Bereich der Websiteerstellung eingesetzt.

Es hat sich herausgestellt, dass die Arbeitsweise des Generators nicht für jeden Anwender vollständig zu verstehen ist. Dadurch kommt es zu häufigen Rückfragen und einer fehlerhaften Bedienung durch einige Anwender. Schwierigkeiten haben vor allem Anwender, die keinerlei Programmiererfahrung mit Sprachen, die einen Compiler benötigen, haben.

Der Generator zeigt durch den vermehrten Einsatz jedoch auch seine Schwächen. Es hat sich mehrfach als Nachteil herausgestellt, dass der Generator nur einzelne Tags erkennt und nicht eine Verschachtelung von Tags. Sollte der Einsatz des Generators in gleichem Maße zunehmen wie bisher, so wäre es sinnvoll, diese Funktionalität noch nachzurüsten. Dies würde die Leistungsfähigkeit des Generators nicht unerheblich steigern.

Schlusswort

Sowohl die Arbeitsumgebung als auch der Generator haben ihren Zweck erfüllt. Die Arbeitsumgebung mit ihren Inhalten wird in absehbarar Zeit online gehen, und der Generator wird inzwischen auch in anderen Projekten eingesetzt.

Für mich war es eine interessante Erfahrung, den Generator mit seinen zwei Compilern und dem Interpreter zu erstellen. Diese Erfahrungen werden mir vermutlich noch häufig eine große Hilfe sein, da es in der Informatik nicht selten vorkommt, dass ein spezieller Compiler benötigt wird.

Mindestens ebenso interessant und hilfreich ist das Wissen über die Erstellung von Internetseiten, das ich mir während der Erstellung der Diplomarbeit aneignen konnte. In diesem Gebiet gibt es viele interessante Bereiche mit denen ich mich wehrend dieser Zeit ausgiebig befassen konnte.

Anhang

Glossar

- JDK

  Abkürzung für „Java Development Kit“ der Firma Sun. Es umfasst die Entwicklungs- und Laufzeitumgebung für JAVA.

- SDK

  Abkürzung für „Standard Development Kit“ der Firma Sun. Es umfasst die Entwicklungs- und Laufzeitumgebung für JAVA.

- JVM

  Abkürzung für „Java Virtual Machine“ der Firma Sun. Es enthält die Laufzeitumgebung für JAVA.

- JRE

  Abkürzung für „Java Runtime Environment“ der Firma Sun. Es enthält die Laufzeitumgebung für JAVA.

- GUI

  Abkürzung für „Graphic User Interfaces“. Es stellt die visuelle Schnittstelle zwischen dem Programm und dem Anwender dar.

- CUP

  Ein Parser Generator, der in Java geschrieben ist und Java Code erzeugt.

- JLex

  Ein Generator für lexikalische Analysen, der in Java geschrieben ist und Java Code erzeugt.

- MSIE

  Microsoft Internet Explorer

- Generator

  Programm, das Websites mit erweitertem Befehlssatz in Standard Html übersetzt

- Arbeitsumgebung
  Die Website, welche die Navigation zu den Lerninhalten zur Verfügung stellt.
- Makro
  Kleines Programm in einer in dieser Arbeit entwickelten Sprache.
- Dokument
  Eine Html-Datei,die sich aus zusätzlichen, im Html-Standard nicht enthaltenden, Sprachelementen zusammensätzt.
- Erweiterung
  Eine Datei, welche die Definition für einen zusätzlichen Tag enthält.
- Übersetzer
  D*e*r Teil des → Generators, der die Übersetzung der → Dokumente durchführt.
- Inhalt
  Html-Seiten, die gelernt werden sollen, also zu einer → Lerneinheit gehören.
- Lerneinheit
  Ein abgeschlossenes Kapitel zu einem Thema. (Teilweise als Buch bezeichnet)
- Verlinkung
  Eine Verknüpfung zu einer anderen stelle.

Quellennachweise

Produkte

- J-Builder V3.5
  Eingetragenes Warenzeichen von Borland Inprise Corporation
  http://www.borland.de/
- HomeSite V4.5
  Eingetragenes Warenzeichen von Allaire Corporation
  http://www.allaire.com/
- JLex
  Entwickelt von Elliot Berk, Princeton University
  Weiterentwickelt von C. Scott Ananian
  http://www.cs.princeton.edu/~appel/modern/java/JLex/
- CUP
  Entwickelt von Scott Hudson, 1995
  Weiterentwickelt von Frank Flannery, 1996
  Weiterentwickelt von C. Scott Ananian
  http://www.cs.princeton.edu/~appel/modern/java/CUP/

- Java™
  Eingetragenes Warenzeichen von Sun Microsystems, Inc.
  http://www.javasoft.com/
- JavaScript
  Eingetragenes Warenzeichen von Sun Microsystems, Inc.
  http://www.sun.com/
- Microsoft Internet Explorer
  Eingetragenes Warenzeichen von Microsoft Corporation
  http://www.microsoft.com
- Netscape® Navigator
  Eingetragenes Warenzeichen von Netscape Communications Corporation
  http://www.netscape.com

Firmen

- Netscape®
  Eingetragenes Warenzeichen von Netscape Communications Corporation
  http://www.netscape.com
- Microsoft
  Eingetragenes Warenzeichen von Microsoft Corporation
  http://www.microsoft.com
- Borland
  Eingetragenes Warenzeichen von Borland Inprise Corporation
  http://www.borland.de/

Litteratur

- Seehusen
  Prof. Dr. Silke Seehusen
  Programmierrichtlinien für Java
  Fachhochschule Lübeck, 1997
  http://www.informatik.fh-luebeck.de/ti/Seehusen/Local/Programm
- Dai
  Prof. Dr. -Ing. Kechang Dai
  Compilerbau
  Fachhochschule Lübeck, 1999

- Krueger
  Guido Krueger
  Java 1.1 lernen
  Online-Version
  Addison-Wesley, Bonn, 1997
  http://medoc.informatik.tu-muenchen.de/Java/krueger/index.html
- Harold
  Elliotte Rusty Harold
  Java FAQ list and Tutorial
  Online-Version, 1995
  http://www.informatik.fh-luebeck.de/~java/javafaq.html
- Münz
  Stefan Münz
  Selfhtml 7.0
  Online-Version, 1998
  http://www.teamone.de/selfaktuell/
- Ananian
  Prof. C. Scott Ananian
  Modern Compiler Implementation in Java
  Online-Version, 1998
  Http://www.cs.princeton.edu/~appel
- Aho
  Alfred V. Aho, Raui Sethi, Jeffrey D. Ullman
  Compiler Bau
  Addison-Wesley GmbH, 1988
  ISBN 3-89319-150-X

Beispiele

Um eine bessere Vorstellung von der Vereinfachung, die der Einsatz des Generators bewirkt, zu bekommen, habe ich hier einige Seiten vor und nach der Übersetzung aufgeführt. Die Zunahme an Übersichtlichkeit ist nicht zu übersehen.

Seite aus den Beispielen:

Es folgt die Seite *flash.html*, welche eine Flash-Animation enthält. Den Bereich, der die Flash-Animation einbindet, habe ich zum besseren Vergleich fett hervorgehoben. In diesem Beispiel wird gleich ein Alternativbild mit angegeben, welches angezeigt wird, wenn der Browser kein Flash-Pluin hat.

Vorher (So wurde die Seite erstellt):

```
<html>
<head>
```

```
<meta name="next"         content="realplayer.html">
<meta name="description"  content="finden sich auf dieser Seite.">
<title>Flash</title>
</head>
<body>
<p align="center">
<H1>Flash</H1>
</p>
<p>
<img src="../images/pakete.gif" swfsrc="../images/pakete.swf" width=200
height=200>
</p>
</body>
</html>
```

Nachher (Das macht der Generator daraus):

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head profile="../../profil.rdf">
        <meta name="author"       content="">
        <meta name="description"  content="">
        <meta name="keywords"     content="">
        <link rel="stylesheet"   href="../../htm/style.css" type="text/
css">
        <script language="JavaScript" type="text/javascript"><!--
                window.onerror = handle_error; function
handle_error(){return true;}
                if( !top.UTILITY || !top.UTILITY.UTILITY )
                      document.writeln( "<script language='JavaScript'
src='../../scripts/utilities.js' type='text/javascript'></script>" );
                else
                        top.UTILITY.Stylesheet(self);//-->
        </script>
        <title>Flash</title>
</head>
<body bgcolor="#FFFFFF" leftmargin="0" topmargin="0" marginwidth="0"
marginheight="0" style="background:url(../../images/5_bg.gif);
background-repeat:no-repeat; background-color:#FFFFFF;">
<table cellspacing="0" cellpadding="0" border="0">
<tr valign="top" align="left">
<td width="180">
<!-- Links -->
<img src="../../images/empty.gif" alt="" width="180" height="1"
border="0">
</td>
<td>
<table cellspacing="0" cellpadding="0" border="0">
<tr valign="top" align="left">
<td width="500">
<!-- Mitte -->
<p>
<H1>Flash</H1>
</p>
</td>
<td width="100">
<!-- Rechts -->
<script language="JavaScript"><!--
top.UTILITY.ShowNotiz( self ); // -->
</script>
</td>
</tr>
<tr valign="top" align="left">
<td width="500">
```

```
<p>
<script language="JavaScript">
if( top.flash ){
document.write( "<embed src='images/pakete.swf' width='200'
height='200' loop='false' autostart='true' quality='high'
type='application/x-shockwave-flash'></embed>" );
}
else{
document.write( "<img src='images/pakete.gif' width='200' height='200'
alt='' border='0'>" );
document.write( "<p>Es wird ein <a
href=\"javascript:top.NaviBar.OpenWindow( self, 'http://
www.macromedia.com/shockwave/download/
download.cgi?P1_Prod_Version=ShockwaveFlash', 'Flash', '480', '420',
'scrollbars=yes' )\">Flash Plugin</a> benötigt um diese Seite
vollständig anzuzeigen!</p>" );
}
</script>
</p>
</td>
<td width="100">
<!-- Rechts -->
<script language="JavaScript"><!--
top.UTILITY.ShowNotiz(); // -->
</script>
</td>
</tr>
</table>
</tr>
</table>
<p>
 <table cellspacing="0" cellpadding="0" border="0">
<tr valign="top" align="left">
<td width="180">
<!-- Links -->
<img src="../../images/empty.gif" alt="" width="180" height="1"
border="0">
</td>
<td>
<table cellspacing="0" cellpadding="0" border="0">
<tr valign="top" align="left">
<td width="500">
<!-- Mitte -->
<p>
 <br>Hier kommen Sie <a class="button"
href="javascript:top.NaviBar.SeiteWeiter();"
onmouseover="status='Weiter'; return true;" onmouseout="status='';
return true;" onClick=""> Weiter </a> </td>
<td width="100">
<!-- Rechts -->
<script language="JavaScript"><!--
top.UTILITY.ShowNotiz(); // -->
</script>
</td>
</tr>
</table>
</tr>
</table>
<p>
</body>
<!-- Ermoeglicht das Drucken unter IExplorer -->
<SCRIPT LANGUAGE="VBScript">
<!--
sub print
```

```
on error resume next
WB.ExecWB 6, 1
end sub
//-->
</SCRIPT>
<OBJECT ID="WB"
WIDTH=0 HEIGHT=0
CLASSID="CLSID:8856F961-340A-11D0-A96B-00C04FD705A2">
</OBJECT>
</html>
```

Seite aus der Lerneinheit „Brandels Barkunde“:

Der abgedruckte Code ist ein Auszug aus der Seite *fundament_1.html*. Der vollständige Code befindet sich auf der beiligenden CD.

Vorher:

```
<html>
<head>
<meta name="next"         content="fundament_2.html">
<meta name="titel"        content="Die wichtigsten Fundamente der Bar">
<meta name="author"       content="Andrea Greulich">
<meta name="description"  content="1. Fundament">
</head>
<body>
<p align="left">
<!img src="../images/.gif" alt="Barkunde" width=160 height=200
hspace=10 vspace=4>
</p>
<p align="center">
<H1>1. Angenehme Atmosph&auml;re</H1>
</p>
<p><b>Wer sich wohlfühlt, bleibt länger und konsumiert mehr!</b></p>
<p>Die Einrichtung einer Bar muss zum gewollten Stil passen. Der Stil
wiederum hängt davon ab, welche Gäste angesprochen werden sollen,
welchen Standort das Lokal hat, wie die räumlichen Gegebenheiten sind
usw.</p>

...

<p>Damit sich Ihre Gäste wohl fühlen, sind unangenehme Zeitgenossen,
die andere belästigen oder sich in anderer Weise unziemlich aufführen,
höflich, aber bestimmt zum Verlassen der Bar aufzufordern.
</p>
</body>
</html>
```

Nachher:

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head profile="../../profil.rdf">
        <meta name="author"       content="">
        <meta name="description"  content="">
        <meta name="keywords"     content="">
        <link rel="stylesheet"   href="../../htm/style.css" type="text/
css">
        <script language="JavaScript" type="text/javascript"><!--
                window.onerror = handle_error; function
handle_error(){return true;}
                if( !top.UTILITY || !top.UTILITY.UTILITY )
                      document.writeln( "<script language='JavaScript'
src='../../scripts/utilities.js' type='text/javascript'></script>" );
                else
```

```
                        top.UTILITY.Stylesheet(self);//-->
        </script>
        <title>Die wichtigsten Fundamente der Bar</title>
</head>
<body bgcolor="#FFFFFF" leftmargin="0" topmargin="0" marginwidth="0"
marginheight="0" style="background:url(../../images/4_bg.gif);
background-repeat:no-repeat; background-color:#FFFFFF;">
<table cellspacing="0" cellpadding="0" border="0">
<tr valign="top" align="left">
<td width="180">
<!-- Links -->
<p>
<! img src="../images/.gif" alt="Barkunde" width=160 height=200
hspace=10 vspace=4 >
</p>
<img src="../../images/empty.gif" alt="" width="180" height="1"
border="0">
</td>
<td>
<table cellspacing="0" cellpadding="0" border="0">
<tr valign="top" align="left">
<td width="500">
<!-- Mitte -->
<p>
<H1>1. Angenehme Atmosph&auml;re</H1>
</p>
</td>
<td width="100">
<!-- Rechts -->
<script language="JavaScript"><!--
top.UTILITY.ShowNotiz( self ); // -->
</script>
</td>
</tr>
<tr valign="top" align="left">
<td width="500">
<p>
 <b>Wer sich wohlfühlt, bleibt länger und konsumiert mehr!</b></p>
</td>
<td width="100">
<!-- Rechts -->
<script language="JavaScript"><!--
top.UTILITY.ShowNotiz(); // -->
</script>
</td>
</tr>
<tr valign="top" align="left">
<td width="500">
<p>
 Die Einrichtung einer Bar muss zum gewollten Stil passen. Der Stil
wiederum hängt davon ab, welche Gäste angesprochen werden sollen,
welchen Standort das Lokal hat, wie die räumlichen Gegebenheiten sind
usw.</p>

...

</td>
<td width="100">
<!-- Rechts -->
<script language="JavaScript"><!--
top.UTILITY.ShowNotiz(); // -->
</script>
</td>
</tr>
```

```
<tr valign="top" align="left">
<td width="500">
<p>
 Damit sich Ihre Gäste wohl fühlen, sind unangenehme Zeitgenossen, die
andere belästigen oder sich in anderer Weise unziemlich aufführen,
höflich, aber bestimmt zum Verlassen der Bar aufzufordern.
</p>
</td>
<td width="100">
<!-- Rechts -->
<script language="JavaScript"><!--
top.UTILITY.ShowNotiz(); // -->
</script>
</td>
</tr>
</table>
</tr>
</table>
<p>
 <table cellspacing="0" cellpadding="0" border="0">
<tr valign="top" align="left">
<td width="180">
<!-- Links -->
<img src="../../images/empty.gif" alt="" width="180" height="1"
border="0">
</td>
<td>
<table cellspacing="0" cellpadding="0" border="0">
<tr valign="top" align="left">
<td width="500">
<!-- Mitte -->
<p>
 <br>Hier kommen Sie <a class="button"
href="javascript:top.NaviBar.SeiteWeiter();"
onmouseover="status='Weiter'; return true;" onmouseout="status='';
return true;" onClick=""> Weiter </a> </td>
<td width="100">
<!-- Rechts -->
<script language="JavaScript"><!--
top.UTILITY.ShowNotiz(); // -->
</script>
</td>
</tr>
</table>
</tr>
</table>
<p>
</body>
<!-- Ermoeglicht das Drucken unter IExplorer -->
<SCRIPT LANGUAGE="VBScript">
<!--
sub print
on error resume next
WB.ExecWB 6, 1
end sub
//-->
</SCRIPT>
<OBJECT ID="WB"
WIDTH=0 HEIGHT=0
CLASSID="CLSID:8856F961-340A-11D0-A96B-00C04FD705A2">
</OBJECT>
</html>
```

Nachtrag

Zugriff von Java auf JavaScript

Es hat sich entgegen meinen bisherigen Ermittlungen herausgestellt, dass es doch möglich ist, von einem Java Applet auf eine JavaScript Funktion zuzugreifen. Dies Funktionalität wird von der Klasse `netscape.javascript` bereitgestellt. Entgegen dem was der Name der Klasse suggeriert, funktioniert der Zugriff über diese Klasse auch im MSIE.

CD

Auf der beiliegenden CD befinden sich alle im Rahmen dieser Diplomarbeit entstandenen Produkte. Das sind im einzelnen Folgende Teile:

- Diese Diplomarbeit im Html- und Pdf-Format
- Die Arbeitsumgebung mit den bereits umgesetzten Lerninhalten
- Der Generator mit Quellen und Api

Index