Diplomarbeit

zur Erlangung des akademischen Grades

Diplomkaufmann

an der Wirtschaftswissenschaftlichen Fakultät

der

Universität Augsburg

Einsatz von Markup-Languages

und

Web-Technologien

Eingereicht am Extraordinariat für Wirtschaftsinformatik und Neue Medien

- Prof. Dr. Rony G. Flatscher -

Eingereicht von: Thorsten Schädler

Betreuer : Prof. Dr. Rony G. Flatscher

Augsburg, im Februar 2004

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
ASCII	American Standard Code For Information Interchange
ASP	Active Server Pages
AVI	Audio Video Interleaved
AWT	Abstract Window Toolkit
B2B	Business to Business
B2C	Business to Customer
BMP	Bitmap
BSF	Bean Scripting Framework
BWS	BSFWeb Scripting
CGI	Common Gateway Interface
CLSID	Class Identifier
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets
DCOM	Distributed COM
DHTML	Dynamic HTML
DLL	Dynamic Linked Library
DOM	Document Object Model
DSO	Dynamic Shared Object
DSSSL	Document Style Semantics and Specification Language
.dtd	DTD-Datei
DTD	Document Type Definition
ECMA	European Computer Manufactures Association
FTP	File Transfer Protocol
GIF	Graphics Interchange Format
GML	Generalized Markup Language
GUI	Graphical User Interface

GXA	Global XML Web Services Architecture
.html	HTML-Dokument
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	http Secure
IBM	International Business Machines Corporation
ID	Identifier
IID	Interface Identifier
IIS	Internet Information Services
IP	Internet Protocol
ISO	International Standard Organization
JDK	Java Development Kit
JPEG	Joint Photographic Experts Group
JRE	Java Runtime Environment
JSP	Java Server Pages
MathML	Mathematical Markup Language
MP3	Moving Pictures Experts Group – Audio Layer 3
MPEG	Moving Pictures Experts Group
MS	Microsoft
OLE	Object Linking and Embedding
PDA	Personal Digital Assistant
PHP	Personal Home Page / PHP Hypertext Preprocessor
PNG	Portable Network Graphics
ProgID	Program Identifier
RDF	Resource Description Framework
Rexx	Restructured extended executor
RGB	Rot-Grün-Blau
RMI	Remote Method Invokation
RPC	Remote Procedure Call
RSP	Rexx Server Pages

SDK	Software Development Kit
SGML	Standard Generalized Markup Language
.smil	SMIL-Dokument
SMIL	Synchronized Multimedia Integration
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SSI	Server Side Includes
.svg	SVG-Dokument
SVG	Scalable Vector Graphics
TTS	Text-To-Speach
UDDI	Universal Description Discovery Integration
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WBMP	Wireless Bitmap
Windows ME	Windows Millenium Edition
Windows XP	Windows eXPerience
.wml	WML-Dokument
WML	Wireless Markup Language
.wmls	WMLScript-Datei
WSC	Windows Script Component
WSDL	Web Service Description Language
WSH	Windows Script Host
WWW	World Wide Web
XHTML	eXtensible Hypertext Markupt Language
.xml	XML-Dokument
XML	eXtensible Markup Language
XPath	XML Path Language
.xsd	XML Schema Datei
.xsl	XSL-Datei

XSL	eXtensible Stylesheet Language
XSLFO	eXtensible Stylesheet Language Formatting Objects
XSLT	eXtensible Stylesheet Language Transformation

1. Einleitung

Diese Diplomarbeit entstand auf der Basis eines Seminars von Prof. Dr. Rony G. Flatscher mit dem Titel „Überblick über das objektorientierte Paradigma und seine exemplarische Anwendung (Object Rexx)“, welches mein Interesse an Themen wie HTML, Skriptsprachen und anderen Web-Technologien geweckt hat. Diese Diplomarbeit hat zum Ziel einen Überblick und Einblick in bedeutende Informationstechnologien und deren Zusammenhänge zu geben.

Die Angaben zu den einzelnen Themen sind dabei nur auf die wichtigsten Aspekte begrenzt, da sonst der Rahmen dieser Arbeit gesprengt würde.

Nachdem die einzelnen Technologien in den verschiedenen Kapiteln theoretisch angesprochen als auch erläutert werden, wird deren praktische Anwendung anhand verschiedener Beispiele dargestellt. Zum Nachvollziehen dieser Beispiele sollte der Internet Explorer 6[1] verwendet werden, welcher kostenlos im Internet zum Download bereit steht.

Daneben kann in den meisten Fällen auch der Netscape Navigator (Version 7.1)[2] oder Opera (Version 7.21) [3] als Browser verwendet werden.

In einem ersten Schritt wird der geschichtliche Hintergrund als auch der allgemeine Aufbau der ursprünglichen Metasprache GML bzw. deren Standardisierung in Form von SGML betrachtet. In Kapitel 3 wird daran anschließend etwas tiefer auf die wohl bekannteste Auszeichnungssprache HTML eingegangen. Hierbei wird gezeigt wie HTML zum Erstellen von Benutzeroberflächen für Web Seiten verwendet werden kann.

Da ein reines HTML-Dokument ohne entsprechende Formatierung eher blass wirkt, wird in Kapitel 4 erläutert wie mit Hilfe von Cascading Style Sheets (CSS) ein solches HTML-Dokument optisch aufgewertet werden kann. Ferner werden client- und serverseitige Technologien vorgestellt. Unter Verwendung dieser

---

[1] http://www.microsoft.com/windows/ie/downloads/critical/ ie6sp1/default.asp, Abruf am 2003-06-24.
[2] http://www.netscape.de/netscapeprodukte/netscape71/download.html, Abruf am 2003-08-12.
[3] http://www.opera.com/download/, Abruf am 2003-10-22.

Technologien ist es möglich Dynamik in die ansonsten statischen HTML-Dokumente und damit verbundene E-Commerce Anwendungen zu bringen. In diesem Zusammenhang wird auch auf die Themen, ActiveX, Component Object Model (COM) und Windows Script Host (WSH) eingegangen.

Kapitel 4.4 zeigt anschließend worin die Unterschiede zwischen dem Microsoft Ansatz für Dynamisches HTML (DHTML) und dem Document Object Model (DOM) liegen. Dabei wird auf das `all`- und das `node`-Objekt näher eingegangen und anhand von Beispielen gezeigt, wie diese eingesetzt werden können.

In Kapitel 4.5 wird der allgemeine Einsatz von Skriptsprachen innerhalb eines HTML-Dokuments dargestellt. Dabei wird auch auf Unterschiede zwischen Javascript und Object Rexx eingegangen. Anhand verschiedener Object Rexx Beispiele wird abschließend der Zusammenhang einzelner Technologien aus Kapitel 4 demonstriert und erläutert.

Im darauf folgenden Kapitel wird die so genannte e**X**tensible **M**arkup **L**anguage (XML) etwas genauer betrachtet. XML ist eine Teilmenge von SGML und wird zum Verfassen von Dokumenttypdefinitionen bzw. Auszeichnungssprachen verwendet. XML definiert dabei verschiedene Sprachen mit welchen es möglich wird ein XML-Dokument zu transformieren, zu präsentieren oder es mit anderen XML-Dokumenten zu vernetzen. Diese Sprachen bieten somit die Möglichkeit ein einfaches XML-Dokument zu modifizieren. Solche Modifikationen bzw. Ergänzungen sind XSL, XML Linking und XML-Namensräume.

Kapitel 6 gibt daran anschließend einen Überblick über die XML-basierten Auszeichnungssprachen XHTML, WML, SMIL, SVG und MathML, welche auch als XML-Derivate [Sht03f] bezeichnet werden. Um die Einsatzmöglichkeiten dieser Auszeichnungssprachen zu verdeutlichen, werden einige Beispiele angeführt.

Aufgrund der zunehmenden Bedeutung für B2B- als auch B2C-Bereiche wird im Schlusskapitel auf das Konzept der XML Web-Services und den damit verbundenen Komponenten SOAP, WSDL und UDDI etwas genauer eingegangen. Dabei wird gezeigt wie diese Komponenten mit XML definiert werden und welche Rolle sie im Rahmen der XML Web-Services spielen.

Allgemeine Bemerkungen

Im Rahmen dieser Arbeit ist jeder Code in englischer Sprache verfasst und in einem umrahmten Fenster dargestellt. Der Quelltext ist dabei in schwarzer und die dazugehörigen Kommentare in grüner Farbe gehalten.

```
Sourcecode                                    -- Comment
```

Als Schriftart zur Darstellung solcher Quelltexte als auch zur Darstellung von Beispielen, Syntax, Tags und Programmcode wird Schriftart `Courier New` verwendet. Dabei sind wichtige Teile des Programmcodes in blauer Farbe und/oder **Fett** gehalten.

Des Weiteren werden hervorzuhebende Wörter innerhalb des normalen Textes als auch Attributwerte *kursiv* dargestellt.

Die Object Rexx Skripte in dieser Arbeit wurden unter Verwendung der Object Rexx Version 2.1.2 auf einem Windows 98 und XP Betriebssystem erstellt.

Danksagung

Ein besonderer Dank gilt Herrn Professor Dr. Rony G. Flatscher, der mein Interesse an diesen Technologien geweckt und das Schreiben dieser Arbeit ermöglicht hat. Auch stand er mir stets mit Rat und Tat zur Seite.

Ferner noch ein herzliches Dankeschön an Herrn Florian Helmecke und Herrn Tobias Specht, die mich ebenfalls beim Erstellen dieser Arbeit mit Ratschlägen unterstützten.

2. GML/SGML

Dieses Kapitel hat zum Ziel einen kleinen Einblick in die Entstehung als auch Entwicklung von Meta- und Auszeichngssprachen zu geben. Darüber hinaus wird auf die allgemeine Struktur eines SGML-basierten Dokuments als auch auf das Prinzip der Verarbeitung von SGML-basierten Dokumenten eingegangen.

2.1 Geschichte und Prinzip von GML/SGML

Der Ursprung der **G**eneralized **M**arkup **L**anguage (GML) liegt im Jahre 1969, in welchem Charles **G**oldfarb, Edward **M**osher und Raymond **L**orie für IBM eine Metasprache entwickelten, welche semantische und syntaktische Elemente für die formale Beschreibung von Auszeichnungssprachen (Markup-Languages) zur Verfügung stellte. Die Grundidee welche dahinter steht ist, dass man Struktur und Layout eines Dokuments von einander trennt, und somit ein plattform- und anwendungsunabhängiges Dokument erhält [Vgl. AnCr03, S.3].

Frühere Auszeichnungssprachen haben als wesentliches Kennzeichen ihren *prozeduralen* Charakter, wonach Formatierungsanweisungen oder beliebige andere Verarbeitungsanweisungen in den Text integriert werden [Vgl. HäPeS00, S.437].

GML basierte dagegen bereits auf einem anderen Prinzip der Auszeichnung, welches man als so genanntes *deskriptives Markup* bezeichnet. Hierbei werden Textkomponenten, welche man auch als Elemente bezeichnet, identifiziert und Elementtypen zugeordnet. Dabei ist die Art der Verarbeitung der Komponenten nicht determiniert und kann beliebige Formen annehmen [Vgl. HäPeS00, S.437].

In einem zweiten Schritt werden deskriptive Markierungen üblicherweise auf Verarbeitungsregeln wie beispielsweise Formatierungsanweisungen abgebildet. Neben der Formatierung können die entsprechenden Dokumente aber auch anderen Verarbeitungsalgorithmen wie beispielsweise der Indexerstellung oder Verzeichniserzeugung unterzogen werden [Vgl. HäPeS00, S.437].

Welche Markierungen benutzt werden dürfen, welche zwingend erforderlich sind und wie Markierungen vom eigentlichen Textinhalt abzugrenzen sind, beschreiben die Syntax einer Auszeichnungssprache. Der Aspekt der Bedeutung einer Markierung wird als Semantik einer Auszeichnungssprache definiert [Vgl. HäPeS00, S.438].

Im Jahr 1986 wurde der GML Nachfolger **S**tandard **G**eneralized **M**arkup **L**anguage (SGML) als ISO Standard 8879 verabschiedet . Zudem ist er auch in der Europanorm EN 28879 von 1990 und in der DIN EN 28873 von1991 beschrieben [Vgl. HäPeS00, S.438].

SGML gehört dabei wie sein Vorgänger GML zu den so genannten Auszeichnungssprachen und basiert ebenfalls auf dem Prinzip der deskriptiven Markierung bzw. Auszeichnung von Textkomponenten. Zielsetzung hierbei ist es, diese ausgezeichneten Textkomponenten einer bestimmten Art der Formatierung oder irgendeiner anderen Art weitergehender Verarbeitung zu unterziehen [Vgl. HäPeS00, S.437].

SGML ist nicht für das WWW vorgesehen [AnCr02, S.3] und keine Auszeichnungssprache im bereits oben erwähnten Sinne. Es ist eine Sprache mit welcher die Syntax von Auszeichnungssprachen definiert wird [Vgl.HäPeS00, S.438].

Um eine Systemunabhängigkeit der Dokumente zu erreichen, trennt auch SGML die Dokumentstruktur vom Dokumentlayout. Layout und Struktur können anschließend in einen Browser wieder zusammengeführt werden [Vgl. AnCr03, S.3].

2.2 Dokumentverarbeitung in SGML

Beim Prinzip der Dokumentverarbeitung in SGML, wird zunächst ein Quell-Dokument /SGML Dokument von einem Parser in Zwischencode umgewandelt. Dieser Zwischencode dient anschließend als Input für weitere Verarbeitungsprogramme. Als Verarbeitungsprogramm wird meistens ein Formatierer ver-

wendet. Ausgehend von einem Dokument kann er verschiedene Arten der Darstellung erzeugen [Vgl. HäPeS00, S. 438].

Abbildung 1 veranschaulicht dieses Prinzip der Dokumentverarbeitung.

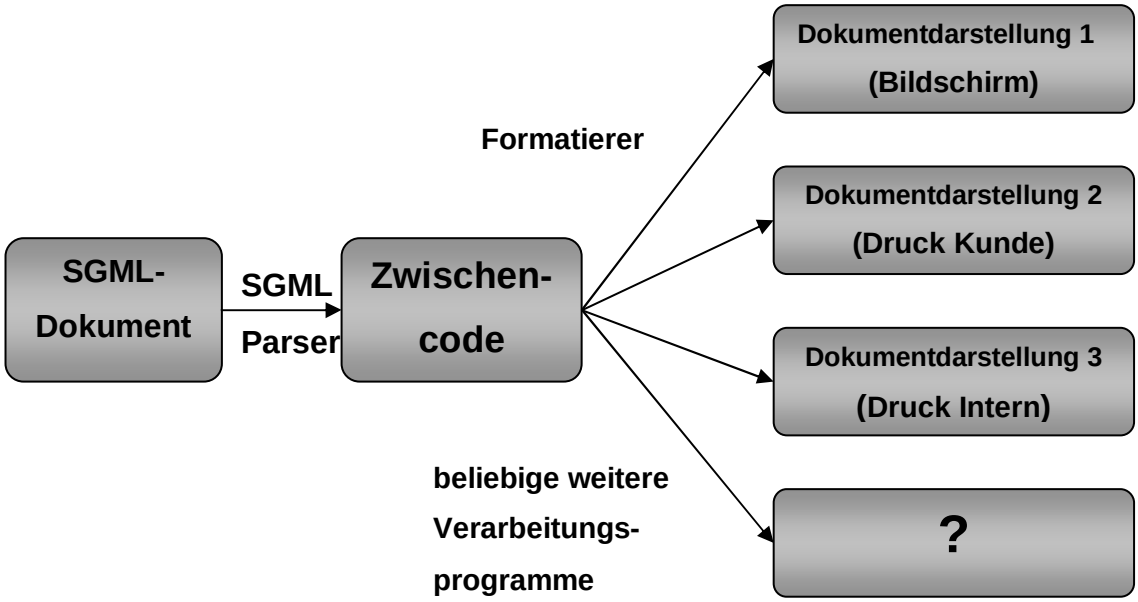

Abbildung 1 : Prinzip der Dokumentverarbeitung in SGML[4]

2.3 Sprachkonstrukte

Die wesentlichen Sprachkonstrukte einer SGML-basierten Sprache sind die *Elemente*, *Attribute* und *Entities*.

Unter einem *Element* versteht man eine strukturelle Komponente eines Textes. Es gibt verschiedene Typen von Elementen, welche über einen eindeutigen Namen identifiziert werden. Ein Element besteht im allgemeinen aus einem Starttag (`<xyz>`), dem Inhalt des Elements und dem Endtag (`</xyz>`). Elemente können auch geschachtelt werden, wodurch eine hierarchische Dokumentenstruktur[5] entsteht. Diese kann mittels einer Baumstruktur visualisiert werden [Vgl. HäPeS00, S.438f].

[4] Nach [HäPeS00, S.438].
[5] S.a. 4.4.3 .

Welches Element in einem Dokument vorkommen darf bzw. kann ist dabei in der entsprechenden DTD definiert.

Attribute werden verwendet um einzelne Elemente genauer zu spezifizieren. Sie werden innerhalb der Elemente platziert ohne jedoch zum Textinhalt eines Elements zu gehören. In der DTD wird der Typ und der Wertebereich eines Attributs ebenso wie ein möglicher Standardwert festgelegt.

Unter *Entity* versteht man den Oberbegriff für eine Fülle an verschiedenartigen Gegenständen. So können Dateien, Teile von Dateien oder gar nur einzelne Zeichen Entities sein [Vgl. Duen03]. Eine häufige Verwendung von Entityreferenzen sind Kürzel und die Ersatzdarstellungen von Sonderzeichen. In SGML beschränkt man sich auf den ISO 646-Zeichensatz (entspricht ASCII) und ersetzt Sonderzeichen durch Referenzen [Vgl. HäPeS00, S.440].

So werden auch in einem Word-Dokument vordefinierte Entities verwendet. Gibt man innerhalb eines Word-Dokuments beispielsweise „mfg" ein, so wird daraus automatisch der Text „Mit freundlichen Grüßen“ erstellt.

2.4 DTD - Dokumenttyp-Definition

Nachdem nun die drei wesentlichen Sprachkonstrukte von SGML-basierten Auszeichnungssprachen wie beispielsweise HTML (SGML-DTD) veranschaulicht wurden, wird nun die Definition einer solchen Sprache mittels SGML betrachtet. Dokumenttypen sind der Kern einer solchen metasprachlichen Definition und werden in Dokumenttypdefinitionen (DTD) spezifiziert. Ein Dokument ist eine Instanz einer solchen Dokumenttypdefinition und muss die in der DTD festgelegten Regeln einhalten [Vgl. HäPeS00, S.440].

„Eine SGML-DTD Deklaration besteht u.a. aus Entity-, Element- und Attributdeklarationen“ [6] [HäPeS00, S.440].

[6] S.a. 5.2 .

In einer DTD wird so mit Hilfe dieser Deklarationen festgelegt welches Element welche Attribute als auch anderen HTML-Elemente (Verschachtelung) beinhalten darf.

2.5 DSSSL

Die **D**ocument **S**tyle **S**emantics and **S**pecification **L**anguage (ISO/IEC 10179) ist eine Sprache zum Formatieren von SGML-Dokumenten und Transformieren von SGML-Dokumenten in andere Dokumente.

Für SGML-Elemente kann mittels DSSSL eine beliebige Bearbeitungssemantik definiert werden [Vgl. HäPeS00, S.444]

Ausgehend von einem SGML-Dokument, kann mittels der Präsentationssprache, Transformationssprache, Dokumentenabfragesprache oder der Sprache zur Formulierung von Ausdrücken, welche Bestandteil von DSSSL sind, die Layout-Struktur konstruiert werden [Vgl. HäPeS00, S.444].

So besteht sowohl die Möglichkeit ein SGML-Dokument entsprechend den Bedürfnissen wie Druck oder für elektronische Medien aufzubereiten als auch ein SGML-Dokument in ein anderes zu transformieren, ohne dabei das SGML-Dokument ändern zu müssen [Vgl. BüMe03, S.6].

Da DSSSL allerdings sehr vielseitig und damit auch sehr komplex ist, konnte es sich wie SGML nie richtig durchsetzten. Genau wie XML eine Vereinfachung von SGML darstellt, wurde auch DSSSL in Form von XSL[7] vereinfacht [Vgl. BüMe03, S.8].

[7] S.a. 5.3.1 .

2.6 Von SGML zu HTML und XML

Tim Berner Lee, der heutige Direktor des W3C, entwickelte die erste Version der Auszeichnungssprache HTML[8], um einen einfachen Austausch von Dokumenten wie beispielsweise Forschungsergebnissen zu ermöglichen. Um eine rasche Verbreitung gewährleisten zu können, basierte HTML anfänglich nicht auf SGML, sondern wurde lediglich an dessen Syntax angelehnt. Erst seit der Version 2.0, welche im November 1995 als offizieller Sprachstandard eingeführt wurde, ist HTML eine in SGML als DTD definierte Auszeichnungssprache [Vgl. AnCr03, S.3f].

Mittels HTML sollte lediglich die logische Struktur eines Dokumentes definiert werden, während die gestalterische Interpretation dieser Struktur Aufgabe des Browsers war [Vgl. HäPeS00, S.445]. „Das Layout eines HTML-Dokumentes ist im Browser quasi „hart codiert “. “ [HäPeS00, S.445].

Mit dem Laufe der Zeit und dem raschen Wachstum des WWW, stiegen für alle Browser, hinsichtlich eines ansprechenden Designs und einer einheitlichen Darstellung, die Anforderungen an die Gestaltung von Web-Seiten. So kamen im HTML-Standard immer wieder neue Tags hinzu, mit welchen es möglich wurde neue gestalterische Elemente wie Tabellen[9], Formulare[10], Frames[11] und dergleichen zu generieren. Insgesamt führten diese Erweiterungen allerdings zu einer „Verwässerung“ in der Trennung von Struktur und Layout der ursprünglichen SGML-Idee. Um diesem Problem entgegenzuwirken, wurde 1994 vom W3C die so genannten Cascading Style Sheets (CSS)[12] als Ergänzung zu HTML eingeführt. Dennoch bleibt in HTML eine andere Problematik weiterhin bestehen. Da es sich bei HTML um eine SGML-DTD und keine Metasprache handelt, ist diese vom Benutzer nicht erweiterbar [Vgl. HäPeS00, S.445].

---

[8] S.a. 3 .
[9] S.a. 3.5.4 .
[10] S.a. 3.5.5 .
[11] S.a. 3.5.6 .
[12] S.a. 4.1 .

Eine mögliche Lösung wäre also der direkte Einsatz von SGML selbst. Um allerdings ein SGML-Dokument (Instanz) als Baumstruktur darstellen zu können, muss dieses von einem SGML-Parser immer gegen seine DTD auf Validität geprüft werden. Ein solches Dokument ist dabei nur darstellbar wenn es gültig (valide) ist, wodurch eine erschwerte Austauschbarkeit von SGML-Instanzen entsteht. Ein weiteres Argument gegen den Einsatz von SGML ist dessen Komplexität. Insofern ist der Einsatz von SGML selbst auch keine geeignete Lösung [Vgl. HäPeS00, S.446].

Hier kommt nun die in Kapitel 5 näher erläuterte e**X**tensible **M**arkup **L**anguage (XML) zum tragen, welche eine SGML-Teilmenge darstellt, die auf einige der komplexeren aber selten genutzten SGML-Elemente und einige SGML-Syntaxregeln verzichtet. So kann in XML beispielsweise die in SGML zwingend erforderliche DTD[13] weggelassen werden [Vgl. HäPeS00, S.447].

Abbildung 2 soll nun noch einmal den Zusammenhang zwischen SGML, HTML, XML und den anderen Auszeichnungssprachen veranschaulichen.

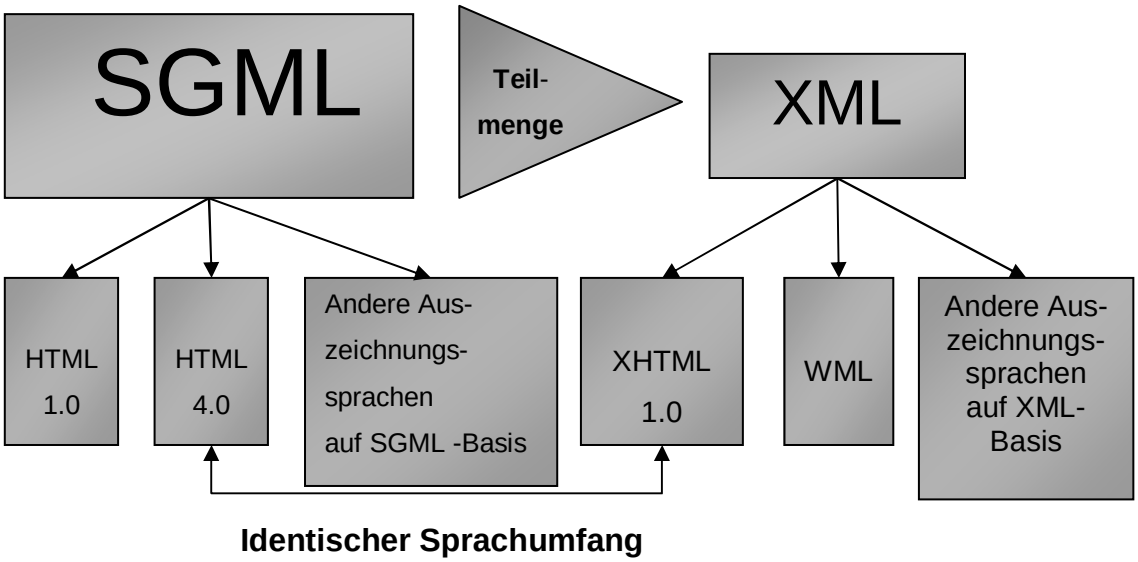

Abbildung 2 : Abhängigkeiten zwischen SGML,HTML und XML[14]

[13] S.a. 2.4 und 5.2 .
[14] Nach [HäPeS00, S.451] .

3. HTML

Um Informationen weltweit und für jedermann lesbar zu machen, benötigt man eine universell verständliche Publikationssprache. Die vom World Wide Web derzeit am häufigsten eingesetzte Publikationssprache ist die Hypertext Markup Language (HTML) .

Bei HTML handelt es sich wie bereits erwähnt um eine SGML-DTD bzw. eine Auszeichnungssprache, welche die Aufgabe hat, Bestandteile wie Verweise[15], Tabellen[16] oder Grafiken[17] und den Aufbau eines Dokumentes zu beschreiben.

Im Laufe der Zeit entwickelten sich immer neue HTML-Versionen und somit auch neue Möglichkeiten. Die aktuelle Version ist HTML 4.0.1.

Wie der Name schon sagt besteht HTML aus den beiden Komponenten Hypertext (HT) und Markup Language (ML).

3.1 Hypertext

Ted Nelson, der 1965 Xanadu entwickelte, gilt als Schöpfer des Begriffs Hypertext. In ihrer ursprünglichen Form handelt es sich bei Hypertexten um elektronische Texte [Vgl. AnCr03, S.3].

Ein besonderes Merkmal solcher Texte besteht vor allem darin, dass sie über Verweise (Links) in miteinander verbundene Informationseinheiten untergliedert sind. Durch solche Verweise wird es dem Leser ermöglicht, diese Einheiten in beliebiger Reihenfolge zu lesen [Vgl. HäPeS00, S.359]

Moderne Hypertexte können dabei unterschiedliche Komponenten wie statische – oder bewegte Grafiken, Videos, Sprache und Ton enthalten [Vgl. HäPeS00, S.359].

[15] S.a. 3.5.1 .
[16] S.a. 3.5.4 .
[17] S.a. 3.5.2 .

3.2 Markup

Markup bedeutet Auszeichnung. Unter einer Auszeichnung versteht man dabei einen „Text, der zu den eigentlichen Inhalten eines Dokumentes hinzugefügt wird, um zusätzliche Informationen über das Dokument bereitzustellen“ [TuKr02, Folie 52].
Ein Dokument besteht dabei aus den drei Grundbestandteilen Inhalt, Struktur und Layout. Inhalt und Struktur sind Bestandteile der so genannten *beschreibenden Auszeichnung*, welche „die Struktur und andere Attribute eines Dokumentes in einer systemunabhängigen Weise beschreibt. Die beschreibende Auszeichnung ist unabhängig von jeglichen Verarbeitungsanweisungen, von der sie ggf. ausgewertet wird.“ [TuKr02. Folie 52]
Unter einer *Verarbeitungsanweisung* ist in diesem Zusammenhang eine Auszeichnung zu verstehen, welche aus systemspezifischen Anweisungen besteht, „die definieren, wie ein Dokument verarbeitet werden soll .“ [TuKr02, Folie 52]

3.3 Struktur und Syntax von HTML

Alle HTML-Anweisungen stehen in so genannten Tags, welche durch spitze Klammern markiert sind. Nahezu alle HTML-Elemente bestehen aus einem einleitenden `<...>` und einem abschließenden `</...>` Tag. Es spielt dabei keine Rolle ob die Tags in Form von Groß- oder Kleinbuchstaben notiert werden. Innerhalb dieser beiden Tags befindet sich der Text, für welchen diese Auszeichnugsanweisungen gelten.

Code 1 zeigt die Struktur eines einfachen HTML-Dokuments.

```
<html>
    <head>
       <title>...</title>
       <meta  ..... >
    </head>
    <body>
     ...
    </body>
</html>
```

Code 1 : Grundlegende Struktur eines HTML-Dokuments

Der gesamte Inhalt eines HTML-Dokuments wird in den Tags `<html> ...</html>` bzw. dem `html`-Element eingeschlossen. Innerhalb des einleitenden `<head>` Tags und dessen Gegenstück `</head>`, werden Titel, Meta- Angaben oder auch Formatierungsanweisungen für das HTML-Dokument notiert. Solche Meta-Tags enthalten dabei unter anderem Angaben zum Autor und Inhalt eines Dokuments. In Code 1 ist dabei auch zu sehen, dass es möglich ist Elemente ineinander zu verschachteln.

Der eigentliche Inhalt des Dokuments wird anschließend im Textkörper `<body>... </body>` notiert. Alles was innerhalb dieses Textkörpers steht wird im WWW-Browser entsprechend der Struktur und Layoutanweisungen angezeigt.

Um das Layout eines HTML-Dokuments festzulegen, können Tags durch Attribute[18] weiter spezifiziert werden. In HTML gibt es folgende fünf Arten von Attributen [Vgl. MüNe01, S.61f] :

- Allein stehende Attribute
- Attribute mit Wertzuweisungen
- Attribute mit numerischen Wertzuweisungen
- Attribute mit prozentualen Wertzuweisungen
- Attribute mit variablen Namen

Allerdings kann nicht jedes Attribut in jedem Tag angegeben werden. Welches Attribut in welchem Tag angegeben werden darf, ist dabei in der entsprechenden DTD[19] geregelt. Attribute welche in nahezu allen Tags verwendet werden dürfen, nennt man Universalattribute[20] [Vgl. MüNe01, S.319].

Aufgrund der wachsenden Anzahl möglicher Sprachversionen und Besonderheiten, welche ein WWW-Browser beim Lesen von HTML-Dokumenten beherrschen muss, ist es durchaus sinnvoll, jedoch nicht erforderlich, die SGML-gerechte Angabe zu der im Dokument verwendeten HTML-Version zu verwenden. Hier kommen nun die Dokumenttypdefinitionen zum tragen. Die Angabe

[18] S.a. 2.3 .
[19] S.a. 2.4 und 5.2 .
[20] S.a. 3.6 .

zur verwendeten HTML-Version wird am Anfang des HTML- Dokuments notiert, und hat standardmäßig folgendes Aussehen [Vgl. MüNe01, S.63f.].

Syntax:
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML [Version]//
    [Sprache]" "[URL]">
```

3.4 Eigenschaften von HTML

HTML-Dokumente bestehen aus ASCII Text, und können insofern mit jedem einfachen Texteditor bearbeitet bzw. geschrieben werden. Sie sind somit softwareunabhängig und können auf jedem Rechner erstellt und dargestellt werden. Neben der Eigenschaft Elemente auszuzeichnen und Benutzeroberflächen zu generieren, ist es eine weitere wichtige Eigenschaft von HTML eine Vernetzung von Dokumenten innerhalb eines Netzes herzustellen. HTML ist nicht auf den Einsatz im WWW beschränkt. HTML-Dokumente können auch lokal auf jedem Rechner dargestellt werden. Somit besteht die Möglichkeit HTML-Dokumente zum Erstellen von Readme-Dateien oder Dokumentationen zu verwenden [Vgl. MüNe01, S.32-35].

3.5 Erstellen von Benutzeroberflächen

Innerhalb von HTML ist es möglich verschiedenste Elemente zu referenzieren. So ist es unter anderem möglich Grafiken, Tabellen, Verweise, Formulare, Objekte und Frames innerhalb eines HTML-Dokuments einzeln oder kombiniert einzusetzen, um beliebige Benutzeroberflächen zu generieren. Angesichts der Möglichkeit solche Benutzeroberflächen als Basis für E-Commerce Anwendungen einzusetzen, soll nun auf diese Elemente und deren Einsatz etwas genauer eingegangen werden.

3.5.1 Verweise

Verweise bzw. Links sind ein zentraler Bestandteil eines jeden Hypertext Projektes und daher in beinahe jedem HTML-Dokument enthalten. Durch den Ein-

satz von Verweisen wird aus einer losen Sammlung von Dateien ein strukturiertes Projekt, welches zum Einsatz im WWW geeignet ist [Vgl. MüNe01, S.141].

Um innerhalb des Körpers eines HTML-Dokumentes einen Verweis zu definieren, steht in HTML das `anchor`-Element zur Verfügung. Dieses Element darf nur im Körper stehen und kann durch bestimmte Attribute spezifiziert werden.

Um ein Verweisziel anzugeben wird das `href`-Attribut benötigt. Dieses Attribut beinhaltet als Wertzuweisung das Verweisziel. Ein Verweisziel kann eine Stelle innerhalb desselben Dokuments, ein anderes HTML-Dokument, eine E-Mail Adresse, eine WWW-Adresse oder eine andere beliebige Datei wie beispielsweise ein Word-Dokument sein. Der wichtigste Aspekt bei einem Verweis ist ein aussagekräftiger Verweistext, welcher den Zweck des Verweises verdeutlichen soll.

Somit ergibt sich folgende grundlegende Struktur eines Verweises:

Syntax : Verweistext

Verweisziele innerhalb eines HTML-Dokuments, stellen sozusagen einen Spezialfall für Verweise dar, da hier zum Einen ein Anker für Verweise definiert, und zum Zweiten ein Verweis auf einen Anker gesetzt werden muss.

1. **Definieren eines Zielankers :**

Bei der Definition eines Zielankers, muss ein über das `name`- oder `id`-Attribut dokumentweit eindeutiger und einmaliger Name für das Ziel angegeben werden.

Syntax : <[Starttag] [*Attribut*]="[Zielname]">……</[Endtag]>
Beispiel : <h1 id="*target*">Headline</h1>

2. **Verweis auf Zielanker setzten :**

Nachdem ein Ziel definiert wurde, kann ein Verweis auf die vorab verankerte Stelle gesetzt werden. Ein solcher Verweis kann dabei innerhalb aber auch au-

ßerhalb dieses HTML-Dokuments gesetzt werden. Dies erreicht man, indem man das gewöhnliche Schema für HTML-Verweise verwendet. Es ist jedoch erforderlich, das Verweisziel zusätzlich mit einem Gatterzeichen [#] zu versehen.

Syntax : Verweistext
Beispiel : top

Eine andere Art von Verweis stellt das `Link`-Element dar. Dieses Element darf nur im Kopf eines HTML-Dokuments vorkommen. Es wird beispielsweise dazu verwendet um externe CSS-Dateien in ein HTML-Dokument einzubinden[21] oder Informationen für Suchmaschinen bereitzustellen [Vgl. W3C02, S.154].

3.5.2 Grafiken

Eine weitere Möglichkeit ein HTML-Dokument interessanter zu gestalten, ist der Einsatz von Bildern bzw. Grafiken. Hierfür steht in HTML das `image`-Element zur Verfügung, mit welchem es möglich wird Grafiken in ein HTML-Dokument einzubinden.

Das `image`-Element hat folgende Grundstruktur :

Syntax: `<img src="[`*`Pfadangabe`*`].[`*`Grafikformat`*`]">`

Als Wertzuweisung des Attributs `src` muss die *Pfandangabe* für das Verzeichnis, in welchem das Bild gespeichert ist, angegeben werden. Das referenzierte Verzeichnis, kann sich dabei auf dem eigenen Rechner als auch auf einem anderen WWW-Server befinden.

Als gängigste *Grafikformate* werden im WWW GIF(.gif) bzw. JPEG.(jpg) verwendet. Um die Vorteile von GIF und JPEG zu vereinen wurde vom W3C das PNG-Format entwickelt [Vgl. MüNe01, S.183].

[21] S.a. 4.1.2.2

Innerhalb des `<img>` Tags können neben dem `src`-Attribut noch weiter Attribute zur Spezifikation des Bildes angegeben werden. Fehlen diese Attribute, so verwendet der Browser zur Darstellung des Bildes die jeweiligen Default-Werte.

Die wohl wichtigsten Attribute sind `width` und `height`, mit welchen die anzuzeigende Größe des Bildes festgelegt werden kann.

Tabelle 1 enthält die nach HTML 4.0 verwendbaren Attribute.

Attribut	Zweck	erlaubte + sinnvolle Werte
alt	Alternativer Text zur Grafik..	beliebige Zeichenkette
border*	Rahmen um Grafik zeichnen.	0-10
name	Namen zu Identifikation der Grafik.	Beliebige Zeichenkette
longdesc	Verweis auf Stelle, wo die Grafik in Textform näher beschrieben wird.	#[Verweisziel]
width	Breite des Bildes im Browser.	% oder Zahl (Pixel)
height	Höhe des Bildes im Browser.	% oder Zahl (Pixel)
align*	Ausrichtung des Bildes.	Left, right, center,Top, middle, bottom, Texttop, absmiddle, absbottm
hspace*	Horizontaler Abstand zwischen Grafik und daneben stehenden Elementen.	Pixel
vspace*	Vertikaler Abstand zwischen Grafik und darüber oder darunter befindlicher Elemente.	Pixel

Tabelle 1 : Attribute für Grafiken[22]

[22] Basiernd auf [MüNe01, S.167-175]

Seit HTML 4.0.1 sind im `image`- als auch im `object`-Element, Attribute für die visuelle Ausrichtung und Darstellung missbilligt. An ihrer Stelle sollen nun Style Sheets[23] verwendet werden [Vgl. W3C02, S.179].

Die in Tabelle 1 angeführten Attribute `align`, `border`, `hspace` und `vspace` sollten daher nicht mehr verwendet werden.

Darüber hinaus ist es auch möglich eine Grafik in einen Verweis[24] einzubinden. Anstelle des Verweistextes setzt man hier mit Hilfe des oben erläuterten `image`-Elements eine beliebige Grafik ein. Somit ist es möglich, diese Grafik als Verweis anzuklicken [Vgl. MüNe01, S.176].

3.5.3 Objekte

HTML bietet die Möglichkeit alle Arten von Dateien bzw. Objekten wie Bilder, Videos, Musikdateien, SVG[25], Java Applets[26] oder ActiveX Controls[27] in ein HTML-Dokument einzubinden [Vgl. MüNe02, S.341]. Sollte es sich um ein Dateiformat handeln, welches der Web-Browser nicht selbst anzeigen kann, so muss ein entsprechendes Plugin[28] installiert sein [Vgl.MüNe02, S.342].

Um nun eine Multimediadatei wie beispielsweise mp3, AVI oder MPEG in ein HTML-Dokument einzubinden, gibt es in HTML das `object`-Element[29]. Dieses Element wird innerhalb des Textkörpers an der gewünschten Stelle eingebaut und bindet über das `data`- oder `classid`-Attribut eine entsprechende Datei in das HTML-Dokument ein [Vgl. MüNe01, S.254-257f]. Das `object`-Element kann darüber hinaus auch im Kopfbereich eines HTML-Dokuments notiert werden [Vgl MüNe02, S.344].

Allerdings wird dieses Element noch nicht vollständig von den Web Browsern unterstützt [Vgl. MüNe02, S.341].

---

[23] S.a. 4.1 .
[24] S.a. 3.5.1 .
[25] S.a. 6.4 .
[26] S.a. 4.2.4.3 .
[27] S.a. 4.3 .
[28] S.a. 4.2.1.1 .
[29] S.a. 4.2.4.4 .

Neben diesem `object`-Element steht in HTML auch noch das von Netscape eingeführte `embed`-Element zur Verfügung. Dieses Element gehört jedoch nicht zum HTML-Standard [Vgl. MüNe02, S.362]. Mit `<embed src="[Pfadangabe]">` wird die Plugin-Referenz definiert. An Stelle von `[Pfadangabe]` wird das Verzeichnis und die sich dort befindliche Datei angegeben. Hierbei ist die Dateiendung von besonderer Bedeutung, da sie dem Browser vermittelt, welche Art von Plugin zu verwenden ist [Vgl. MüNe01, S.266].

Auf diese Weise kann beispielsweise der Windows Media Player mit all seinen Funktionen in ein HTML-Dokument eingebunden werden. Für Plattenfirmen besteht somit eine Möglichkeit, die Musikstücke ihrer Künstler für jedermann online zum Probehören anzubieten oder Filmverleihe können Filme oder Trailer online zum Probesehen anbieten. Ein Download ist dabei nicht möglich. Dieses Feature ist allerdings standardmäßig nur für den MS Internet Explorer verfügbar. Ferner sollte eine ausreichend hohe Datenübertragungsrate, wie sie beispielsweise mittels DSL erreicht werden kann, zur Verfügung stehen.

3.5.4 Tabellen

Tabellen werden in HTML definiert bzw. eingesetzt, um Daten tabellarisch, tabellarische Daten und/oder Text und Grafiken attraktiver darstellen zu können [Vgl. MüNe02, S.233].

Im Folgenden soll nun der Aufbau einer grundsätzlichen Tabelle und deren wichtigsten Elemente und Attribute dargestellt werden.

Eine Tabelle wird innerhalb des Textkörpers (`<body>...</body>`) definiert und hat die in Code 2 dargestellte Grundstruktur:

```
<table>                    -- start of the table
  <tr>                     -- creates a table row
     <th>…….</th>          -- creates the first table head
     <th>…….</th>          -- creates a second table head
      ……
  </tr>                    -- end of table row for table heads
  <tr>                     -- new table row for table data
     <td>……</td>           -- creates a first table data
```

```
      <td>…….</td>            -- creates a second table data
     …….
   </tr>                      -- end of last table row
</table>                      -- end of table
```

Code 2 : Grundstruktur einer Tabelle in HTML

In jeder Tabellenzeile sollte dabei die Anzahl der Zellen gleich sein, so dass die Tabelle durchweg die gleiche Anzahl an Spalten hat .

Da der Browser erst die gesamte Tabelle einlesen muss bevor er irgendwas darstellen kann, gibt es seit HTML 4.0 die Möglichkeit mittels des `colgroup`-Elements die Anzahl der Spalten vorab zu definieren, so dass der Browser sofort weiß, wie viele Spalten er darzustellen hat. Somit kann der Browser bereits Teile der Tabelle am Bildschirm anzeigen, bevor die ganze Tabelle eingelesen ist [Vgl. MüNe01, S.200].

Um nun eine Tabelle optisch aufzuwerten, stehen dem einleitenden `<table>` Tag die in Tabelle 2 dargestellten Attribute zur Verfügung.

Attribut	Erlaubte Werte	Zweck
border	Pixel	Sichtbarer Rahmen und sichtbare Gitternetzlinien in Tabelle erzeugen.
align[30]	left, center, right	Tabelle horizontal ausrichten.
width	Pixel, %	Breite der Tabelle bestimmen.
cellspacing	Pixel	Dicke der Gitternetzlinien innerhalb der Tabelle bestimmen.
cellpadding	Pixel	Abstand zwischen Zellenrand und Zelleninhalt bestimmen.
frame*	Box, void, above, below, hsides, vsides, lhs, rhs	An welchen Seiten soll ein Tabellenrahmen gezogen werden.

[30] Das align-Attribute ist seit der HTML-Spezifikation 4.0.1 missbilligt und sollte nicht mehr verwendet werden. Stattdessen sollte für eine solche Formatierung CSS eingesetzt werden [Vgl. W3C02, S.113].

rules*	None,rows,cols,groups,all	Regeln für Gitternetzlinien bestimmen.
bgcolor	Farbname oder Hexadezimaler RGB-Wert.	Hintergrundfarbe für gesamte Tabelle festlegen.

Tabelle 2 : Verwendbare Attribute im `<table>` Tag[31]

Vorraussetzung für die mit * gekennzeichneten Attribute ist der Einsatz des `border`-Attributs mit einem Wert größer als 0 (Standardwert ist 0) [Vgl. MüNe02, S.243-245].

Als weitere Attribute können die Universalattribute[32] verwendet werden. Darüber hinaus können auch mit Hilfe der CSS[33] alle Elemente einer Tabelle einzeln oder gruppenweise formatiert werden [Vgl. MüNe01, S.204].

3.5.5 Formulare

Mit Hilfe spezieller Befehle ist es in HTML möglich ein oder mehrere Formulare zu erstellen. Solche Formulare sind vor allem hinsichtlich einer Interaktion von Client und Server von Bedeutung. So ermöglicht ein Formular beispielsweise das Suchen in einer Datenbank, das Beisteuern von Daten in eine Datenbank oder das Tätigen von Bestellungen [Vgl. MüNe01, S.277].

Um ein Formular in einem HTML-Dokument zu erzeugen, muss innerhalb des Dateikörpers ein Formularbereich angegeben werden. Mit `<form .... >` wird das Formular definiert. In diesem einleitenden `<form>` Tag kann neben den CSS-Angaben[34] , mit dem `action`-Attribut angegeben werden, was mit den ausgefüllten Formulardaten geschehen soll. Des Weiteren wird die gewünschte Übertragungsmethode (http Befehl)[35] mittels des `method`-Attributs angegeben.

[31] Basierend auf [MüNe02, S.240-252] und [W3C02 , S.113f].
[32] S.a. 3.6 .
[33] S.a. 4.1 .
[34] S.a. 4.1.1 .
[35] S.a. 4.2.6.3 .

Als Übertragungsmethode kommen hierbei `POST` oder `GET` in Frage [Vgl. MüNe01, S.278f].

Um mittels einer Skriptsprache[36] auf ein bestimmtes Formular zugreifen zu können, muss im einleitenden `<form...>` Tag mittels des `name`-Attributs ein dokumentweit eindeutiger Name für das Formular vergeben werden.

Das Universalattribut `id` kann in diesem Zusammenhang nicht verwendet werden.

`Action, method` und `name` sind dabei Eigenschaften (Attribute) des `forms`-Objektes[37], welches sich in der JavaScript Objekthierarchie[38] unterhalb des `document`-Objekts befindet [Vgl. MüNe01, S.586].

Innerhalb dieses einleitenden `<form>` Tags und dessen Gegenstück `</form>`, werden die gewünschten Bestandteile des Formulars wie Eingabefelder, Auswahllisten, Buttons zum Abschicken des Formulars, Checkboxen aber auch Tabellen[39] und dergleichen angegeben [Vgl. MüNe01, S.278f].

Um solche Bestandteile eines Formulars zu erzeugen, bedarf es weiterer in HTML zur Verfügung gestellter Elemente, auf welche im Folgenden etwas genauer eingegangen werden soll.

Diese Elemente können natürlich auch außerhalb eines Formulars verwendet werden. Da sie allerdings zumeist im Rahmen von Formularen Verwendung finden, werden sie auch in dieser Arbeit im Zusammenhang mit Formularen erläutert. So kann beispielsweise ein Button, unabhängig von einem Formular, Bestandteil eines HTML-Dokuments sein.

In den nun folgenden Unterkapiteln werden diese Elemente erläutert und mit je einem Beispiel und einer entsprechenden Abbildung nochmals veranschaulicht.

---

[36] S.a. 4.2.2 .
[37] S.a. 4.4.1.3 .
[38] S.a. 4.2.2.1.2.2 .
[39] S.a. 3.5.4 .

3.5.5.1 Einzeiliges Eingabefeld

Mit `<input type="text"` wird ein einzeiliges Eingabefeld definiert. Mit den zusätzlichen Attributen `name="[interner Bezeichnername]", size="[#Zeichen]"` und `maxlength="[#Zeichen]",` wird der Name, die Anzeigelänge und die interne Feldlänge des Eingabefeldes in Zeichen festgelegt. Zusätzlich kann mittels des `value`-Attributes ein vorbelegter Textinhalt im Eingabefeld angegeben werden. Ein abschließendes `</input>` Tag ist jedoch für das `input`-Element nicht zulässig [Vgl. MüNe01, S.281].

Beispiel:`<input type="text" name="text1", size="8" maxlength="12">`

Abbildung 3 zeigt wie das obige Beispiel in einer HTML-Datei angezeigt wird.

Abbildung 3 : einzeiliges Textfeld

3.5.5.2 Mehrzeiliges Eingabefeld

Um beispielsweise Nachrichten oder Kommentare innerhalb des Formulars schreiben zu können, stellt HTML das `textarea`-Element zur Verfügung, mit welchem ein mehrzeiliges Eingabefeld erzeugt wird. Genau wie das einzeilige Eingabefeld, muss auch das mehrzeilige Eingabefeld einen internen Bezeichnernamen haben. Als weitere Attribute müssen hier jedoch nicht `size` und `maxlength`, sondern `cols` und `rows`, also die gewünschte Anzahl an Spalten und Zeilen angegeben werden [Vgl. MüNe01, S.284]. Es besteht die Möglichkeit aus diesem Eingabefeld ein Ausgabefeld zu machen. Hierzu muss lediglich im einleitenden `<textarea>` Tag noch das Attribut `readonly` angegeben werden [Vgl. MüNe01, S.286]. Dies ist beispielsweise von Bedeutung, wenn mittels einer Skriptsprache Werte innerhalb eines Formularfeldes ausgegeben werden sollen. Abschließend ist es zwingend erforderlich das mehrzeilige Eingabefeld mit dem `</textarea>` Tag zu beenden [Vgl. MüNe01, S.284].

Beispiel: `<textarea name="Mehrzeiler" rows=7 cols=40></textarea>`

Abbildung 4 zeigt wie das obige Beispiel in einer HTML-Datei angezeigt wird.

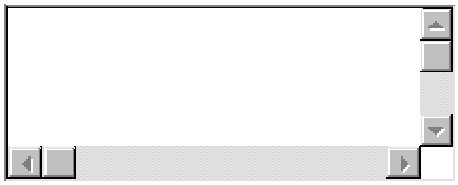

Abbildung 4 : mehrzeiliges Textfeld

3.5.5.3 Auswahlliste

Manchmal werden innerhalb eines Formulars fest vorgegebene Antworten benötigt, aus denen der Interviewte eine oder mehrere Antworten auswählen soll. In HTML kann eine solche Auswahlliste mittels `des select`-Elements eingeleitet werden. Innerhalb des einleitenden `<select>` Tags werden anschließend die bereits bekannten Attribute `name` und `size` definiert. Das Attribut `size` bestimmt dabei, wie viele Einträge im Auswahlfenster angezeigt werden sollen. Falls die Liste mehrere Einträge enthält, als angezeigt werden, wird automatisch eine Scrollleiste erstellt [Vgl. MüNe01, S.287].

Um solche Einträge zu erzeugen, werden in HTML die `option`-Elemente verwendet, welche zwischen dem einleitenden `<select...>` Tag und dem Endtag `</select>` eingebunden werden [Vgl. MüNe01, S.277].

Um in einer Auswahlliste eine Mehrfachauswahl zu erreichen, muss lediglich innerhalb des einleitenden `<select...>` Tags das Attribut `multiple` angegeben werden [Vgl. MüNe01, S.288].

Beispiel :
```
<select name="Fach" size="2">
    <option> Wirtschaftsinformatik
    <option> Controlling
    <option> Financial Engineering
</select>
```

Abbildung 5 zeigt wie das obige Beispiel in einer HTML-Datei angezeigt wird.

Abbildung 5 : Auswahlliste

3.5.5.4 Button

Button können innerhalb eines HTML-Dokuments bzw. eines Formulars verschiedenste Aufgaben erfüllen. So können Button verwendet werden, um Auswahlmöglichkeiten darzustellen, Funktionen aufzurufen oder einfach nur Daten an einen Server zu übergeben.

3.5.5.4.1 Radiobutton und Checkboxen

Möchte man Auswahlmöglichkeiten zum ankreuzen erzeugen, stehen so genannte Radiobutton und Checkboxen zur Verfügung.

Mittels des `<input ...>` Tags bzw. `input`-Elements werden solche Auswahlmöglichkeiten definiert. Innerhalb dieses Tags werden anschließend noch die Attribute `type`, `name` und `value` benötigt. Das `type`-Attribut gibt dabei an, ob es sich um einen Radiobutton (`type="radio"`) oder um eine Checkbox (`type="checkbox"`) handeln soll. `Name` und `value` werden verwendet um einen „internen Bezeichnernamen“ und einen „internen Bezeichnerwert“ zu vergeben. Dieser interne Bezeichnerwert des markierten Button, wird dann beim Abschicken des Formulars übertragen. Alle Buttons desselben Typs, welche den gleichen internen Bezeichnernamen haben, werden zu einer Gruppe zusammengefasst [Vgl. MüNe01, S.291f].

3.5.5.4.2 Klick-Button

Eine weitere Art eines Button stellt der so genannte Klick-Button dar. Der Einsatz eines solchen Buttons ist im allgemeinen nur in Verbindung mit einer Skriptsprache sinnvoll.

Um einen solchen Button zu erzeugen, gibt es seit HTML 4.0 zwei Möglichkeiten [Vgl. MüNe01, S.293ff] :

1. `<input type="button" value="……" [EventHandler]="…….">`

   Wie in Kapitel 3.5.5.4.1 wird auch hier der Button mit Hilfe des `input`-Elements definiert. Anschließend erfolgt die Angabe `type="button"`, ge-

folgt vom `value`-Attribut, mit welchem die Beschriftung des Button vorgenommen wird. Bei [*`EventHandler`*] handelt es sich um ein Universalattribut[40], mit dessen Hilfe eine Verknüpfung zu einer Skriptsprache[41] wie ObjectRexx oder Java Script hergestellt werden kann. Am häufigsten wird hierbei der Event-Handler „`onClick`“ verwendet. Daran anschließend wird in Anführungszeichen dem Browser mitgeteilt, was beim Auslösen des Events zu tun ist. In Kapitel 3.6.1 wird auf die Verwendung der verschiedenen Event-Handler noch näher eingegangen, ehe dann in Kapitel 4.2.2 bzw. 4.5 einige Skriptsprachen vorgestellt und deren Einsatzmöglichkeiten demonstriert werden.

2. `<button name=".." type="button" value="…" [EventHandler]= "…."> ...</button>`

Seit HTML 4.0 besteht die Möglichkeit einen Button mittels des `button`-Elements zu erzeugen. Mit dem `name`-Attribut wird dem Button ein Name gegeben, mit welchem er von einer Skriptsprache aus angesprochen werden kann. In 1. ist dies dagegen nur mit dem Universalattribut `id` zu verwirklichen.

Die Beschriftung des Button erfolgt im zweiten Fall innerhalb des einleitenden und des abschließenden `<button>` Tags. Dies macht diese Form des Buttons flexibler, da es somit auch möglich wird einen definierten Inhalt wie beispielsweise eine Grafik[42] zu verwenden.

3.5.5.4.3 Standard-Button

Innerhalb eines Formulars stellt HTML zwei Standard-Button zur Verfügung. Mit `<input type="reset" value="……">` wird ein Button definiert, mit welchem der Anwender alle Angaben innerhalb des Formulars mit einem Knopfdruck verwerfen kann [Vgl. MüNe01, S.301].

---

[40] S.a. 3.6 .
[41] S.a. 4.2.2 .
[42] S.a. 3.5.2 .

Möchte der Anwender das ausgefüllte Formular abschicken, so muss er den `Submit`-Button verwenden. Ein solcher `Submit`-Button hat in HTML folgendes Aussehen : `<input type=”submit” value=“….”>` [Vgl. MüNe01, S.301].

Mit dem `type`-Attribut wird der Typ des Buttons definiert, während mit dem `value`-Attribut die dem Anwender angezeigte Beschriftung des Buttons angegeben wird [Vgl. MüNe01, S.301].

3.5.6 Frames

Seit HTML 4.0 gehören Frames (Segmente) zum offiziellen HTML-Standard und werden von den bedeutendsten Browsern wie Internet Explorer, Netscape Navigator und Opera problemlos interpretiert. Mit Frames wird der Anzeigebereich eines Browsers in verschiedene frei definierbare Segmente unterteilt [Vgl. MüNe01, S.234].

Die einzelnen Segmente haben dabei verschiedene Aufgaben wie Navigation oder Darstellung und sind miteinander verbunden. Wird beispielsweise in einem Segment zur Navigation ein Verweis angeklickt, so ändert sich entsprechend dem Verweisziel (beliebige SGML- bzw. XML-basiertes Dokument) der Inhalt im Segment zur Darstellung. Hierzu müssen die verschiedenen Segmente allerdings eindeutig identifiziert werden können. Aus diesem Grund wird einem Segment mittels des `name`-Attributs ein eindeutiger Namen zugeordnet [Vgl. MüNe01, S.246].

Um ein Frameset zu generieren wird ein HTML-Dokument erzeugt, welches lediglich ein Frameset der in Code 3 dargestellten Form beinhaltet.

```
<html>
  </head>
  <frameset cols="15%,85%">            -- Creation of a frameset
      -- Creation of a frame for linking
     <frame src="Links.html" name="left">
      -- Creation of a frame for content
     <frame src="Show.html" name="right">
  </frameset>
  <body>
  </body>
</html>
```

***Code 3 : Struktur eines Frames*[43]**

Des Weiteren wird eine zweites HTML-Dokument mit Namen `Links` generiert auf welches innerhalb des Framesets verwiesen wird. Dieses Dokument wird innerhalb des linken Segments der Frameset-Datei angezeigt.

In dem vom Frameset referenzierten HTML-Dokument „*Links*“, wird ein Verweis[44] erzeugt. Mit dem Attribut `target` wird innerhalb dieses Verweises angegeben, in welchem Segment der Inhalt des referenzierten Dokuments ausgegeben werden soll [Vgl. MüNe01, S.277].

Beispiel: `<a href="start.html" target=”right“>….</a>`

3.6 Universalattribute[45]

Mit Universalattribute, werden solche Attribute bezeichnet, die in fast allen HTML-Elementen erlaubt sind. Sie werden dabei wie alle normalen Attribute innerhalb eines einleitenden Tags notiert. Eingesetzt werden diese Attribute, um verschiedene Aufgaben zu erfüllen. So werden sie beispielsweise verwendet, um HTML-Elementen einen individuellen Namen zu geben oder Verknüpfungen zwischen einem HTML-Element und einer Skriptsprache[46] herzustellen.

[43] Nach [MüNe01, S.246].
[44] S.a. 3.5.1 .
[45] Basierend auf [MüNe02, S.377f].
[46] S.a. 4.2.2 .

Bei Letzterem handelt es sich um so genannte Event-Handler, welche eine wichtige Sondergruppe innerhalb der Universalattribute darstellen.

Attrribut	Bedeutung/Zweck
id	Legt einen dateiweit eindeutigen Namen für ein Element fest.
lang	Gibt an, welche Landessprache innerhalb des HTML Elementes verwendet wird.
style	Ermöglicht CSS Definitionen zur individuellen Formatierung eines HTML Elements.
class	Gibt an, dass das HTML Element einer bestimmten Stylesheet-Klasse angehört.
dir	Gibt die Textrichtung der Landessprache an, welche innerhalb des HTML Elements verwendet wird.
title	Erlaubt es, HTML Elemente mit kommentiertem Text zu betiteln.

***Tabelle 3 : Liste der Universalattribute*[47]**

3.6.1 Event-Handler[48]

Da es sich um Attribute handelt, werden auch Event-Handler innerhalb eines einleitenden HTML-Tags angegeben. Ein mit einem Event-Handler versehenes Element, kann auf ein ausgelöstes Ereignis, wie beispielsweise einen Tastendruck oder einen Mausklick, entsprechend reagieren. Event-Handler erkennt man daran, dass sie mit on. . . . . beginnen.

Sie stellen dabei ein wichtiges Bindeglied zwischen HTML und einer Skriptsprache dar, und können in fast allen Elementen notiert werden.

Tabelle 4 enthält die in HTML 4.0.1 spezifizierten Event-Handler.

[47] Nach [MüNe02, S.378]
[48] Basierend auf [MüNe01, S. 322-327].

Event-Handler	Bedeutung
onclick	Beim Anklicken mit der Maus
ondblclick	Beim Doppelklick mit der Maus
onload	Beim Laden des Dokuments
onmousedown	Bei gedrückter Maustaste
onmousemove	Beim Bewegen der Maus
onmouseout	Wenn die Maus den Elementbereich verlässt
onmouseover	Wenn die Maus den Elementbereich erreicht
onmouseup	Beim Loslassen der gedrückten Maustaste
onreset	Beim Zurücksetzen
onselect	Beim Selektieren von Text
onsubmit	Beim Absenden
onunload	Beim Beenden
onkeypress	Wenn Taste gedrückt und wieder losgelassen
onkeyup	Wenn Taste losgelassen
onkeydown	Wenn Taste gedrückt
onfocus	Beim Positionieren auf das Element
onchange	Bei geändertem Wert eines Elements
onblur	Beim Verlassen des Elements

Tabelle 4 : Liste der Event-Handler[49]

Im Zusammenhang mit Event-Handlern, soll hier das `event`-Objekt kurz angesprochen werden.

3.6.2 Das event-Objekt

Das `event`-*Objekt* ist ein Bestandteil der JavaScript Objekthierarche[50] und stellt eine Erweiterung der klassischen Event-Handler dar [Vgl. MüNe02, S.1029].

[49] Nach [MüNe01, S.324-327].
[50] S.a. 4.2.2.1.2.2 .

Mit Hilfe dieses Objekts können Einzelinformationen zu Anwenderereignissen ermittelt und weitergegeben werden. Seit Netscape 4.x als auch dem Internet Explorer 4.x ist dieses Objekt den beiden Browsern bekannt [Vgl. MüNe01, S.636].

Die Implementierung in den beiden Browsern ist jedoch völlig unterschiedlich gelöst. So ist in der Regel eine Eigenschaft des `event`-Objekts entweder nur bei Netscape oder nur beim MS Internet Explorer verfügbar [Vgl. MüNe02, S.883] .

4. Erweiterungen / Ergänzungen für HTML

Wie in den vorangegangenen Kapiteln gezeigt, wird HTML im eigentlichen Sinne lediglich dazu verwendet um die Struktur und den Inhalt eines Dokuments anzugeben.

Um nun ein HTML-Dokument optisch aufzuwerten, Interaktionen zuzulassen, zu dynamisieren und dergleichen, gibt es nun Erweiterungs- bzw. Ergänzungsmöglichkeiten für HTML. Einige dieser Möglichkeiten sollen nun in den folgenden Abschnitten dargestellt und näher erläutert werden.

4.1 CSS - Cascading Style Sheets

Cascading Style Sheets (CSS) stellen eine unmittelbare Ergänzung sowohl für HTML als auch für XML[51] dar. Mit ihrer Hilfe werden die Formateigenschaften einzelner HTML-Elemente definiert [Vgl. MüNe02, S.403].

CSS ist dabei nur eine von mehreren Sprachen zum Definieren von Style Sheets. Neben den CSS gibt es unter anderem noch die Document Style Semantics and Specifications Language (DSSSL)[52], welche für SGML konzipiert wurde und die für XML entwickelte eXtensible Stylesheet Language (XSL)[53] [Vgl. MüNe02, S.404].

Da auch CSS vom W3C Konsortium normiert wird, handelt es sich um einen firmenunabhängigen, offen dokumentierten und frei verwendbaren Standard. 1996 wurde die Version 1.0 herausgegeben, ehe 1998 Version 2 folgte. Momentan wird noch an der dritten Version gearbeitet, welche wiederum neue Möglichkeiten der Formatierung bietet. Da die CSS-Empfehlungen erst einmal in einem Browser implementiert werden müssen, und dies einige Zeit in Anspruch nehmen kann, sollten die neueren bzw. neusten CSS-Angaben jedoch vorerst einmal vorsichtig oder gar nicht eingesetzt werden [Vgl. MüNe02, S.404].

---

[51] S.a. 5 .
[52] S.a. 2.5 .
[53] S.a. 5.3.1 .

4.1.1 Style Sheet Angaben

CSS bieten die Möglichkeit Schrift, Position, Hintergrund, Farbe, Ausrichtung usw. einzelner Elemente oder des ganzen Dokumentes zu formatieren. Hierfür stehen den verschiedenen Elementen bestimmte vordefinierte Attribute zur Verfügung, welchen wiederum bestimmte Werte zugewiesen werden dürfen.

4.1.2 Cascading Style Sheets in HTML einbinden

Um Stylesheets in einem HTML-Dokument zu definieren, gibt es seit Version 1.0 mehrere Möglichkeiten.

4.1.2.1 CSS zentral in einem HTML-Dokument definieren

Um eine dokumentweite Formatierung für HTML-Elemente zu erreichen, kann im Kopfteil eines HTML-Dokuments ein `<style type="text/css">... </style>` Bereich notiert werden. Das im einleitenden `<style>` Tag mit *`text/css`* spezifizierte `type` Attribut entspricht dem MIME-Type[54] der Cascading Stylesheet Sprache.

Innerhalb dieses `style`-Elements wird angegeben, welche Elemente wie formatiert werden sollen. Dabei gibt es wiederum verschiedene Möglichkeiten, welche im Folgenden kurz vorgestellt werden.

Die entsprechenden Style Sheets werden in geschweiften Klammern definiert. Alles was vor den geschweiften Klammern steht wird auch als "*Selektor*" [MüNe02, S.413] bezeichnet.

4.1.2.1.1 Formatieren einzelner HTML-Elemente

Um einzelne HTML-Elemente eines Dateikörpers zu formatieren, wird das zu formatierende Element („*Selektor*" [MüNe02, S.413]) innerhalb des `style`-Elements angegeben [Vgl. MüNe02, S.413].

[54] MIME steht für Multipurpose Internet Mail Extensions. MIME-Typen sind ein Internet Standard um Dateitypen anzugeben.

Beispiel:
```
<style type="text/css">
  h1 {color:red; font-family:helvetia}
  table {font-family:Arial; background-color:blue}
</style>
```

Wird das obige Beispiel in ein HTML-Dokument eingebettet, so werden dort alle `h1`-Elemente[55] in roter Farbe und in der Schriftart `helvetia` dargestellt. Daneben werden alle Tabellen[56] mit einer blauen Hintergrundfarbe versehen. Darin befindliche Texte werden in der Schriftart `Arial` dargestellt.

4.1.2.1.2 Unterklassen definieren

Mit Hilfe von Style Sheets Unterklassen, kann in einem HTML-Dokument dasselbe Element auf unterschiedliche Weise formatiert werden. Unterklassen von HTML-Elementen werden gebildet, indem zuerst das Element angegeben und anschließend ein Punkt, gefolgt vom Namen der Unterklasse, notiert wird [Vgl. MüNe01, S.338].

Innerhalb des Dateikörpers kann dann eine solche Unterklasse mittels des Universalattributs[57] `class` referenziert werden [Vgl. MüNe01, S.338].

Beispiel :
```
<style type="text/css">
     h1.one {color:blue}
     h1.two {color:red}
</style>.....
<body>
  <h1 class="one">Blue Headline </h1>
   .....
  <h1 class="two">Red Headline</h1>
   .....
```

In diesem Beispiels, werden alle Überschriften erster Ordnung die der Unterklasse `one` angehören mit blauer Farbe dargestellt. Alle `h1`-Elemente die der Unterklasse `two` angehören, werden dagegen in roter Farbe dargestellt.

---

[55] Um Hierarchieverhältnisse in Dokumenten abbilden zu können, unterscheidet HTML sechs Überschriftebenen. <h[1-6]> (h= heading =Überschrift) leitet eine Überschrift ein. Die Nummer gibt dabei die Überschriftenebene an, wobei 1 eine Überschrift erster Ordnung (höchste Ebene) und 6 eine Überschrift sechster Ordnung (niedrigste Ebene) erzeugt [Vgl. MüNe01, S.103].

[56] S.a. 3.5.4 .

[57] S.a. 3.6 .

4.1.2.1.3 Verschachtelte HTML Tags definieren

Soll ein Element innerhalb eines anderen Elements anders als dieses formatiert werden, so kann in der zentralen CSS-Definition dies erreicht werden, indem beide Elemente durch ein Leerzeichen getrennt angegeben werden [Vgl. MüNe02, S.415].

Beispiel :
```
<style type="text/css">
    h3 {color:blue}
    h3 i {color:red; font-family:Arioso}
</style>
```

Das Beispiel bewirkt, dass in einem HTML-Dokument alle Überschriften dritter Ordnung in blauer Farbe dargestellt werden. Befindet sich ein `i`-Element[58] innerhalb einer solchen Überschrift dritter Ordnung, so wird der sich darin befindliche Textinhalt in roter Farbe und der Schriftart `Arioso` dargestellt.

4.1.2.1.4 Unabhängige Formate definieren

Des Weiteren besteht die Möglichkeit unabhängige Formate zu definieren, welche einen beliebigen Namen erhalten und über das `id`-Attribut referenziert werden. Den im `style`-Element definierten unabhängigen Formaten muss ein Gatterzeichen # vorangestellt werden [Vgl. MüNe01, S.341].

Beispiel :
```
<style type="text/css">
   #uf1 {color:yellow; font-size:16pt}
</style>
<body> ......
   <h1 id="uf1">....</h1>
      .....
</body> .......
```

Das Beispiel bewirkt, dass der Textinhalt innerhalb des `h1`-Elements in gelber Schrift und der Schriftgröße 16 dargestellt wird.

---

[58] Das `i`-Element ist eine physische Textauszeichnung. Das `i` steht dabei für `italic` und bewirkt, dass der darin befindliche Text kursiv dargestellt wird [Vgl. MüNe01, S.123]. Anstelle des `i`-Elements kann eine solche Formatierung auch mit Hilfe des `style`-Attributs in Verbindung mit der Style-Sheets Angabe `font-style` (Schriftstil) und dem dort zugewiesenen Wert `italic` oder `oblique` vorgenommen werden. Vgl. hierzu auch Kapitel 4.1.2.4 .

4.1.2.1.5 Pseudo-Elemente und Pseudo-Klassen[59]

„CSS führt das Konzept der *Pseudo-Elemente* und *Pseudo-Klassen* ein, um eine Formatierung abhängig von Informationen zu erlauben, die außerhalb des Dokumentbaums liegen.“ [W3Ce04a].

Pseudo-Klassen klassifizieren Elemente nach Charakteristika, die nicht vom Dokumentbaum abgeleitet werden können. Sie können überall in Selektoren auftreten und erscheinen weder im Dokumentbaum noch in der Dokumentquelle.

Tabelle 5 gibt im Folgenden einen Überblick über alle in CSS 2.1 definierten Pseudo-Klassen.

Pseudo – Klasse	Beschreibung
:first-child	Erzielt eine Übereinstimmung mit einem Element, welches das erste Kindelement eines anderen Elements ist.
:link	Gilt für Links, die noch nicht besucht wurden.
:visited	Gilt für einen Link, nachdem er vom Benutzer besucht wurde.
:hover	Wird angewendet, wenn der Benutzer ein Element zwar markiert, es aber nicht aktiviert.
:active	Gilt während ein Element vom Benutzer aktiviert wird
:focus	Gilt während ein Element den Fokus hat
:lang(C)	Erzielt eine Übereinstimmung, wenn das Element im Sprachcode C geschrieben ist.

Tabelle 5 : Pseudo-Klassen

„*Pseudo-Elemente* erzeugen Abstraktionen zum Dokumentbaum, die über die von der Dokumentsprache spezifizierten hinausgehen.“[W3Ce04a].

[59] Basierend auf [W3Ce04a] und [W3C04].

Pseudo-Elemente können nur nach dem Subjekt des Selektors auftreten und erscheinen wie die Pseudo-Klassen ebenfalls weder im Dokumentbaum noch in der Dokumentquelle.

Tabelle 6 zeigt alle gemäß CSS 2.1 verwendbaren Pseudo-Elemente.

Pseudo - Element	Beschreibung
:first-line	Wendet spezielle Stile auf die erste formatierte Zeile eines Absatzes an.
:first-letter	Erzielt eine Übereinstimmung nur mit Teilen der Elemente auf Blockebene.
:before	Wird verwendet um erzeugten Inhalt vor dem Inhalt eines Elements einzufügen.
:after	Wird verwendet um erzeugten Inhalt nach dem Inhalt eines Elements einzufügen.

Tabelle 6 : Pseudo-Elemente

Beispiel :
```
<style type="text/css">
        a:link {color:blue}
        a:visited {color:orange}
        a:active {color:red}
        a:hover {color:yellow}
        h2:first-letter {color:green; font-family:Arioso; font-
           size:200%}
        </style>
```

Im obigen Beispiel werden alle Verweise[60] im HTML-Dokument in blauer Farbe dargestellt (`a:link {color:blue}`). Wird ein solcher Verweis mit dem Mauszeiger berührt, so ändert sich die Schriftfarbe von blau nach gelb (`a:hover {color:yellow}`). Wird der Verweis anschließend angeklickt, wird der Verweis in roter Farbe dargestellt (`a:active {color:red}`). Wird ein anderer Verweis aktiviert, so ändert sich die Farbe des Verweises von rot nach orange (`a:visited {color:orange}`). Darüber hinaus wird der erste Buchstabe einer

[60] S.a. 3.5.1 .

jeden Überschrift zweiter Ordnung in grüner Farbe (`color:green`), doppelter Größe (`font-size:200%`) und der Schriftart Arioso (`font-family:Arioso`) dargestellt.

4.1.2.1.6 Attributbedingte Formatierung

Seit CSS 2.0 besteht auch die Möglichkeit Elemente abhängig von ihren Attributen und/oder Attributwerten zu formatieren. Derzeit ist diese Art der Formatierung allerdings nur mit dem Netscape Navigator und Opera zu verwirklichen. Der Internet Explorer ist derzeit noch nicht in der Lage eine solche Formatierungsangabe zu bearbeiten [Vgl. MüNe02, S.417f].

Um dies zu erreichen muss der „Selektor“ [MüNe02, S.413] noch spezifiziert werden. Hierzu wird nach der Angabe des zu formatierenden Elements in eckigen Klammern das Attribut und bei Bedarf auch der entsprechende Wert angegeben [Vgl. MüNe02, S.418].

Beispiel :
```
<style type="text/css">
   h2[align="center"] {color:green}
   h2[align="right"]  {color:blue}
</style>
```

Dieses Beispiel stellt Überschriften zweiter Ordnung, die mit dem Attribut `align` und der Wertzuweisung *`center`* versehen wurden, in grüner Farbe dar. Wird eine Überschrift zweiter Ordnung dagegen mit `align="`*`right`*“ definiert, so erhält diese Überschrift eine blaue Farbe.

4.1.2.2 CSS zentral in separater Datei definieren

Im Kopfteil eines HTML-Dokuments kann mittels des `link`-Elements eine externe Datei referenziert werden, welche die Formatierungsanweisungen enthält.

Bei der refernzierten Datei muss es sich um eine reine Textdatei mit der Endung “.css“ handeln. Die darin enthaltenen Formatierungsanweisungen gelten dabei ebenfalls dokumentweit. Dies ist sinnvoll wenn beispielsweise eine unternehmensweit einheitliche Formatierung für mehrere HTML-Dokumente erreicht werden soll, da lediglich diese Datei erstellt bzw. geändert werden muss.

Innerhalb des <link> Tags werden die Attribute rel="*stylesheet*", type="*text/css*" und href="*[Pfadangabe]*" notiert [Vgl. MüNe02, S.407].

Sollte sich allerdings das Verzeichnis in welchem sich die Formatierungsdatei befindet oder der Name der Formatierungsdatei ändern, so muss in jedem HTML-Dokument eine entsprechende Änderung der *[Pfadangabe]* erfolgen.

Beispiel : <link rel="*stylesheet*" type="*text/css*" href="[*C:\HTML\CSS\Style.css]*">

4.1.2.3 Das style-Attribut

Um ein einzelnes Element eines HTML-Dokuments zu formatieren muss in diesem das einleitende Tag mit dem style-Attribut versehen werden. Dahinter werden dann in Anführungszeichen die gewünschten Formatdefinitionen notiert [Vgl. MüNe02, S.426].

Beispiel : <h1 style="*color:red*; text-align:*center*">...</h1>

In diesem Beispiel wird lediglich diese Überschrift erster Ordnung in roter Farbe (*color:red*) und zentriert (text-align:*center*) am Bildschirm dargestellt .

4.1.2.4 Das span-Element

Um an einer beliebigen Stelle innerhalb eines Textes eine Formatierung vorzunehmen, muss innerhalb des Textabschnittes eines beliebigen HTML-Elements ein [zu formatierender Text] Element eingebaut werden [Vgl. MüNe01, S.345].

Beispiel: <marquee>Hello Thorsten .How do you do?</marquee>

In diesem Beispiel wird innerhalb eines Lauftextes (marquee-Element) lediglich der Textabschnittes Thorsten kursiv (font-style:italic) und in roter Hintergrundfarbe (background-color:red) dargestellt.

4.1.2.5 CSS für unterschiedliche Ausgabemedien

Neben den verschiedenen Formatierungsmöglichkeiten, kann auch festgelegt werden, wann welches Stylesheet zu verwenden ist. Diese Angaben können dabei in einer separaten Datei gemacht werden oder zentral im HTML-Dokument verankert sein.

4.1.2.5.1 Das media-Attribut

Hierfür wird jedes zu verwendende Stylesheet als separate Datei[61] in das HTML-Dokument eingebunden. Zusätzlich muss das `link`-Element allerdings noch das Attribut `media` enthalten. Mit dieser Angabe wird bestimmt für welches Ausgabemedium die Datei verwendet werden soll. Gemäß CSS 2.0 können derzeit *`print, screen, tv, aural, projection, handheld, braille, embossed, tty`* oder *`all`* als Werte angegeben werden [Vgl. MüNe02, S.409f], [Vgl. W3Ce04b].

Beispiel :
```
<link rel="stylesheet" media="print" href="print.css">
<link rel="stylesheet" media="screen" href="screen.css">
```

Das Beispiel bewirkt, dass zur Formatierung eines HTML-Dokuments, die im Web-Browser angezeigt wird, die CSS-Datei `screen.css` verwendet wird. Soll das HTML-Dokument anschließend ausgedruckt werden, so verwendet der Browser die CSS-Datei `print.css` zur Formatierung.

Eine solche bedingte Formatierung ist beispielsweise sinnvoll, da nicht jede passende Formatvorlage für Bildschirmpräsentationen automatisch eine passende Formatvorlage für Ausdrücke ist [Vgl. MüNe02, S.408f].

4.1.2.5.2 @import-Regel

Die zweite Möglichkeit Angaben zur Formatierung unterschiedlicher Ausgabemedien zu machen ist CSS-Syntax und wird im HTML-Dokument innerhalb des `style`-Elements vorgenommen. Durch die Angabe `@import` wird die Formatie-

[61] S.a. 4.1.2.2 .

rung eingeleitet. Dahinter wird mit `url(....)` die zu verwendende CSS-Datei angegeben. Hinter der Dateiangabe werden anschließend ein oder mehrere gewünschte Ausgabemedien notiert [Vgl. MüNe02, S.410], [Vgl. W3Ce04c].

Beispiel :
```
<style type="text/css">
    @import url(screen.css) screen , tv ;
    @import url(print.css) print ;
</style>
```

Dieses Beispiel ist äquivalent zum Beispiel aus Kapitel 4.1.2.5.1. Allerdings wird eine solche Formatierungsanweisung nicht vom MS Internet Explorer interpretiert.

4.1.2.5.3 @media-Regel

Unterschiedliche Ausgabemedien können auch innerhalb eines HTML-Dokuments definiert werden. Hierzu wird innerhalb eines `style`-Elements die Angabe `@media` gefolgt von einem oder mehreren Ausgabemedien verwendet [Vgl.W3Ce04d], [Vgl. MüNe02, S.410f].

Beispiel:
```
<style type="text/css">
    @media screen, tv
    {
      h2{color:lightyellow; text-align:center}
      body {background-color:black}
    }

    @media print
    {
     h2{color:red; text-align:center}
     body {backround-color:white}
    }
</style>
```

Wird dieses Beispiel in ein HTML-Dokument eingebettet, so wird am Bildschirm ein schwarzer Hintergrund dargestellt, auf welchem alle Überschriften zweiter Ordnung zentriert und in hellgelber Farbe zu sehen sind.

Wird das HTML-Dokument hingegen ausgedruckt, so ändert sich der Hintergrund von Schwarz nach Weiß und den Überschriften zweiter Ordnung wird die Farbe rot zugeordnet.

4.1.3 Beispiel

Das folgende Beispiel soll nun demonstrieren wie wirkungsvoll eine solche Formatierung eines HTML-Dokuments sein kann.

Code 4 zeigt ein HTML-Dokument, welches mit Hilfe der `id` und `class`-Attribute eine elementspezifische Formatierung ermöglicht. Wie diese Elemente als auch andere Elemente formatiert werden sollen, ist in einer externen CSS-Datei festgelegt. Dabei wird für die Bildschirmpräsentation die CSS-Datei `screen.css` (Code 5) und für den Druck die CSS-Datei `print.css` (Code 6) verwendet.

```
<html>
<head>
<title>CSS-Example</title>
<meta name="author" content="Thorsten Schädler">
<link rel="stylesheet" media="screen" href="screen.css">
<link rel="stylesheet" media="print" href="print.css">
</head>
<body >
 <h1>Using CSS in a HTML file</h1>
 <br>
 <p id="uf1">This example uses an <span class="any">extern CSS
file</span> for bringing the HTML file into a good shape</p>
 <p id="uf2">Later, the following table will also be used for
demonstrating <span class="any">XSLT</span></p>
 <hr>
 <table border="3" width="60%" bgcolor="white">
    <tr class="over">
      <td><u><b>Surname</b></u></td>
      <td><u><b>Forename</b></u></td>
    </tr>
    <tr><td>Schädler</td><td>Thorsten</td></tr>
    <tr><td>Design</td><td>Wella</td></tr>
    <tr><td>Mustermann</td><td>Michi</td></tr>
    <tr> <td>Cola</td><td>Coca</td></tr>
 </table>
</body>
</html>
```

Code 4 : HTML-Dokument und CSS

Abbildung 6 zeigt zunächst wie Code 4 ohne die beiden `link`-Elemente (Zeilen 5 und 6) innerhalb des MS Internet Explorers dargestellt wird.

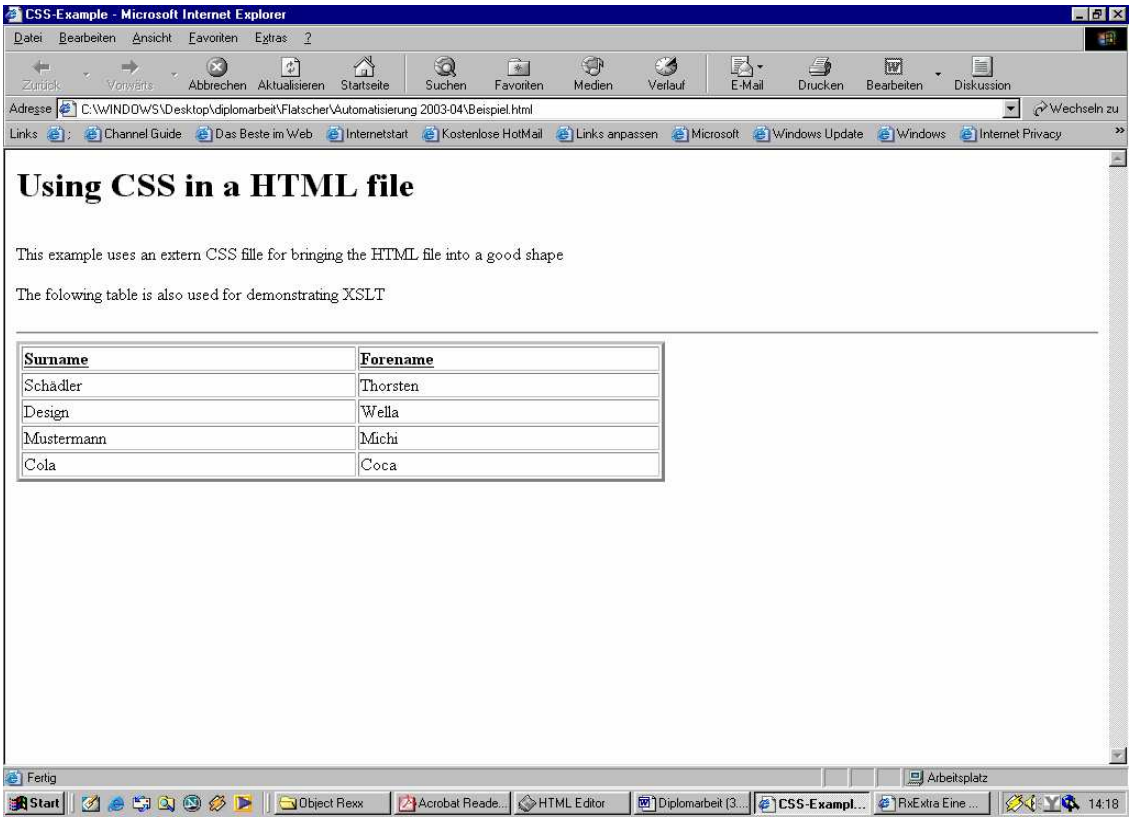

Abbildung 6 : HTML-Dokument ohne Formatierung

Um nun dieses eher blass wirkende HTML-Dokument optisch aufzuwerten wird nun eine CSS-Datei verwendet, welche in Code 5 dargestellt ist.

```
    -- Background is set from white (default) to lightyellow
body {background-color:lightyellow;}
    -- Format for h1-elements
h1 {color:firebrick;text-align:center;font-family:Arioso;}
    -- Format for table-elements
table {color:firebrick;margin-left:200px;margin-top:20px}
    -- Format for a row in a table
tr {text-align:center;}
    -- Format for table rows with a <u> element in it
tr u {color:black;}
    -- Format for elements belong to the class any
.any {background-color:white;color:black;}
    --Format for elements belong to the class over
.over {background-color:orange;font-family:Helmet}
    -- Format for elements with the id uf1
#uf1 {text-align:center;color:blue}
    -- Format for elements with the id uf2
```

```
#uf2 {color:yellowgreen;font-size:16pt;font-family:Chevara;}
```

Code 5 : Formatierung für Bildschirm

Um nun diese Formatierungsanweisungen dem HTML-Dokument zugänglich zu machen, muss die Formatierungsdatei `screen.css` (Code 5) im Kopfteil der HTML Datei über das `link`-Element eingebunden werden (vgl. Code 4 Zeile 5).

```
<link rel="stylesheet" media=”screen” href="sceen.css">
```

Abbildung 7 zeigt nun wie der MS Internet Explorer das HTML-Dokument aus Abbildung 6 darstellt, nachdem die CSS-Datei aus Code 5 eingelesen wurde.

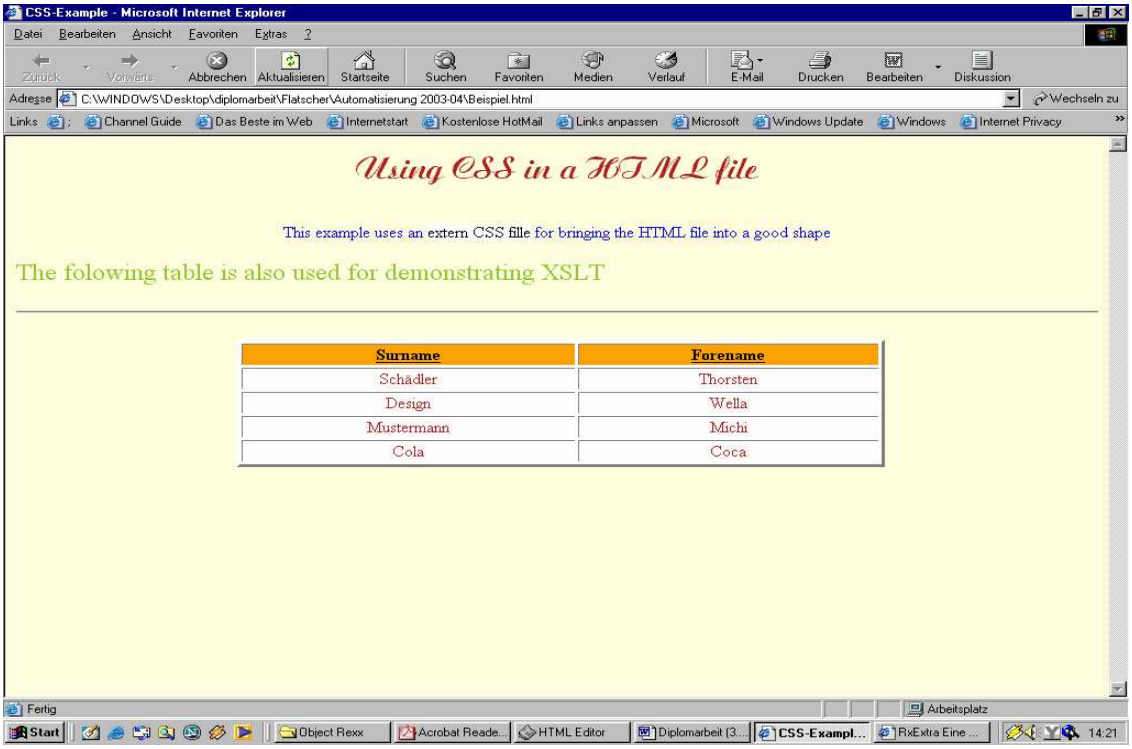

Abbildung 7 : HTML-Dokument mit CSS-Formatierung

Möchte man nun das Dokument drucken, so wird die CSS-Datei `print.css` verwendet, welche ebenfalls über `<link rel="stylesheet" media="print" href="print.css">` in das HTML-Dokument (vgl. Code 4 Zeile 6) eingebunden wird.

Code 6 zeigt die Formatierungsdatei die für den Druck verwendet wird. Dabei sind die Änderungen gegenüber der Formatvorlage für die Bildschirmpräsentation (Code 5) mit blauer Farbe gekennzeichnet.

```
    -- Background is white (default-value) and the document
       has a left,right,top and bottom border of 3cm.
body {background-color:white; margin:3cm}
    -- Format for h1-elements
h1 {color:red;text-align:center;font-family:Arioso;}
    -- Format for table-elements
table {color:red;margin-left:100px;margin-top:20px}
    -- Format for a row in a table
tr {text-align:center;}
    -- Format for table rows with a <u> element in it
tr u {color:black;}
    -- Format for elements belong to the class any
.any {background-color:lightgrey;color:black;}
    --Format for elements belong to the class over
.over {background-color:lightgrey;font-family:Helmet}
    -- Format for elements with the id uf1
#uf1 {text-align:center;color:black}
    -- Format for elements with the id uf2
#uf2 {color:green;font-size:16pt;font-family:Chevara;}
```

Code 6 : Formatierung für Druck

4.2 Web-Technologien

In den Anfängen wurden HTML-Dokumente bzw. Web-Seiten lediglich dazu verwendet um statische Informationen in einem Browser anzuzeigen.

Im Laufe der Zeit wurden die Anforderungen an Web Seiten jedoch immer größer. Aus diesem Grund kam es hier in nur wenigen Jahren zu vielen Neuerungen mit denen Web Seiten nunmehr dynamisch erzeugt werden konnten. Auf Basis dieser so genannten *Web-Technologien* entwickelte sich das WWW von einem Instrument der reinen Informationsdarstellung zu einem Instrument der wechselseitigen Kommunikation und der interaktiven Informationsverarbeitung. Als signifikante Beispiele dieser Entwicklung sind hier der E-Commerce und das Electronic-Banking zu nennen [Vgl. HäPeS00, S.362f].

4.2.1 Überblick[62]

Für das Erzeugen von dynamischen Web-Seiten haben sich mit den *client-basierten* und den *server-basierten* Techniken, zwei prinzipiell unterschiedliche Techniken herausgebildet.

Bevor in diesem Kapitel noch näher auf diese einzelnen Techniken eingegangen wird, soll hier nun zuerst einmal ein kleiner Überblick über diese Techniken und deren Zusammenhänge gegeben werden.

4.2.1.1 Client-basierte Techniken

Die drei wichtigsten client-basierten Techniken sind *Skripte*[63], *Plug-Ins* und *Applets*[64].

Skripte werden entweder in ein HTML-Dokument bzw. eine Web-Seite integriert oder als eigenständige Komponente auf einem Web-Server abgelegt. Hauptaugenmerk solcher Skripte sind die flexible Gestaltung der HTML-Oberfläche als auch die Entlastung des Servers. So besteht für den Benutzer die Möglichkeit, Teile des Dokumentes wie Texte, Grafiken, Video oder Bilder während ihrer Präsentation zu verändern [Vgl. HäPeS00, S.367].

Ein besonders großer Nachteil beim Einsatz dieser Technik besteht allerdings in den Kompatibilitätsproblemen bei Einsatz verschiedener Browser [Vgl. HäPeS00, S.367].

Applets (Java Applets) sind den Skripten sehr ähnlich. Sie werden beim Aufruf vom Server im Byte-Code an den Client übertragen und dort zur Ausführungszeit durch einen Java-Interpreter des Browsers erzeugt [Vgl. HäPeS00, S.368].

Durch einen Compiler des Servers wird der Byte-Code durch einmalige Vorübersetzung erzeugt und kann in dieser Form wiederholt abgerufen werden.

---

[62] Basierend auf [HäPeS00, S.362-369].
[63] S.a. 4.2.2 .
[64] S.a. 4.2.4.3 .

Durch die Verwendung des vollen Java Sprachumfanges und der Plattformunabhängigkeit, ist die Gestaltbarkeit von Anwendungen deutlich größer [Vgl. HäPeS00, S.369].

Da Java Applets in der Java Sandbox ablaufen, haben diese keinen Zugriff auf lokale Dateien, limitierten Zugriff auf Systemfunktionen als auch limitierten Zugriff auf Systemeigenschaften. Außerdem können keine zusätzlichen Netzwerkverbindungen aufgebaut werden [Vgl. TuKr02, Folie 164].

Unter *Plug-Ins* versteht man vom Browser ladbare und dauerhaft gespeicherte Programmmodule, die die Basisfunktionalität eines Browsers erweitern. Jedoch sind Plug-Ins aufgrund der browserspezifischen Schnittstellen hersteller- und plattformabhängig. Das Haupteinsatzgebiet von Plug-Ins liegt in der Darstellung von Multimedia-Anwendungen oder der Darstellung spezieller Dateiformate wie beispielsweise PDF-Dateien [Vgl. HäPeS00, S.367].

Der Vorteil von Plug-Ins gegenüber den Skripten und Applets ist, dass sie keine Übersetzung erfordern und die Sicherheit auf der Client-Seite nicht gefährdet wird. Nachteilig sind allerdings die feste Bindung an den Hersteller des Browsers und die Anwendungen. Bei Fehlen benötigter Plug-In Module, kann die Anwendung nicht ausgeführt werden [Vgl. HäPeS00, S.367].

Abbildung 8 stellt im Folgenden den Ablauf einer clientseitig erzeugten dynamischen Web Seite dar.

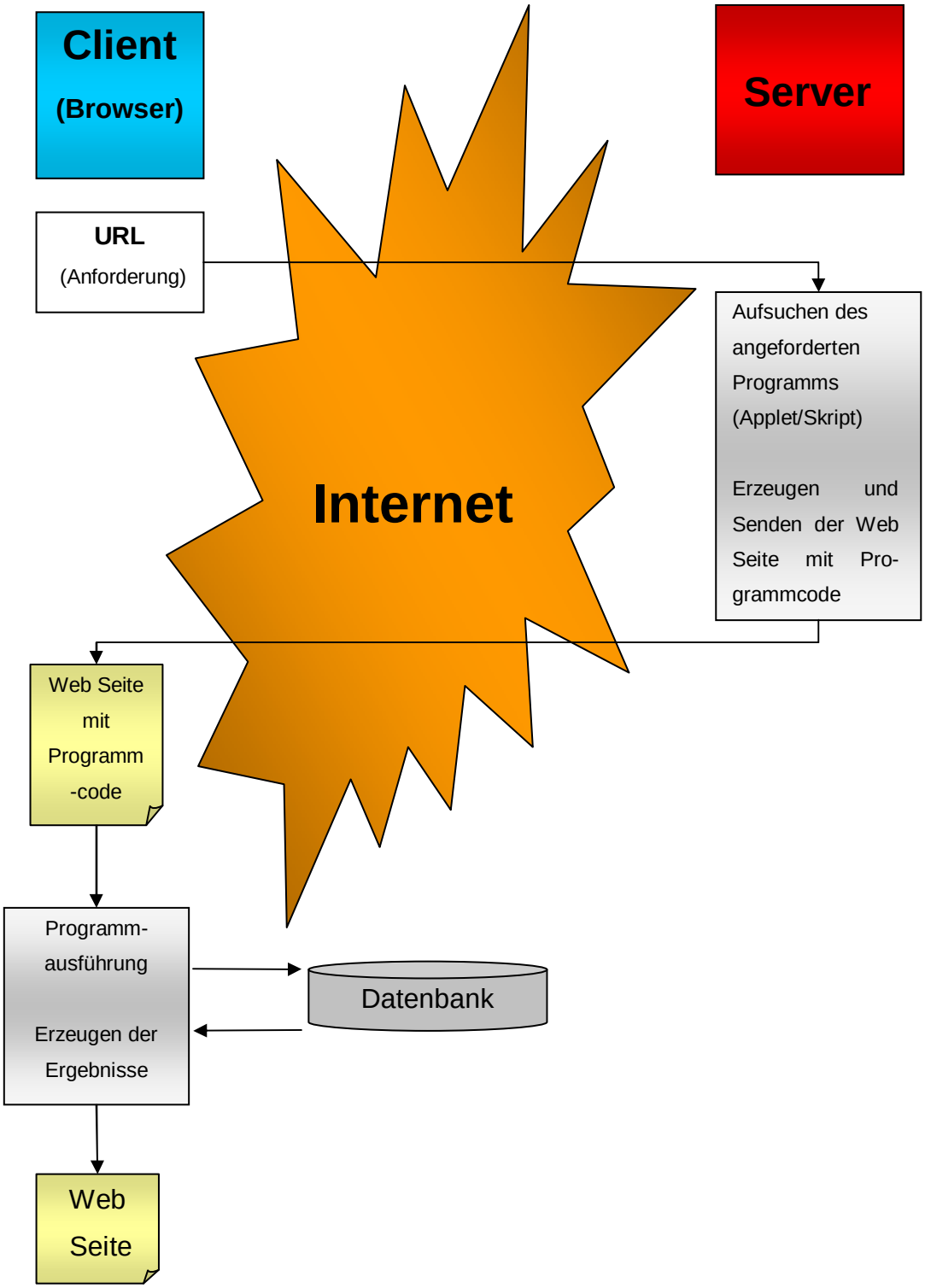

Abbildung 8 : Clientseitige Erzeugung einer dynamischen Web-Seite[65]

[65] Nach [HäPeS00, S.368] .

4.2.1.2 Server-basierte Techniken

Bei den server-basierten Techniken sind zum Einen die „Einbindung von Programmen in die Seite“ und zum Anderen der „Aufruf von Programmen über eine an den Web-Server angebundene Anwendungsschnittstelle oder Bibliotheksaufruf“ zu unterscheiden [HäPeS00, 364].

Beim Einbinden von Programmen, wird eine Anweisung an die Position des HTML-Dokuments eingeführt, an der das Ergebnis dieser Anweisung erscheinen soll. Die Funktionen dieser Programme erweitern dabei die Funktionalität des Servers [Vgl. HäPeS00, S.364].

„Die beiden Grundvarianten sind *Server Side Includes* (SSI)[66] und *Skripte“*.[HäPeS00, S.364].

Um Skripte serverseitig zu verarbeiten, seien hier Microsofts *Active Server Pages* (ASP)[67] , Suns *Java Server Pages* (JSP)[68], Rexx Server Pages (RSP)[69] als auch PHP[70] genannt.

Beim Programmaufruf über eine an den Web Server angebundene Schnittstelle hingegen sind mehrere Varianten denkbar [Vgl. HäPeS00, S.365].

Die älteste und wohl auch bekannteste Variante ist das *Common Gateway Interface* (CGI)[71]. Über diese sprachunabhängigen Schnittstellenspezifikation wird das angeforderte Anwendungsprogramm (CGI-Programm) aufgerufen. Als Programmiersprachen für solche Anwendungsprogramme werden meistens Perl[72], C oder C++ eingesetzt [Vgl. HäPeS00, S.365].

Jedes mal wenn eine Anforderung diese Schnittstelle an das Anwendungsprogramm weitergereicht wird, muss der Server einen neuen Prozess starten und anschließend alle Daten die zur Verarbeitung benötigt werden, übergeben.

---

[66] S.a. 4.2.5 .
[67] S.a. 4.2.10.
[68] S.a. 4.2.9 .
[69] S.a 4.2.12 .
[70] S.a. 4.2.7 .
[71] S.a. 4.2.6 .
[72] S.a. 4.2.7 .

Dies kann zu einem nicht akzeptablen Zeitverhalten führen und stellt somit ein signifikantes Manko von CGI dar [Vgl. HäPeS00, S. 366].

Um diesem Nachteil aus dem Weg zu gehen, werden *Server-APIs* verwendet. Das über ein API gestartete Anwendungsprogramm, läuft unter der Steuerung des Serverprozesses und ist im Vergleich zu einem CGI-Programm und deutlich schneller [Vgl. HäPeS00, S.366].

Jedoch hat auch dieser Ansatz Nachteile. Zum Einen handelt es sich um herstellerspezifische Schnittstellen und zum Zweiten kann ein Absturz eines Anwendungsprogramms den Absturz des gesamten Serversystem nach sich ziehen [Vgl. HäPeS00, S.366].

Servlets[73] stellen wie die Server-APIs oder Apache-Module eine Servererweiterung dar und liegen in ihrer Funktionalität zwischen diesen und der CGI Technologie. Es handelt sich dabei um Java Programme. Diese laufen auf einer Virtual Java Maschine (VJM) ab und werden nach dem Thread-Verfahren im Hauptprozess des Servers platziert [Vgl. HäPeS00, S.366]

„Ein *Thread* ist ein Ablauf innerhalb eines Programms, der unabhängig vom Rest des Programms abgearbeitet werden kann. Existieren mehrere Threads in einem Programm, so können diese für den Benutzer scheinbar parallel ausgeführt werden.“ [KnSW98, S.436]

„Durch den Verzicht auf die Ausführung zeitaufwendiger Prozesswechsel durch die Prozessverwaltung des Betriebssystems ist dieser Ansatz sehr leistungsfähig“ [HäPeS00, S.366].

Abbildung 9 stellt den Ablauf einer serverseitig erzeugten Web-Seite dar.

[73] S.a.4.2.9.2 .

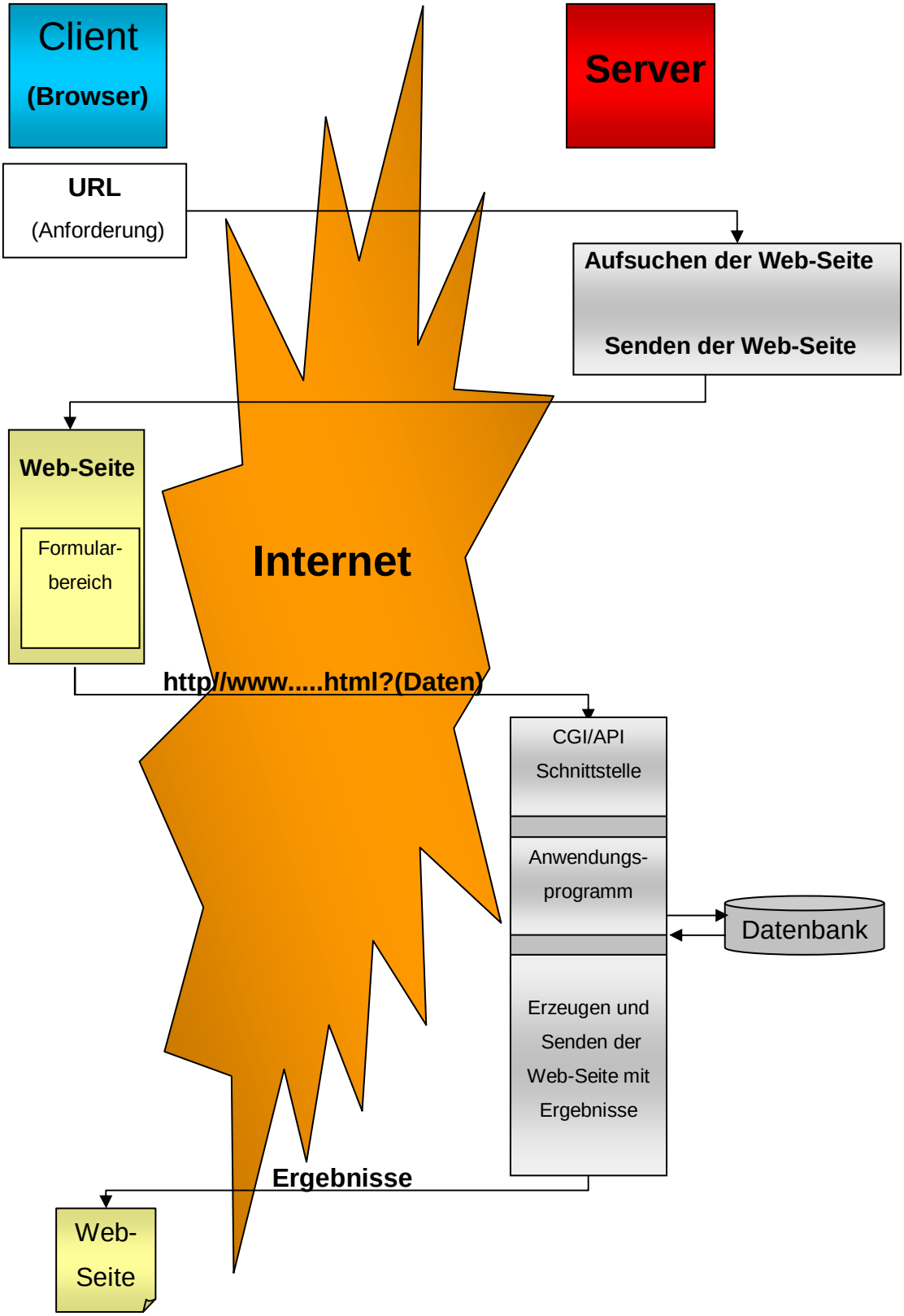

Abbildung 9 : Serverseitige Erzeugung einer dynamischen Web-Seite[74]

[74] Nach [HäPeS00, S.365] .

4.2.2 Skriptsprachen

Skriptsprachen sind neben den CSS eine weitere Möglichkeit einfache HTML-Dokumente interessanter, interaktiver als auch anspruchsvoller gestalten zu können. Mit ihrer Hilfe werden Anwendungen und Dokumente, die über das Internet laufen bzw. gestartet werden können, erstellt.

Skriptsprachen ermöglichen beispielsweise das Prüfen von Formularen auf Vollständigkeit und Plausibilität, das Vorlesen eines HTML-Dokuments mit Hilfe einer Text-To-Speach-Engine (TTS-Engine), den nachträglichen Zugriff auf HTML-Elemente während der Laufzeit oder das Automatisieren von Windows Anwendungen. Letztlich bilden sie zusammen mit HTML[75] und CSS[76] die Grundlage für Dynamisches HTML (DHTML)[77], wodurch faszinierende neue Effekte machbar werden, welche mit reinem HTML nicht möglich wären.

4.2.2.1 JavaScript

JavaScript ist eine von Netscape geschaffene, plattformunabhängige und objektorientierte Skriptsprache [Vgl. Net03a] Es bietet die Möglichkeiten aktiv auf HTML-Dokumente zuzugreifen und diese zu lesen, zu schreiben oder zu verändern. Um eine standardisierte Sprachversion von JavaScript zu entwickeln, arbeitet Netscape mit der ECMA[78] zusammen. Hieraus resultierte letztlich ECMAScript, welches als ECMA-Standard in der ECMA Spezifikation ECMA-262 dokumentiert ist [Vgl. GrüF03, S.4].

4.2.2.1.1 Bestandteile

JavaScript besteht aus den Komponenten JavaScript Core (JSC), Client-Side JavaScript (CSJS) und Server-Side JavaScript (SSJS). Deren Zusammenhang ist in Abbildung 10 dargestellt.

---

[75] S.a. 3.
[76] S.a. 4.1 .
[77] S.a. 4.4 .
[78] http://www.ecma.ch .

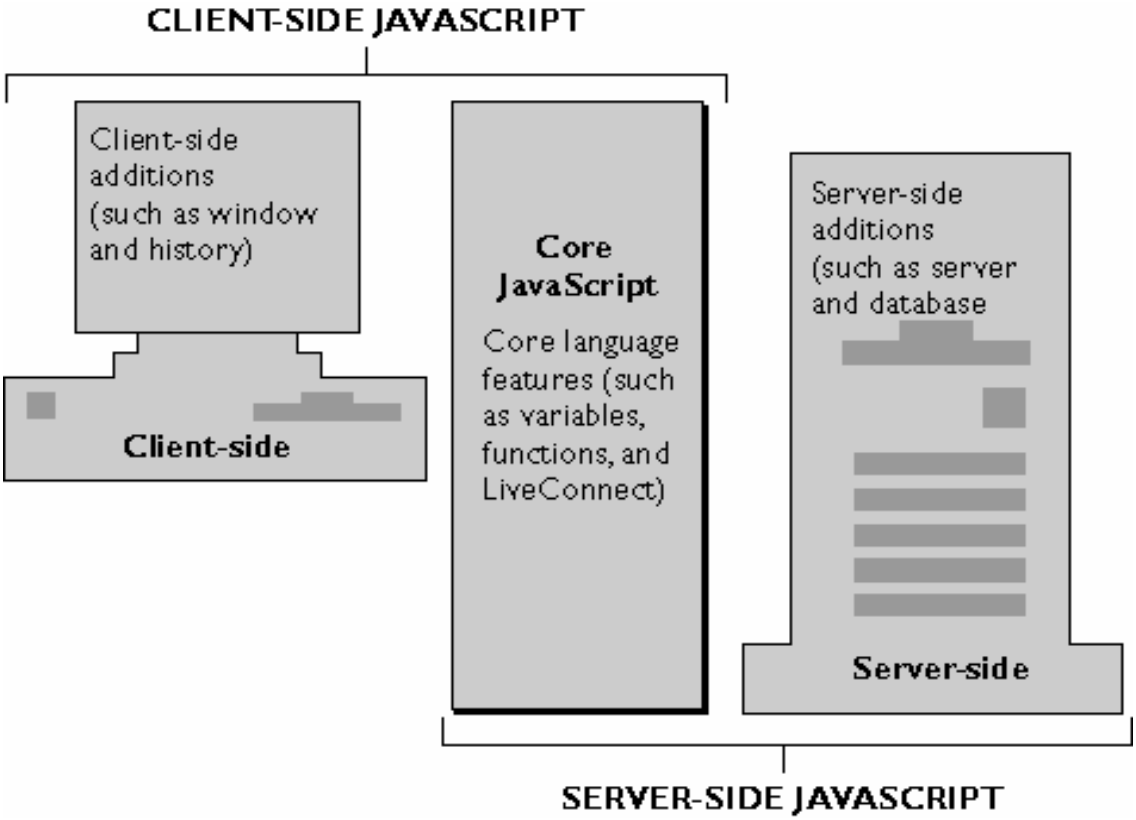

Abbildung 10 : Komponenten von JavaScript[79]

4.2.2.1.1.1 Core JavaScript (CJS)

Core JavaScript stellt das Grundgerüst für CSJS als auch SSJS. Unter anderem enthält es die Definitionen für die Verarbeitung von Anweisungen, (Grund-) Objekte wie z.B. `Date` und `Array` und deren Methoden [Vgl. WinJ03].

4.2.2.1.1.2 Server-Side JavaScript (SSJS)

SSJS ist eine Erweiterung von CJS. Es enthält Objekte, Methoden und Eigenschaften, welche es ermöglichen auf den Server, die aktuelle Verbindung oder Datenanbindungen zuzugreifen, zu lesen oder zu schreiben/verändern. SSJS wird wie CSJS in das HTML-Dokument eingebettet und anschließend von einem Server ausgeführt bzw. verarbeitet . Anschließend werden die Resultate dann an den Client gesendet [Vgl. WinJ03].

---

[79] Entnommen aus http://devedge.netscape.com/library/manuals/2000/javascript/1.3/guide/intro.html, Abruf am 2003-12-22.

4.2.2.1.1.3 Client-Side JavaScript (CSJS)[80]

CSJS ergänzt die Kernfunktionalität von CJS um Objekte, Eigenschaften und Methoden, welche es ermöglichen auf den Browser, die angezeigte(n) Seite(n) und sonstige benutzerspezifische Dinge einzugreifen, zu lesen oder zu schreiben bzw. zu verändern. CSJS wird von einem Browser ausgeführt und verarbeitet [Vgl. WnJ03].

4.2.2.1.2 Klassen, Objekte, Eigenschaften und Methoden

Klassenbasierte und objektorientierte Sprache wie Java[81] oder C++ haben zwei grundlegende Bestandteile. Einerseits (abstrakte) Klassen in welcher Eigenschaften und Methoden definiert werden und zum Zweiten ein Objekt welches eine Instanz einer Klasse darstellt. JavaScript 1.5 hingegen ist eine *prototyp*-basierte Sprache in welcher lediglich Objekte existieren. Dabei sind alle Objekte Instanzen. *Prototypische Objekte* dienen als Vorlage für neue Objekte. Jedes beliebige Objekt kann seine eigenen Eigenschaften entweder bei der Erstellung oder zur Laufzeit spezifizieren. Ferner kann jedes beliebige Objekt als Prototyp für ein anderes Objekt dienen. Dies erlaubt dem zweiten Objekt die Eigenschaften des ersten Objekts zu verwenden [Vgl. Net03b].

In JavaScript 2.0, das sich noch in der Entwicklung[82] befindet, wird sich dies allerdings ändern und es werden Klassen eingeführt.

Eigenschaften eines Objekts können innerhalb des JavaScript Codes jederzeit ausgelesen werden. In vielen Fällen können die Werte von Eigenschaften auch geändert werden. Um Objekteigenschaften ansprechen zu können, notiert man zuerst den Namen des Objekts, gefolgt vom Nachrichtenopertor und anschließend den Namen der Eigenschaft. In JavaScript ist der Nachrichtenoperator der „Punkt“(**.**) [Vgl. MüNe01, S.445f].

Syntax : `Objekt.Eigenschaft;`

---

[80] Vergleiche hierzu 4.2.2.1.2.2 .
[81] S.a. 4.2.4 .
[82] http://www.mozilla.org/js/language/js20/index.html , Abruf am 2004-01-29.

Methoden sind Funktionen, die bei Bedarf Aktionen ausführen. Im Gegensatz zu allein stehenden Funktionen sind sie an ein bestimmtes Objekt gebunden. Sie werden wie Objekteigenschaften angesprochen. Jedoch muss nach dem Methodennamen noch eine öffnende und eine schließende Klammer notiert werden [Vgl. MüNe01, S.446f].

Syntax : `Objekt.Methode();`

4.2.2.1.2.1 Erzeugen eigener Objekte

Um ein eigens Objekt zu erzeugen, stehen in JavaScript zwei Möglichkeiten zur Verfügung. Die Verwendung eines *Objektinitialisierers* (object initializer) oder einer Konstruktorfunktion (constructor function) [Vgl. Net03c].

Um mit Hilfe einer Konstruktorfunktion ein eigenes Objekt zu definieren, muss zuerst ein Objekttyp definiert werden. Um einen Objekttyp zu definieren wird eine Funktion für diesen Objekttyp erzeugt. Diese Funktion spezifiziert den Namen, die Eigenschaften und Methoden des Objekttyps. Nachdem der Objekttyp definiert wurde, können Instanzen dieses Objekttyps angelegt werden. Dies erfolgt mit Hilfe des `new`-Operators. Eine solche Instanz kann eine Eigenschaft besitzen die selbst ein Objekt ist [Vgl. Net03c].

Syntax : `Objekt = new Objekttyp("[Parameter]")`

4.2.2.1.2.2 Vordefinierte Objekte [83]

Neben der Möglichkeit eigene Objekte und deren Eigenschaften und Methoden selbst zu definieren, gibt es in Client-Side JavaScript (CSJS)[84] vordefinierte Objekte, welche dem Anwender den Zugriff auf die Daten eines HTML-Dokuments und deren Umgebung ermöglichen. Diese Objekte haben jeweils vordefinierte Eigenschaften und Methoden. Die Werte solcher Objekteigenschaften können vom Anwender ausgelesen und in manchen Fällen sogar geändert werden.

---

[83] Basierend auf [MüNe01, S.516-646] .
[84] S.a. 4.2.2.1.1.3 .

Abbildung 11 stellt diese vordefinierten Objekte und deren Hierarchie dar. Die in CSJS vordefinierten Objekte werden dabei um das `all`-Objekt[85], welches nicht zum offiziellen Sprachstandard von JavaScript gehört [Vgl. MüNe01, S.560], und das `node`-Objekt[86] erweitert. „Nach der HTML-Variante des Document Object Model (DOM) stellt jedes HTML-Element in einem HTML-Dokument ein Objekt dar.“ [MüNe02, S.700]. Aus diesem Grunde werden in Abbildung 11 die HTML-Elementobjekte angeführt. Da das Netscape 4 spezifische `layers`-Objekt seit der Version 6.0 nicht mehr im Netsacape Navigor unterstützt wird [Vgl. MüNe02, S.862], wird es in Abbildung 11 nicht mehr dargestellt.

In diesem Zusammenhang sei insofern auch auf das DOM[87] verwiesen, welches Klassen bzw. Objekte, Eigenschaften und Methoden definiert, die eine DOM-fähige Skriptsprache umsetzen sollte. [Vgl. MüNe02, S.987].

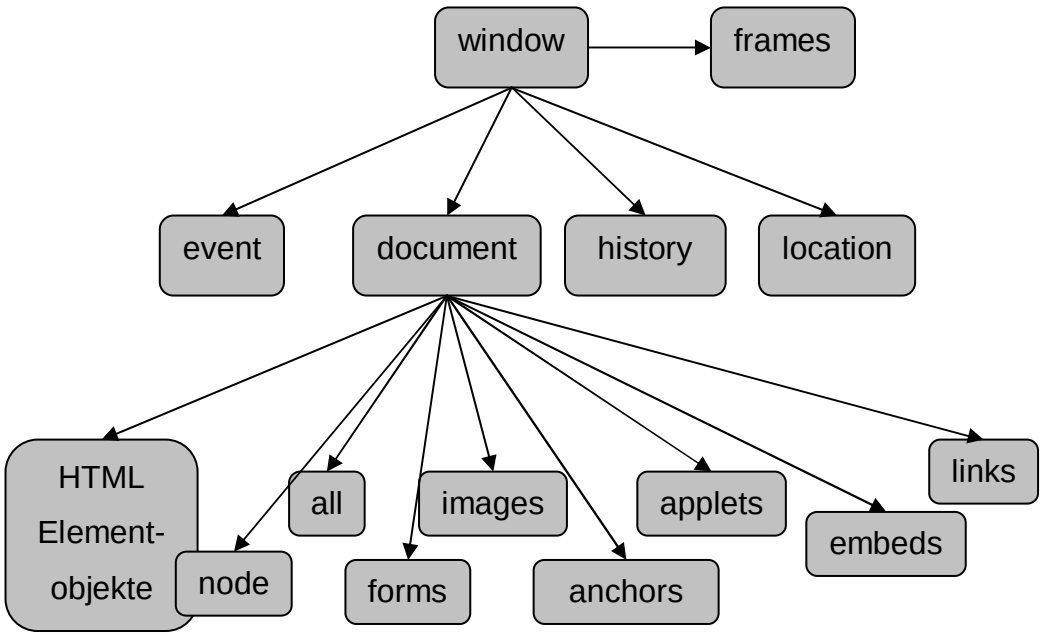

Abbildung 11 : Objekthierarchie

[85] S.a. 4.4.1.1 .
[86] S.a. 4.4.3.2 .
[87] S.a. 4.4.3 .

Das `window`-*Objekt* ist das oberste Objekt in der JavaScript-Objekthierarchie. Es ist sozusagen der Container für alles, was in einem Anzeigefenster eines WWW-Browsers angezeigt werden kann [Vgl. MüNe01, S.520] .

Das `frames`-*Objekt* ist lediglich eine Variante des `window`-Objekt. Es stellt ein eigenes Objekt dar, da es neben den Eigenschaften und Methoden des `window`-Objektes die zusätzliche Eigenschaft `lenght` enthält, und weil bei der Adressierung von Frame-Fenstern ein paar Besonderheiten zu beachten sind [Vgl. MüNe01, S.546].

Wie in Abbildung 11 dargestellt, besitzen das `window`- und das `frames`-Objekt vier Unterobjekte. Jedem dieser Unterobjekte stehen wiederum vordefinierte Eigenschaften und Methoden zur Verfügung, mit welchen sich die verschiedensten Aufgaben erfüllen lassen.

Das wohl bedeutendste und am meisten eingesetzte Objekt ist das `document`-Objekt, welches zugleich auch Ausgangsobjekt für das vom W3C verabschiedete *Document Object Model* (*DOM*)[88] ist [Vgl. MüNe02, S.682].

Wie in Abbildung 11 zu erkennen ist, spiegeln sich die in Kapitel 3.5 dargestellten HTML-Elemente Verweise[89], Grafiken[90] und Formulare[91] als Unterobjekte des `document`-Objekts wieder.

In Tabelle 7 werden diese vier Unterobjekte und deren Zweck bzw. Einsatzmöglichkeiten dargestellt.

Objekt	Zweck / Einsatzmöglichkeiten
event	Einzelinformationen zu Anwenderereignissen wie Mausklicks oder Tastatureingaben ermitteln.
document	Zugriff auf den Inhalt innerhalb eines Browser-Fensters. Ausgangsobjekt des DOM.

[88] S.a. 4.4.3 .
[89] S.a. 3.5.1 .
[90] S.a. 3.5.2 .
[91] S.a. 3.5.5 .

history	Zugriff auf die besuchten WWW-Seiten eines Anwenders.
location	Zugriff auf die vollständige URL-Adresse der aktuell angezeigten WWW-Seite.

Tabelle 7 : Unterobjekte des window-Objekts

Neben diesen Objekten, gibt es in JavaScript auch noch weitere vordefinierte Objekte, auf welche im Rahmen dieser Arbeit jedoch nicht weiter eingegangen werden soll.

Hinsichtlich der Bedeutung des `document`-Objekts für Dynamisches HTML[92], wird im Rahmen dieser Arbeit auf dieses Objekt und dessen Unterobjekte zu einem späteren Zeitpunkt noch näher eingegangen.[93]

4.2.2.2 Object Rexx

1979 entwickelte Mike F. Colishow (IBM-Fellow) einen Nachfolger für die kryptische Skriptsprache „Exec“. Dieser Nachfolger heißt Rexx und zeichnet sich durch eine einfache und schnell zu erlernende Syntax aus [Vgl. Fla03a, S.6].

Auf Initiative der einflussreichen IBM-Benutzervereinigung „SHARE“ wurde Anfang der 90er eine objektorientierte Version von Rexx entwickelt. Diese intern vollständig objektorientiert aufgebaute Version heißt Object Rexx und ist vollständig kompatibel zum bisherigen prozeduralen Rexx. Neben der weiterhin einfachen Syntax, besitzt Object Rexx ein mächtiges Objektmodell und ist für viele Betriebssysteme wie für Windows, OS/2, AIX oder Linux verfügbar [Vgl. Fla03a, S.7].

Ein Object Rexx Programm kann entweder wie JavaScript in eine Web Seite eingebettet werden oder für sich selbst innerhalb der Object Rexx Workbench ablaufen.

---

[92] S.a. 4.4 .
[93] S.a. 4.4.3 .

Im Gegensatz zu JavaScript ist Object Rexx nicht in die Browser integriert.

Um nun Object Rexx als Skriptsprache verwenden zu können, muss der Object Rexx Interpreter auf dem System installiert sein und der MS Internet Explorer als Browser verwendet werden. Dies liegt daran, dass Object Rexx eine Windows Script Host Engine (WSE) ist und zur Kommunikation einen Windows Script Host (WSH)[94] benötigt. Im Gegensatz zum Opera Browser als auch dem Netscape Navigator ist der MS Internet Explorer ein solcher WSH. Es besteht allerdings die Möglichkeit Object Rexx Skripte dennoch innerhalb anderer Web Browser mit Hilfe des *BSFWebScripting* Ansatzes zum Laufen zu bringen. Insofern wird auf diesen Ansatz in Kapitel 4.2.3 näher eingegangen.

Die folgenden Kapitel sollen nun lediglich einen kleinen Einblick in das Arbeiten mit Object Rexx geben, und zum besseren Verständnis der später folgenden Beispiele dienen.

4.2.2.2.1 Prozeduren (Operationen) und Funktionen

Eine *Prozedur* ist ein Unterprogramm, welches mit einer Sprungmarke (z.B. `call`) beginnt. Anschließend werden die Unterprogrammanweisungen ausgeführt. Die `RETURN`-Anweisung gibt zum Schluss die Kontrolle an die Anweisung zurück, die unmittelbar auf den Unterprogrammaufruf folgt [Vgl. Fla03b, S.3].

Eine *Funktion* ist eine Prozedur, die einen Wert („Funktionswert“) mit Hilfe der `RETURN`-Anweisung an den Aufrufer zurückgibt [Vgl. Fla03b , S.5].

Rexx stellt mehrere zur Sprache gehörenden Funktionen zur Verfügung [Vgl. Fla03b, S.8].

4.2.2.2.2 Klassen, Objekte, Attribute und Methoden

Object Rexx stellt einige Klassen und darin enthaltene vordefinierte Methoden zur Verfügung. Object Rexx bietet die Möglichkeit eigene Klassen zu definie-

[94] S.a. 4.3.3 .

ren. Diese *Klassen* werden durch *Methoden* (Operationen) und *Attribute* (Eigenschaften) spezifiziert [Vgl. Fla03d, S.9].

Unter einem *Objekt* versteht man eine Instanz einer Klasse. Alle Objekte einer Klasse weisen dieselben vordefinierten Operationen und Attribute auf. Objekte können Nachrichten erhalten und auf diese entsprechend reagieren. Um einem Objekt eine Nachricht zu senden, wird der gewünschte Methodenname in der Nachricht angegeben. Objekte können dabei nur auf Nachrichten reagieren dessen Methode sie kennen [Vgl. Fla03d, S.10f].

Um Nachrichten zu versenden wird der Object Rexx Nachrichtenoperator „**~** “ (TildeZeichen - „Twiddle“) benötigt [vgl. Fla03d, S.11].

Syntax: `object~method`

Um eine neues Objekt einer Klasse zu instanzieren, wird der Klasse die `New`-Methode geschickt [Fla03d, S.10].

Syntax: `object = .class~New()`

4.2.2.2.3 Direktiven

In einem HTML-Dokument können ein oder mehrere Object Rexx Programme eingebettet sein. Um in einem HTML-Dokument diese Programme bei Bedarf zu starten, wird die sog. `::ROUTINE`-Direktive in Verbindung mit einem Event-Handler[95] benötigt.

Neben dieser Direktive gibt es in Objekt Rexx noch die `CLASS` -, die `METHOD` - und die `REQUIRES`-Direktive.

Eine Direktive beginnt immer mit einem doppelten Doppelpunkt (::).

Die `::CLASS [classname]` Direktive wird verwendet um eine Rexx Klasse zu erstellen. Diese erzeugte und mit einem Namen [`classname`] versehene Klasse, wird Programmen über das Rexx Umgebungssymbol “ **.** “ , gefolgt vom Na-

[95] S.a. 3.6.1 .

men der Klasse zur Verfügung gestellt. Diese Klasse erwirbt alle in den nachgestellten ::METHOD-Direktiven definierten Methoden [Vgl. IBM03a, S.105] .

Beispiel : ::CLASS person

Über die optionalen Schlüsselwörter „PUBLIC“, „SUBCLASS“, „MIXINCLASS“, „METACLASS“ als auch „INHERIT“ kann die Eigenschaft einer Klasse näher bestimmt werden [Vgl. Fla03e,S.18].

Die ::METHOD [methodname] Direktive erzeugt ein Methodenobjekt und definiert neben den Methoden (Operationen), die Attribute (Eigenschaften) und damit die interne Datenstruktur einer Klasse. Eine solche ::METHOD-Direktive kann auch außerhalb einer ::CLASS-Direktive stehen und über das **.**METHODS Archiv den Programmen zugänglich gemacht werden [Vgl. IBM03a, S.107f.].

Beispiel :
```
::CLASS person
::METHOD name ATTRIBUTE
```

Die ::REQUIRES [programname] Direktive gibt an, dass das auszuführende Programm/Skript Zugang zu den Klassen und Objekten dieser Rexx Programms [programname] benötigt [Vgl. IBM01a, S.109f.]. Alle öffentlichen Klassen und Routinen, welche in dem referenzierten Programm [programname] enthalten sind, werden dem ausführenden Programm zur Verfügung gestellt.

Der Aufruf des referenzierten Programms erfolgt während der Initialisierungsphase ehe die restlichen Direktiven befolgt werden [Vgl. Fla03c, S.24]

Syntax : ::REQUIRES “[programname].[filetype] “

Beispiel : ::REQUIRES “OREXXOLE.CLS“-- Klassendefinition OREXXOLE

Die ::ROUTINE-Direktive kann ein Unterprogramm (Prozedur) als auch eine Funktion darstellen [Vgl. Fla03c, S.16].

Wird dem Routinenamen noch das Schlüsselwort „`public`“ angehängt, so wird die Routine allen übergeordneten (aufrufenden) Programmen zugänglich gemacht [Vgl. Fla03c, S.16]

Code 7 stellt den Zusammenhang zwischen den Direktiven `::CLASS`, `::METHOD` und `::ROUTINE` dar.

```
P = .person~New      -- creates an instance of the class person
p~firstname="Thorsten"    -- assign the value “Thorsten” to the
                             attribute firstname
p~surname="Schaedler"    -- assign the value “Schaedler” to the
                             attribute surname
p~number="1001"           -- assign the value 1001 to the
                             attribute number
    -- Output
say "Hello " p~firstname p~surname " your old number is: "
p~number
    -- Output
say "Your new number is now : " p~~change~number
call doit                  -- invokes the ::routine doit

::CLASS person             -- definition of the class person
                              with the ::CLASS directive
::METHOD firstname ATTRIBUTE   -- specify the attributes of the
::METHOD surname   ATTRIBUTE      class person using the
::METHOD number    ATTRIBUTE      ::METHOD directives
::METHOD change            -- creates a method by using the
                                ::METHOD directive
  EXPOSE number            -- get access to the number attribute
   number = 0815           -- changes the number from 1001 to
                              0815
::Routine doit public                -- the ::ROUTINE doit
 say "I am the routine doit "        -- Output
```

Code 7 : Person.rex

Code 7 erzeugt unter Verwendung der eigens definierten Klasse `person` innerhalb der Object Rexx Workbench die in Abbildung 12 dargestellte Ausgabe.

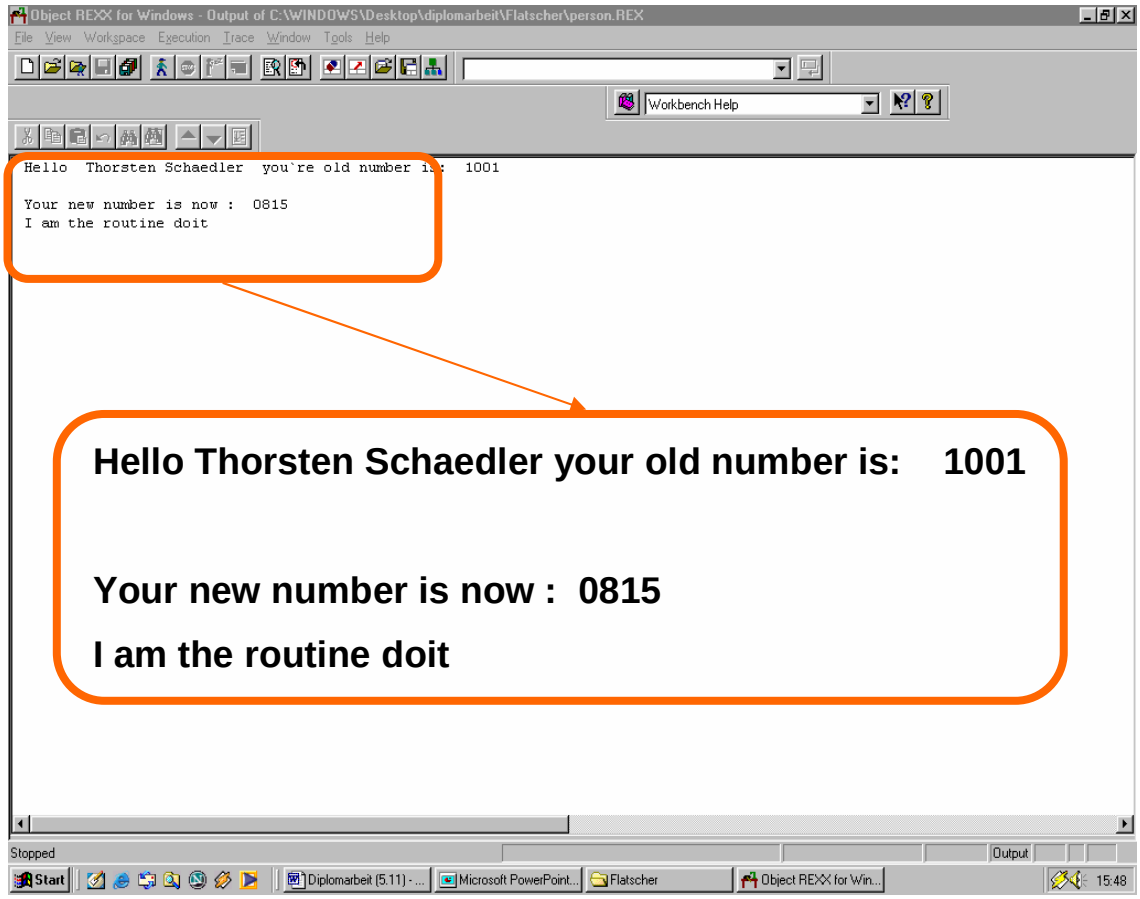

Abbildung 12 : Object Rexx Workbench

4.2.2.2.4 Object Rexx für Windows

Object Rexx stellt Windows Interface Klassen für die Kommunikation mit Microsofts Betriebssystem Windows 95/98/NT/2000/XP zur Verfügung [Vgl. Fla01, Folie 20].

Das folgende Object Rexx Skript (Code 8) zeigt, wie mit Hilfe der Windows Program Manager Klasse ein Ordner mit Verknüpfungen (lnk-Dateien) im Startmenü erzeugt werden kann. Hierzu muss der Object Rexx Programmcode (Code 8) über das `script`-Element in Code 55 eingebettet werden.[96]

Der Anwender hat dabei die Möglichkeit einen beliebigen Namen für den Ordner zu vergeben (vgl. Code 55 im Anhang). Ferner kann der Anwender in einer

[96] Vergleiche hierzu Kapitel 4.5.1 .

HTML-Tabelle (vgl. Code 55 im Anhang) bis zu drei Dateinamen und deren Dateiendung angeben, von denen dann eine Verknüpfung innerhalb des Ordners angelegt wird.

```
::requires "winsystm.cls"   -- the winsystm class definition is
                               implemented into the script code
-- the following code creates a file with .ink files in it by
   pressing the "Add" buttton
::routine addgroup public"
 -- an instance of the Windows Program Manager class is created
pm = .WindowsProgramManager~new
 -- value of input for groupname is saved in the variable n
n=document~getElementById("name")~value
 -- value of input for Ink Number 1 is saved in the variable x
x=document~getElementById("1")~value
 -- value of input for Ink Number 2 is saved in the variable y
y=document~getElementById("2")~value
 -- value of input for Ink Number 3 is saved in the variable z
z=document~getElementById("3")~value
 -- value of input for extension 1 is saved in the variable xe
xe=document~getElementById("e1")~value
 -- value of input for extension 2 is saved in the variable ye
ye=document~getElementById("e2")~value
 -- value of input for extension 3 is saved in the variable ze
ze=document~getElementById("e3")~value
 -- adds a new group into the start menu
pm~AddGroup(n)
pm~AddItem(x , x"."xe, x"."xe ,0)   -- adds the first ink file
                                        into the group
pm~AddItem(y ,  y"."ye,,,,1)        -- adds the second ink file
                                        into the group
pm~AddItem(z , z"."ze ,,,"c:\")     -- adds the third ink file
                                        into the group
pm~ShowGroup(n, "MAX")              -- shows the created group
  -- the following code deletes the new group from the start
     menu by pressing the “Delete” Button
::routine deletegroup public
pm = .WindowsProgramManager~new
n=document~getElementById("name")~value
pm~DeleteGroup(n)  -- deletes the new Group from the start menu
pm~deinstall       -- uninstalls the pm instance
```

Code 8 : Object Rexx für Windows[97]

Wichtig ist in diesem Beispiel, dass die ::REQUIRES-Direktive vor der ::ROUTINE-Direktive im Programmcode angegeben wird, da sonst das Skript nicht auf die

[97] Basiernd auf dem IBM Beispiel „Desctop.rex“ (...\Samples).

benötigte `WINSYSTM`-Klassendefinition und die darin definierten Klassen und Methoden zugreifen kann.

Um Zugriff auf die HTML-Elemente aus zu erlangen, wird die `getElementById()`[98] Methode des `document`-Objekts[99] verwendet.

Abbildung 13 zeigt den Einsatz von Code 8 im entsprechenden HTML-Dokument.[100]

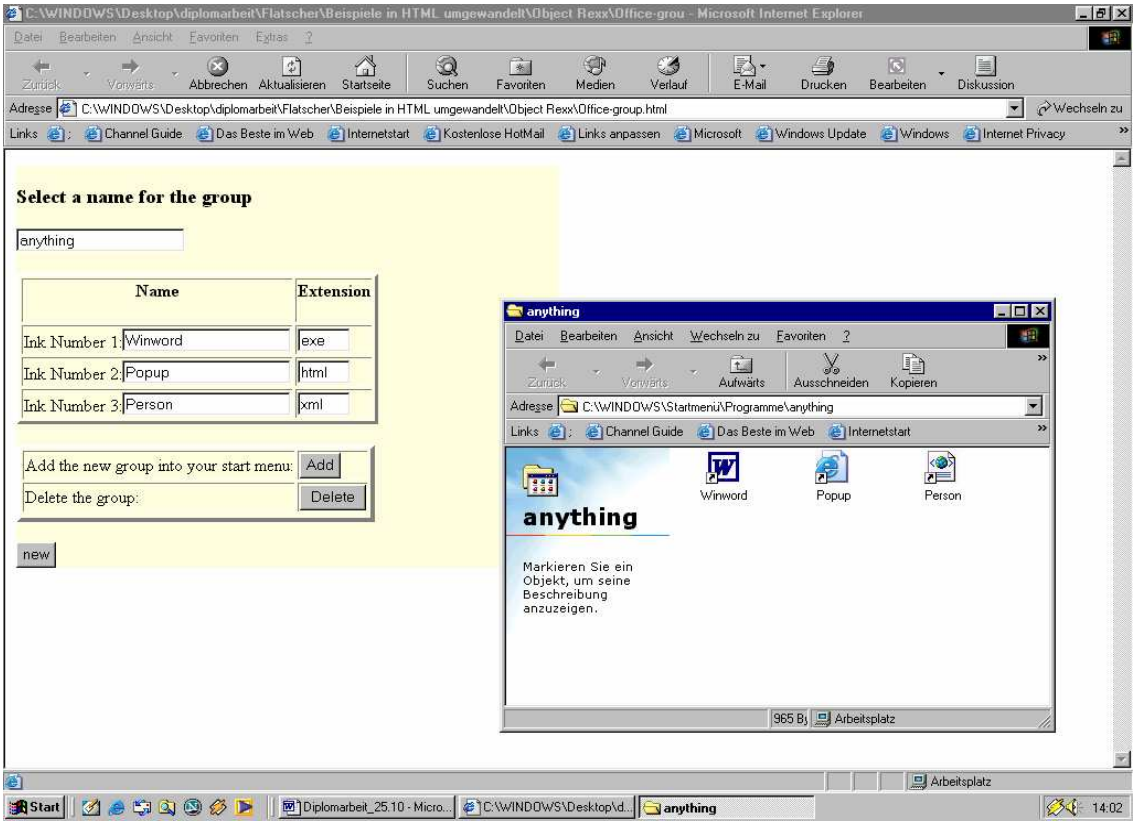

Abbildung 13 : Object Rexx für Windows

4.2.2.2.5 Benutzeroberflächen

Neben den Windows Interface Klassen, stellt Object Rexx in der so genannten OODialog Systemumgebung mehrere Klassen zur Verfügung, mit welchen ver-

[98] S.a. 4.4.3 .
[99] S.a. 4.2.2.1.2.2 .
[100] Der Quelltext dieses Dokuments ist im Anhang als Code 55 dargestellt.

schiedene grafische Benutzeroberflächen erstellt und kontrolliert werden können [IBM03b, S. 19].

Code 9 soll nun zeigen wie eine Instanz einer solchen Klasse innerhalb eines HTML-Dokuments eingesetzt werden kann.

```
::requires "OODPLAIN.CLS"   -- the OOPPlain class definition is
                               implemented into the script code
::routine browser public
    -- creates an instance of the TimedMessage class
dlg = .TimedMessage~new("You are using the " navigator~appName
": Version " navigator~appVersion " as your browser !",
"Browser", 5000)
dlg~execute                  -- the TimedMessageBox is started
```

Code 9 : Object Rexx - Timed Message Box[101]

Wichtig ist in diesem Beispiel, dass die ::REQUIRES-Direktive vor der ::ROUTINE-Direktive im Programmcode angegeben wird, da sonst das Skript nicht auf die benötigte OODPLAIN-Klassendefinition und die darin definierten Klassen und Methoden zugreifen kann.

Der new-Methode müssen drei Argumente übergeben werden. Das erste Argument enthält die Nachricht , das zweite Argument gibt den Titel der Message Box an und das dritte Argument gibt in Form von Millisekunden an wie lange die Box sichtbar sein soll [Vgl. IBM03b, S.21f.].

Abbildung 14 zeigt ein HTML-Dokument[102] in welchem Code 9 immer ausgeführt wird, wenn der „Browser?“-Button gedrückt wird. Die Timed Message Box mit dem Titel „Browser“ erscheint daraufhin für 5000 Millisekunden am Bildschirm und zeigt dem Anwender an, welchen Browser er in welcher Version verwendet.

Um die Browser Version und den Namen zu bestimmen, wurde das navigator-Objekt mit dessen Eigenschaften appName und appVersion verwendet.

[101] Nach [IBM03b, S.21f] .
[102] Siehe Code 56 im Anhang .

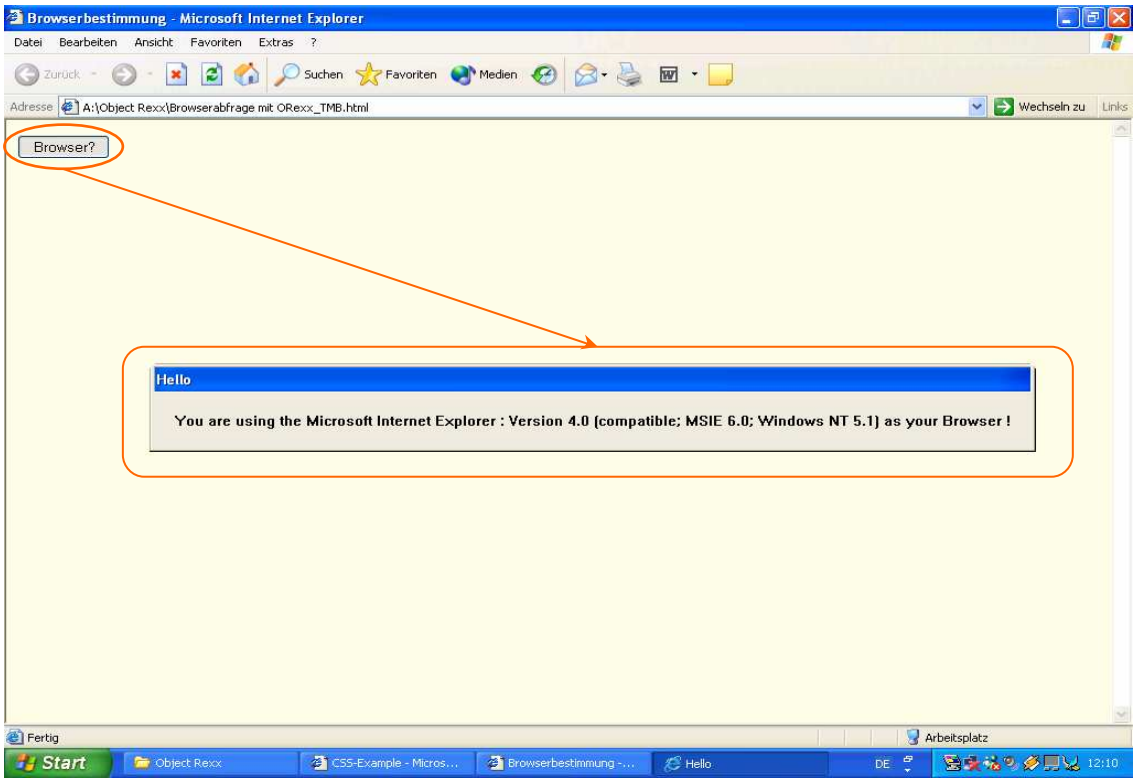

Abbildung 14 : Timed Message Box in einem HTML-Dokument

4.2.2.2.6 OleObject

Ein OleObject ist eine Instanz der OLEObject-Klasse. Diese Klasse ist die Schnittstelle von Object Rexx für OLE[103]. Sie ermöglicht OLE Automation[104], wodurch eine Applikation Zugriff auf ein Objekt einer anderen Applikation erlangen kann [Vgl. IBM03a, S.302].

In Kapitel 4.5 wird dies anhand zweier Beispiele näher erläutert.

In früheren Versionen war diese Klasse jedoch nicht automatisch Bestandteil von Object Rexx. Im Programmcode wurde daher die Direktive ::`REQUIRES` "`OREXXOLE.CLS`", welche die Klassendefinitionen enthält, benötigt. In der neuesten Version (2.1.2) wird diese jedoch automatisch geladen und braucht im Programmcode nicht mehr angegeben werden [Vgl. HelF03, S.94].

---

[103] S.a. 4.3 .
[104] S.a. 4.3.2.

Ein OleObject kann über seine `CLSID` oder seine `ProgID` (Programm ID) instanziert werden [Vgl. IBM03a, S.302].

Eine `CLSID` kann beispielsweise mit dem Object Rexx basierten und frei erhältlichen *RGF_Oleinfo.hta* -Tool[105] von Prof. Rony G. Flatscher ermittelt werden.

4.2.2.3 JScript

JScript ist die Microsoft-Implementierung der ECMA-262 Sprachspezifikation zusätzlich einiger Erweiterungen, welche die Funktionen des Microsoft Internet Explorer nutzen [Vgl. MSJS03].

„JScript ist eine interpretierte objektbasierte Skriptsprache“ [MSJS03]. Auch wenn über einen geringeren Funktionsumfang als größere objektorientierte Sprachen wie C++, Java oder Object Rexx verfügt, ist JScript für den Einsatz innerhalb eines HTML-Dokuments mehr als nur ausreichend leistungsstark [Vgl. MSJS03].

JScript unterliegt jedoch auch einigen Restriktionen. So können keine selbständigen Anwendungen in ihr geschrieben werden, „und sie bietet nur begrenzte Funktionen zum Lesen und Schreiben von Dateien. Darüber hinaus können JScript Dateien nur bei vorhandenem Interpreter ausgeführt werden, entweder über einen Web Server oder einen Web Browser.“ [MSJS03] .

4.2.2.4 VBScript

VBScript ist ein Mitglied der Visual Basic Programmiersprachen, und ermöglicht das Erstellen von interaktiven Skripten für das WWW [Vgl. MSVBS03].

„VBScript verwendet ActiveX™ Scripting zur Kommunikation mit Host-Anwendungen. ActiveX™ Scripting erspart Browsern und anderen Host-Anwendungen die Implementierung von speziellem Integrations-Code für jede Scripting-Komponente. ActiveX™ Scripting ermöglicht einem Host die Kompilierung von Skripten, die Ermittlung und den Aufruf von Einsprungpunkten und

[105] http://wi.wu-wien.ac.at/rgf/ rexx/orx13/tmp/ .

die Verwaltung des Namensraums, der dem Entwickler zur Verfügung steht. Mit ActiveX™ Scripting können Compiler-Hersteller Standard-Laufzeitmodule der jeweiligen Sprache für Skripten erstellen. Microsoft wird Laufzeitunterstützung für VBScript und Visual Basic anbieten. Microsoft arbeitet mit verschiedenen Internet-Gruppen an der Definition des ActiveX™ Scripting-Standards, um die Austauschbarkeit von Scripting-Modulen zu fördern. ActiveX™ Scripting wird im Microsoft Internet Explorer und im Microsoft Internet Information Server verwendet.“ [MSVBS03].

4.2.3 BWS - BSFWebScripting

Wie zu Beginn des Kapitels 4.2.2.2 bereits erwähnt, funktioniert ein Object Rexx Skript bzw. jede ActiveX-Scripting Sprache wie JScript[106] und VBScript[107], standardmäßig nur im MS Internet Explorer (WSH), während das Skripten von HTML-Dokumenten mit Java/ECMAScript in jedem beliebigen Web-Browser möglich ist.

Um nun diesen Lock-In Effekt von ActiveX-Scripting Sprachen umgehen zu können, wurde vom Augsburger Studenten Tobias Specht im Rahmen eines Seminars bei Herrn Prof. Dr. Rony G. Flatscher im WS 2002/03 eine Lösung präsentiert. Diese Lösung nennt sich *BSFWebScripting (BWS)* und verwendet einen (beliebigen) Web Browser als Anwendungsplattform. Derzeit wird diese Lösung von Tobias Specht noch im Rahmen seiner Diplomarbeit weiter ausgearbeitet.

4.2.3.1 Bestandteile

Um ein lauffähiges BWS-Dokument zu erstellen, bedarf es innerhalb eines normalen HTML-Dokuments einiger zusätzlicher Bestandteile, welche im Folgenden kurz vorgestellt werden.

---

[106] S.a. 4.2.2.3 .
[107] S.a. 4.2.2.4 .

Abbildung 15 gibt vorab einen Überblick über die allgemeine Struktur eines solchen modifizierten HTML-Dokumentes, welches auch als BWS-Dokument bezeichnet wird.

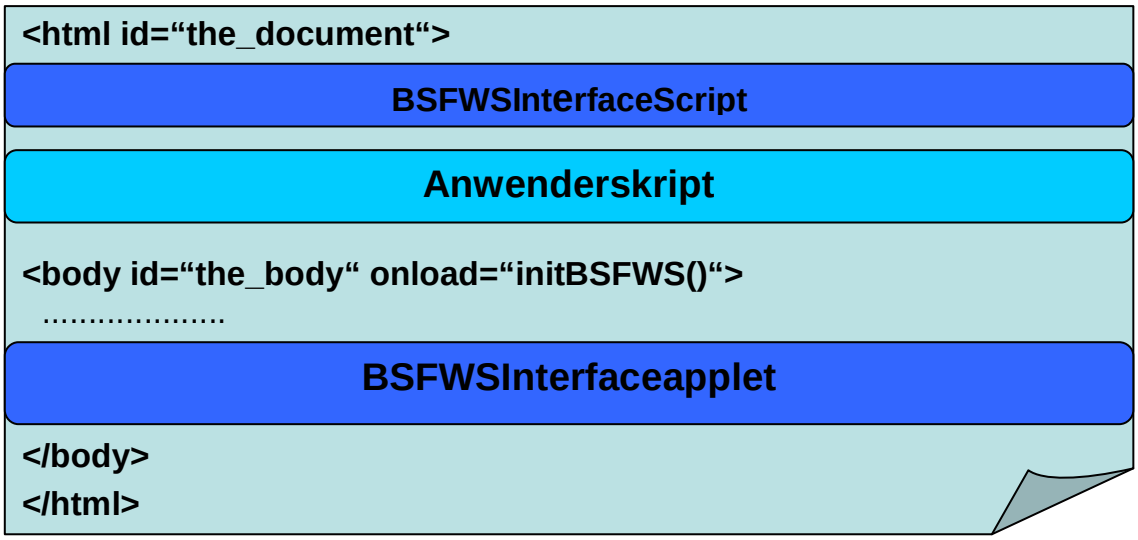

Abbildung 15 : Struktur eines BWS-Dokumentes

In einem HTML-Dokument müssen demnach folgende Modifikationen vorgenommen werden [Vgl. Fla03f, S.3].

- Das einleitende `<HTML>` Tag muss das `id`-Attribut mit der Wertzuweisung *`the_document`* beinhalten.
- Das einleitende `<Body>` Tag muss zum Einen das Attrtibut `id` als auch den Event-Handler `onload` enthalten. Dem `id` Attribut muss dabei der Wert *`the_body`* zugewiesen werden. Beim Öffnen des Dokuments wird mit Hilfe des `onload`-Attributs die JavaScript Funktion `initBSFWS`() aufgerufen.
- Der Aufruf eines Skriptes erfolgt über einen Verweis.

  Syntax : `<a href="lang:Skriptname">`

4.2.3.1.1 BSFWSInterfaceScript

Das `BSFWInterfaceScript` ist ein JavaScript Programm, welches mit Hilfe eines `script`-Elements als externe Datei in ein HTML-Dokument eingebettet

wird und welches den `<body>` eines HTML-Dokuments so modifiziert, dass ein eingebettetes BWS-Skript an das Bean Scripting Framework (BSF) weitergeleitet werden kann. Es besteht dabei aus der zentralen Funktion `InitBWS()` welche wiederum andere benötigte Funktionen aufruft [Vgl. SpeT03a, S.13].

Syntax : `<script type="text/JavaScript" id="BSFWSInterfaceScript" src="BSFWSInterfaceScript.js"></script>`

4.2.3.1.2 BSFWSInterfaceApplet

Das `BSFWSInterfaceApplet` ist ein Java Programm, welches mittels des `object-` oder `applet`-Elements in das HTML-Dokument eingebettet wird. Es ist der Teil von BSFWebScripting, welcher den von JavaScript erhaltenen Skriptcode an die entsprechende Script-Engine weiterleitet und diese mit benötigten DOM-Schnittstellen versorgt [Vgl. SpeT03a, S.12].

Das Einbetten dieses Applets erfolgt dabei im Internet Explorer und Mozilla auf verschiedene Weise [Vgl. SpeT03a, S.12] .

4.2.3.1.3 JSNode

JSNode ist eine Java Klasse welche den Zugang zu den wichtigsten DOM-Funktionen erleichtert [Vgl. Fla03b, S.4].

„JSNode ist die Java Implementierung eines DOM `node`." Da von Java aus der Zugriff über LiveConnect auf das DOM sehr umständlich ist, sind über JSNode alle Methoden von `node` direkt in Java verfügbar [Vgl. SpeT03c].

4.2.3.2 Basistechnologien

BSFWebScripting basiert auf der Implementierung zweier Schnittstellentechnologien. Zum Einen LiveConnect und zum Zweiten dem Bean Scripting Framework (BSF) [Vgl. SpeT03a, S.6].

4.2.3.2.1 LiveConnect[108]

LiveConnect ist eine Technologie, welche die Kommunikation zwischen Java, JavaScript und Browser Plug-Ins ermöglicht. Es wurde 1996 von Netscape entwickelt und erstmals in den Netscape Navigator 3.0 eingebaut. Der Opera Browser unterstützt LiveConnect seit der Version 6.0 [Vgl. SpeT03a, S.8].

Sun übernahm es und baut es seit Java 1.3 in seine Java Runtime Environment (JRE) ein [Vgl. Fla03f, S.4].

Im Rahmen von BSFWebScripting wird LiveConnect zur Kommunikation zwischen dem Interface Applet (`BSFWSInterfaceApplet.java`) und dem DOM[109] verwendet [Vgl. SpeT03a, S.8].

4.2.3.2.2 BSF - Bean Scripting Framework

Das BSF ist seit Anfang 2001 ein IBM Developer Works Projekt und kann derzeit in Version 2.2[110] kostenlos aus dem Internet herunter geladen werden. 2002 wurde es dem Jakarta Projekt der Apache Organisation übergeben und wurde dort unter BSF 2.3 veröffentlicht [Vgl. Fla03g, S.3].

Das BSF ermöglicht die Kommunikation zwischen Java und unterstützten Skriptsprachen. Über das BSF stehen vier verschiedene Methoden zur Verfügung, mit denen ein Skript von Java aus aufgerufen werden kann. Diese vier Methoden sind `exec`, `eval`, `apply` und `call` [Vgl. Fla03h,S.2f].

4.2.3.2.3 BSF4Rexx

Das Bean Scripting Framework for Rexx (BSF4Rexx) ermöglicht, dass ein Rexx - oder Object Rexx Skript in Verbindung mit dem Bean Scripting Framework verwendet werden kann. Dabei gibt es zwei Versionen dieses BSF4Rexx. Die ältere Essener Version, welche 2001 entstand und die Augsburger Version aus dem Jahre 2003. Die Augsburger Version ist voll kompatibel zu der Esse-

---

[108] S.a. 4.2.4.6 .
[109] S.a. 4.4.3 .
[110] http://oss.software.ibm.com/developerworks/ projects/bsf .

ner Version und verfügt über einige Neuerungen bzw. Erweiterungen, zu welchen auch die Möglichkeit gehört, Java von einem Rexx Programm aus zu starten. Auf diese Weise wird Java zum weltweit größten externen Funktionspacket für Rexx bzw. Object Rexx [Vgl. Fla03g, S.2].

Die Abbildung 16 und 17 veranschaulichen den Zusammenhang zwischen den einzelnen Technologien und die Unterschiede der beiden Versionen.

Abbildung 16 zeigt zunächst die Architektur der Essener Version.

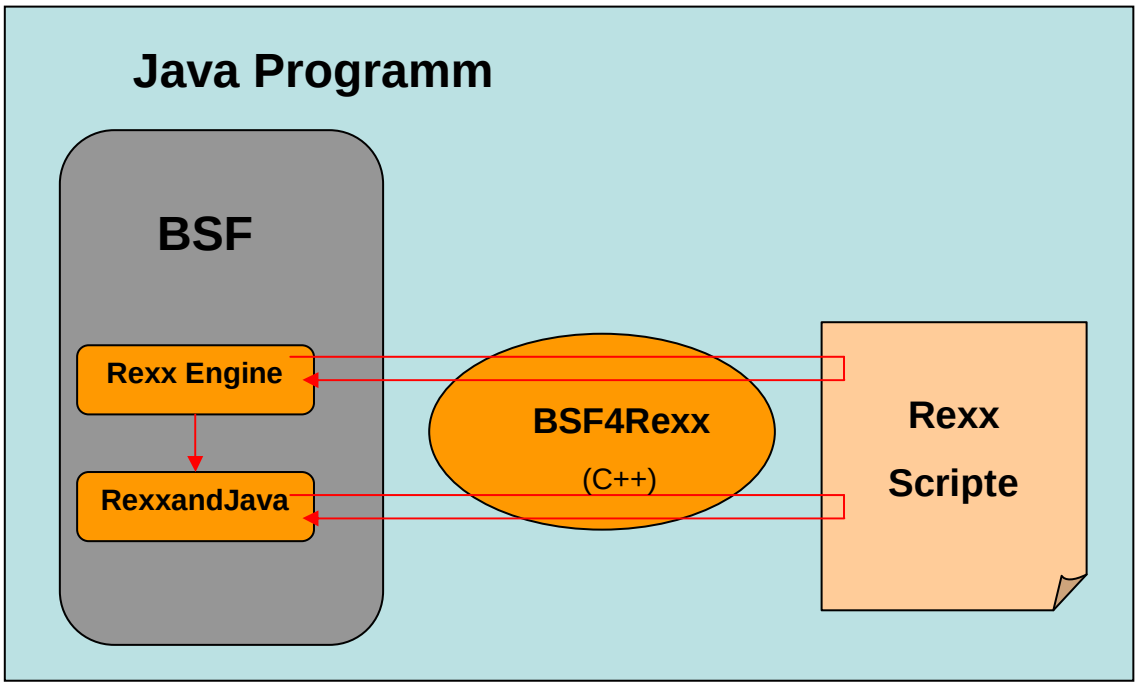

Abbildung 16 : BSF Architektur und die Essener Version von BSF4Rexx[111]

Bei der Essener Version erzeugt zuerst das Java Programm eine Instanz der Java `BSFManager` Klasse, welche das Laden und Ausführen von Programmen ermöglicht die in einer von BSF unterstützten Sprache geschrieben wurden. Das Java Programm verwendet anschließend das `BSFManager` Objekt um die Rexx Engine, welche als das Java Programm `RexxEngine.java` implementiert ist , zu laden und initialisiert die Schnittstelle zu Rexx (`BSF4Rexx`). Ab diesem Zeitpunkt ist es der Rexx Engine möglich, Rexx Programme aufzurufen. Hierzu

[111] Nach [Fla03g, S.4]

werden der Rexx Code und die Rexx Argumente dem Rexx Interpreter übergeben, welcher in BSF4Rexx aufgerufen wird. Zuvor wird jedoch die externe Rexx Funktion BSF() über den Rexx Interpreter registriert. Diese Funktion erlaubt den Rexx Programmen den Zugang zu Java. Rexx Skripte welche Zugang zu Java erhalten, können eine Reihe an Funktionen verwenden, welche über das Java Programm RexxAndJava zur Verfügung stehen [Vgl. Fla03g, S.3f].

Abbildung 17 zeigt nun die Architektur der Augsburger Version von BSF4Rexx.

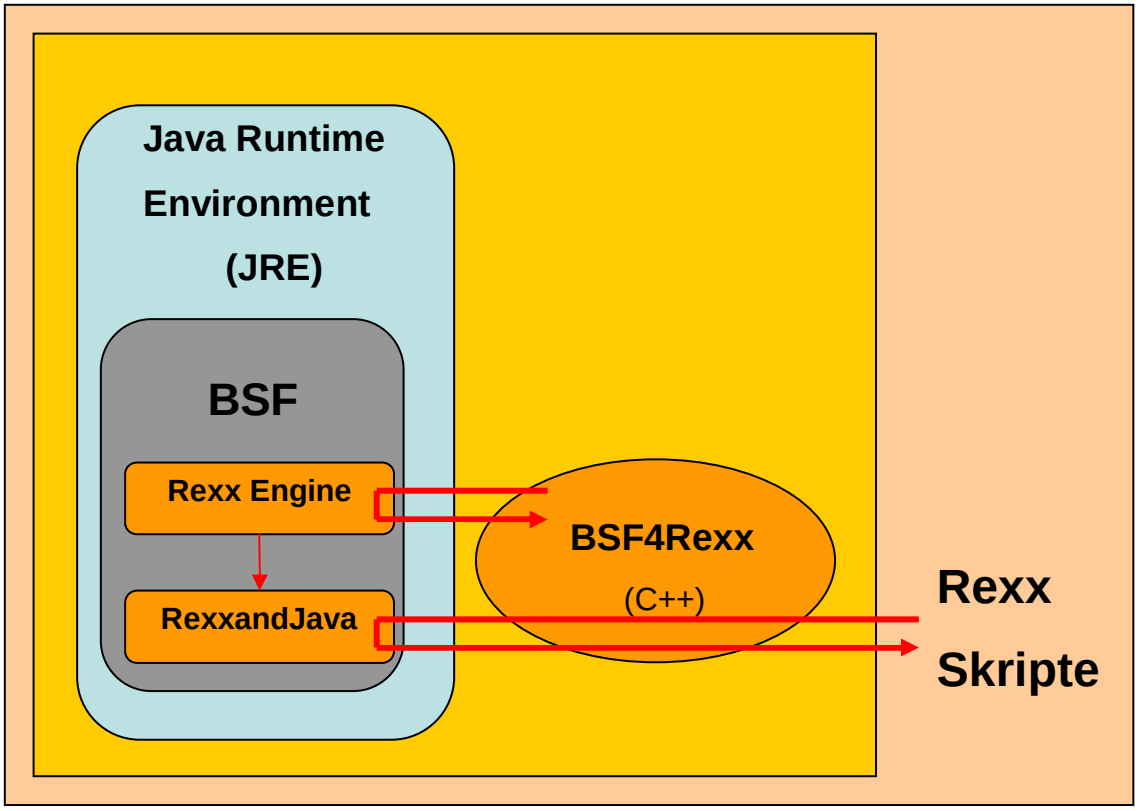

Abbildung 17 : BSF und die Augsburger Version von BSF4Rexx [112]

4.2.3.3 Ablaufschema [113]

Die BWS Funktionalität ist in die zwei Teile *Initialisierung* und *Skriptaufruf* unterteilt, welche im Folgenden graphisch veranschaulicht sind.

[112] Nach [Fla03g, S.7] .
[113] Nach [SpeT03b] .

4.2.3.3.1 Initialisierung

Bei der Initialisierung wird vom Browser automatisch zuerst das eingebettete `BSFWSInterfaceApplet` geladen und dessen Startup-Methoden aufgerufen und ausgeführt. Anschließend wird das `BSFWSInterfaceScript` ausgeführt.

Abbildung 18 gibt einen Überblick über den Ablauf bei der Initialisierung.

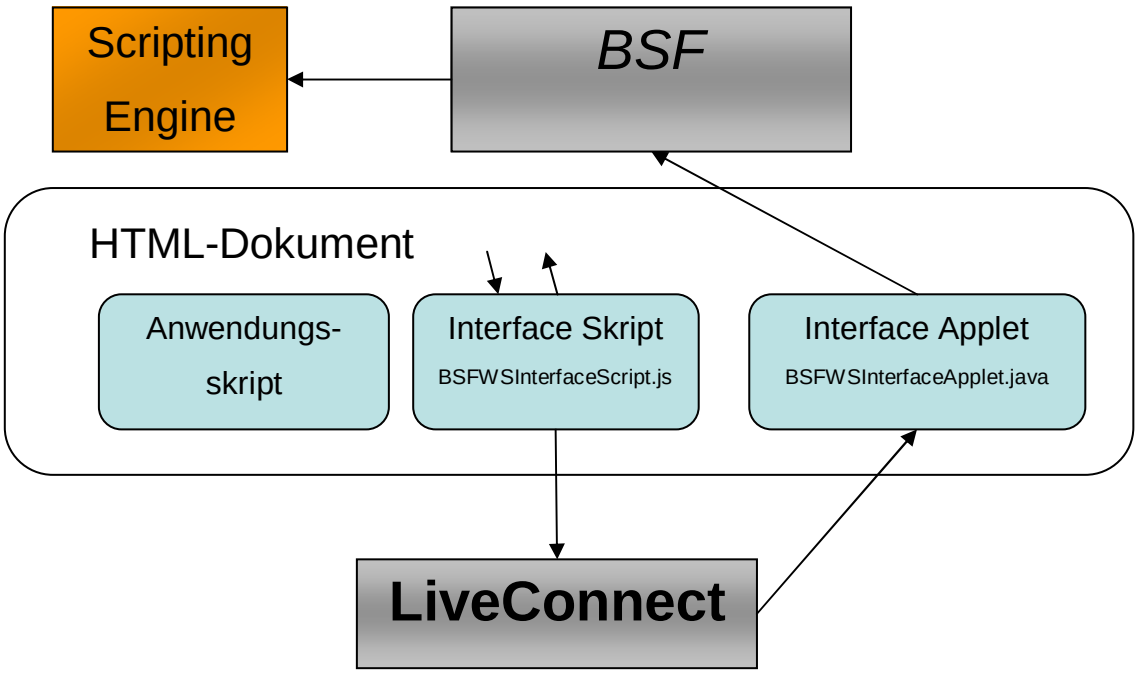

Abbildung 18 : BWS - Initialisierung[114]

4.2.3.3.2 Skriptaufruf

Bei einem entsprechenden Skriptaufruf wird über LiveConnect das Interface Applet angesprochen, welches wiederum über das BSF die entsprechende Script Engine aufruft, in welcher dann das Skript verarbeitet wird.

Abbildung 19 gibt abschließend einen Überblick über den Ablauf bei einem Skriptaufruf.

[114] Nach [SpeT03b] .

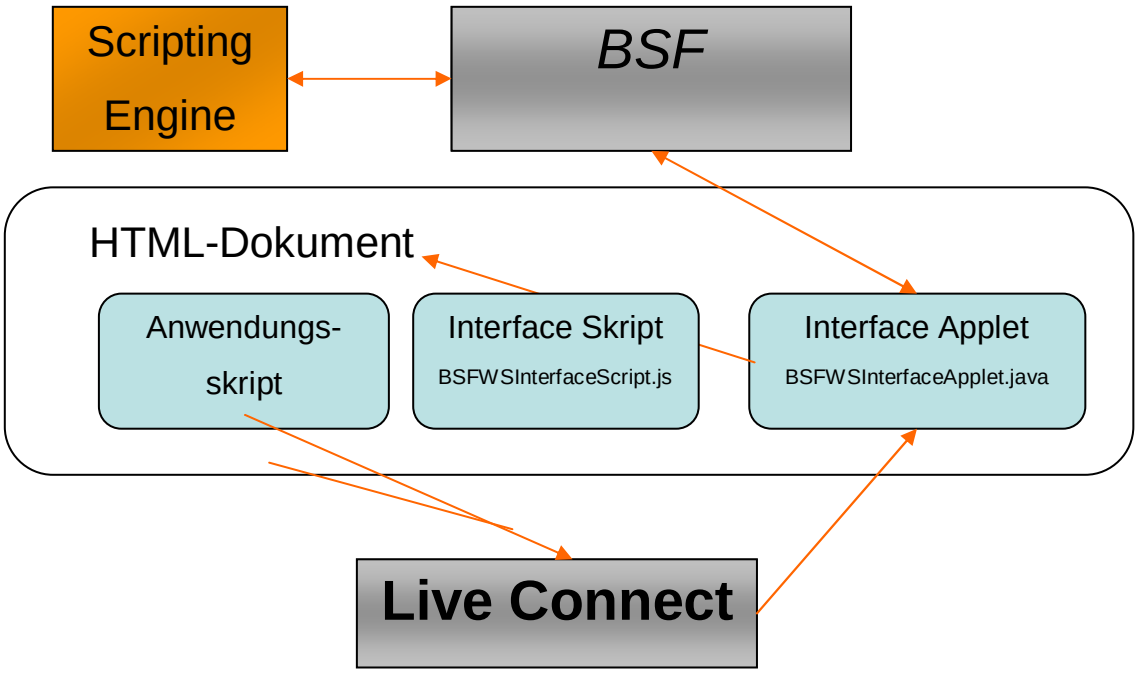

Abbildung 19 : BWS – Skriptaufruf[115]

4.2.4 Java

Neben den in Kapitel 4.2.2 angesprochenen Skriptsprachen gibt es auch noch eine weitere Möglichkeit, Leben in ein statisches HTML-Dokument zu bekommen. Diese Möglichkeit heißt Java.

Das Hauptaugenmerk dieses Abschnitts besteht darin, zu zeigen, wie diese Programmiersprache zum Erstellen einer Benutzeroberfläche (Applet) für HTML-Dokumente (Web Seiten) eingesetzt werden kann.

Java wurde 1995 als neue Programmiersprache für das Web vorgestellt, und löste eine Euphorie aus. Dabei bietet Java eigentlich nichts Neues. Vielmehr verbindet es die Vorteile unterschiedlicher Programmiersprachen wie C, C++ und Smalltalk [Vgl. KnSW98, S.12].

Java ist eine vollständige und klassenbasierte Programmiersprache, mit welcher herkömmliche Applikationen als auch Java-Applets erstellt werden kön-

[115] Nach [SpeT03b] .

nen. Grundbestandteil von Java sind dabei die Klassen bzw. Objekte und deren Attribute und Methoden.

Bei *Java-Applikationen* handelt es sich um vollständige, eigenständige Programme, welche über einen Interpreter oder einen Just-in-time Compiler auf einer Systemplattform ausgeführt werden [Vgl. Sche00, S.239].

Java Applets hingegen sind Programme, die mit dem HTML-Code weitergegeben werden und so in Web-Seiten eingebunden werden können. Um Java-Applets zu starten, wird ein javafähiger Browser oder ein spezieller Appletviewer benötigt [Vgl. Sche00, S.239].

Die Unterscheidung von einem Applet und einer Applikation ist innerhalb des Java Bytecode möglich [Vgl. KnSW98, S.14] .

4.2.4.1 Das JDK

Das Java Development Kit (JDK 1.2) enthält alle notwendigen Tools, mit denen Java-Applets erstellt werden können. Um Java-Applets oder Java-Applikationen erstellen zu können, sind im JDK einige Entwicklungswerkzeuge beigefügt worden. So sind unter anderem der Appletviewer (appletviewer) , der Java Interpreter (java), der Java Debugger (jdb) und der Java Compiler (javac) darin enthalten [Vgl. KnSW98, S.21].

Mit der Java Runtime Environment (JRE) steht ein Plug-In für den Browser zur Verfügung.

4.2.4.2 Klassen, Objekte, Eigenschaften und Methoden

Da es sich bei Java um eine klassenbasierte Programmiersprache handelt werden auch hier Klassen bzw. Objekte mit ihren Eigenschaften und Methoden verwendet. Ein Objekt ist eine Instanz einer Klasse.

4.2.4.3 Applets

Java Applets können aus den verschiedensten Gründen entwickelt werden.

Zum Programmieren des Quelltextes eines solchen Applets, wird lediglich ein ASCII-Editor wie beispielsweise Notepad, der Windows beiliegt, benötigt.

In Kapitel 3.5 wurde gezeigt wie mit Hilfe von HTML eine Benutzeroberfläche erstellt werden kann. Derartige Oberflächen kann man auch mit Java entwickeln. Im Folgenden werden nun die wichtigsten Aspekte zur Umsetzung einer solchen Oberfläche mittels Java dargestellt.

4.2.4.3.1 Erstellen einer Benutzeroberfläche

Für die Erstellung solcher Benutzeroberflächen, Grafiken und Bildern, stellt Java das „Abstract Window Toolkit“ (AWT) zur Verfügung. Mit diesem AWT können Steuerelemente in ein Programm aufgenommen werden.

Tabelle 8 stellt einige dieser Elemente kurz vor.

Steuerelement	Beschreibung	Konstruktor
Frame	Fenster mit Rahmen.	New Frame(String)
Canvas	Fläche für graphische Ein- und Ausgabe.	New Canvas()
Label	Textfeld	New Label(String)
Button	Schaltfläche	New Button(String)
Checkbox	Kontrollkästchen mit Beschriftung.	new Checkbox (String, boolean)
Choice	Auswahlliste mit einfacher Auswahl.	New Choice()
List	Auswahlliste mit mehrfacher Auswahl.	New List(int)
TextField	Einzeiliges Textfeld für Tastatureingaben.	new Textfield(String, columns)
TextArea	Mehreiliges Textfeld für Tastatureingaben.	new Textarea(String, rows, columns)
Dialog	Kleines Fenster mit Text. Beispielsweise verwendbar für Feh-	new Dialog(Frame, String, true)

	lermeldungen.	
FileDialog	Menü zum Auswählen einer Datei.	new FileDialog(Frame, String, FileDialo.LOAD) new FileDialog(Frame, String, FileDialo.SAVE)
MenuBar	Menüleiste	New MenuBar()
Menu	Hauptmenü in der Menüleiste.	New Menu(String)
MenuItem	Untermenü eines Hauptmenüs.	New MenuItem(String)
MenuShortcut	Tastenkombination für Menü.	new MenuShortcut(key-code, shift)
Checkbox-MenuItem	Untermenü mit Kontrollkästchen.	new CheckboxMenuItem (String, boolean)
PopupMenu	Popup-Menü	New PopupMenu(String)
ScrollBar	Horizontale oder vertikale Bildlau-fleiste.	new Scollbar(orientation, value, size, min, max)
ScrollPane	Rahmen mit Scrollbars für ein zu großes Canvas.	New ScrolPane()

Tabelle 8 : AWT-Steuerelemente[116]

Um das AWT innerhalb eines Java Programms nutzen zu können, werden über die `import`-Anweisung die Packages `java.awt` und `java.awt.event` aus dem AWT in den Programmcode eingebunden. Eine solche `import`-Anweisung steht dabei am Anfang des Quellcodes [Vgl. Sche00, S.243].

Um das Aussehen solcher Steuerelement nach dem Erstellen verändern zu können, benötigt man eine oder mehrere Methoden. Allen Steuerelementen stehen dabei die in Tabelle 9 angeführten Methoden zur Verfügung.

Methoden	Beschreibung
setForeground	Schriftfarbe festlegen
setBackground	Hintergrundfarbe festlegen

[116] Nach [Sche00, S.250-252] .

setSize	Größe festlegen
getSize	Größe erfahren
addXXXListener	Einen Listener an das Objekt anhängen
setVisible	Sichtbar bzw. unsichtbar machen
paint, update, repaint	Bei Grafiken für das Zeichnen, Aktualisieren und Neuzeichnen

Tabelle 9 : Methoden für Steuerelemente[117]

Darüber hinaus haben die einzelnen Steuerelemente zum Setzten der Eigenschaften ihre eigenen Methoden wie beispielsweise `setText`(), `getText`() oder `getFile`() [Vgl. Sche00, S.253].

4.2.4.3.2 Das Reagieren auf Ereignisse

Um auf Aktionen eines Anwenders reagieren zu können, steht in HTML mit den Event-Handlern[118] ein Universalattribut zur Verfügung. In Java ist dies jedoch ein wenig anders.

In einem Java Programm muss an ein Steuerelement ein so genannter „Listener“ angehängt werden. Dieser überwacht permanent ob ein Ereignis ausgelöst wird. Sollte ein Ereignis ausgelöst werden, ruft er die entsprechende Funktion auf, und führt die darin enthaltenen Befehle aus. Hierbei stehen verschiedene Listener zur Verfügung, welche jeweils bestimmte Funktionen beinhalten [Vgl. Sche00, S.257].

4.2.4.3.3 Kompilieren von Java Programmen

Nach der Installation des JDK steht dem System ein Java Compiler zur Verfügung, welcher den Quelltext (`[Dateiname].java`) in plattformunabhängigen Bytecode umwandelt (`[Dateiname].class`).

---

[117] Nach [Sche00, S. 253] .
[118] S.a. 3.6.1 .

Um den Compiler auf MS-DOS Basis verwenden zu können, muss folgendes unternommen werden :

1.) Eine Verknüpfung mit der Datei „Command.com“, welche sich auf dem Root- Laufwerk befindet, muss hergestellt werden.
2.) Mit der rechten Maustaste auf die Verknüpfung klicken und anschließend Eigenschaften auswählen.
3.) In der Kategorie „Programm“ beim Textfeld Arbeitsverzeichnis den Namen des Verzeichnisses „bin“ in welches das JDK installiert wurde eingeben.
4.) Um einen Programmcode zu kompilieren muss anschließend in die Befehlszeile der gestarteten Verknüpfung folgender Befehl eingegeben werden : `javac [Dateiname].java`

Daraufhin wird vom Compiler im selben Verzeichnis eine .class Datei erstellt, welche zum Einbetten in ein HTML-Dokument verwendet werden kann.

4.2.4.4 Applets in ein HTML-Dokument einbinden

Um Java Applets einzubinden stellt HTML zwei verschiedene Elemente zur Verfügung. Die in Java geschriebenen Java Applets müssen jedoch vorab mit dem Java Compiler zu einem ausführbaren Programm (.class) zusammengebunden werden.

Zum Einen kann das ältere `applet`-Element[119] verwendet werden. Innerhalb des einleitenden `<applet ...>` Tag werden dem Browser mit den Attributen `code` und `codebase` der Name und das Verzeichnis des gewünschten Applets mitgeteilt. Das Attribut `codebase` ist dabei nur anzugeben, falls sich das Applet in einem anderen Verzeichnis oder auf einem anderen Server als das HTML-Dokument befindet [Vgl. MüNe01, S.271f].

[119] Das applet-Element ist seit der HTML Spezifikation 4.0.1 vom W3C missbilligt. An seiner Stelle sollte das object-Element verwendet werden.

Seit HTML 4.0 gibt es mit dem `object`-Element eine weitere Möglichkeit ein Applet in ein HTML-Dokument einzubinden. Mit der Angabe `classid="java:[Applet]"` wird dem Browser mitgeteilt welches Applet er in die Webseite integrieren soll. Befindet sich dieses auszuführende Applet in einem anderen Verzeichnis als das referenzierte HTML-Dokument, muss auch hier mit Hilfe des Attributs `codebase` noch das genaue Verzeichnis angegeben werden [Vgl. MüNe01, S.257].

Code 10 zeigt nun den Quelltext eines Java Applets, welches anschließend in ein HTML-Dokument (Code 11) eingebettet wird. Dieses Applet ermöglicht einem Anwender die Eingabe seines Namens. Daraufhin wird dieser mit seinem Namen begrüßt. Dieses Beispiel soll lediglich den Einsatz der in Tabelle 8 dargestellten AWT-Steuerelemente als auch deren Methoden verdeutlichen.

```
-- embeds the file java.awt needed for creating control elements
Import java.awt.* ;
-- embeds the file java.applet needed for nearly every applet
import java.applet.* ;
-- embeds the file java.event needed for eventhandling
import java.awt.event.* ;

public class DA extends Applet implements ActionListener
{
 -- declaration of all variables used in the applet
    private TextField txtInput;
    private Label lblText;
    private TextArea txtOutput;
    private Button cmdOK;
    private String text;
    Color BG1 = Color.white;
 -- function init which is invoked by loading the web side
 public void init( )
   {
    -- the applet has a white background color
    this.setBackground(BG1);
    -- puts the text in the brackets into the applet
    lblText = new Label("Please type anything into the
    Textfield below and press the OK Button");
    add(lblText);
    -- the text has a yellow background
    lblText.setBackground(Color.yellow);
    -- puts a textfield into the applet
    txtInput = new TextField("",30);
    add(txtInput);
    -- the letters of the textfield have orange color
```

```
    txtInput.setForeground(Color.orange);
    -- adds a button for eventhandling into the applet
    cmdOK = new Button("OK");
    cmdOK.addActionListener(this);
    add(cmdOK);
    -- adds a textarea to the applet
    txtOutput = new TextArea("",5,50);
    add(txtOutput);
    -- the textarea has a white background
    txtOutput.setBackground(BG1);
    -- the predefined Layout Manager FlowLayout is used to
      control the arrangement and size of the control elements
    setLayout (new FlowLayout() );
    -- Visible
    setVisible(true);
   }
-- function for eventhandling
 public void actionPerformed (ActionEvent e)
  {
-- the variable text gets the text from the textfield
  "txtInput"
  text = txtInput.getText( );
 -- Output which is shown in the Textarea
  txtOutput.setText("Hello "+ text ".My name is Thorsten. How
                    do you do?");
  }
}
```

Code 10 : Quelltext des Java Applets (DA.java)

Code 11 zeigt den Quellcode eines HTML-Dokuments, in welches das Java Applet aus Code 10 über das `object`-Element eingebettet wird.

```
<html>
<head>
<title>Hallo</title>
<meta name="author" content="Thorsten Sch&auml;dler">
</head>
<body bgcolor="lightyellow">
</script>
<h1 style="margin-top:50pt">This is a demonstration of using a
Java-Applet in a HTML document</h1><br>
 -- the Java applet DA is embedded into the html file by using
    the object-element
<object classid="java:DA.class" width="500" height="200"
style="margin-left:200pt"></object>
</body></html>
```

Code 11 : Java Applet in einem HTML-Dokument

Abbildung 20 zeigt das in das HTML-Dokument (Code 11) eingebettete Java Applet aus Code 10 unter Verwendung des MS Internet Explorers.

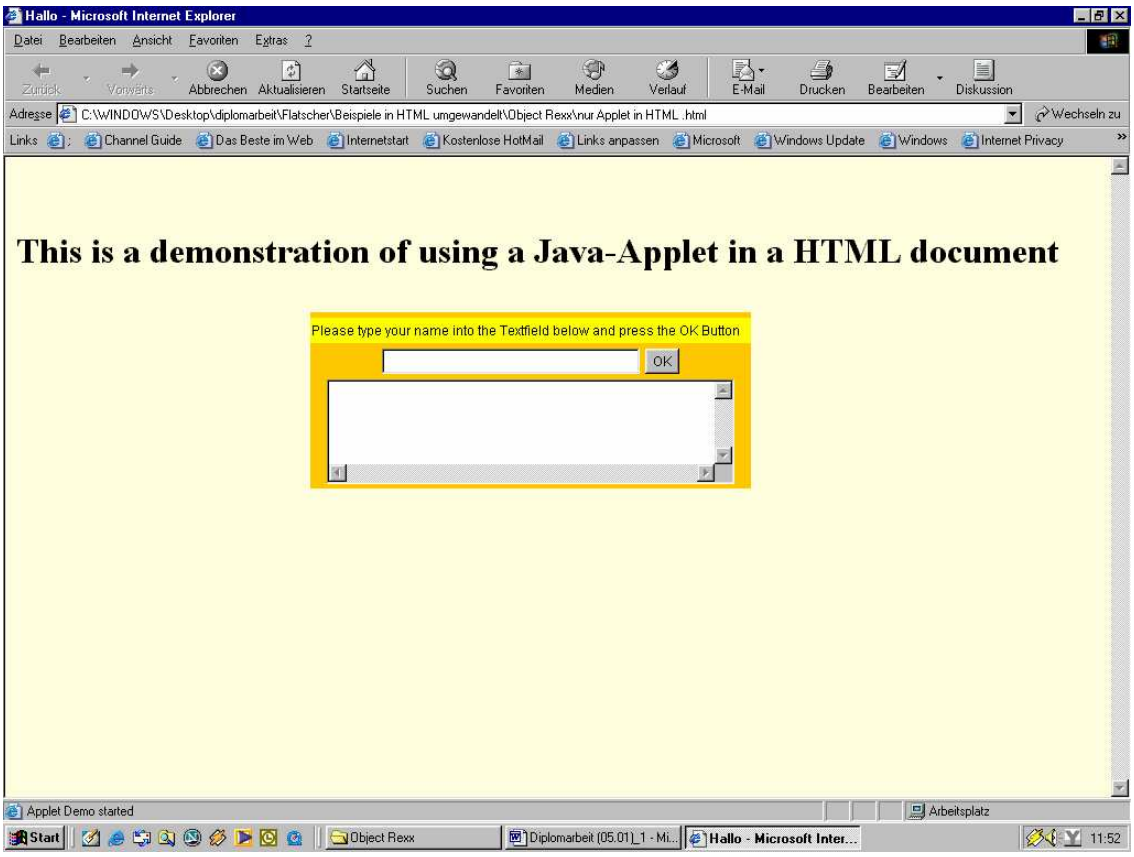

Abbildung 20 : Java-Applet in einem HTML-Dokument

4.2.4.5 Das applets-Objekt

Das `applets`-Objekt[120] bietet Zugriff auf Java-Applets, die in einer HTML-Dokument definiert sind. Mit der Eigenschaft `length` kann die Anzahl der Java Applets in einem Dokument abgefragt werden. Dabei werden alle `applet`-Elemente, die sich in dem HTML-Dokument befinden gezählt und in dieser Variablen `length` gespeichert [Vgl. MüNe01, S.583].

Ferner werden allerdings auch alle `object`-Elemente in dieser Variablen gespeichert .

[120] S.a. 4.2.2.1.2.2 .

Das `applets`-Objekt kann somit auch in Code 11 verwendet werden. Hierzu muss ein Funktionsaufruf definiert werden. Im Rahmen von Code 11 wurde hierfür mittels `<input type="button" value= "Number" onclick="call num">` ein Button[121] definiert, welcher beim Anklicken die in Code 12 dargestellte Object Rexx Routine **num** aufruft. Diese Routine aus Code 12 muss ebenfalls in Code 11 eingebettet werden.[122] Die Routine verwendet daraufhin die `alert`-Methode um dem Anwender über ein Meldungsfenster die Anzahl der Applets bzw. Objekte mitzuteilen.

```
::routine num public
   -- using the applets object for getting knowledge about the
      number of applets in the HTML file
alert("There is (are)" document~applets~length "applet(s) in
this web side")
```

Code 12 : Das applets-Objekt

Da allerdings das `applet`-Element vom W3C missbilligt ist und die Eigenschaft `length` sowohl alle `applet`-Elemente als auch alle `object`-Elemente eines HTML-Dokuments speichert, ist die weitere Verwendung dieses JavaScript-Objekts eher fraglich.

Abbildung 21 zeigt das modifizierte HTML-Dokument aus Code 11, nachdem der „*Number*“ Button gedrückt wurde.

Das `applets`-Objekt bietet auch die Möglichkeit des Zugriffs auf den Java Code in einem Applet. Hierzu muss dieses Java Applet entweder über seine Indexnummer, seinen Indexnamen oder seinen Namen angesprochen werden [Vgl. MüNe02, S.824].

[121] S.a. 3.5.5.4 .
[122] Vergleiche hierzu 4.5.1.1 .

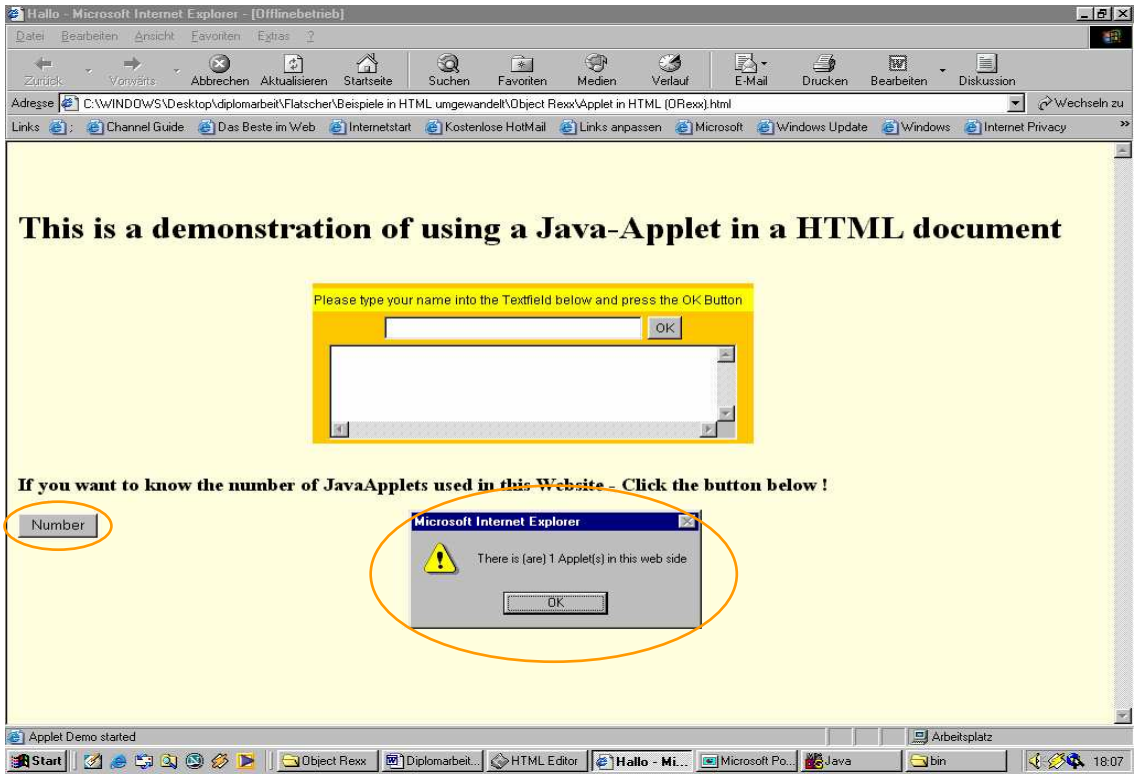

Abbildung 21 : Das applets-Objekt

4.2.4.6 LiveConnect

LiveConnect wurde von Netscape im Jahre 1996 entwickelt. LiveConnect ist eine Technologie welche eine Interaktion zwischen Java, JavaScript und Browser Plug-Ins ermöglicht [Vgl. Net03d].

4.2.4.6.1 Verhüllungen

Um ein JavaScript Objekt nach Java zu senden wird ein Java Wrapper (Hülle) vom Typ `JSObject` erzeugt. Beim Übermitteln eines `JSObjekts` von Java zu JavaScript, wird es wieder in seinen ursprünglichen JavaScript-Objekttyp enthüllt. Die `JSObject`-Klasse stellt eine Schnittstelle bereit um JavaScript Methoden aufzurufen und JavaScript Eigenschaften zu untersuchen [Vgl. Net03d].

4.2.4.6.2 Zugriff von Java auf JavaScript

Um JavaScript Objekte in Java verwenden zu können, muss das `netscape.javascript` Packet in die Java Datei eingebaut werden. Dieses Packet

stellt die Java Klassen `netscape.javascript.JSObject` und `netscape.javascript.JSException` zur Verfügung. Die Klasse `netscape.javascript.JSObject` ermöglicht Java-Code den Zugriff auf JavaScript Methoden und Eigenschaften [Vgl. Net03d].

4.2.4.6.3 Zugriff von JavaScript auf Java

Für den Zugriff von JavaScript auf Java werden die speziellen LiveConnect-Objekte `JavaArray`, `JavaObject`, `JavaClass` und `JavaPackage` verwendet. `JavaArray` ist ein verhülltes Java Array das von JavaScript-Code aus erreichbar ist. `JavaClass` ist eine JavaScript-Referenz auf eine Java Klasse. `JavaObject` ist ein verhülltes Java Objekt das von JavaScript-Code aus erreichbar ist. `JavaPackage` ist eine JavaScript Referenz auf ein Java Packet [Vgl Net03d].

Mit dem `id`- oder `name`-Attribut kann einem Applet ein Name gegeben werden. Mit diesem Namen kann das Applet anschließend von JavaScript aus angesprochen werden.

Im Folgenden wird anhand eines einfachen Beispiels gezeigt, wie mit Hilfe von JavaScript auf ein Java-Applet zugegriffen werden kann, um dessen Inhalt zu ändern. Dabei kann von einem Anwender in einem HTML-Textfeld ein beliebiger Text eingegeben werden, welcher anschließend als Inhalt des Applets (graues Feld) erscheint.

Abbildung 22 wird dies im Anschluss nochmals verdeutlichen. Vorab wird jedoch in Code 13 ein entsprechendes Java Applet definiert.

```
Import java.awt.Graphics;   -- access to the class awt.Graphics
import java.applet.Applet;  -- access to the class awt.Applet
 -- class Orexx
public class testapplet extends Applet
{
 String Appout =""; -- the variable Appout is empty
 -- creates a graphic called go with the variable Appout in it
 public void paint(Graphics textfield)
 {
  textfield.drawString(Appout, 25, 20);
 }
 -- changes the value of Appout into the value of the HTML
```

```
    textfield by pressing the "Applet" Button (Code 14)
 public void change(String HTMLtext)
 {
  Appout = HTMLtext;        -- Appout gets the value of the HTML
                               textfield
  repaint();
 }
}
```

Code 13 : Zugriff auf Java-Code (1)[123]

In Code 14 ist ein entsprechendes HTML-Dokument dargestellt, in welches dieses Applet über das `object`-Element eingebunden ist .

```
<html><head>
            <!- Beginning of JavaScript code -->
<script type="text/JavaScript">
-- function Applet which is invoked by pressing the “Applet”
   button
function Applet()
{
document.output.change(document.getElementById("input").value)
}
-- function reset which is invoked by pressing the “New” button
function reset()
{
document.getElementById("input").value = "";
document.output.change(document.getElementById("input").value)
}
</script>
                <!- End of JavaScript code -->
</head>
<body bgcolor="lightyellow">
<h1>This is a demonstration of getting access to a Java App-
let</h1><br>
<h3>Please write anything into the textfield below and press
the Applet button.</h3>
<input type=”text” id="input">
<input type="button" value="Applet" onclick="Applet()">
<input type="button" value="New" onclick="reset()"><hr>
<h2>The Java Applet :</h2>
  -- embeds the applet of Code 13
<object classid="java:testapplet.class" name="output"
  width="250" height="40"></object>
</body></html>
```

Code 14 : Zugriff auf Java-Code (2)

[123] Basierend auf [MüEl03, S.11] .

Wird im HTML-Dokument der "`Applet`" Button gedrückt, so wird dem Applet mit dem Namen „***output***" über die JavaScript Funktion `Applet()`, der Inhalt des Textfeldes (`document.getElementById("input").value`) mit der `id` „**input**" übergeben. Innerhalb von Code 13 übernimmt die Methode `change` diesen Inhalt und ändert den Inhalt der vorab definierten Variablen „`Appout`" in den Inhalt des HTML-Textfeldes (`Appout = HTMLtext`). In Code 13 wird dieser Inhalt in der Variablen „`HTMLtext`" gespeichert. Anschließend wird mit Hilfe der `repaint`-Methode die Grafik mit dem neuen Inhalt (`HTMLtext`) gefüllt bzw. neu gezeichnet.

Um den Inhalt des Textfeldes und des Applets zu löschen, muss der "`New`"-Button, welcher die `reset`-Funktion aufruft, gedrückt werden.

Abbildung 22 zeigt das Zusammenspiel von Code 13 und Code 14 .

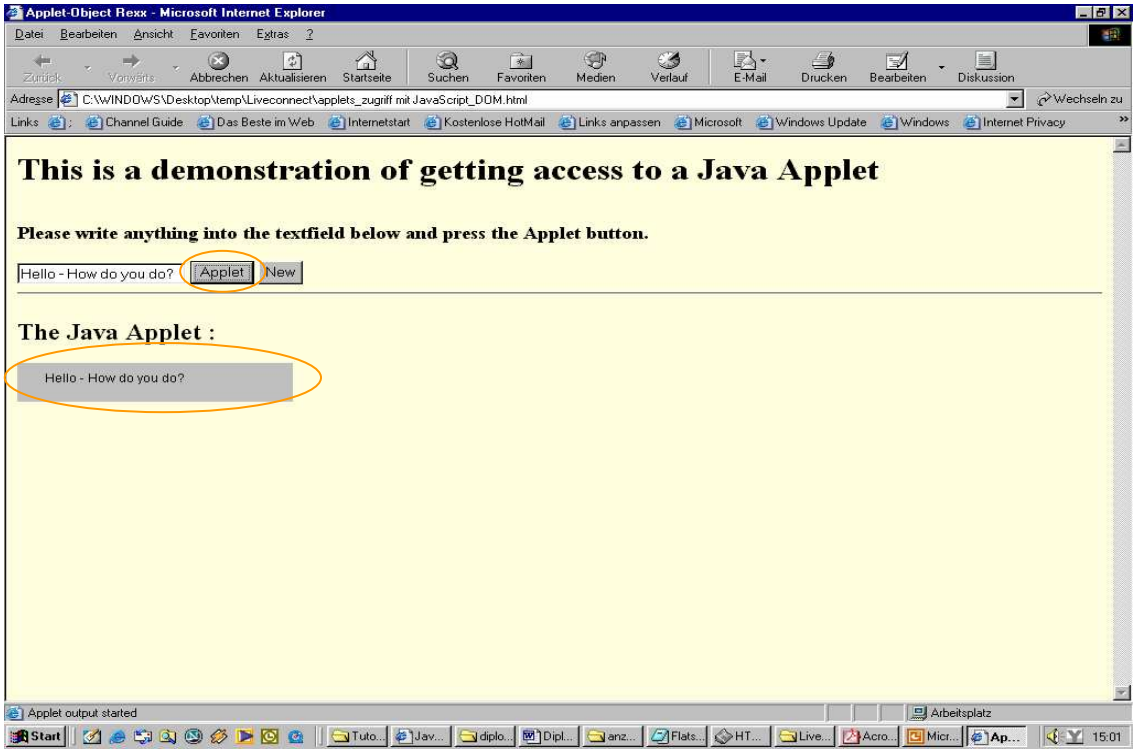

Abbildung 22 : Zugriff auf Java-Code

4.2.5 SSI - Server Side Includes

Auch ein Web-Server kann für die Gestaltung von Web-Seiten eingesetzt werden. Dazu stehen auf vielen (nicht allen) Servern die so genannten Server Side Includes (SSI) bereit. Dabei handelt es sich um einen Satz relativ einfacher Befehle, die in ein HTML-Dokument eingebettet werden können und direkt auf dem Server ausgeführt werden. Damit ein WWW-Server ein HTML-Dokument mit SSI sofort erkennt und ausführt, muss diese Datei mit der Endung .shtml , .shtm oder .sht anstatt .html gekennzeichnet sein. SSI-Anweisungen beginnen mit einem Gatterzeichen #, und müssen in einem HTML-Dokument innerhalb eines Kommentars `<!-- .....-- >` notiert werden. Hinter der Anweisung folgt ein Parameter und dessen Wert [Vgl. MüNe01, S.831] .

Tabelle 10 stellt verschiedene SSI-Anweisungen, deren Parameter, Zweck und Werte dar .

Anweisung	Parameter	Werte	Zweck
#config	1.errmsg= 2.sizefmt= 3.timefmt=	„Fehlermeldung“ „“Wert“ „“Wert“	Fehlermeldung, falls SSI nicht funktioniert.
#echo	Var=	“CGI Umgebungsvariable“ DOCUMENT_NAME DOCUMENT_URL LAST_MODIFIED DATE_LOCAL DATE_GMD QUERY_STRING_UNESCAPED	Die entsprechenden Inhalte werden am Bildschirm ausgegeben.
#exec	Cmd= cgi=	“[Pfad/Programmdatei}“ “[Pfad/CGISript}“	Starten eines Programms auf dem Server.

#fsize	File=	“[Pfad/Datei]“	Gibt die Größe der angegebenen Datei an.
#flastmod	File=	“[Pfad/Datei]“	Gibt den Zeitstempel der angegebenen Datei aus.
#include	File= Virtual=	“[Pfad/Datei]“	Gibt den Inhalt der angegebenen Datei aus.

Tabelle 10 : SSI-Anweisungen[124]

4.2.6 CGI - Common Gateway Interface

Das Common Gateway Interface (CGI) ist eine sprachenunabhängige Schnittstellenspezifikation, welche die Möglichkeit bietet, Programme oder Skripte in einem Netz bereitzustellen, und diese von einem HTML-Dokument (z.B. Formular, SSI Anweisung) aus aufzurufen. Ein CGI-Programm kann in verschiedenen Programmiersprachen geschrieben werden. Eine verwendete Programmiersprache muss lediglich das Lesen von `stdin`, das Schreiben von `stdout` erlauben und Umgebungsvariablen lesen können. Somit besteht die Möglichkeit Sprachen wie beispielsweise Perl[125], SSI[126], C, Pascal oder PHP[127] innerhalb des CGI einzusetzen. Damit das Programm auf dem Server ausführbar ist, muss auf dem Server entweder ein Laufzeitinterpreter (Perl-Interpeter, PHP-Interpreter) welcher das Programm ausführt, vorhanden sein oder das Programm muss für die Betriebssystemumgebung des Servers als ausführbares Programm kompiliert worden sein [Vgl. MüNe01, S.49].

Bei CGI-Programmen fordert der Web-Browser vom Web-Server über HTTP eine Datei an, die der Web-Server als auszuführendes Skript erkennt. Um zu

[124] Nach [MüNe01. S. 832f] .
[125] S.a. 4.2.6 .
[126] S.a. 4.2.4 .
[127] S.a. 4.2.7 .

wissen, dass es sich bei der vom Browser angeforderten Datei um ein auszuführendes CGI-Skript handelt, prüft der Web-Server folgende Kriterien [Vgl. Sht03j]:

- Das aufgerufene Skript befindet sich in einem Verzeichnis, das aus Sicht des Web-Servers als mögliches Verzeichnis für CGI-Skripte definiert ist (z.B. *cgi-bin* und dessen Unterverzeichnisse).
- Die Lese- und Ausführrechte sind bei der Skriptdatei so gesetzt, dass der Web-Server berechtigt ist, die Datei auszuführen.
- Die Skriptdatei hat eine Dateiendung, die vom System als Endung für CGI-Skripte erkannt wird, z.B. *.pl* für Perl, .php für PHP oder *.cgi*.

Sind diese Faktoren erfüllt, so reagiert der Web-Server entsprechend und sendet den Quelltext des Skripts nicht einfach an den Browser, sondern ruft den entsprechenden Interpreter für das CGI-Skript auf [Vgl. Sht03j].

Diese Programme werden anschließend auf dem Server ausgeführt. Sie erzeugen dabei selbst HTML-Code und senden diesen an den Web-Browser des Anwenders. Insofern sind solche CGI-Skripte nur ausführbar, wenn eine http-basierte Kommunikation zwischen einem Web Server und einem Web Browser stattfindet [Vgl. MüNe01, S.49].

Der Nachteil bei der Verwendung des CGI besteht darin, dass jedes Mal wenn eine Anforderung über diese Schnittstelle an das CGI-Programm weitergereicht wird, ein neuer Prozess vom Server gestartet wird. Anschließend muss er alle zur Verarbeitung nötigen Daten übergeben [Vgl. HäPeS00, S.366]. „Dieser Sachverhalt kann je nach Anwendung, aufgrund dieser hohen Belastung des Servers zu einem nicht akzeptablen Zeitverhalten führen, insbesondere dann, wenn eine größere Folge von Datenbankoperationen durchzuführen ist“ [HäPeS00, S.366].

Der entscheidende Vorteil von CGI besteht jedoch darin, dass es sich um einen kostenlosen und produktübergreifenden Standard handelt [Vgl. MüNe01, S.817].

4.2.6.1 Bestandteile des CGI

Das Common Gateway Interface besteht zum Einen aus einem bestimmten *Verzeichnis* (im allgemeinen cgi-bin) auf dem Server, in welchem die CGI Programme enthalten sind, und zum Zweiten aus den so genannten *CGI-Umgebungsvariablen* [Vgl. MüNe01, S.816].

Ein WWW-Server stellt solche Umgebungsvariablen auf Betriebsystemebene zur Verfügung. Bei einem Aufruf eines CGI-Programms, füllt der Server einige dieser Variablen mit bestimmten Informationen. Diese Variablen können anschließend von dem CGI-Programm ausgelesen und für eigene Zwecke verwendet werden [Vgl. MüNe01, S.828].

Es besteht daneben auch die Möglichkeit diese Variablen mit einem CGI-Programm selbst zu füllen oder zu verändern [Vgl. MüNe01, S.828].

Um eine CGI-Umgebungsvariable aus einem CGI-Programm heraus nutzen zu können, muss die in der verwendeten Programmiersprache vorgesehene Technik zum Auslesen von Umgebungsvariablen verwendet werden [Vgl. MüNe01, S.829].

In Tabelle 11 sind die verschiedenen CGI-Umgebungsvariablen und deren Erläuterung dargestellt [Vgl. MüNe01, S.829f].

CGI- Umgebungs-variable	Erläuterung
CONTENT_LENGTH	Enthält die Anzahl der Zeichen, die beim Aufruf des CGI-Skripts über die POST-Methode[128] übergeben wurden.
CONTENT_TYPE	Bei einem Aufruf über die POST-Methode enthält diese Variable den MIME-Typ der übergebenen Daten.
GATEWAY_INTERFACE	Enthält die Version des CGI.

[128] S.a. 4.2.6.3.2 .

http_ACCEPT	Enthält die Liste der MIME-Typen, die der aufrufende WWW-Browser akzeptiert.
http_REFERER	Enthält die URL-Adresse der WWW-Seite, von der aus das CGI-Skript aufgerufen wurde.
http_USER_AGENT	Enthält Produkt- und Versionsinformationen zum aufrufenden WWW-Browser.
PATH_INFO	Wenn das Skript über die `GET`-Methode[129] aufgerufen wurde, enthält diese Variable spezielle Pfadinformationen, und zwar relativ zu den im WWW-Server eingestellten Root Verzeichnissen.
PATH_TRANSLATED	Im Unterschied zu PATH_INFO enthält diese Variable die Pfadangeben entsprechend der Verzeichnisstruktur des WWW-Servers.
QUERY_STRING	Enthält beim Aufruf über die GET-Methode eine Zeichenkette mit Daten, die dem Skript übergeben wurden.
REMOTE_ADDR	Enthält die IP-Adresse des Servers, über den das CGI Skript aufgerufen wurde, wenn es ein anderer Server ist als der, auf dem das CGI-Skript liegt.
REMOTE_HOST	Enthält die Domain-Adresse des Servers, über den das CGI-Skript aufgerufen wurde, wenn es ein anderer Server ist als der, auf dem das CGI-Skript liegt.
REMOTE_IDENT	Enthält Protokollinformationen, wenn auf dem Server das `ident` Protokoll für geschützte Zugriffe läuft.
REMOTE_USER	Enthält den Benutzernamen, welcher das CGI Programm aufgerufen hat.
REQUEST_METHOD	Enthält den http Befehl, mit dem das CGI Programm aufgerufen wurde. (`GET` oder `POST`)
SCRIPT_NAME	Enthält die relative Adresse des CGI Skripts.

[129] S.a. 4.2.6.3.1 .

SERVER_NAME	Enthält den Namen des Servers.
SERVER_PORT	Enthält die Portnummer (normal: 80)
SERVER_PROTOCOL	Enthält die Version des http Protokolls.
SERVER_SOFTWARE	Produktbezeichnung der installierten Server-software.

Tabelle 11 : CGI-Umgebungsvariablen[130]

4.2.6.2 CGI und HTML

Die Kommunikation zwischen CGI und HTML erfolgt in beide Richtungen. Im Allgemeinen wird ein auf dem Server liegendes CGI-Programm innerhalb eines HTML-Dokumentes (durch z.B. Drücken des Submit–Button[131] in einem Formular) aufgerufen, und sendet den entsprechenden HTML Code an den Browser des Anwenders zurück [Vgl. MüNe01, S.817].

Beim Ausführen des CGI Programms können dabei Daten, welche innerhalb des HTML-Dokuments vom Anwender eingegeben wurden, verarbeitet werden. Somit kann ein CGI-Skript dazu verwendet werden, Datenbanken zu durchsuchen und dem Anwender anschließend das Ergebnis im Browser zu präsentieren oder Online-Bestellungen entgegenzunehmen und zu verarbeiten. Daneben besteht auch die Möglichkeit Daten auf dem Server zu speichern und zu einem später Zeitpunkt auszulesen [Vgl. MüNe01, S.817f].

Um ein solches CGI Programm aufzurufen gibt es dabei mehrere Möglichkeiten. Tabelle 12 zeigt, auf welche Arten ein CGI-Skript aus einer HTML-Dokument heraus aufgerufen werden kann [Vgl. MüNe01, S.818] .

Aufruf über :	Struktur
Formular	`<form action=“[Pfadangabe]“ >`
Verweis	`<a href=“[Pfadangabe]“ >`
Grafikreferenz	`<img src=“[Pfadangabe]“>`

[130] Nach [MüNe01, s.829f] .
[131] S.a. 3.5.5.4.3.

SSI Anweisung	`<!-- #exec cgi="[Pfadangabe]"--`
Automatisches Laden	`<meta URL="[Pfadangbabe]">`

Tabelle 12 : CGI-Aufrufe aus einem HTML-Dokument

Mit `[Pfadangabe]` ist dabei die Angabe des Verzeichnisses und das darin befindliche, auszuführende CGI-Programm gemeint.

4.2.6.3 Formulardaten übertragen[132]

Eine der am häufigsten eingesetzten HTML Elemente ist das Formular[133]. Insofern soll nun gezeigt werden, wie die Formulardaten an den Server geschickt werden können.

Um Daten zwischen einem WWW-Server und einem WWW Browser austauschen zu können, wird ein Übertragungsprotokoll benötigt. Dieses Übertragungsprotokoll ist das Hypertext Transfer Protocol (`http`). Im Zusammenhang mit Formularen sind dabei zwei `http`-Befehle von Bedeutung. Zum Einen die `GET`-Methode und zum Anderen die `POST`-Methode.

4.2.6.3.1 GET-Methode

Um die `GET`-Methode zum Versenden der Daten an den Server zu verwenden, muss im einleitenden `<form ...>` Tag die Angabe `method="GET"` erfolgen. Bei dieser Angabe werden die vom Anwender ausgefüllten Formulardaten als erstes an die Server-Software übertragen und anschließend in der `QUERY_STRING` Umgebungsvariablen[134] zwischengespeichert. Ein im einleitenden Formulartag durch `action="[Pfadangabe]"` aufgerufenes CGI-Skript, welches die Daten aufnehmen und verarbeiten soll, muss den Inhalt dieser Umgebungsvariablen auslesen, um an die Formulardaten heranzukommen [Vgl. MüNe01, S.834].

---

[132] Basierend auf [MüNe01, S.833-835] .
[133] S.a. 3.5.5.
[134] S.a. 4.2.6.1.

Der Nachteil dieser Methode ist, dass die vom Anwender angegebenen Daten direkt an den URL des CGI-Programmaufrufs angehängt werden und somit einsehbar sind [Vgl. MüNe01, S.834] .

4.2.6.3.2 POST-Methode

Bei dieser Übertragungsmethode muss als Wertzuweisung des `method`-Attributs, “POST“ angegeben werden. Dies bewirkt, dass die ausgefüllten Formulardaten direkt an die im `action`-Attribut angegebene Adresse übertragen werden. Das mit `action` aufgerufene CGI-Programm muss um an die Formulardaten heranzukommen die Standardeingabe (`stdin`) auslesen. Da im übergebenen Datenstrom das Datenende nicht gekennzeichnet ist, muss das entsprechende Programm die Umgebungsvariable `CONTENT_LENGTH` auslesen, um zu wissen wie viele Zeichen es von der Standardeingabe lesen soll [Vgl. MüNe01, S.834] .

4.2.6.3.3 Datenstrom bei Übertragung von Formulardaten

Bei der Datenübertragung eines vom Anwender ausgefüllten Formulars an das entsprechende CGI-Skript, wird eine bestimmte Kodierungsmethode verwendet. Dies ist erforderlich, damit das aufgerufene CGI-Programm erkennt, aus welchen Feldern das Formular besteht und welche Daten der Anwender in welches Feld eingetragen hat.

Diese Kodierungsmethode benutzt dabei folgende Regeln [MüNe01, S.834f]:

- Namen und Daten werden durch ein Gleichheitszeichen voneinander getrennt.
- Leerzeichen werden durch ein Pluszeichen + ersetzt.
- Einzelne Formularelemente inklusive deren Daten werden durch das kaufmännische & voneinander getrennt.
- Eingeleitet durch das Prozentzeichen % , werden alle Zeichen mit den ASCII-Werten 128 bis 255 (hexadezimal 80 bis FF) durch eine Hexadezimal-Zeichenfolge umschrieben.

- Alle Zeichen, die in diesen Regeln als Steuerzeichen vorkommen (z.B. =, %, & ...) werden genauso wie höherwertige ASCII Zeichen hexadezimal umschrieben.

Somit ergibt sich bei einer Übertragung beispielsweise folgender Datenstrom :

```
“ forename=Thorsten&surname=Schaedler&text=How+do+you+do “
```

4.2.6.4 Der Apache Web Servers

Um lokal auf dem eigenen PC ein CGI zu haben, muss auf diesem ein eigener WWW-Server installiert sein. Ein WWW-Server ist eigentlich ein gar nicht besonders großes Software-Programm, welches prinzipiell auf jedem Rechner lauffähig ist. Für eine Kommunikation zwischen Client und Server ist nicht einmal eine Internetverbindung notwendig. Es ist lediglich ein TCP/IP-Socket von Nöten . Unter Windows ist dies die winsock.dll [Vgl. MüNe01, S.821f].

Der Apache Web-Server[135] kann kostenlos aus dem Internet herunter geladen und anschließend auf dem PC installiert werden. Mit seiner Hilfe können sowohl die nachfolgenden Perl-Skripte[136], PHP-Skripte[137], Rexx Server Pages[138] als auch andere serverseitige Technologien auf dem eigenen Rechner getestet werden.

„Das Erfolgsrezept des Apache liegt insbesondere in der außerordentlich guten Dokumentation und in einem modularen Konzept, welches die problemlose Erweiterung des Servers mittels externer Module ermöglicht.“ [AdWe03, S.1].

Apache Module sind Kodesegmente. Sie entsprechen der Apache-API Spezifikation und können entweder statisch oder dynamisch in den Apache Web Server geladen werden. Statische Apache Module werden in den http-Daemon[139] und dynamische Apache Module über die Konfigurationsdatei des Webservers geladen [Vgl. ATvN04].

---

[135] http://archive.apache.org/dist/httpd/old/apache_1_3_6_win32.exe, Abruf am 2003-08-12.
[136] S.a. 4.2.7 .
[137] S.a. 4.2.8 .
[138] S.a. 4.2.12 .
[139] Vergleiche hierzu http://www.boeny.de/apache/Apache_Ausarbeitung.html , Abruf am 2004-01-26.

„Die Unterstützung dynamischer Module (Dynamic Shared Object DSO) ist eine der wichtigsten Funktionen des Apache Web-Servers. DSO erlaubt es, Funktionen und Fähigkeiten von Apache genau dann zu erweitern, wenn diese gebraucht werden.“ [ATvN04].

Im Rahmen dieser Arbeit wurde der Apache Server in der Version 1.3.6 verwendet. Der aktuellste Apache Server ist allerdings 2.048[140] welcher der Version 2.0.x angehört und einige Änderungen und Erweiterungen im Vergleich zur Version 1.3.x in sich birgt.

4.2.7 Perl

Perl bedeutet **P**ractical **E**xtraction and **R**eport **L**anguage [Vgl. MüNe01, S.825] und ist eine serverseitige Programmiersprache, die eine Mischung aus klassischer Programmiersprache wie C und Skriptsprachen wie JavaScript darstellt, und mit dessen Hilfe CGI-Skripte erstellt werden können. Um Perl als Sprache für CGI-Skripte einzusetzen, muss neben dem Perl Interpreter noch ein Web-Server wie beispielsweise der Apache Server auf dem System installiert sein.

Der Perl Interpreter kann kostenlos aus dem Internet herunter geladen werden. Im Unterschied zu clientseitig eingesetzten Skriptsprachen wie JavaScript[141] bzw. Client-Side-JavaScript oder Object Rexx[142] werden die Befehle nicht vom Client (Browser) sondern vom Server ausgeführt. Der Browser stellt dabei lediglich das Ergebnis am Bildschirm dar.

Der Perl Interpreter kann mit einem Perl Skript (Datei mit Perl Code) aufgerufen werden. Solche Dateien besitzen die Endung .pl

Die erste Zeile eines Perl-Skripts sollte immer eine so genannte "*Shebang*"-Anweisung enthalten. „Diese Anweisung ist für ein auslesendes Programm der Hinweis darauf, welches Programm für die Interpretation des Quelltext dieser Skriptdatei aufgerufen werden soll.“[Sht03j].

---

[140] http://httpd.apache.org/download.cgi , Abruf am 2004-01-25.
[141] S.a. 4.2.2.1 .
[142] S.a. 4.2.2.2 .

Beispiel : `#!c:\programme\perl\bin\perl.exe oder #!c:\usr\bin`

Perl ist allerdings nicht nur für CGI-Skripte gedacht. Perl-Skripte können zahllose Aufgaben auf einem Rechner übernehmen. So kann Perl beispielsweise eingesetzt werden um die Rechnerauslastung zu analysieren, Backups zu organisieren, CGI-Umgebungsvariablen zu verwenden oder in Dateien nach etwas zu suchen und es eventuell durch etwas anderes zu ersetzen [Vgl. Sht03j].

Um Perl-Skripte auszuführen, die nicht als CGI-Skripts eingesetzt werden, muss ein Zugang zu einer Shell oder Kommandozeile auf dem Rechner eingerichtet sein. Am lokalen PC und unter MS Windows kann ein DOS-Fenster geöffnet werden. Anschließend kann der Perl-Interpreter vom DOS-Prompt aus aufgerufen werden [Vgl. Sht03j].

Der Perl-Interpreter wird normalerweise einfach durch Eingabe von `perl` gestartet. Ein Perl-Skript kann demnach mit „ `perl [Dateiname].pl` “ gestartet werden. Gegebenenfalls muss sowohl der Perl-Interpreter als auch das Perl-Skript mit den korrekten Pfadnamen aufgerufen werden. [Vgl. MüNe01, S.836].

Perl wird in HTML oft zum Verarbeiten von Formularen bzw. Formulareingaben eingesetzt und um benutzerspezifische Angaben oder Antworten zu erzeugen und diese dem Anwender zukommen zu lassen.

4.2.7.1 Formulardaten einlesen

Um Daten aus einem vom Anwender abgeschickten HTML Formular einzulesen, werden einige der CGI Umgebungsvariablen[143] benötigt.

So wird beim verwenden der `GET`-Methode, zum Einlesen der Daten die CGI-Umgebungsvariable `QUERY_STRING` benötigt. Bei `POST` dagegen wird hierfür die Umgebungsvariable `CONTENT_LENGTH` benötigt [Vgl. MüNe01, S.829f].

[143] S.a. 4.2.6.1 .

4.2.7.2 HTML-Code an Browser senden[144]

„Eine der wichtigsten Aufgaben von CGI-Skripten ist es HTML-Code an den Browser zu senden.“ [MüNe01, S. 892]. Dabei können Variablen des Perl Skripts beliebig in den HTML-Code mit eingebaut werden. Mit Hilfe der `print`-Funktion kann in Perl, HTML-Code geschrieben werden. Dies kann beispielsweise sinnvoll sein um sich für das Ausfüllen des Formulars beim Anwender zu bedanken oder entsprechende CGI-Umgebungsvariablen als auch entsprechende Daten aus der Datenbank des Servers dem Anwender anzuzeigen.

In Code 15 wird mit Perl HTML-Code erzeugt. Dabei werden auch verschiedene CGI Umgebungsvariablen, welche im Code mit blauer Farbe gekennzeichnet sind, verwendet. Diese werden vom Server entsprechend ausgefüllt und anschließend dem Browser (Client) als HTML Code übermittelt.

In Perl kann über `$Env{'[Variablenname]'}` auf CGI-Umgebungsvariablen zugegriffen werden [Vgl. MüNe01, S.829].

```
#!c:\programme\perl\bin\perl.exe
-- the  following perl code creates a HTML Page which includes
   some CGI environment variables (blue color)
print "Content-type: text/html\n\n";
print '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transi-
tional//EN">', "\n";
print "<html><head><title>Environment variables</title></head>
       <body bgcolor=\"lightyellow\">\n";
print "<p>You are connected with the <b><i>$ENV{'SERVER_NAME '}
</i></b>, which is an <b><i>$ENV{'SERVER_SOFTWARE'} }</i></b>
Server.</p>\n";
print "<p>For any question you can send an e-mail to the Admin-
istrator <b><i> $ENV{'SERVER_ADMIN'} </i></b></p>\n";
print "<p>You are using the <b><i>$ENV{'HTTP_USER_AGENT'}
</i></b> as your Web Browser</p>\n";
print "</body></html>\n";
```

Code 15 : CGI-Umgebungsvariablen mit Perl

Abbildung 23 zeigt die mittels des Apache Web Servers erzeugte Darstellung von Code 15 im Opera Browser.

---

[144] Basierend auf [MüNe01, S.892] .

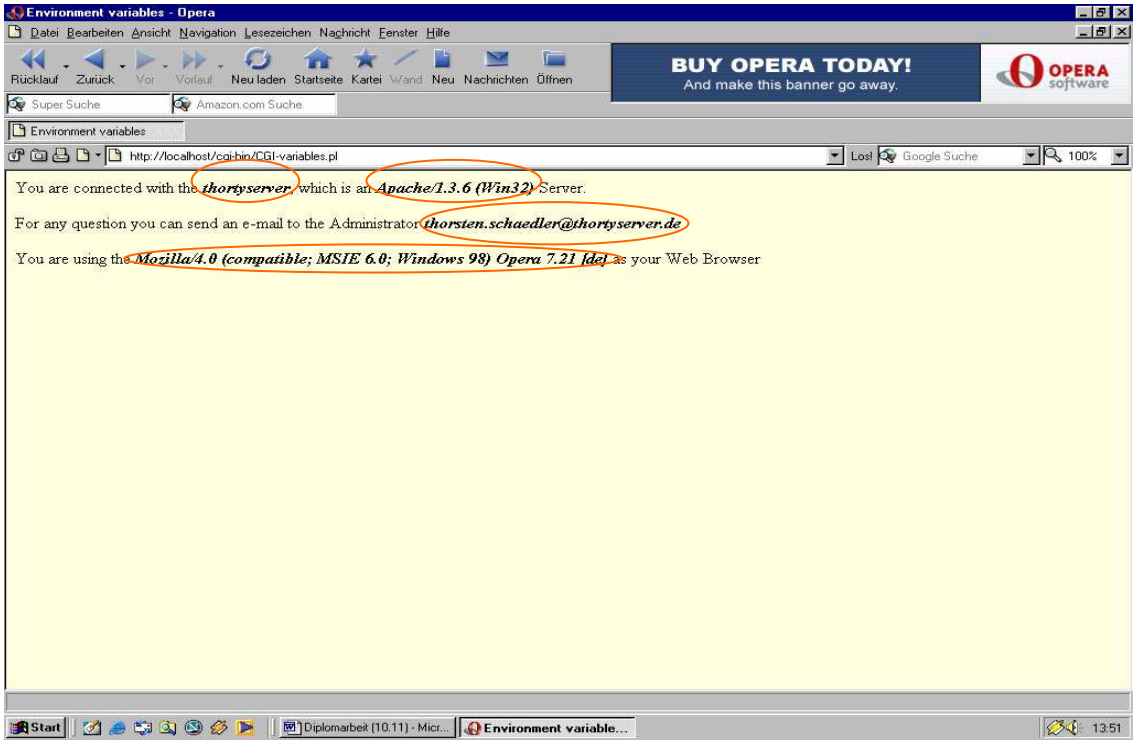

Abbildung 23 : CGI-Umgebungsvariablen mit Perl

4.2.7.3 PerlScript in HTML

Mit PerlScript besteht die Möglichkeit einen Text innerhalb eines HTML-Dokumentes zu erzeugen. Im Gegensatz zum vorherigen Kapitel wird hierbei kein Server benötigt. Allerdings muss der Perl-Interpreter auf dem System installiert sein. Um innerhalb eines HTML-Dokuments einen Text mit PerlScript erzeugen zu können, wird das `script`-Element benötigt.[145] Code 16 zeigt wie PerlScript innerhalb eines HTML-Dokuments verwendet werden kann.

```
<html>....
<script type="text/PerlScript">
    -- creates “any Text” in a html file
$window->document->write("[any Text]");
 </script>
 ....
</html>
```

Code 16 : PerlScript in einem HTML-Dokument

[145] S.a. 4.5.1.1 .

4.2.7.4 Perl Module

Der Perl Interpreter verfügt standardmäßig über viele Funktionen, mit welchen die verschiedensten Aufgaben wie Berechnungen, Betriebsystemaufrufe, Datei- und Verzeichnismanagement und dergleichen erfüllt werden können. Je mehr Funktionen allerdings im Perl Interpreter fest eingebaut werden, desto langsamer und ressourcenunfreundlicher wird dieser. Aus diesem Grund besteht die Möglichkeit Funktionen und Variablen über Perl-Skripte bei Bedarf zur Verfügung zu stellen. Diese speziellen Perl Skripte werden Module genannt und werden in die Perl-Datei eingebunden. Mittlerweile gibt es Tausende frei verfügbarer Perl Module [Vgl. Sht03h].

Es werden zwei Arten von Modulen unterschieden. Zum Einen gibt es so genannte *Standardmodule*, welche zusammen mit dem Perl Interpreter ausgeliefert werden. Bei der zweiten Form handelt es sich um so genannte *CPAN Module* (CPAN = Comprehensive Perl Archive Network), welche im CPAN Verzeichnis enthalten sind. CPAN Module können in einem Perl Skript allerdings erst verwendet werden, wenn sie auf dem System installiert wurden [Vgl. Sht03h].

Module können mit den Funktionen `use` und `require` in eine Perl Datei eingebunden werden. Bei `use` wird das Modul zu Beginn der Datei eingebunden, wohingegen die `require` Funktion erst beim Aufruf aktiviert wird [Vgl. Sht03h].

4.2.7.5 Das CGI Modul

Das CGI-Modul gehört seit Perl 5.004 zu den Standardmodulen. Das CGI-Modul besteht aus einem Hauptmodul und mehreren Untermodulen.

Das CGI-Modul kann dabei objektorientiert als auch funktionsorientiert eingesetzt werden, wobei die objektorientierte Verwendungsweise als eleganter gilt und das Definieren mehrer unabhängiger CGI-Objekte im gleichen Skript erlaubt [Vgl. Sht03g].

Code 17 zeigt wie die Verarbeitung eines Formulars auf objektorientierte Weise mit dem CGI-Modul bewerkstelligt werden kann.

```
<html>...
<body>
<body bgcolor="lightyellow">
<h1>Please insert your data</h1>
<form method="POST" action="/cgi-bin/back.pl">  -- invokes the
back.pl file on the Apache Server by pressing the submit button
<table border="3">
<tr>
<td>Forename:</td><td><input type="text" name="forename"></td>
</tr><tr>
<td>Surname:</td><td><input type="text" name="surname"></td>
</tr><tr>
<td>Tel.:</td><td><input type="text" name="tel"></td>
</tr><tr>
<td>E-mail:</td><td><input type="text" name="mail"></td>
</tr><tr>
<td><input type="submit" value="Send"></td>
<td><input type="reset" value="New"></td>
</tr>
</table>
```

Code 17 : HTML-Formular (contact.html) für Perl Antwort

In diesem Beispiel muss allen Eingabefeldern, welche im Perl Skript verwendet werden sollen mit dem Attribut `name` ein eindeutiger Name zugewiesen werden.

Abbildung 24 zeigt das vom Anwender ausgefüllten Formular aus Code 17 im MS Internet Explorer .

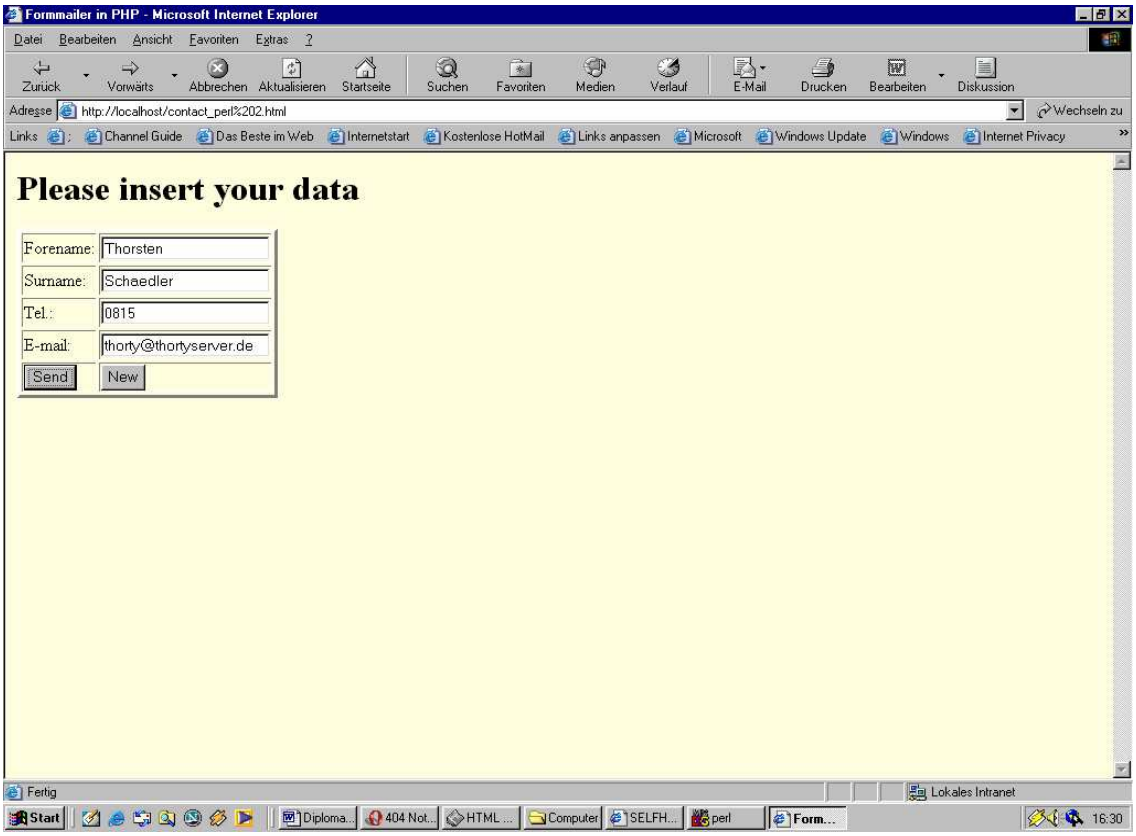

Abbildung 24 : HTML- Formular (contact.html) für Perl Antwort

In Code 18 ist das Perl-Skript (back.pl) dargestellt, welches beim Drücken des Submit-Button aus Code 17 aufgerufen wird.

```
#!c:\programme\perl\bin\perl.exe
use CGI;            -- embeds the CGI Modul
my $cgi = new CGI;  -- creates a new instance of the CGI Object
my @fieldnames = $cgi->param();
  -- the following script code creates a HTML file with the
     input values of the form of Code 17
print $cgi->header(),                -- creates a http header
-- creates the head of the HTML file with  the title, an extern
   Style and the value for the background color in it
$cgi->start_html(-title=>'CGI-Feedback',
                 -style=>{'src'=>'/styles/perl_back.css'},
                 -BGCOLOR=>'#FFFFCC',),
 -- the body of the HTML file
      $cgi->h1('Thank you ');
print $cgi->p('Your Input :');
      foreach my $field (@fieldnames)      -- loop
      {
        print $cgi->b($field),
              $cgi->b(' : '),
              $cgi->param( $field), "<br>";
```

```
        }
print $cgi->p('will be saved.');
print $cgi->end_html();
```

Code 18 : Antwort des Apache Web-Servers (back.pl)[146]

In einer Schleife werden die mit dem `name`-Attribut versehenen Eingabefelder und die darin enthaltenen Angaben des Anwenders in das HTML-Dokument gedruckt.

Abbildung 25 zeigt zum Abschluss dieses Kapitels die Antwort, welche der Apache Servers unter Verwendung von Code 18 erzeugt und dem Browser als HTML-Code sendet.

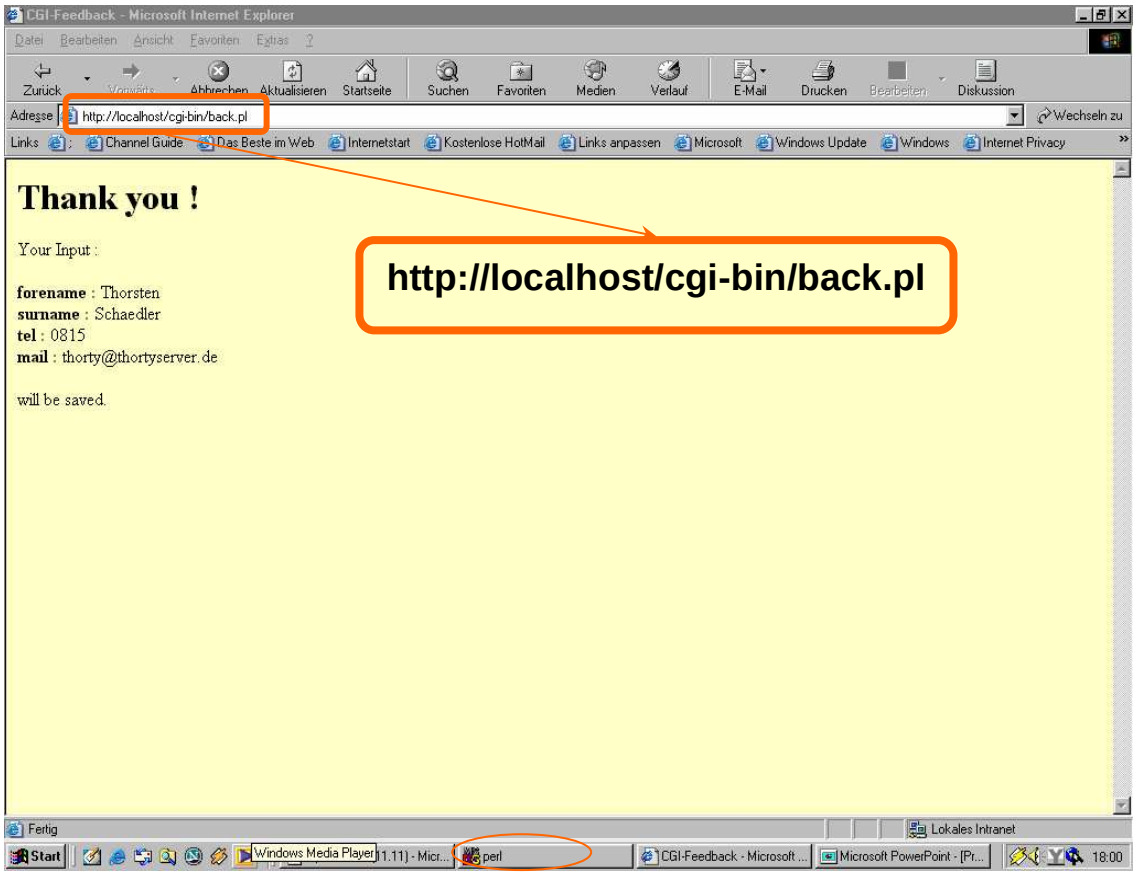

Abbildung 25 : Antwort (back.pl) des Apache Servers

[146] Basierend auf [Sht03g] .

4.2.8 PHP

PHP ist wie Perl[147] eine frei verfügbare serverseitige Skriptsprache und wurde im Herbst 1994 vom Dänen Rasmus Lerdorf für den privaten Gebrauch entwickelt [Vgl. Stae03]. PHP bietet die Möglichkeit bestimmte Informationen von einem Server zu laden und diese anschließend im Browser des Anwenders auszugeben. Die Syntax von PHP ist dabei an C, Java und Perl angelehnt [Vgl. KrWeg03, S.16].

Alles was mit Perl und CGI möglich ist, kann auch mit PHP realisiert werden. Genau wie beim Perl Interpreter wird auch beim PHP-Interpreter zuerst einmal kompiliert und anschließend ausgeführt. Da der PHP-Interpreter allerdings viel stärker als der Perl-Interpreter auf aktuelle Belange des Web-Publishing ausgerichtet ist, sind manche Dinge wie beispielsweise PDF-Dateien dynamisch zu generieren und diese anschließend druckreif an den Browser zu senden mit PHP wesentlich einfacher zu realisieren. Allerdings platzt der PHP Interpreter aufgrund solcher zahlreichen Features aus allen Nähten, was sich durchaus in der Performance niederschlagen kann [Vgl. MüNe02, S.84f].

4.2.8.1 Entwicklung

Die erste Version stammt von Anfang 1995 und nannte sich "Personal Home Page Tools". Aus „Personal Home Page“ wurde schließlich die Abkürzung „PHP“. Heute steht PHP für „PHP Hypertext Preprocessor“. Die erste Version bestand aus einem einfachen Parser, der einige spezielle Makros verstand und mit welchem sich einige andere "Spielzeuge" wie Gästebücher und Counter realisieren ließen [Vgl. Stae03].

Mit PHP/FI Version 2 folgte Mitte 1995 ein überarbeiteter Parser. FI steht für Form Interpreter und kommt von einem zweiten von Lerdorf entwickelten Programmpaket, das HTML-Formulare interpretieren konnte. Des Weiteren kam bei der Version 2 SQL Unterstützung hinzu [Vgl. Stae03].

---

[147] S.a. 4.2.7 .

Die Entwicklung von PHP wurde von den beiden israelischen Studenten Zeev Suraski und Andi Gutmans stark vorangetrieben, bis schließlich im Sommer 1997 PHP 3 veröffentlicht und zum weltweiten Erfolg wurde [Vgl. KrWeG03, S.16] .

PHP 4 wurde am 22.5.2000 freigegeben und war ein komplett neu geschriebener PHP-Parser, der neben besserer Performance (5 bis 200 fache Performance-Steigerungen im Vergleich zu PHP 3.0) auch mehr Funktionalität bot [Vgl. Stae03].

Den Kern von PHP 4 bildet die ZendEngine, welche von Andi Gutmans und Zeev Suraskis Firma Zend entwickelt wurde. Heute ist PHP das am meisten installierte Apache-Modul [Vgl. KrWeG03, S.17].

Die Bedeutung und der Einsatz von PHP steigt ständig. Dies belegt auch das in Abbildung 26 dargestellte Wachstum.

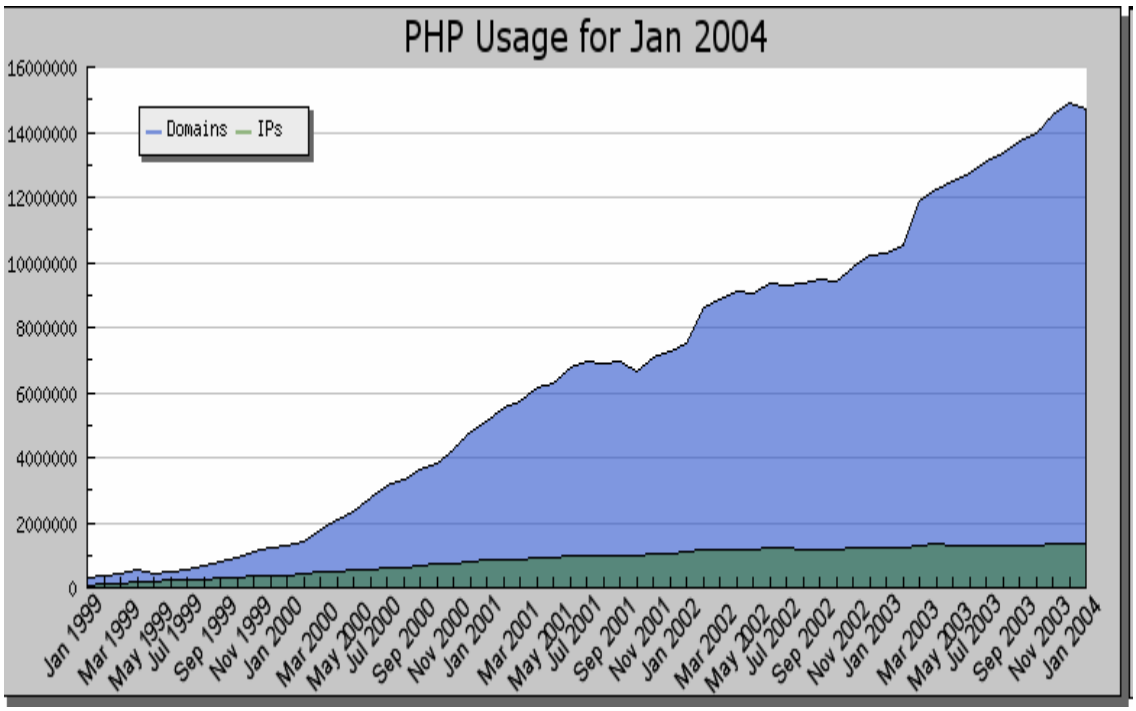

Abbildung 26 : PHP Wachstum[148]

[148] Entnommen aus [http://www.php.net/usage.php] .

4.2.8.2 PHP und HTML

PHP-Code kann im Gegensatz zu Perl genau wie JavaScript[149], Object Rexx[150], VBScript[151] oder SSI[152] direkt in ein HTML-Dokument an einer dafür geeigneten Stelle eingebettet werden [Vgl. KrWeG03, S.37].

Damit PHP überhaupt weiß ob PHP-Code ausgeführt werden soll, muss der Webserver allerdings wissen, welche Dateiendung PHP nach Code durchsuchen soll. Dies kann über die Konfiguration des (Apache-) Webservers (http.conf) eingestellt werden [Vgl. KrWeG03, s.37].

Seit PHP 4.0 wird standardmäßig die Endung .php verwendet [KrWeG03, S.37]. Somit müssen HTML-Dokumente welche PHP-Code enthalten mit .php anstatt .html gekennzeichnet sein

Um PHP von normalem HTML oder gar XML unterscheiden zu können, benötigt PHP bestimmte Tags. Hierfür können momentan vier verschiedene Möglichkeiten verwendet werden [Vgl. KrWeG03, S.37f.].

1. XML konforme Variante :
   `<?php .... ?>`
2. SGML konforme Variante :
   `<? ..... ?>`
3. ASP Variante :
   `<% ..... %>`
4. Skript konforme Variante :
   `<script language="php" type="text/php">.....</script>`

Beispiel :

.....

```
<p> Diese Diplomarbeit wurde von <?php echo "Thorsten Schaedler"
?> geschrieben. </p> .....
```

---

[149] S.a. 4.2.2.1 .
[150] S.a. 4.2.2.2 .
[151] S.a. 4.2.2.4 .
[152] S.a. 4.2.5 .

4.2.8.2.1 HTML-Code mit PHP erzeugen

Mit Hilfe der `print`- oder `echo`-Anweisung kann beliebiger HTML-Code erzeugt werden. Der Server schickt anschließend reinen HTML-Code an den Browser des Anwenders [Vgl. KrWe03, S.38-41].

Code 19 zeigt wie ein HTML-Dokument mittels PHP erzeugt werden kann.

```
<?php
 $table="<table border=”2”><tr><td>This table was created by
          using PHP</td></tr></table>";
 $text =  "<h1>This site was created with PHP</h1>";
 echo $text;-- prints the text stored above into the HTML file
 print $table;   -- prints the table defines above into the
                    HTMLfile
 echo "<br>";    -- prints an empty space
 print "<p>Hello</p>";     -- prints the text Hello into the
                             HTML file
?>
```

Code 19 : HTML-Code mit PHP erzeugen

4.2.8.2.2 PHP in einem HTML-Formular

PHP wird häufig in HTML-Formularen eingesetzt. Hierzu wird ein auf dem Server befindliches PHP-Programm benötigt, welches innerhalb eines HTML-Formulars mit dem `action`-Attribut referenziert wird.

Code 20 zeigt das HTML-Dokument aus Code 17. Der Unterschied besteht allerdings darin, dass beim Abschicken dieses Formulars nicht der Perl-Interpreter, sondern der PHP-Interpreter vom Server aufgerufen wird und die Daten auswertet.

```
<html>
<head>
<title>Formmailer in PHP</title>
</head>
<body bgcolor="lightyellow">
<h1>Please insert your data</h1>
<form method="POST" action="back.php"> --invokes the back.php
file on the Apache server by pressing the submit button
<table border="3">
```

```
<tr>
<td>Forename:</td><td><input type="text" name="forename"></td>
</tr>
<tr>
<td>Surname:</td><td><input type="text" name="surname"></td>
</tr>
<tr>
<td>Tel.:</td><td><input type="text" name="tel"></td>
</tr>
<tr>
<td>E-mail:</td><td><input type="text" name="mail"></td>
</tr>
<tr>
<td><input type="submit" value="Send"><td>
<td><input type="reset" value="New"></td>
</tr>
</table>
</form>
</body>
</html>
```

Code 20 : Formular (contact.php) für PHP-Antwort

Abbildung 27 zeigt die Darstellung von Code 20 im Netscape Navigator.

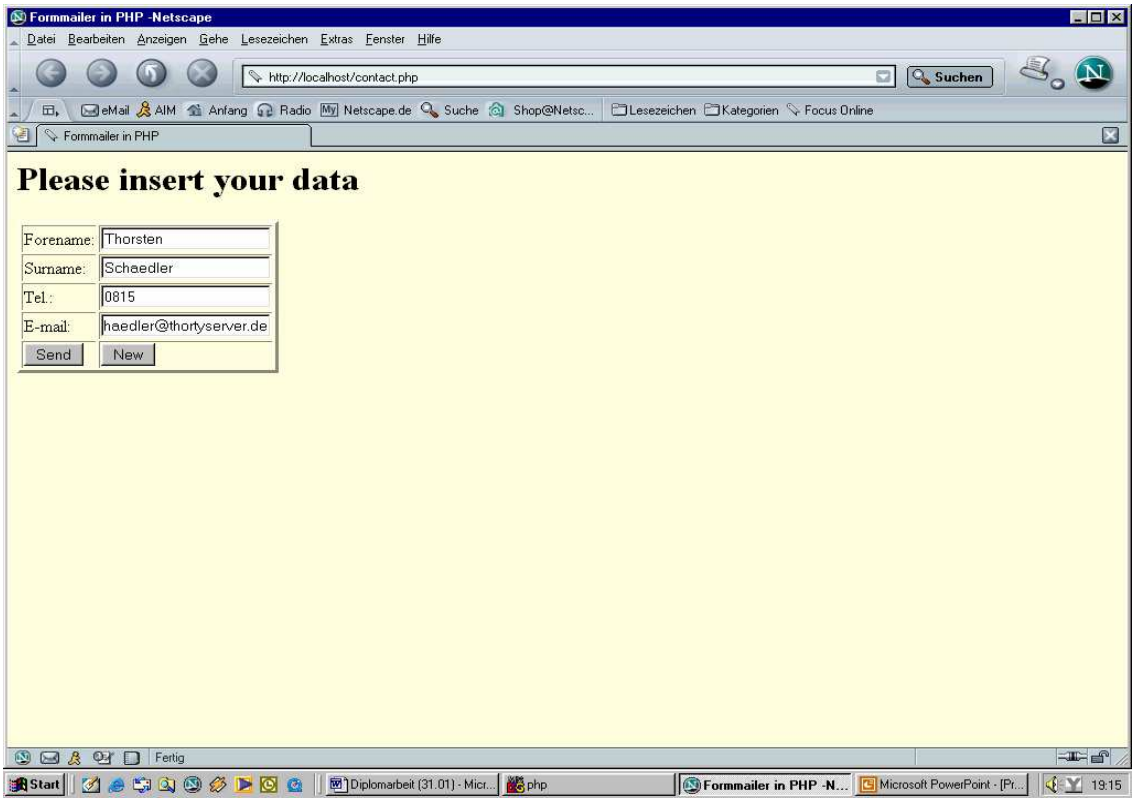

Abbildung 27 : HTML-Formular (contact.php) für PHP-Antwort

Um die vom Benutzer eingegebenen Daten für eine entsprechende Antwort verwenden zu können, müssen diese Eingaben in einem PHP-Skript verarbeitet werden. Eine solche Verarbeitung ist in Code 21 dargestellt.

```
<html>
  -- the html-file has a lightyellow background
<body bgcolor="lightyellow">
<h1>Thank you!</h1>
<br>
<p style="font-size:18">Hello <i style=”color:red”>
-- PHP generated Output is in red color
-- the following PHP-Code puts the values of forename and
   surname into the server response
<?php
 print  $_POST["forename"]." ".$_POST["surname"];
?> </i> , thanks for your interest.</p>
<p style="font-size:18">Your data will be saved</p>
</body>
</html>
```

Code 21 : PHP-Code (back.php) für Antwort des Apache Servers[153]

Abbildung 28 zeigt die vom Apache Server[154] generierte Antwort (Code 21). Der mittels PHP erzeugte Text (Eingaben aus den Textfeldern mit Namen `forename` und `surname` des HTML-Formulars aus Code 20) wird dabei kursiv und in roter Farbe dargestellt.

[153] Basiernd auf [KrWeG03, S.41]
[154] S.a. 4.2.6.4 .

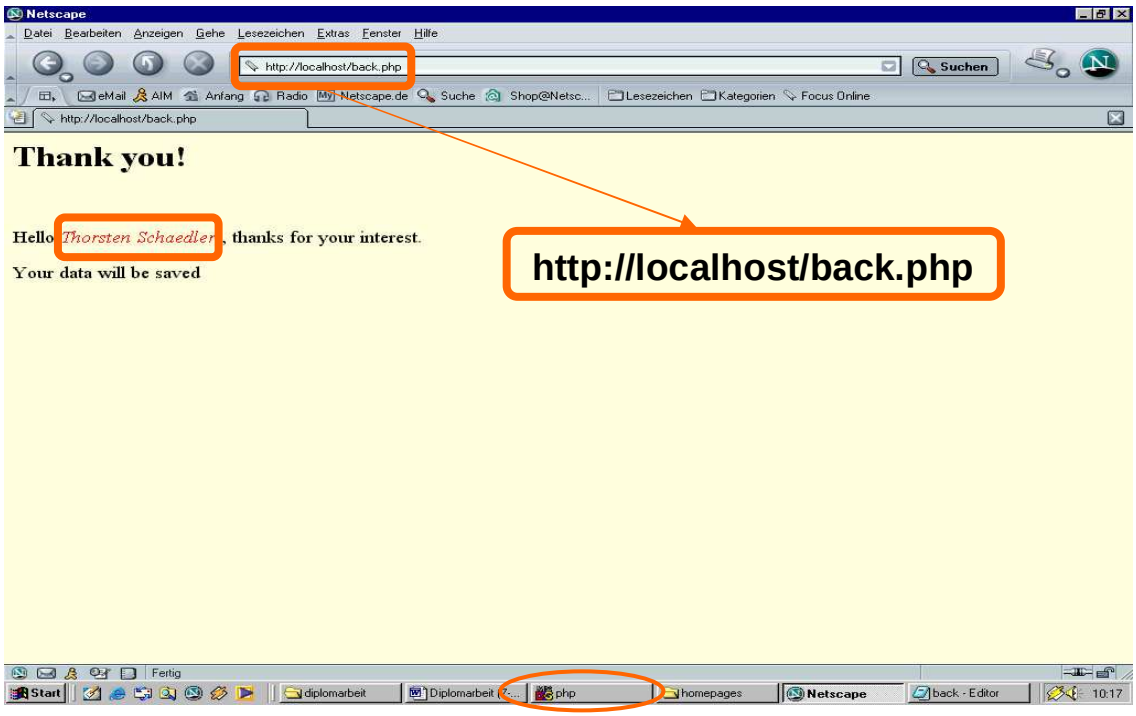

Abbildung 28 : PHP-Antwort (back.php) des Apache Servers

4.2.9 JSP – Java Server Pages

Bei Java Server Pages (JSP) handelt es sich um Textdokumente welche neben statischem Text, der mittels einem textbasierten Format wie HTML[155], XML[156] oder einem XML-Derivat[157] ausgezeichnet wird, noch JSP-Elemente enthalten, welche den dynamischen Inhalt der Seite generieren [Vgl. Sun03, S.643].

Das Standardformat zum Generieren von statischem Text ist HTML. Um ein anderes Format verwenden zu können, muss eine *page*-Direktive mit dem `content-type` Attribut, welches Angaben zum entsprechenden Format enthält, am Beginn der JSP-Seite eingefügt werden [Vgl. Sun03, S.655].

JSP laufen auf dem Server ab und bieten dabei eine Vielzahl an Entwicklungstechniken wie die Verwendung von Java-Code gemischt mit HTML-Code, Java Beans[158] und das Erstellen von eigenen Tags [Vgl. BePiW03b].

---

[155] S.a. 3 .
[156] S.a. 5 .
[157] S.a. 6 .
[158] S.a. 4.2.9.6 .

JSP sind wie HTML aufgebaut, und ermöglichen es Standard HTML und neue Skript-Tags miteinander zu kombinieren [Vgl. BePiW03b].

Diese werden anschließend in so genannte *Servlets*[159] umgewandelt und kompiliert, wenn sie zum ersten Mal aufgerufen werden. Das resultierende Servlet setzt sich aus den HTML Elementen und dem dynamischen JSP Code zusammen. JSP's müssen aber keinesfalls HTML Code enthalten [Vgl. BePiW03b].

4.2.9.1 Grundelemente[160]

Die JSP Elemente sind die Komponenten, die den dynamischen Teil einer JSP betreffen. In der JSP Spezifikation gibt es die Grundelemente *Direktiven*, *Skriptelemente* und *Standardaktionen*.

4.2.9.1.1 Direktiven

Direktiven dienen als Nachrichten, welche von einer JSP-Seite zum JSP-Container übermittelt werden. Eine Direktive ist innerhalb eines Tags mit dem Zeichen @ gekennzeichnet und ist in der gesamten JSP-Datei gültig.

Eine JSP Seite kennt die drei Direktiven *page*, *include* und *taglib*.

Mit der *page*-Direktive werden wichtige Attribute definiert. Dabei können in einer JSP-Seite mehrere page-Direktiven enthalten sein. Diese werden während des Übersetzungsvorgangs gesammelt und zusammen auf die Seite angewendet.

Die *taglib*-Direktive ermöglicht es benutzerdefinierte Tags in der JSP-Seite zu verwenden.

Die *include*-Direktive benachrichtigt den Container, dass in dieser JSP-Seite der Inhalt einer Ressource an der angegebenen Position eingebettet werden soll.

---

[159] S.a. 4.2.9.2 .
[160] Nach [BePiW03b].

4.2.9.1.2 Skriptelemente

Skriptelemente werden benötigt um Skriptcode in die JSP-Seite einzubinden, Variablen und Methoden zu deklarieren und Ausdrücke auszuwerten.

In Java Server Pages können drei Skriptelemente verwendet werden.

- Eine *Deklaration* ist ein Java Code Abschnitt welcher innerhalb der Zeichen <%! und %> vorgenommen wird und für die gesamte Klasse gültig ist.
- Bei einem *Scriptlet* handelt es sich um Java Code, der innerhalb der Zeichen <% und %> angegeben und während der Anfrageverarbeitung ausgeführt wird.
- Ein *Ausdruck* ist die Kurzform eines Scriptlets und wird innerhalb von <%= und %> angegeben.

4.2.9.1.3 Standardaktionen

Aktionen sind spezifische Tags, die sich auf das Laufzeitverhalten einer JSP-Seite und auf die Antwort auswirken, welche zum Client zurückgesendet werden.

In der JSP-Spezifikation werden einige Standardaktionen aufgeführt, die einige Grundfunktionen zur Verfügung stellen.

Aktionen werden beispielsweise dazu verwendet um Java Beans mit einer JSP-Seite zu verknüpfen oder Anfragen an eine andere JSP-Seite, ein anderes Servlet oder eine andere statische Ressource weiterzuleiten.

4.2.9.2 Servlet[161]

Ein Servlet ist eine serverseitige Webkomponente, und stellt somit eine mächtige Erweiterung der Serverfunktionalität dar, mit welcher der Inhalt einer Web-

[161] Nach [BePiW03a] .

seite dynamisch generiert werden kann. Servlets basieren auf der plattformunabhängigen Java-Technologie und können somit auf allen gängigen Plattformen (Web-Servern) zum Einsatz kommen .

So kann auch der kostenlose Apache Web-Server für das Arbeiten mit Servlets bzw. JSP eingesetzt werden.

In ihrer Funktionalität liegen Servlets zwischen Servererweiterungen wie den Apache Modulen oder Netscape Server API und dem CGI[162].

Im Gegensatz zu CGI-Skripten bei denen für jede http Anfrage ein neuer und teurer Betriebssystemprozess gestartet wird, erzeugt beim Verwenden von Servlets die Java Virtual Maschine (JVM) für jede Anfrage einen eigenen Lightweight Java Thread.

„Ein *Thread* ist ein Ablauf innerhalb eines Programms, der unabhängig vom Rest des Programms abgearbeitet werden kann. Existieren mehrere Threads in einem Programm, so können diese für den Benutzer scheinbar parallel ausgeführt werden.“ [KnSW98, S.436] .

Bei n gleichen Anfragen existiert so nur eine Kopie der Servletklasse im Speicher wobei n Threads mit dieser Klasse arbeiten. So wird im Vergleich zur CGI-Technologie Speicherplatz und somit auch Kosten gespart, da dort für jede Anfrage der Programmcode in den Speicher geladen werden muss.

Im Gegensatz zu CGI-Skripten bleiben Servlets nach dem Abarbeiten der Anfrage im Speicher, wodurch die Möglichkeit besteht, komplexe Daten zwischen den Anfragen auszutauschen.

Um die von den Servlets erzeugten Antworten zu verarbeiten, werden verschiedene Netzwerkservices benötigt. Die Bereitstellung dieser Services als auch die Verwaltung der Servlets übernimmt der so genannte *Servlet-Container*. Darüber hinaus steuert und überwacht er den Ablauf des genau definierten Lebenszyklus eines Servlets.

---

[162] S.a. 4.2.5 .

4.2.9.3 Servlet-Container

Der *Servlet-Container* kann entweder als Erweiterung in einen Web- und Applikationsserver wie beispielsweise den Apache Server[163], IBM WebSphere, Sun One Web Server oder Sun Web Application Server eingebunden oder direkt in den Webserver integriert werden. Wichtig ist in jedem Fall, dass der Servlet Container das Hypertext Transfer Protokoll (http) unterstützt [Vgl. BePiW03a].

Die Verwendung eines Servlet-Containers zwischen dem Web-Server und dem JSP-Container bietet dabei den Vorteil, dass Entwickler von Web Applikationen lediglich die schnell und einfach zu erlernende Servlet-API kennen müssen um mit JSP vertraut zu werden und zu arbeiten. Des Weiteren verringert es die Komplexität bei der Implementierung eines JSP-Containers, da dieser vollständig auf der Servlet-API aufbaut [Vgl. BePiW03a].

Es besteht zudem die Möglichkeit, ohne großen Aufwand Code aus Servlets und anderen Klassen in JSP-Seiten zu verwenden und umgekehrt .

4.2.9.4 Der JSP-Container

Anhand der Dateiendung .jsp, erkennt der Web Server, dass es sich bei der angeforderten Ressource um eine Java Server Page handelt und somit von der JSP-Maschine behandelt werden muss. Wird die JSP-Seite zum ersten Mal aufgerufen oder ändert sich ihr Inhalt, so wird sie in eine Java Klasse transformiert, aus welcher im Anschluss ein Servlet kompiliert wird. Insofern kommt es lediglich in diesen Fällen zu kleinen Verzögerungen. Bei jedem normalen Aufruf wird die Anfrage direkt an das fertige Servlet im Speicher des Servers weitergeleitet [Vgl. BePiW03b].

Abbildung 29 veranschaulicht das Zusammenspiel von JSP-Dokumenten, Servlets, dem JSP-Container, dem Servlet-Container und dem Web-Server.

---

[163] S.a. 4.2.6.4 .

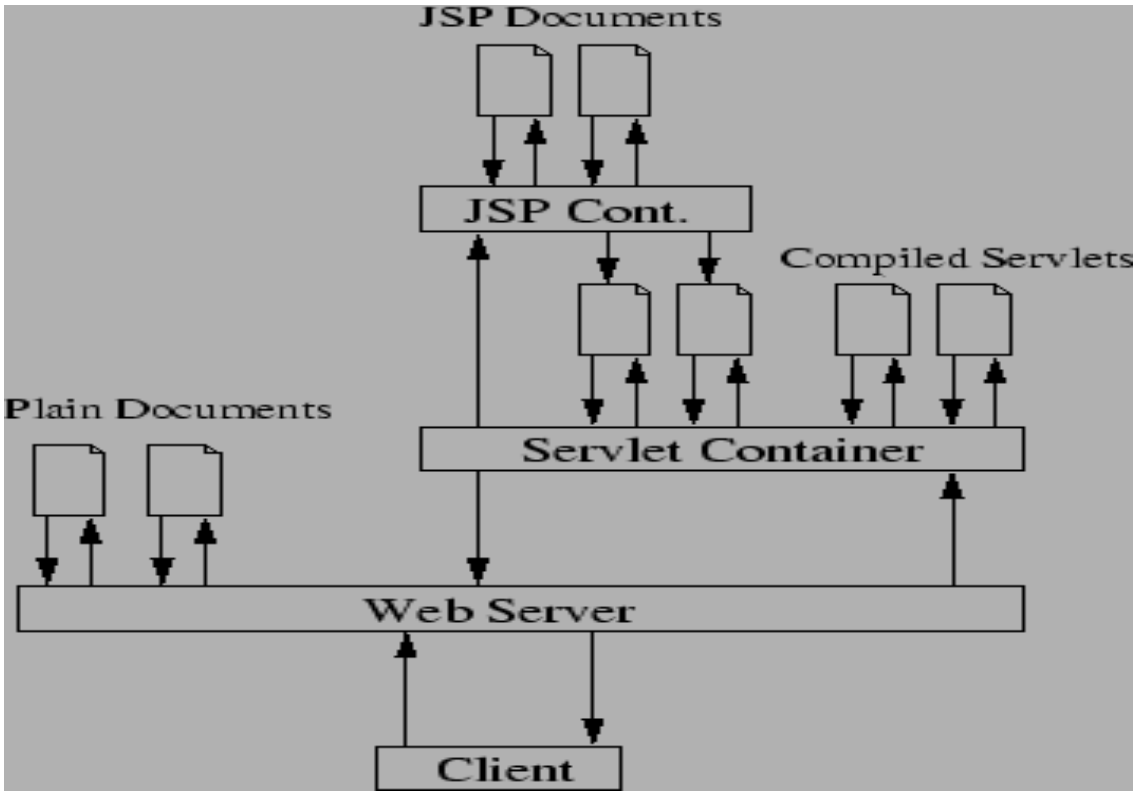

Abbildung 29 : Schichtmodell eines Web Application Servers[164]

4.2.9.5 Eigenschaften

Java Server Pages haben folgende wichtige Eigenschaften [Vgl. BePiW03b]:

- Können den Zustand des Servers zwischen einzelnen Anfragen speichern.
- Für jede Anfrage wird ein neuer Thread erzeugt.
- Es muss nach der Initialisierung nicht erneut geladen werden.
- Einmalige Übersetzung in Byte-Code, welcher lediglich bei Änderung des Inhalts neu erstellt werden muss.
- Laufen auf allen wichtigen Web Servern ab.
- Durch die Verwendung von Java Beans, Enterprise Java Beans und eigenen Tag Bibliotheken ist eine bessere Trennung von Daten und Programmcode möglich.
- JSP laufen auf dem Server ab und sind daher universell einsetzbar.

[164] Entnommen aus [BePiW03b] .

4.2.9.6 Java Beans

Eine der leistungsstärksten Aspekte bei der Verwendung von JSP besteht in der Möglichkeit des Einsatzes von Java Beans. Java Beans sind wieder verwertbare Softwarekomponenten, wie beispielsweise ein Button (GUI-Element) oder eine ganze Applikation. Sie wurden erstmals 1996 während der Java One Konferenz der Weltöffentlichkeit vorgestellt [Vgl. BePiW03c].

Um JavaBeans in eine JSP Seite einzubinden, können die drei Standardaktionen[165] `jsp:useBean`, `jsp:setProperty` und `jsp:getProperty` verwendet werden [Vgl. BePiW03b].

4.2.9.7 Benutzerdefinierte Tags

Das Verwenden von JavaBeans in JSP ist allerdings auch mit Problemen verbunden. So besteht der Quellcode aus HTML und Java-Scriptlets und wird mit jedem zusätzlichen Scriptlet komplexer und damit auch unübersichtlicher. Auch gibt es keine klare Trennung zwischen Präsentation und Logik. Des Weiteren sind die Tags für das Einbinden von Java Beans statisch und dadurch sehr begrenzt [Vgl. BePiW03b].

Um diesen Problemen entgegenzuwirken werden benutzerdefinierte Tags eingeführt, welche mit XML definiert werden [Vgl. BePiW03b].

4.2.10 ASP – Active Server Pages

ASP steht für Active Server Pages (.asp.) und wurde von Microsoft ins Leben gerufen. ASP sind der bedeutendste Konkurrent zur JSP[166] Technologie und haben ähnliche Methoden zum serverseitigen Erzeugen von dynamischen Webseiten [Vgl. BePiW03c].

---

[165] S.a. 4.2.9.1.3 .
[166] S.a. 4.2.8 .

So werden auch ASP Seiten auf dem Server ausgeführt und das Ergebnis in Form von HTML, XML oder XML-basierten Dokumenten an den Client gesendet.

Im Gegensatz zu JSP können ASP allerdings nicht auf allen Servern eingesetzt werden, sondern sind auf den Microsoft IIS und den kleineren PWS beschränkt [Vgl. BePiW03c].

Im Gegensatz zu PHP ist ASP keine Skriptsprache sondern eine lizenz- und kostenpflichtige Umgebung in welcher Vorzugsweise VBScript und JScript eingesetzt werden. VBScript[167] ist dabei die Standardskriptsprache von ASP [Vgl. Sht03i].

Um die Standardskriptsprache zu ändern, muss wie in JSP am Anfang der ASP Datei mit `<%@ language="...." %>` die neue Standardskriptsprache angegeben werden. So kann beispielsweise auch Object Rexx[168] als Standardskriptsprache für ASP verwendet werden [Vgl. WinB04].

Die Skripte in einer ASP Datei werden von der ASP-Umgebung auf dem Server ausgeführt [Vgl. Sht03i].

In ASP können auch mehrere Skriptsprachen verwendet werden. Hierzu wird das HTML `script`-Element zusammen mit dem `language`-Attribut und dem `runat`-Attribut in die ASP Datei eingebettet [Vgl. WinB04].

4.2.11 ASP.NET

ASP.NET ist die nächste Generation von ASP und eine Grundstruktur für die Programmierung, welche auf der Common Language Runtime (CLR) basiert und auf einem Server zum Erstellen leistungsstarker Webanwendungen eingesetzt werden kann [Vgl. gbn03] .

ASP.NET Seiten haben die Endung .aspx .

---

[167] S.a. 4.2.2.4 .
[168] S.a. 4.2.2.2 .

ASP.NET ist Teil des .NET Framework von Microsoft, mit welchem Web Applikationen und XML Web Services erstellt werden können. ASP.NET-Seiten und XML Web-Services[169] Dateien enthalten serverseitige Logik, welche in Microsofts Visual Basic.NET, Visual C#.NET oder einer anderen .NET-Framework kompatiblen Sprache geschrieben wurde. [Vgl. msAN03].

Gegenüber ASP hat ASP.NET einige Neuerungen. So verfügt ASP.NET mit der Verwendungsmöglichkeit von ADO.Net und der Unterstützung von Visual Basic, C++ und C# über eine bessere Sprachunterstützung [Vgl. W3s03g].

ASP.NET enthält ein umfangreiches Angebot an HTML-Controls. Beinahe alle HTML-Elemente einer Seite können als ASP.NET Control-Objekte definiert werden, die wiederum über Skripte kontrolliert werden können. Darüber hinaus basieren ASP.NET Komponenten auf XML und alle ASP.NET Objekte einer Web Seite können Events auslösen, welche mittels ASP.Net-Code bearbeitet werden können [Vgl. W3s03g].

4.2.12 RSP - Rexx Server Pages

Rexx Server Pages (RSP) sind den Active Server Pages[170], Java Server Pages[171] und PHP[172] prinzipiell sehr ähnlich, da auch hier in einer RSP-Datei neben normalen HTML-Elementen auch eigene Bereiche definiert werden können, welche (Object-) Rexx-Code enthalten. Diese Bereiche werden vom Web-Server entsprechend bzw. dynamisch ausgefüllt.

Prinzipiell stehen drei Möglichkeiten zur Verfügung um Rexx-Anweisungen innerhalb einer RSP-Datei von normalem HTML abzugrenzen [Vgl. AshD03b]:

- **<?rexx** /* rexx statement */ **?>**
- **<script type="rexx">** /*rexxstatement */ **</script>**
- **<script language="rexx">** /*rexxstatement */ **</script>**

[169] S.a. 7 .
[170] S.a. 4.2.9 .
[171] S.a. 4.2.8 .
[172] S.a. 4.2.7 .

Um RSP-Dateien auf dem Apache Server[173] verwenden zu können muss Object Rexx[174], Regina Rexx oder ein anderer Rexx-Interpreter auf dem System installiert sein und *Mod_Rexx* muss in den Apache Server integriert werden [Vgl. AshD03a].

4.2.12.1 Mod_Rexx[175]

Da im Rahmen dieser Arbeit der Apache Server 1.3.6 verwendet wurde, kann aus Gründen der Kompatibilität nicht die aktuellste Version 2.0 sondern lediglich Version 1.2[176] verwendet werden.

Das Apache-Modul Mod_Rexx kann derzeit unter Windows, Unix, Os/2 und Linux eingesetzt werden und ist vergleichbar mit anderen Apache-Modulen wie Mod_Perl und Mod_PHP. Die Datei *Mod_rexx.dll* muss dabei in das Verzeichnis *\Apache\ modules* und die Datei rspcomp.rex (Compiler) in das Verzeichnis *\Apache\bin* kopiert werden .
Um Rexx CGI Programme unter Mod_Rexx zum Laufen zu bringen, müssen innerhalb der Apache Konfigurationsdatei *httpd.conf* einige Änderungen vorgenommen werden, welche in der Dokumentation (Readme.html) von Mod_Rexx erklärt werden.

Über die Mod_Rexx kann ein Rexx Programmierer volle Kontrolle über den Apache Anfrageprozess erlangen. Eine dem Apache eingehende Anfrage wird in Phasen unterteilt, wobei jede dieser Phasen einen Teil der Anfrage verarbeitet.

4.2.12.2 RSP und HTML[177]

Genau wie PHP oder Perl kann auch RSP innerhalb eines HTML-Formulars verwendet werden. Hierzu muss in Code 20 lediglich im Attribut `action` *“back.php“* durch *“back.rsp“* ersetzt werden.

---

[173] S.a. 4.2.5.4 .
[174] S.a. 4.2.2.2 .
[175] Basierend auf [AshD03a] .
[176] http://www-124.ibm.com/ developerworks/projects/modrexx , Abruf am 2003-12-09 .
[177] Basierend auf [AshD03b] und [AshD03c].

In Code 22 wird dabei die REXX Variable **wwwargs.n.!value** verwendet um den Inhalt der Inputfelder aus dem HTML-Dokument in die RSP-Datei zu übernehmen.

```
<html>
<body bgcolor="lightyellow">
<h1>Thank you!</h1><br>
<p style="font-size:18">Hello <i style="color:red">
<?rexx
say wwwargs.1.!value
say wwwargs.2.!value
?>
</i>
, thanks for your interest.<br><br>Your data will be saved</p>
<hr>
<p>You are using the <I style="color:blue">
<script type="rexx">
say wwwhttp_user_agent
</script>
</i>as your Browser</p>
</html>
```

Code 22 : Antwort des Apache Servers (back.rsp)

Im Folgenden soll nun gezeigt werden wie mit Hilfe von Object Rexx der Zugang zu einer Seite per Passwortabfrage gewährleistet werden kann. Dabei wird in einem HTML-Dokument die Eingabe des Namens und Passworts verlangt. Nachdem das Formular abgeschickt wurde, wird innerhalb der vom Apache Server aufgerufenen RSP-Datei „password.rsp“ überprüft, ob die Passworteingabe des Anwenders dem Wert “ORexx“ entspricht. Ist dies der Fall so wird die Seite mit dem Inhalt aus Code 24 gefüllt und verschiedene Rexx Variablen werden in einer Tabelle dargestellt. Ist die Passworteingabe allerdings ungültig, so erscheint das Apache Standard-Dokument FORBIDDEN 401 und teilt dem Anwender mit, dass der Zugriff auf den Inhalt der RSP-Datei verweigert wurde.

Code 23 zeigt das Dokument, in welches der Name und das Passwort eingetragen werden müssen. Beim Drücken des „*Send*“-Button wird die Datei password.rsp aufgerufen.

Diese Datei ist in Code 24 dargestellt.

```
<html>
<head>
<title></title>
</head>
<body bgcolor="lightyellow">
<h1>Please insert your Name and your password !</h1>
 -- the RSP file is invoked by pressing the submit button
<form method="POST" action="password.rsp">
<table border="3">
<tr>
<td>Name:</td><td><input type="text" name="name"</td>
</tr>
<tr>
<td>Password:</td><td><input type="password"
name="password"</td>
</tr>
<tr>
<td><input type="submit" value="Send"></td>
<td><input type="reset" value="New"></td>
</tr>
</table>
</form>
</body>
</html>
```

Code 23 : Passwortabfrage mit Object Rexx (1)

Abbildung 30 zeigt die Darstellung von Code 23 im MS Internet Explorer.

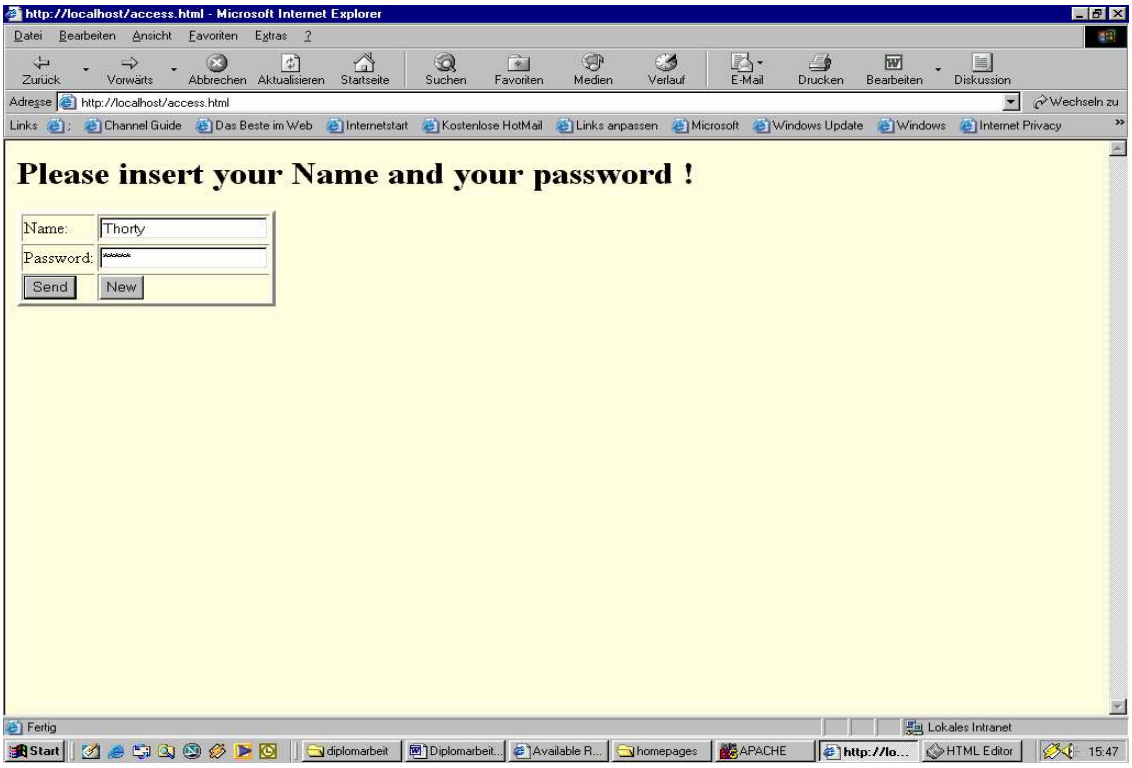

Abbildung 30 : Passwortabfrage mit Object Rexx (1)

In Code 24 wird anschließend das Passwort mit Hilfe der Rexx-Variablen **wwwargs.2.!value** überprüft.

```
<html>
<head>
<style type="text/css">                -- this RSP file uses CSS
#color {color:blue}
</style>
</head>
<body bgcolor="lightyellow">
<script type="rexx">
-- Begin of Object Rexx Script
FORBIDDEN = 401                        -- Main document forbidden
-- the user input of the password field is saved in the
   variable “wwwargs.2.!value”. If the entered password is
   correct (ORexx) than the user gets access
if wwwargs.2.!value = 'ORexx' then
say Hello wwwargs.1.!value  -- Content of the Rexx Server Page
-- if the password is incorrect the Apache Main document
   FORBIDDEN 401 appears
else return FORBIDDEN
 -- End of Object Rexx script
</script><hr>
 -- Content of the Rexx Server Page
<p style="font-size:18pt"><b>You are now able to see some <i
id="color">Rexx variables.</i></b></p>
<table border="3">
<tr><td>Your Browser :</td><td id="color">
<script language="rexx">
say wwwhttp_user_agent                          -- Rexx variable
</script>
</td></tr>
<tr><td>Your data :</td><td id="color">
<script type="rexx">
say wwwpost_string                              -- Rexx variable
</script>
</td></tr>
<tr><td>The Server-Software :</td><td id="color">
<script language="rexx">
say wwwserver_software                          -- Rexx variable
</script>
</td></tr>
</table>
<p>Today is a the <i id="color">
<script language="rexx">
say date()                                      -- Rexx function
</script>
</i>and you opend this side at <i id="color">
<script type="rexx">
say time()                                      -- Rexx function
</script></p></html>
```

Code 24 : Passwortabfrage mit Object Rexx (2) – (password.rsp)

Abbildung 31 zeigt das vom Apache Server geschickte FORBIDDEN 401 Dokuments, welches bei Eingabe eines falschen Passworts im Browser des Anwenders angezeigt wird.

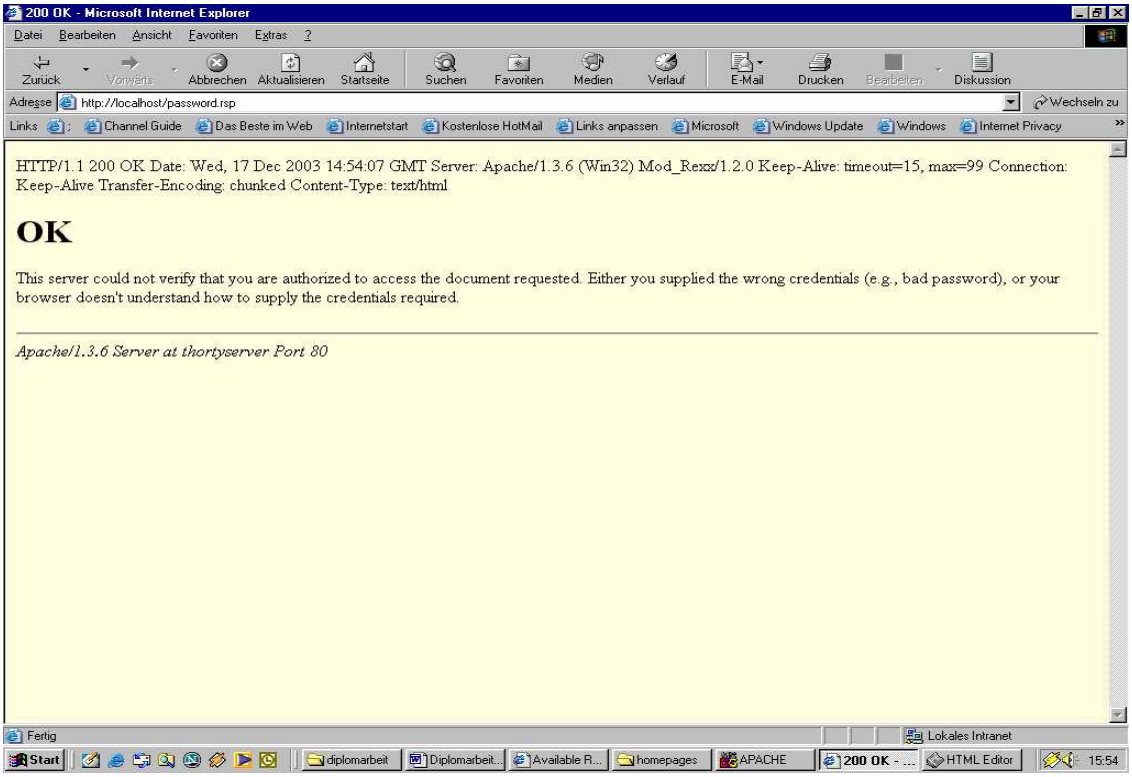

Abbildung 31 : Falsches Passwort - Forbidden = 401

Ist hingegen das Passwort korrekt, so wird der Anwender mit dem von ihm eingegebenen Namen (`wwwargs.1.!value`) begrüßt. Daran anschließend wird in einer HTML-Tabelle gezeigt welchen Browser der Anwender verwendet, wie der entsprechende Datenstrom aussieht und welche Serversoftware verwendet wurde. Hierfür werden die Rexx-Variablen ***wwwhttp_user_agent***, **wwwpost_*string*** und ***wwwserver_ software*** verwendet.

Auch werden mit den Rexx Funktionen ***date()*** und ***time()*** das aktuelle Datum als auch die Uhrzeit angegeben [Vgl. Fla03b, S.8].

Abbildung 32 soll dies im Folgenden nochmals veranschaulichen.

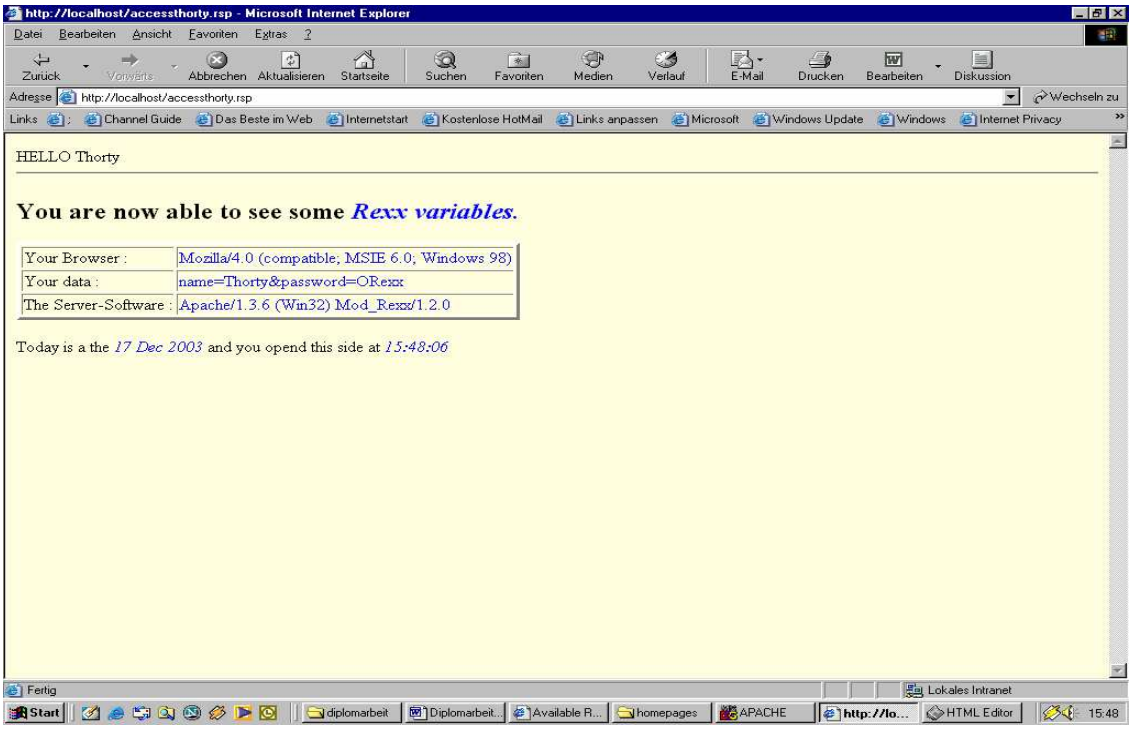

Abbildung 32 : Korrektes Passwort

4.2.12.3 HTML-Dokument mit Object Rexx erzeugen[178]

Sind Mod_Rexx und der Object Rexx Interpreter auf dem Apache Server installiert, so kann genau wie Perl oder PHP auch Object Rexx dazu verwendet werden ein HTML-Dokument serverseitig dynamisch zu erzeugen und als reines HTML an den Client zu senden. Hierzu wird die Rexx-Anweisung `say` verwendet. So kann in einer (Object-) Rexx-Datei der entsprechende Inhalt generiert werden. Das Beispiel in Code 25 soll dies nun veranschaulichen.

```
say "<html>"
say "<head>"
say "<title>Using Mod_Rexx</title>"
say "<body bgcolor='lightyellow'>"
say "<h1 style='color:red;align:center'>This is a Rexx
      Page</h1>"
say "<img src='C:\Programme\Wamp\homepages\powrexx.jpg'>"
say "<p>This side was created by using Object Rexx</p>"
say "</body>"
say "</html>"
```

Code 25 : RexxToHTML.rex

[178] Basierend auf [AshD03d] .

Abbildung 33 zeigt die Darstellung von Code 25 im MS Internet Explorer.

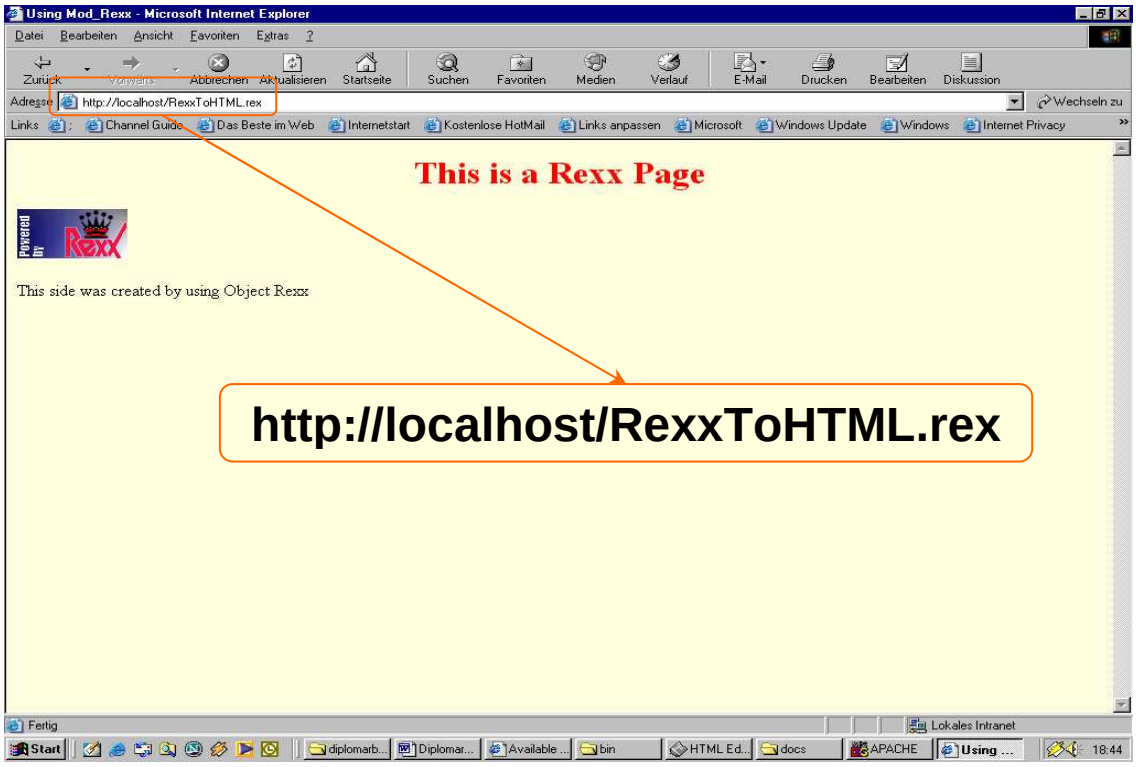

Abbildung 33 : HTML-Dokument mit Object Rexx erzeugen

4.3 ActiveX/OLE

ActiveX ist der Marketingname von Microsoft für **OLE** (**O**bject **L**inking and **E**mbedding) [Vgl. Kurz03, S.11].

ActiveX/OLE ist ein Überbegriff für verschiedene Softwarekomponenten, welche auf dem Component Object Model (COM)[179] basieren und mit dessen Hilfe Windows Anwendungen, die OLE unterstützen, ferngesteuert und miteinander verknüpft werden können [Vgl. Kurz03, S.11].

Mit Hilfe von ActiveX/OLE werden spezifische Eigenschaften des Microsoft Windows Betriebssystem für Web-Seiten nutzbar. So besteht beispielsweise die Möglichkeit, mit Hilfe einer Skriptsprache wie Objekt Rexx[180], den Internet

[179] S.a. 4.3.1 .
[180] S.a. 4.2.2.2 .

Explorer fernzusteuern oder Daten aus einem HTML-Formular[181] direkt in ein Word-Dokument oder eine Excel-Tabelle einzulesen (und umgekehrt)[182] [Vgl. MüNe01, S.54].

ActiveX besteht im wesentlichen aus ActiveX-Controls, Active-Documents und Active-Scripting und wird derzeit lediglich vom MS Internet Explorer direkt ausgeführt. Für Netscape gibt es ein ActiveX-Plugin [Vgl. MüNe02, S.88].

ActiveX-Controls sind Programme die sich wie Java Applets in ein HTML-Dokument einbinden lassen. Der Programmcode wird dabei im Arbeitsspeicher des Client-Rechners ausgeführt [Vgl. MüNe02, S.88].

Im Gegensatz zu Java Applets sind sie jedoch stärker in der Windows Welt verankert [Vgl. MüNe01, S.258].

ActiveX-Controls können über das `object`-Element in ein HTML-Dokument als Objekt eingebunden werden. Dabei muss das ActiveX-Control über den Klassenbezeichner ClassID (`CLSID`) referenziert werden. Über diese ID kann jede Applikation, welche OLE unterstützt, eindeutig identifiziert werden. Mit der `ClassID "05589FA1-C356-11CE-BF01-00AA0055595A"` kann beispielsweise ein ActiveX-Control in Form des Windows Media Player in ein HTML-Dokument eingebunden werden, welcher es erlaubt sämtliche dem Media Player bekannte Sound- und Videodateien[183] abzuspielen. Ein ActiveX-Control wird mit Hilfe der `name`- und `value`-Attribute innerhalb des `param`-Elements weiter spezifiziert [Vgl. MüNe02, S.347f].

Active-Documents dient zur Anzeige von Nicht-HTML-Dokumenten wie beispielsweise PDF-, Word- oder Excel-Dokumenten in einem Browser [Vgl. DSZ03].

„Axtive-Scrpting ermöglicht das Verwalten und die Kommunikation von ActiveX-Contols“ [DSZ03].

---

[181] S.a. 3.5.5 .
[182] Vergleiche hierzu 4.5.5.1 und 5.3.4 .
[183] S.a. 3.5.3 .

4.3.1 COM[184]

„Das Component Object Model (COM) ist ein objektorientiertes Programmiermodell für die Verknüpfung und Steuerung von unterschiedlichen Anwendungen.“[Kurz03, S.10].

Es ist wie ActiveX eine Microsoft Technologie und beinhaltet ein Objektmodell, welches eine Interaktion zwischen Objekten ermöglicht.

Eine Komponente des COM ist ein in sich abgeschlossener, programmiersprachenneutraler und plattformunabhängiger Softwarebaustein. Sie besteht dabei aus einer COM-Klasse, welche zur Instanzierung des Objektes dient, dem COM-Objekt, welches eine Instanz der COM-Klasse darstellt und einem oder mehreren Interfaces, welche die zur Kommunikation notwendigen Eigenschaften und Methoden bereitstellt.

Ein solches *COM-Objekt* kann mit Hilfe der Windows Script Components (WSC) auf Basis von Skriptcode erstellt werden [Vgl. Kurz03, S.15].

Ein solches *COM-Interface* wird durch einen Interface Identifier (IID) eindeutig identifiziert und bildet die Schnittstelle des Objekts bzw. das Bindeglied zwischen der aufrufenden Anwendung und der Komponente. Bevor eine Komponente allerdings verwendet werden kann, muss sie mittels des Global Unique Identifier (GUID) in der Windows Registrierung eingetragen werden.

4.3.2 OLE Automation

OLE Automation ist eine Sonderform eines COM Interfaces und ermöglicht einem Client (aufrufende Anwendung), ein COM-Objekt aufzurufen ohne das genaue COM-Interface mit dessen Methoden und Eigenschaften kennen zu müssen [Vgl. Kurz03, S.12].

---

[184] Nach [Kurz03,10f] .

4.3.3 WSH

In den ersten MS Windows Betriebssystemen gab es noch keine Möglichkeit zum programmgesteuerten Zugriff auf die Windows Systemumgebung. Dies änderte sich jedoch mit der Einführung von Windows 98, in welchem der Windows Script Host (WSH) integriert wurde. Der Windows Script Host ist sprachenunabhängig und in Windows 98, ME, 2000 und XP enthalten. Für Windows 95 und NT kann er kostenlos von der Microsoft Homepage herunter geladen werden. Der WSH erlaubt den Einsatz beliebiger Skriptsprachen, unterstützt den Ablauf von Skript-Dateien unter Windows und ermöglicht die Fernsteuerung von Windows Anwendungen über die OLE-Schnittstelle [Vgl. Kurz03, S.5].

Der WSH enthält verschiedene Windows Script Engines (WSE). Mit der Installation von Object Rexx[185] wird eine solche Script Engine für den WSH zur Verfügung gestellt.

Für den Zugriff auf die Windows Systemressourcen stellt der WSH einige COM Objekte zur Verfügung (WSH-Objektmodell) [Vgl. Kurz03, S.5].

4.4 DHTML - Dynamisches HTML

DHTML ist ein Überbegriff für das Zusammenspiel von HTML[186], CSS[187] und Skriptsprachen[188] zum dynamischen Ändern eines HTML-Dokumentes.

Es gibt dabei verschiedene Ansätze und Möglichkeiten DHTML zu erzeugen.

4.4.1 Das DHTML Modell von Microsoft

Das 1997 mit dem MS Internet Explorer 4.0 eingeführte Konzept, ermöglichte auf alle beliebigen HTML-Elemente inklusive deren CSS-Eigenschaften zuzugreifen und diese zu ändern, auszutauschen, zu löschen, neu zu erzeugen

[185] S.a. 4.2.2.2 .
[186] S.a. 3 .
[187] S.a. 4.1 .
[188] S.a. 4.2.2 .

und dergleichen. Die Vorraussetzung dieser Effekte sind beim Internet Explorer die bereits angesprochenen Cascading Style Sheets [Vgl. MüNe02, S.1001].

Mittels einer Skriptsprache kann auf ein Element zugegriffen werden, und durch Angabe neuer Style Sheets (Attributwerte) das Aussehen des Element dynamisch geändert werden. Hierbei ist, nach dem Ansatz des MS Internet Explorer ab Version 4.0 (1997), das aus der JavaScript Objekthierarchie[189] bereits bekannte `all`-Objekt[190] der Schlüssel zu DHTML. Dieses `all`-Objekt gehört allerdings nicht zum offiziellen JavaScript Sprachstandard, sondern wurde von Microsoft für den Internet Explorer 4.0 implementiert [Vgl. MüNe01, S.560].

Eine wichtige Vorraussetzung für Dynamisches HTML ist in vielen Fällen die Interaktion zwischen Anwender und dem angezeigten HTML-Dokument. Hierbei sind die bereits erläuterten Event-Handler[191] als auch das `event`-Objekt[192] von besonderer Bedeutung [Vgl. MüNe02, S. 1009}.

Der Internet Explorer reagiert beim Einfügen von dynamischem Inhalt wie eine Textverarbeitungssoftware und positioniert dabei automatisch alles was dahinter stand neu. Der 1997 dominierende Netscape Navigator war hierzu allerdings nicht in der Lage. Für Seiten, welche mit dem Netscape Navigator dargestellt werden sollten, musste also weiterhin die umständliche *Layer-Technik* eingesetzt werden [Vgl. MüNe02, S.1001].

Obwohl dieses Microsoft Modell ein firmeneigener Vorstoß ohne eine Absprache mit einem unabhängigen Normierungs-Gremium (W3C) war, war es dennoch richtungweisend. Der Internet Explorer eroberte von Monat zu Monat mehr Marktanteile und ist heute der am weitesten verbreitete Browser [Vgl. MüNe02, S.1001].

Abbildung 34 zeigt sowohl das kontinuierliche Wachstum des MS Internet Explorers als auch den gravierenden Rückgang in der Verwendung des Netscape Navigators seit 1996.

---

[189] S.a. 4.2.2.1.2.2 .
[190] S.a. 4.4.1.1 .
[191] S.a. 3.6.1 .
[192] S.a. 3.6.2 .

In wie weit der DHTML-Ansatz von Microsoft diesen Trend beeinflusst bzw. beeinflusst hat, kann im Rahmen dieser Arbeit nicht gesagt werden.[193] Die Grafik soll lediglich die Entwicklung im Browsermarkt verdeutlichen.

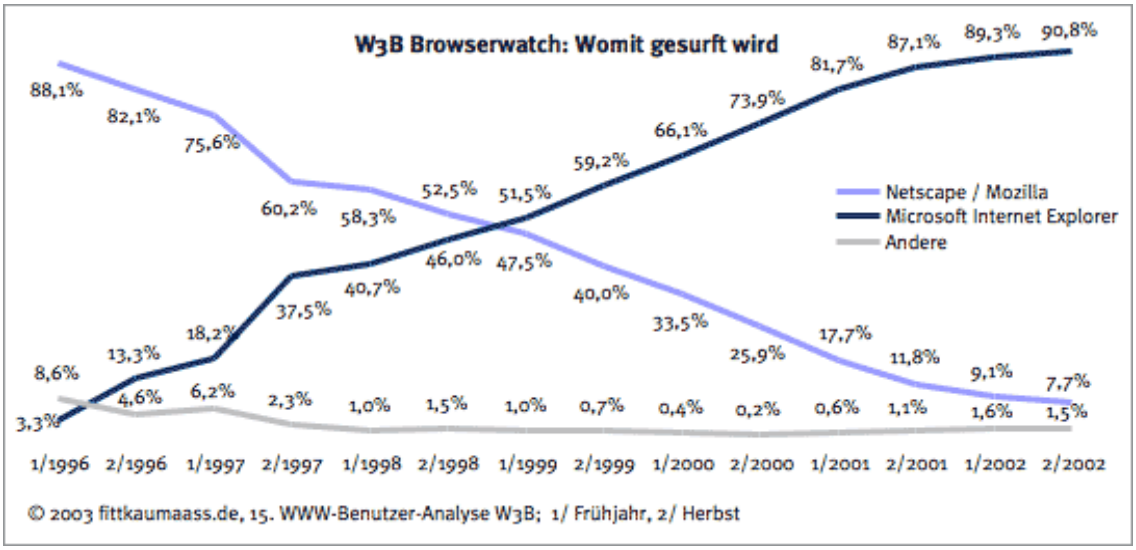

Abbildung 34 : Browserentwicklung[194]

Insofern war das DHTML-Modell von Microsoft ein wichtiger Meilenstein auf dem Weg zum heute empfohlenen Document Object Model (DOM) [Vgl. MüNe02, S.1002].

Mit den neuen Browser Generationen und dem DOM wurde das `all`-Objekt allerdings durch die neuen HTML-Elementobjekte und das `node`-Objekt verdrängt [Vgl. MüNe02, S.797].

Im Folgenden wird nun ein Einblick in dieses `all`-Objekt gegeben. Des Weiteren werden auch die JavaScript-Objekte `forms`, `images` und `links`, welche allesamt Unterobjekte des `document`-Objekts[195] sind, kurz vorgestellt.

---

[193] Eine weitaus größere Rolle bei dieser Entwicklung dürfte allerdings die Tatsache spielen, dass der MS Internet Explorer mit den MS Windows Betriebssystemen ausgeliefert wird. Alle anderen Browser müssen dagegen erst aus dem WWW herunter geladen werden.

[194] Entnommen aus http://www.w3b.org/trends/browserwatch.html , Abruf am 2004-01-29.

[195] S.a. 4.2.2.1.2.2 .

4.4.1.1 Das all-Objekt

Mit Hilfe des `all`-Objektes und dessen Eigenschaften und Methoden erlangt man mittels einer Skriptsprache Zugriff auf alle einzelnen Elemente und Attribute eines HTML-Dokumentes [Vgl. MüNe01, S.560].

Die meisten Eigenschaften können gelesen und geändert werden. Die Methoden des `all`-Objekts ermöglichen das Einfügen oder Entfernen von HTML-Elementen als auch von Angaben innerhalb eines HTML-Tag. Hierdurch wird ein dynamischer Zugriff auf alle Bestandteile eines Dokuments ermöglicht [Vgl. MüNe02, S.797].

4.4.1.2 Das style-Objekt

Durch das `style`-Objekt, welches ein Unterobjekt des `all`-Objektes ist, erhält man darüber hinaus Zugriff auf die Style Sheet Eigenschaften (Attribute) dieses HTML-Elementes. Somit wird es beispielsweise möglich die Größe, Farbe, Position eines Elementes dynamisch zu ändern [Vgl. MüNe01, S.578].

Beispiel : `document~all~any~style~color =“red“`

Darüber hinaus ist dieses `style`-Objekt auch im DOM[196] (Version 2.0) enthalten. Um auf die Methoden und Eigenschaften des `style`-Objekts zugreifen zu können, wird ein Elementknoten benötigt, auf welchen über die Methoden[197] `getElementById`(), `getElementByTagName`() oder `getElementByName`() des `document`-Objekts zugegriffen werden kann [Vgl. MüNe02, S.815].

Beispiel : `document~getElementById(“any“)~style~color=“red“`

4.4.1.3 Das forms-Objekt

Das `forms`-Objekt ist ein weiteres Unterobjekt des `document`-Objektes und bietet Zugriff auf alle Formulare eines HTML-Dokuments. Formulare können entweder über die Indexnummer, den Namen des Formulars als Indexnamen oder

[196] S.a. 4.4.3 .
[197] S.a. 4.4.3 .

direkt mit dem Namen des Formulars angesprochen werden. Die Eigenschaften und Methoden des `forms`-Objekts betreffen nur Bestandteile des gesamten Formulars [Vgl. MüNe01, S.586f].

4.4.1.4 Das elements-Objekt

Das `elements`-Objekt ist das Unterobjekt des `forms`-Objektes und bietet Zugriff auf einzelne Elemente die innerhalb eines Formulars definiert sind. In einem Formular definierte Elemente können entweder mit dem Namen oder der Indexnummer angesprochen werden [Vgl. MüNe01, S.593f].

4.4.1.5 Das images-Objekt

Das `images`-Objekt ermöglicht den Zugriff auf alle Grafiken, welche in einem HTML-Dokument definiert sind. So besteht beispielsweise die Möglichkeit eine Grafik dynamisch durch eine andere zu ersetzten. Ein Grafikobjekt kann entweder durch seine Indexnummer oder seinen Namen angesprochen werden [Vgl. MüNe01, S.610f].

4.4.1.6 Das links-Objekt

Das `links`-Objekt ermöglicht den Zugriff auf Verweise, die in einem HTML-Dokument eingebettet sind. Das `links`-Objekt besitzt lediglich die Eigenschaft `length`, welche die Anzahl der Verweise in einem HTML-Dokument speichert. Darüber hinaus ermöglicht es auch den Zugriff auf die Verweisziele der definierten Verweise [Vgl. MüNe01, S.635].

4.4.2 Das DHTML Modell von Netscape

Neben dem Microsoft Ansatz für DHTML, gab es auch einen Ansatz von Netscape. Allerdings kann mit diesem Ansatz nicht auf alle Elemente eines HTML-Dokuments beliebig zugegriffen werden [Vgl. MüNe02, S.1012].

DHTML gibt es bei Netscape nur im Zusammenhang mit positionierten Elementen. Hierfür hat Netscape im Netscape Navigator 4.x die *Layer*-Technik imple-

mentiert. Allerdings wurde diese Layer-Technik vom W3C-Konsortium abgelehnt und daraufhin in der Version 6.0 nicht mehr implementiert. Stattdessen wurde in diese Version das Document Object Model (DOM) implementiert [Vgl. MüNe02, S.1012].

4.4.3 DOM – Document Object Model

Um den Missstand dieser beiden verschiedenen Standards und einer damit verbundenen Doppelprogrammierung zu beenden, wurde vom W3C ein allgemeines Modell für Objekte eines Dokumentes entwickelt. Vom W3C erhielt dieses Modell die Bezeichnung Document Object Model (DOM) [Vgl. MüNe02, S.986].

Das Document Object Model ist eine vom W3C spezifizierte plattform- und sprachenunabhängige Schnittstelle, welche Programmen und Skripten den dynamischen Zugriff auf Inhalt, Struktur und Layout (also alle beliebigen Elemente und Attribute) eines XML kompatiblen Auszeichnungssprachen-Dokumentes erlaubt [Vgl. W3C03a].

Das DOM definiert dabei lediglich die Klassen bzw. Objekte, Eigenschaften und Methoden, welche eine Skriptsprache umsetzten sollte. Diese Objekte, Eigenschaften und Methoden sollen auf alle Dokumente, die in einer XML-basierten Auszeichnungssprache geschrieben sind, anwendbar sein [Vgl. MüNe02, S.986f].

HTML- als auch XML-basierte Dokumente werden von einem Parser zerlegt und in einer hierarchischen Baumstruktur, in welcher die Elemente die Knoten sind, abgebildet. Code 26 und Abbildung 35 werden dies im Folgenden nochmals veranschaulichen.

```
<html>
<head>
<title>DOM</title>
<style type="text/css">
table {bgcolor:lightyellow}
h1 {color:red}
</style>
```

```
</head>
<body >
<h1><i>Document Object Model<i></h1>
<form>
<table border="2">
<tr><td>forename :</td><td>Thorsten </td></tr>
<tr><td>surname:</td><td>Schaedler</td></tr>
</table>
<p>created by Thorsten Schaedler</p>
</body>
</html>
```

Code 26 : Beispiel für DOM

Abbildung 35 zeigt eine solche Baumstruktur, welche auf Basis von Code 26 vom HTML-Parser generiert wird.

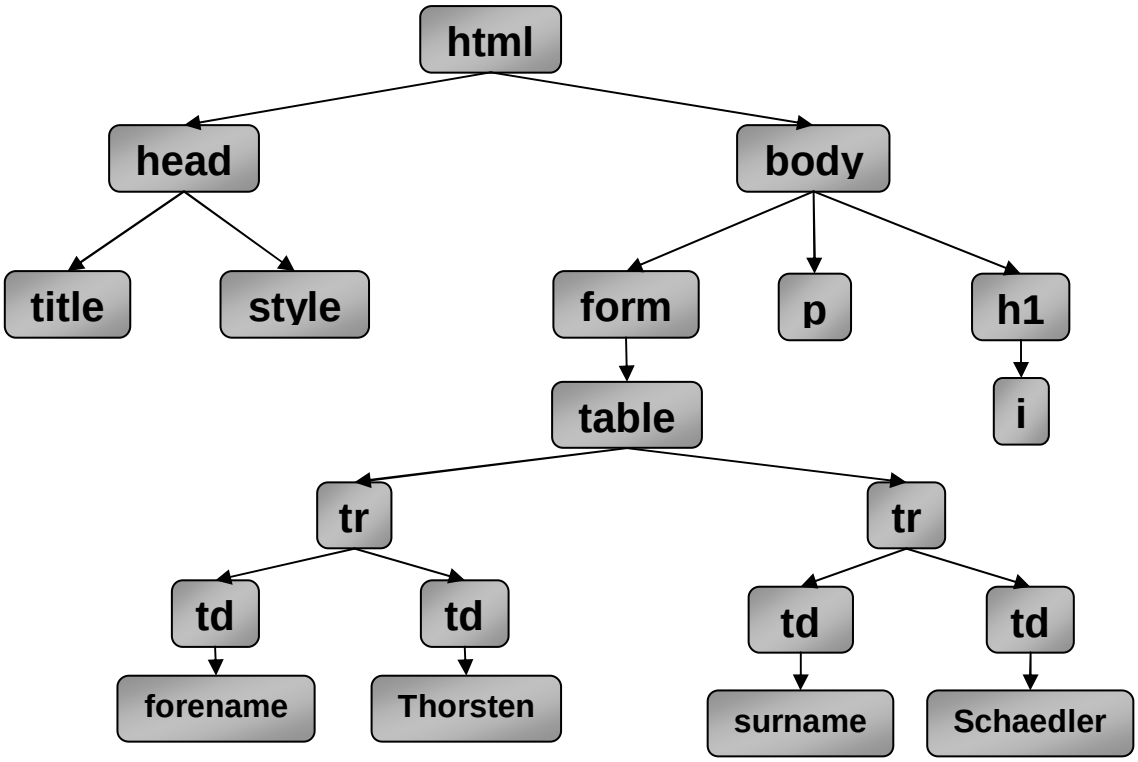

Abbildung 35 : Beispiel eines Elementbaumes

Innerhalb eines gewöhnlichen HTML-Dokuments gibt es verschiedene Knotentypen. Die drei wichtigsten Knotentypen, welche in allen HTML-Dokumenten vorkommen, sind der Elementknoten, der Attributknoten und der Textknoten.

Die Baumstruktur einer Web-Seite kann somit sehr umfangreich und tief verschachtelt sein. Daher muss es in einer Skriptsprache möglich sein, möglichst schnell und effizient auf einzelne Knoten zugreifen zu können. Aus diesem Grunde gibt es im `document`-Objekt des DOM drei wichtige Methoden mit denen auf jeden beliebigen Elementknoten direkt zugegriffen werden kann [Vgl. MüNe02, S.987].

Diese drei Methoden des `document`-Objekts sind [Vgl. MüNe02, s.988]:

1. `getElementById`(), mit welcher auf Elemente zugegriffen werden kann, die ein dokumentweit eindeutiges *`id`*-Universalattribut enthalten.
2. `getElementsByName`(), mit welcher auf Elemente zugegriffen werden kann, welche mittels des *`name`*-Attributs einen dokumentweit eindeutigen Namen erhalten haben.
3. `getElementsByTagName`(), mit welcher über den Elementbaum auf alle Elemente einer HTML-Dokuments zugegriffen werden kann.

Wurde ein Elementknoten ermittelt, so kann mit Hilfe von Eigenschaften und Methoden des DOM auf dessen Attribute und Inhalt zugegriffen werden.

Um in einem HTML-Dokument auf Kinderknoten und Attributknoten eines Elements zugreifen zu können, besteht zum Einen die Möglichkeit die Eigenschaften und Methoden des node-Objekts zu verwenden oder aber HTML-Elementobjekte zu verwenden [Vgl. MüNe02, S.988].

Unter Kinderknoten werden andere Elemente innerhalb eines Elements als auch Textknoten verstanden [Vgl. MüNe02, S.776].

Das DOM ist nicht auf die Client-Seite (Web Browser) beschränkt, sondern es lässt sich ebenso gut in serverseitigen Skripten einsetzen [Vgl. MüNe02, S.987].

4.4.3.1 HTML-Elementobjekte

Nach der HTML-Variante des DOM, stellt in einem HTML-Dokument jedes HTML-Element ein Objekt dar. Mit Hilfe einer Skriptsprache kann auf ein solches HTML-Element (Knoten) nur zugegriffen werden, wenn es im einleitenden Tag mit dem Universalattribut `id` oder dem `name` Attribut einen dokumentweit eindeutigen Namen erhalten hat. Mit dem erweiterten XML-DOM lässt sich jedoch auch ohne das `name` oder `id` Attribut auf jedes beliebige HTML-Element zugreifen [Vgl. MüNe02, S.700].

In einem HTML-Element stellt jedes dort erlaubte Attribut eine DOM-Eigenschaft dieses Elements dar. Für einige der HTML-Elemente definiert das DOM auch Methoden [Vgl. MüNe02, S.700].

Da jedes HTML-Element nach dem DOM einen Knoten im Elementbaum darstellt, kann für jedes HTML-Elemente die Methoden und Eigenschaften des `node`-Objekts angewendet werden [Vgl. MüNe02, S.701].

4.4.3.2 Das node-Objekt

Das `node`-*Objekt* ist ein Unterobjekt des `document`-Objekts[198] und zugleich das zentrale Objekt des DOM. Das `node`-Objekt ist nicht auf HTML beschränkt und stellt Eigenschaften und Methoden bereit, mit welchen auf jeden Knoten des Objektbaums zugegriffen werden kann [Vgl. MüNe02, S.776].

Jedes Element stellt gemäß dem DOM einen Knoten im Elementbaum dar, weshalb für jedes Element auch alle Eigenschaften und Methoden des `node`-Objekts gelten. Um allerdings auf die Eigenschaften und Methoden des `node`-Objektes zugreifen zu können, muss über die bereits vorgestellten Methoden `getElemetById()`, `getElementByName()` oder `getElementByTagName()`des `document`-Objekts ein Element angesprochen werden. Anschließend kann von diesem Element aus auf dessen Kindknoten und Attributknoten mit den Eigen-

[198] S.a. 4.2.2.1.2.2 .

schaften und Methoden des node-Objekts zugegriffen werden [Vgl. MüNe02, S.776f.].

4.4.3.2.1 Eigenschaften und Methoden

Tabelle 13 stellt die verschiedenen Eigenschaften des node-Objekts vor.

Eigenschaft	Bedeutung
attributes	Speichert einen Array aus verfügbaren Attributen eines Elements.
childnodes	Speichert einen Array aus verfügbaren Kinderknoten eines Elements.
data	Speichert Zeichendaten eines Textknotens.
firstChild	Speichert das Objekt des ersten Kindknotens eines Knoten.
lastchild	Speichert das Objekt des letzten Kindknotens eines Knoten.
nextSibling	Speichert aus Sicht eines Knotens den unmittelbar nächstfolgenden Knoten im Strukturbaum.
previousSibling	Speichert aus Sicht eines Knotens den unmittelbar vorhergehenden Knoten im Strukturbaum.
nodeName	Speichert den Namen eines Knotens.
nodeType	Speichert den Typ eines Knotens in Form einer Nummer.
nodeValue	Speichert den Wert oder Inhalt eines Knotens.
parentNode	Speichert den Elternknoten eines Elements.

Tabelle 13 : Eigenschaften des node-Objekts[199]

Tabelle 14 stellt die verschiedenen Methoden des node-Objekts vor.

Methode	Bedeutung
appendChild()	Hängt einen neu erzeugten Knoten in die bestehende Knotenstruktur ein.
appendDate()	Fügt einem Textknoten oder dem Wert eines Attri-

[199] Nach [MüNe02, S.777-785] .

	butknotens am Ende Daten hinzu, ohne die bestehenden Daten zu überschreiben.
cloneNode()	Erstellt eine identische Kopie eines Knotens.
deleteData()	Löscht Daten eines Textknotens oder den Wert eines Attributknotens.
getAttribute()	Ermittelt den Wert eines bestimmten Attributs in einem Element.
getAttributeNode()	Ermittelt einen bestimmten Attributknoten.
hasChildNodes()	Ermittelt ob ein Knoten Kinderknoten hat.
insertBefore()	Fügt innerhalb eines Knotens einen Kindknoten vor einem anderen Kindknoten ein.
insertData()	Fügt Zeichendaten in einen Textknoten ab einer bestimmten Zeichenposition ein.
removeAttribute()	Löscht aus einem Element die Wertzuweisung an ein Attribut.
removeAttributeNode()	Löscht aus einem Element einen Attributknoten.
removeChild()	Löscht aus einem Element einen Kindknoten.
replaceChild()	Ersetzt einen Kindknoten durch einen anderen.
replaceData()	Ersetzt Zeichendaten im Inhalt eines Elements oder in der Wertzuweisung an ein Attribut.
setAttribute()	Setzt in einem Element einen Attributwert neu.
setAttributeNode()	Fügt in ein Element einen neuen Attributknoten ein.

Tabelle 14 : Methoden des node-Objekts[200]

4.4.3.2.2 Zugriff auf Elemente

Mit Hilfe des `node`-Objekts kann direkt auf jeden beliebigen Knoten (Element) eines HTML-Dokuments zugegriffen werden.

Im Code 27 wird nun gezeigt, wie mit Hilfe des `node`-Objekts und des `document`-Objekts in Verbindung mit Object Rexx[201], einerseits Kinderknoten

[200] Nach [MüNe02, S.785-795] .
[201] S.a. 4.2.3 .

eines vorab referenzierten Knotens ermittelt werden können (`::routine dom`) und zum Zweiten ein Element in ein HTML-Dokument eingefügt (`::routine add`) und anschließend wieder entfernt (`::routine remove`) werden kann. Hierzu wird ein Skript Bereich in das HTML-Dokument eingebaut.[202]

```
<html>
<head>
            <!-- Beginning of the script code -->
<script type="text/Object Rexx">
  -- routine for getting knowledge about the first child node
     of the headline
::routine dom public
          -- transfer of the element with the id headline to
             the variable mother
mother = document~getElementById("headline")
          -- transfers the content of the firstChild to the
             variable child
child = mother~firstChild~nodeValue
         -- using the method alert to display the content of
            the variable child on screen
alert("the first child node is : " child)

  -- routine for adding a new node into the HTML file
::routine add public
          -- creates a new h2-element node called newnode
newnode = document~createElement("h2")
          -- creates a text node with the content “Hello I
             am a new node” which is saved in the
             variable content
content = document~createTextNode("Hello i am a new node")
              -- adds the new element node (newnode) and its
                 textnode (content) to the HTML file
docment~getElementById("body")~appendChild(newnode)~appendChild
(content)
  -- routine which removes the node created before
::routine remove public
           -- the new node created by the add routine gets the
              value delete
delete = document~getElementsByTagName("h2")[0]
           -- uses the removeChild method of the node-object to
              remove the node
document~getElementById("body")~removeChild(delete)
              <!-- End of script code -->
</script>
</head>
<body bgcolor="lightyellow" id="body">
<h1 id="headline" align="center">Using the node-object in a <i
style="color:blue">HTML-Dokument</i></h1>
<hr>
<h3>You can see the first Child of the headline above by press-
```

[202] Das Einbetten eines Skriptbereichs in ein HTML Dokument wird in Kapitel 4.5 näher erläutert .

```
ing the Child button</h3>
<input type="button" value="Child" onclick="call dom">
<hr>
<h3>You can add a new node into the DOM by pressing the"Add"
button and erase it by pressing the "Remove" button</h3>
<input type="button" value="Remove" onclick="call remove">
<input type="button" value="Add" onclick="call add">
</body>
</html>
```

Code 27 : Verwenden des node-Objekts in einem HTML-Dokument

Wie in Abbildung 36 zu sehen ist wird beim Drücken des „*Child*“-Buttons, der erste Kinderknoten („Using the node-object in a“) der Überschrift mit der `id` “*headline*“ über ein Meldungsfenster am Bildschirm ausgegeben.

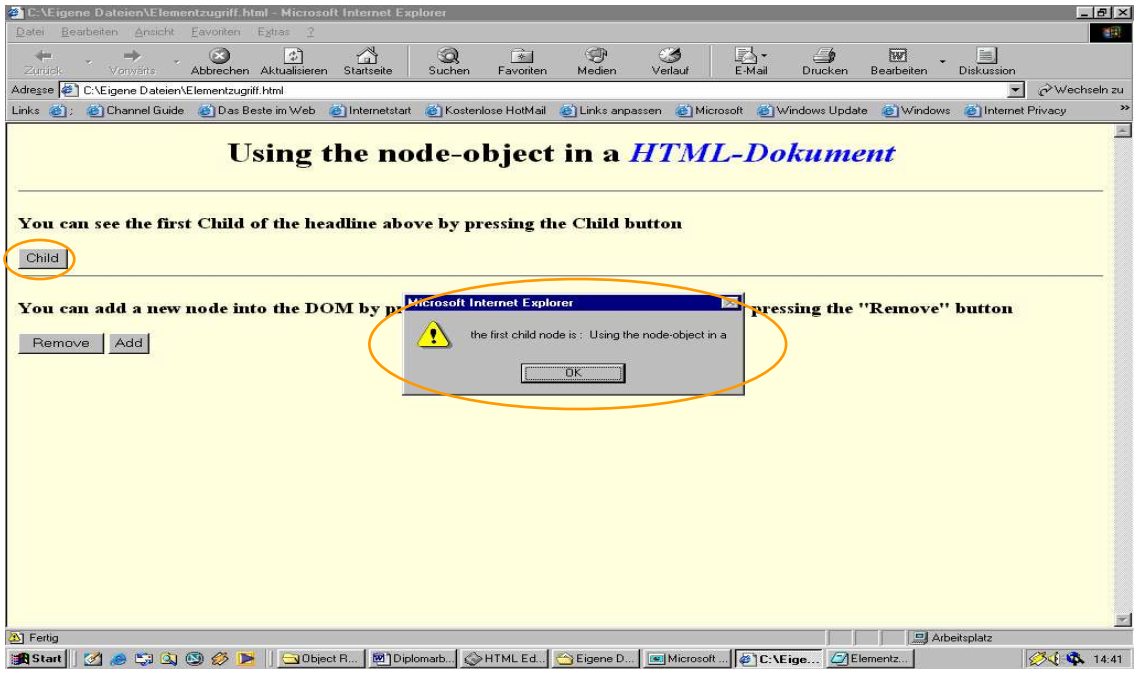

Abbildung 36 : Abfrage eines Kinderknotens

Abbildung 37 zeigt im Anschluss wie der MS Internet Explorer eine Überschrift zweiten Grades (h2-Element), mit dem in Code 27 definierten Inhalt „*Hello i am a new node*“, durch Drücken des „*Add*“– Buttons in das HTML-Dokument einbaut .

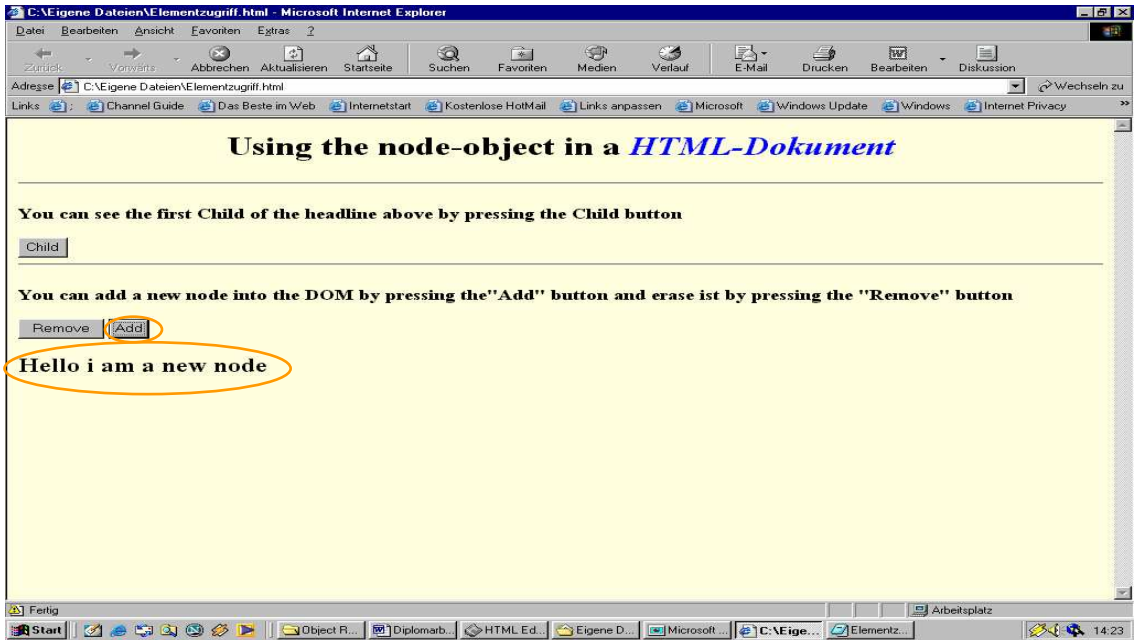

Abbildung 37 : Hinzufügen eines Elements

Abbildung 38 zeigt wie daran anschließend durch Drücken des „*Remove*“-Buttons, diese vorab dynamisch erzeugte Überschrift wieder entfernt werden kann.

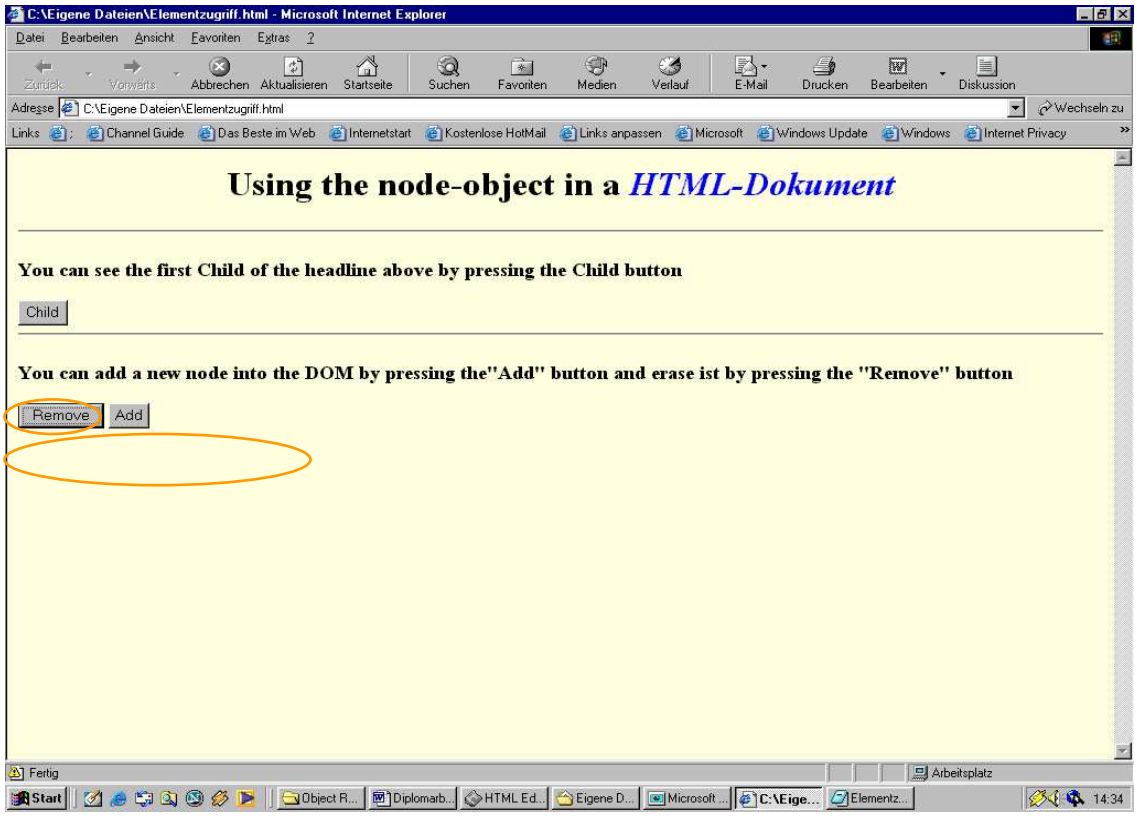

Abbildung 38 : Entfernen eines Elements

4.4.3.2.3 Zugriff auf Attrtibute

Wie in Tabelle 14 zu sehen ist, verfügt das node-Objekt über verschiedene Methoden, welche es ermöglichen Attribute eines Elements zu entfernen, hinzuzufügen oder dessen Wertzuweisung zu ändern.

Code 28 soll nun zeigen wie das HTML-Dokument aus Code 4 mit Hilfe des node-Objekts und Object Rexx so modifiziert werden kann, dass eine Formatierung per Knopfdruck vom Anwender vorgenommen bzw. ausgelöst werden kann. Die entsprechenden Modifikationen sind in blauer Farbe dargestellt.[203]

```
<html>
<head>
<title>CSS-Example</title>
<meta name="author" content="Thorsten Sch&auml;dler">
<link rel=”stylesheet” media=”screen”>
<link rel="stylesheet" media="print" href="print.css">
</head>
<body >
 <h1>Using CSS in a HTML file</h1>
 <br>
 <p id="uf1">This example uses an <span class="any">extern CSS
file</span> for bringing the HTML file into a good shape</p>
 <p id="uf2">Later, the folowing table will also be used for
demonstrating <span class="any">XSLT</span></p>
 <hr>
 <table border="3" width="60%" bgcolor="white">
  <tr class="over">
    <td><u><b>Surname</b></u></td>
    <td><u><b>Forename</b></u></td>
  </tr>
    <td>Sch&auml;dler</td>
    <td>Thorsten</td>
  </tr>
   </tr>
    <td>Design</td>
    <td>Wella</td>
  </tr>
   </tr>
    <td>Mustermann</td>
    <td>Michi</td>
  </tr>
   </tr>
    <td>Cola</td>
    <td>Coca</td>
  </tr>
 </table>
<hr>
```

[203] Das Einbetten eines Skriptbereichs in eine HTML Datei wird in Kapitel 4.5 näher erläutert.

```
  -- invokes the routine CSS by pressing
<input type="button" value="CSS" onclick="call CSS"
  language="Object Rexx">
  -- invokes the routine remove by pressing
<input type="button" value="No-CSS" onclick="call rem">
        <!- Beginning of the script code -->
<script type="text/Object Rexx">
::routine CSS public     -- routine CSS for embedding the Style
                            Sheets file into the html file
 -- createa a new attribute node
new = document~createAttribute("href")
 -- the new attribute node has the value “screen.css”
new~nodeValue="screen.css"
 -- new attribute node is set into the link-element
document~getElementsByTagName("link")[0]~setAttributeNode(new)
</script>
<script type="text/Object Rexx">
::routine rem public   -- routine for removing the Style Sheet
                          file out of the html file
docment~getElementsByTagName("link")[0]~removeAttribute("href")
</script>
            <!- End of script code -->
</body>
</html>
```

Code 28 : Modifikation von Code 4 mit Hilfe des node-Objekts

Um das erste `link`-Element anzusprechen wird innerhalb eines Objekt Rexx Skripts die `document`-Methode `getElementByTagName()` verwendet. Zum Hinzufügen des `href` Attributs zu diesem `link`-Element, wird die `node`-Eigenschaft `nodeValue` sowie die Methoden `createAttribute()` und `setAttributeNode()` verwendet. Zum Entfernen des Attributknotens wird die `node`-Methode `removeAttribut`() verwendet.

Abbildung 39 zeigt die Darstellung der modifizierten HTML-Dokument aus Code 28 im MS Internet Explorer.

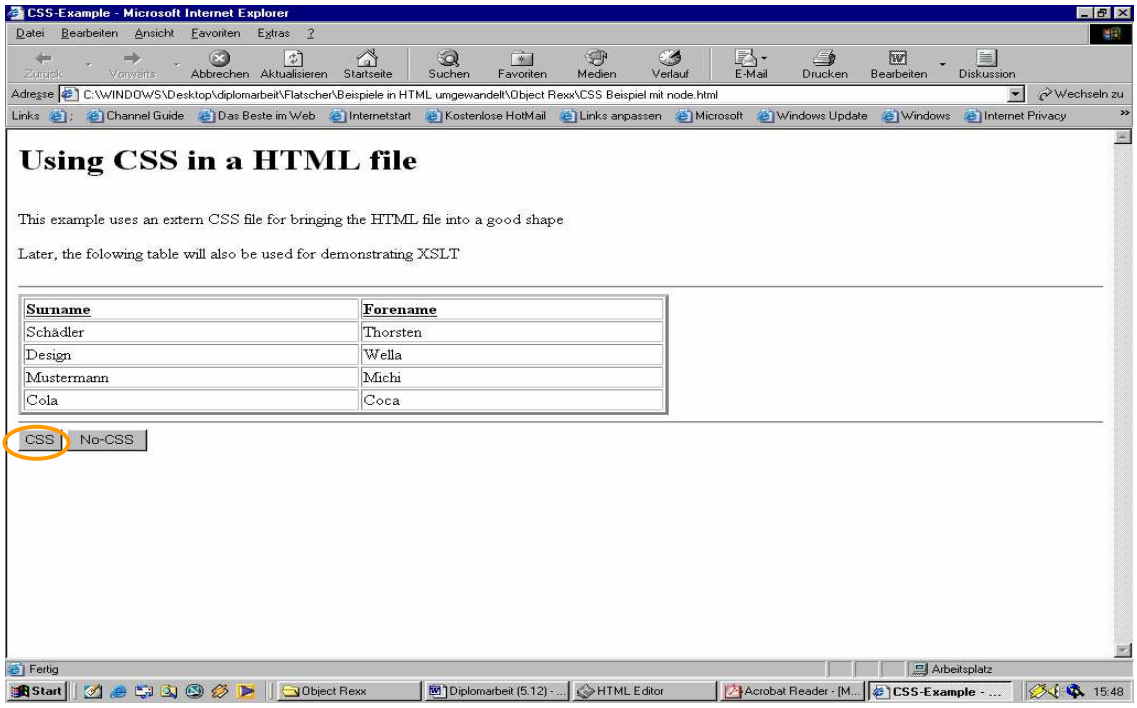

Abbildung 39 : Formatierung per Knopfdruck (1)

Abbildung 40 zeigt abschließend wie das HTML-Dokument nach dem Drücken des „*CSS*“-Buttons aussieht. Durch Drücken des „*NO-CSS*“-Buttons kann anschließend die Formatierung wieder Rückgängig gemacht werden.

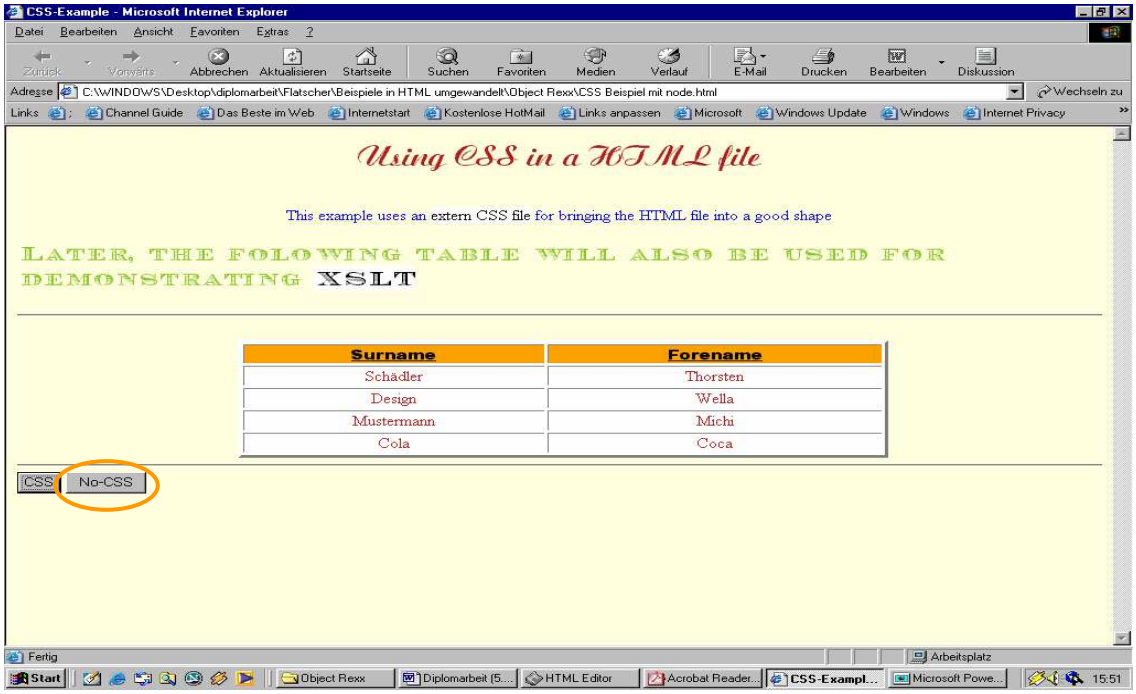

Abbildung 40 : Formatierung per Knopfdruck (2)

4.5 Einsatz von Skriptsprachen in HTML

Wie bereits in den vorangegangenen Kapiteln gesehen, kann ein Object Rexx Skript innerhalb eines HTML-Dokuments eingebaut werden. Dieses Kapitel soll nun die allgemeine Handhabung des Skripten anhand der beiden Programmiersprachen JavaScript[204] und Objekt Rexx[205] näher erläutern. Dabei wird auch auf Unterschiede in den beiden Skriptsprachen eingegangen werden.

4.5.1 Skript Bereiche in HTML definieren

Zum Definieren von Skriptbereichen steht in HTML das `script`-Element zur Verfügung. Dieses `script`-Element wird im allgemeinen in einem HTML-Dokument innerhalb des Kopfteils angegeben. Dies ist jedoch nicht zwingend erforderlich.

Skripte können in HTML auf zwei Arten eingebettet werden.

4.5.1.1 Script-Element in einem HTML-Dokument

Im einleitenden `<script>` Tag ist es seit HTML 4.0.1 Pflicht, das `type`-Attribut und den damit verbundenen MIME-Typen anzugeben. MIME ist ein Internet Standard zur Angabe des Dateityps und steht für Multipurpose Mail Extensions. MIME Typen werden nach dem Schema *Kategorie/Unterkategorie* angegeben. Als Kategorie wird hier `text` und als Unterkategorie die verwendete Skriptsprache angegeben [Vgl. W3C02, S.252ff].

Beispiel : `<script` **`type`**`=“`*`text/Object Rexx`*`“>`

Eine andere Möglichkeit besteht in der Angabe des `language`-Attributes anstelle des `type`-Attributes, jedoch ist dieses Attribut seit der HTML-Spezifikation 4.0.1 missbilligt [Vgl. W3C02, S. 252].

Beispiel : `<script` **`language`**`=“`*`Object Rexx`*`“>`

[204] S.a. 4.2.2.1 .
[205] S.a. 4.2.2.2 .

4.5.1.2 Separate Skriptdatei

Um ein Skript in mehreren Dateien verwenden zu können, ist es sinnvoll das Skript nicht in das HTML-Dokument einzubetten, sondern es als separate Datei abzuspeichern.

Ein HTML-Dokument kann dann bei Bedarf auf dieses Skript zugreifen. Hierzu muss allerdings im einleitenden `<script>` Tag noch das Attribut `src` angegeben werden. Diesem Attribut wird als Wert der URI der separaten Datei zugewiesen.

Beispiel : `<script src=“Script.js“ type=“text/JavaScript“>`

Dies funktioniert standardmäßig nicht mit Object Rexx. Das BWS allerdings ermöglicht es, ein externes Object Rexx Skript wie ein JavaScript-Skript in ein HTML-Dokument einzubinden.

Syntax:
```
<script type="bws/rexx" id="rexxscript" src="[Pfadanga
      be]" />
```

Als Pfadangabe sind sowohl absolute als auch relative URI erlaubt [Vgl. SpeT03c]

4.5.2 Funktionen aufrufen

Um Funktionen bzw. Routinen innerhalb eines Skript Bereiches bei Bedarf aufzurufen, können die Event-Handler[206] eingesetzt werden. Innerhalb eines HTML-Tag wird der gewünschte Event-Handler angegeben.

In JavaScript, JScript und in VBScript wird dem Event-Handler als Wert der Name der Funktion (in Anführungsstrichen) zugewiesen.

Beispiel :
```
<input type=“button“ value=“Calculation“
        onClick=“calculation()“>
       <p onmouseover=”calculation()”>Calculation</p>
```

[206] S.a. 3.6.1 .

Um eine Object Rexx Routine[207] aufzurufen, muss neben dem Namen der Routine noch die Sprungmarke „*call*“ angegeben werden.

Beispiel : `<input type=“button“ value=“Calculation“ onClick=“call calculation“>`

4.5.3 Inhalte an Funktionen/Routinen übergeben

In *JavaScript* werden die Werte beim Funktionsaufruf eingelesen und an die Funktion mit übergeben.

Die nachfolgenden Angaben sind auf das Beispiel in Kapitel 4.5.4 bezogen.

```
 ……
         <!- Beginning of the script code -->
<script type=”text/JavaScript”>
function calculate (a, k) –-JavaScript function calculate
 {
 K = a*k
 document.all.pay.value = K
 }
</script>
             <!- End of script code -->
 …..
<input  type=”button”   value="Calculation" onclick="calculate
(document.Depotform.number.value,document.all.sp.value)"> …. .
```

Code 29 : Inhalte an JavaScript-Funktion übergeben

In *Object Rexx* werden vom Anwender angegebene Werte innerhalb der aufgerufenen Routine mit Hilfe des `document`-Objekts in dort definierte Variablen gespeichert, und nicht mit dem Funktionsaufruf übergeben.

```
          <!- Beginning of the script code -->
<script  type="text/Object Rexx">
::routine calculate public –-Object Rexx routine calculate
a=document~all~Depot~number~value
k=document~all~Depot~sp~value
K = k*a
document~all~pay~value = K
</script>
          <!- End of script code -->
```

[207] S.a.4.2.2.2.3 .

```
.....
<input type=button value="Calculation" language="Object Rexx"
   onclick="call calculate">
.....
```

Code 30 : Inhalte an Object Rexx Routine übergeben

4.5.4 Object Rexx / JavaScript Beispiel

Der Einsatz von Skriptsprachen in einem HTML-Dokument soll nun anhand eines kleinen Beispiels dargestellt werden.

Im diesem Beispiel (Code 31) wird JavaScript bzw. Object Rexx in Verbindung mit dem DHTML-Ansatz von Microsoft[208] dazu verwendet, die Eingaben eines Anwenders bezüglich der Kontobelastung eines Aktienkaufs zu verwenden und dem Anwender die Kontobelastung anzuzeigen.

```
<html>
<head>
<title>Calculation of your transaction</title>
<meta name="author" content="Thorsten Sch&auml;dler">
<style type="text/css">
<!--
body { margin-left:300px;margin-top:100px}
input{ size:5; font-style:Arial}
form { color:red}
h3   { font-family:Chevara }
-->
</style>
       <!- Beginning of the Object Rexx script code -->
<script type="text/Object Rexx">
::routine calculate public
a=document~all~Depot~number~value
k=document~all~Depot~sp~value
K = k*a
document~all~pay~value = K
</script>
            <!- End of Object Rexx script code -->
</head>
<body  bgcolor="lightyellow">
<form name="Depot">
<table border=2 bgcolor=”orange”>
<caption align=”top”>
<h3>Calculation of your transaction</h3>
</caption>
```

[208] S.a. 4.4.1 .

```
<tr>
<td>How many shares you want to buy? :</td>
<td><input type=”text” name="number" size=5 maxlength=7></td>
</tr><tr>
<td>selling price? :</td>
<td><input type=”text” id="sp" size=5 maxlength=7></td>
<td>Euro</td>
</tr>
</table><br>
<input type=”reset” value="new calculation">
<input type=”button” value="Calculation" language="Object Rexx"
onclick="call calculate">
</form>
<table border=”5” style="color:firebrick;background:lightgrey;
font-style:oblique;font-weight:bold">
<tr>
<td>You have to pay :</td>
<td><input type=”text” id="pay"></td>
<td>Euro</td>
</tr>
</table>
</body></html>
```

Code 31 : Kalkulation mit Object Rexx (calculate.html)

Abbildung 41 zeigt die Darstellung von Code 31 im Internet Explorer, nachdem die Berechnung der Kontobelastung durchgeführt wurde.

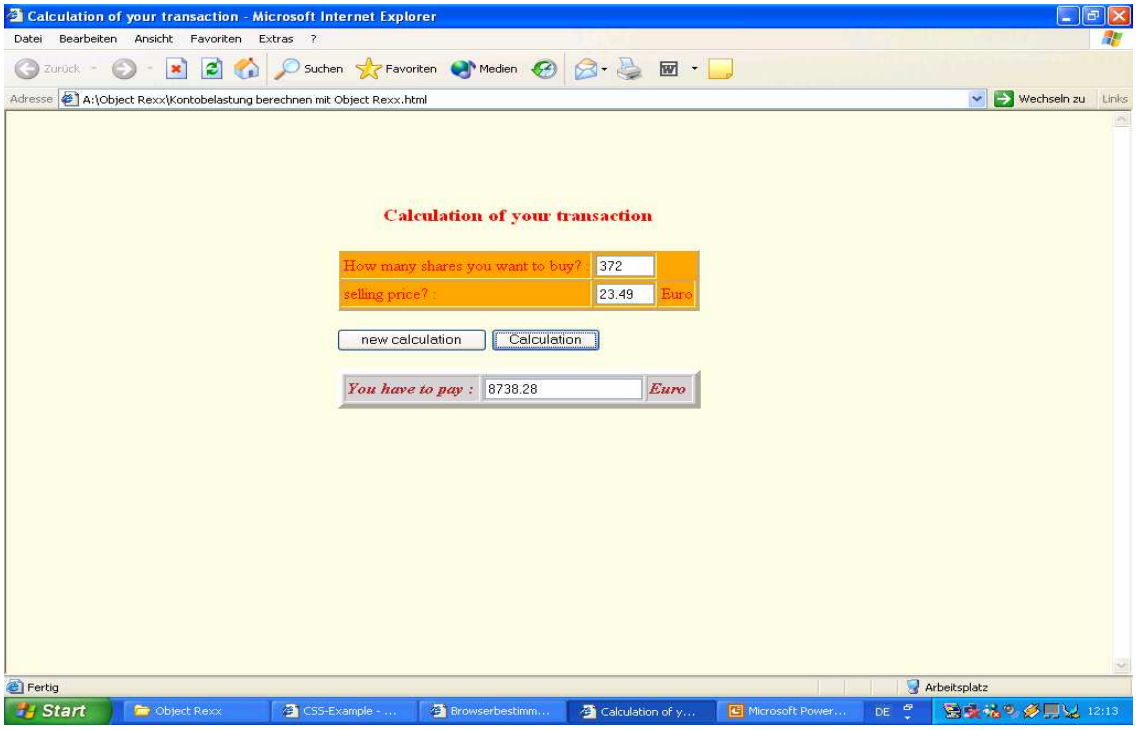

Abbildung 41 : Object Rexx in einem HTML-Dokument (calculate.html)

Um nun dieses Beispiel standardmäßig[209] auch für den Netscape Navigator und Opera lauffähig zu machen, muss als Skriptsprache JavaScript statt Object Rexx eingesetzt werden. Hierzu muss nun die in Kapitel 4.5.2 und 4.5.3 vorgestellten Änderungen vorgenommen werden.

Da der Netscape Navigator das `all`-Objekt nicht unterstützt (Opera hingegen unterstützt dieses `all`-Objekt), muss stattdessen eine der `document`-Methoden `getElementById`() oder `getElementByTagName`() verwendet werden.[210]

Wird also JavaScript als Skriptsprache verwendet, so muss im obigen Beispiel lediglich die Zeile `document.all.pay.value = K` durch die Zeile `document.getElementById("pay").value = K` ersetzt werden.

4.5.5 Einsatz von OLE Objekten mit Object Rexx

Object Rexx bietet die Möglichkeit der Verwendung von OLE Objekten[211].

4.5.5.1 Automatisierung von Word

Code 32 demonstriert und erläutert die Möglichkeit wie vom Benutzer eingegebene Daten aus einem HTML-Formular automatisiert in ein Word-Dokument übertragen werden können. Eine solche Automatisierung kann auch mit den anderen MS Office Produkten realisiert werden.

```
<html><title>HTML-Word</title>
 -- this HTML file uses Cascading Style Sheets
<style type="text/css">
    body { margin-left:120px;margin-top:80px}
    h1  { font-family:Lucida Calligraphy;color:firebrick }
</style>
      <!—Beginning of the Object Rexx script code -->
<script type="text/Object Rexx">
::routine demo public
 --Instantiation of MS Word using the ProgID
Word = .OLEObject~New("Word.Application")
 -- The visible property is set on true. Word can be seen on
    the display
Word~Visible = .true
```

[209] Ohne Verwendung von BSFWebScripting (vgl.4.2.3) .
[210] S.a. 4.4.3 .
[211] S.a. 4.2.2.2.6 .

```
 -- A Word document is added
Word~Documents~Add
Selection = Word~Selection
Selection~WholeStory
 -- The design of the letters from the name field is defined
    for the Word document
design = Selection~Font
design~Name="Arial"-- The following text is of the type “Arial”
design~Size ="16"  -- The following text has the size “16”
 -- the following text has a blue color
design~Color = Word~GetConstant( 'wdColorBlue')
design~Underline = 1  -- the following text is underlined.
 -- Writes the value of the name field and “wrote” into
    the word document
Selection~TypeText(who~value "wrote:")
 -- Empty space is created
Selection~TypeParagraph;
Selection~TypeParagraph;
 -- The design of the letters from the comment field is
    defined for the Word document
design = Selection~Font
design~Name="Arial"-- The following text is of the type “Arial”
design~Size="12"   -- The following text has the size “12”
-- the following text has a black color
design~Color = Word~GetConstant('wdColorBlack')
design~Underline = 0   -- the following text is not underlined
-- Writes the comment of the textarea into to the Word document
Selection~TypeText(comment~value)
-- The Word document is printed out.
Word~PrintOut()
</script>
       <!- End of the Object Rexx script code -->
<body bgcolor="lightyellow">
<form>
<h1><u>From a HTML form to a word document</u></h1><br>
<h3>Your name :</h3>
<input type="text" name="who"><br>
<h3>Please type your comment into the textarea below and press
the <u>"GO !"</u> button !</h3>
<textarea name="comment" rows=15 cols=60></textarea><br><br>
<input type="button" onclick="call demo" value="GO !" >
<input type=”reset” value="NEW">
</form></body></html>
```

Code 32 : Automatisierung von Word [basierend auf HelF03, S.108]

Das Eingabefeld für den Namen als auch das Textfeld für den Kommentar müssen mit dem `name`-Attribut versehen sein. In diesem Beispiel wurde für das `name`-Attribut des Namensfelds der Wert “`who`” und für das Textfeld der Wert “`comment`“ vergeben .

Das erzeugte Word-Dokument kann auch noch automatisch in ein Verzeichnis (hier: `C:\WINDOWS\TEMP`) gespeichert werden. Hierzu muss der Skriptteil aus Code 32 allerdings noch durch Code 33 ergänzt werden.

```
TempDir = VALUE("TEMP",,ENVIRONMENT) –directory C:\WINDOWS\TEMP
FileName = TempDir || "\" || HTMLtoWord
Selection~TypeText(Filename".DOC") -- the file called FileName
                                      gets the extension .doc
Document~SaveAs(FileName)    -- The Word document is saved in
                                C:\WINDOWS\TEMP\HTMLtoWord.doc
call syssleep(2)
```

Code 33 : Speichern eines Dokuments mit Object Rexx[212]

Abbildung 42 zeigt das HTML-Dokuments aus Code 32, in welchem das Namensfeld als auch der Kommentarbereich vom Anwender ausgefüllt wurden.

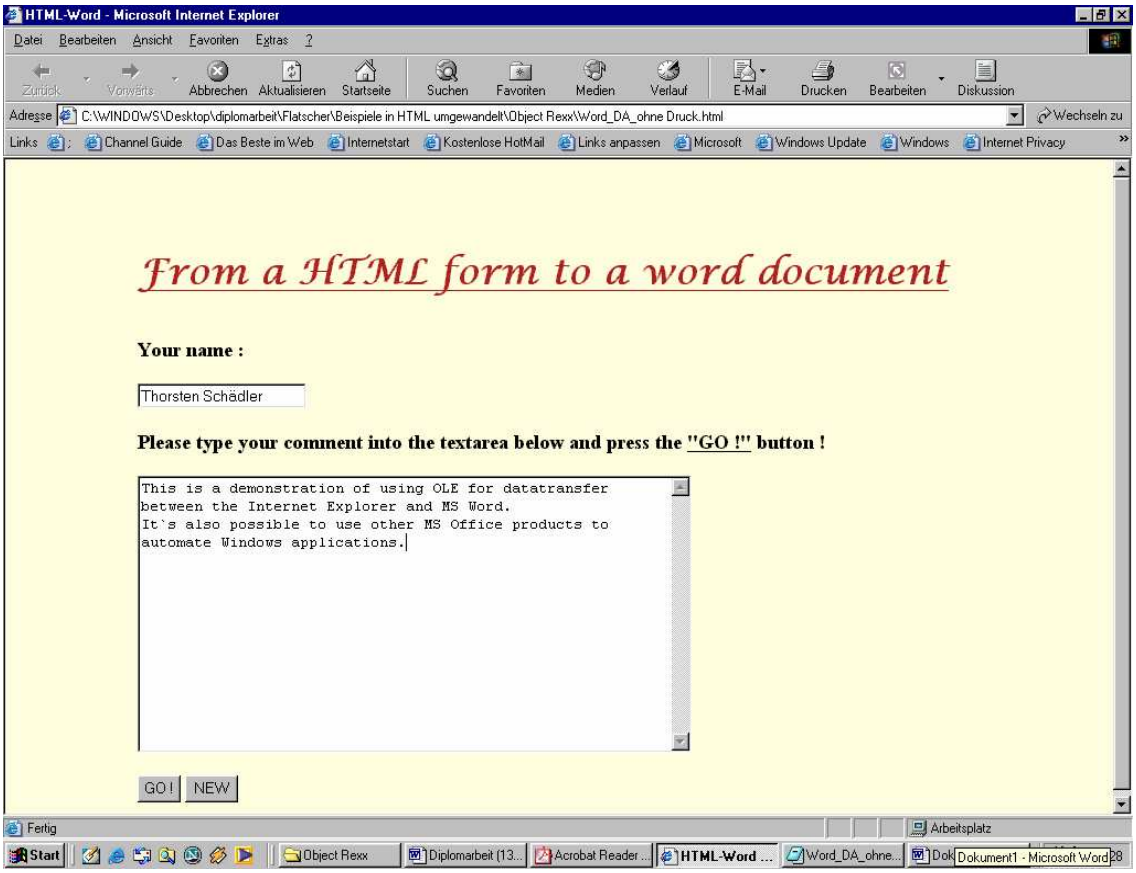

Abbildung 42 : Automatisierung von Word (1)

[212] Nach http://wi3.wiwi.uni-augsburg.de/lehre/ws0203/autowin/aufgaben/Aufgabenteil1-8.htm (Aufgaben zu Einheit 8 – Aufgabe 4), Abruf am 2003-11-23.

Abbildung 43 zeigt anschließend das geöffnete Word-Dokument, welches bei Drücken des „*GO*“ Buttons automatisch erstellt wird. Im rechten unteren Bildschirmrand ist dabei auch der automatisch gestartete Drucker zu sehen.

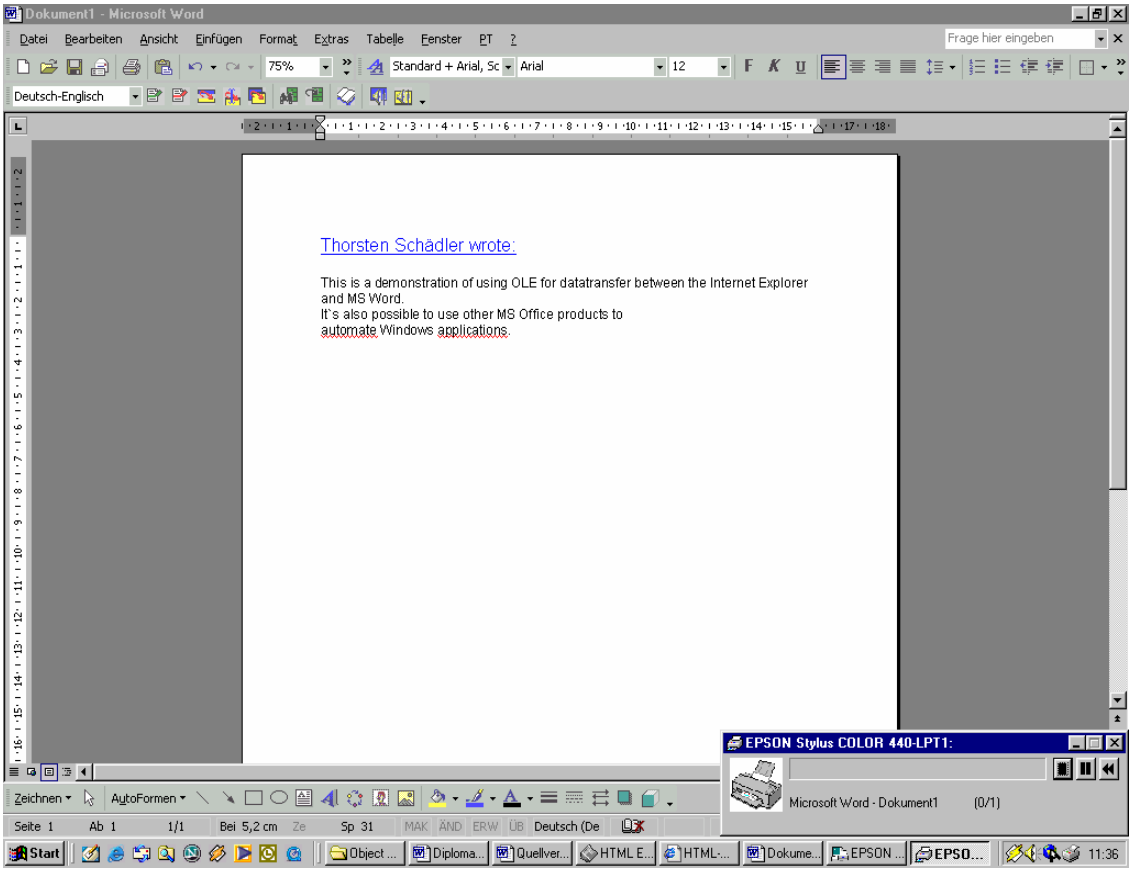

Abbildung 43 : Automatisierung von Word (2)

4.5.5.2 Sprachausgabe

Code 34 soll abschließend zeigen wie Object Rexx mit Hilfe eines Sprachausgabesystem (TextToSpeech Engine – TTS) dazu verwendet werden kann, den Inhalt einer Webseite über die Lautsprecher dem Anwender vorzulesen. Eine solche TTS-Engine kann für mehrere Sprachen von der Microsoft Agent Download-Seite[213] herunter geladen werden. In Windows XP und Windows ME ist eine solche TTS allerdings schon standardmäßig enthalten.

[213] http://www.microsoft.com/msagent/downloads/user.asp#tts, Abruf am 2003-11-19.

Soll ein HTML-Element über die Lautsprecher ausgegeben werden, muss es mit dem `id`- Attribut eindeutig identifiziert werden können.

```
<html>
<head>
  <!—- Beginning of the Object Rexx script code -->
<script type="text/Object Rexx">
::routine speech Public
 -- Instantiation of a TTS object called listen
listen = .OLEObject~new("Speech.VoiceText")
-- The register method has to be used prior to any other method
   being invoked.
listen~register('',"WhatEver")
-- use the speak method in addition to the id attribute and
   innerHTML (property of the all-Object) to get access to the
   text that should be spoken
listen~speak(tts~innerHTML,1)
-- The speed property contols the speed of the voice output
listen~speed = 140
  do while listen~isspeaking
      call SysSleep 1   -- the system sleeps for one second
  end
</script>
      <!-- End of the Object Rexx script code -->
<head>
<body>
<h1 id=”tts”>...</h1>     -- Text that should be spoken
.....
-- Using the Event-Handler “onclick” in a button element to
   start the speech
<button onclick="call speech">Speech</button>
</body>
</html>
```

Code 34 : Sprachausgabe mit Object Rexx

Drückt der Anwender den `Speech`-Button, so wird ihm der Textinhalt der Überschrift erster Ordnung über Lautsprecher vorgelesen.

5. XML

Dieses Kapitel gibt einen Überblick über XML und zeigt wie und zu welchem Zweck XML eingesetzt wird bzw. werden kann.

5.1 Überblick

Aufgrund der in Kapitel 2.6 angesprochenen Defizite von SGML und HTML, beschloss das World Wide Web Konsortium (W3C) nun Ende der 90er Jahre, auf der Basis von SGML eine neue Sprache für das WWW zu entwickeln, welche die Flexibilität und das Prinzip der Trennung von Struktur und Layout von SGML wahren sollte, dabei aber leichter zu erlernen und zu beherrschen sein sollte. Die daraus resultierende e**X**tebsible **M**arkup **L**anguage (XML) stellt dabei eine Teilmenge von SGML dar und wird wie diese zur Definition von Dokumenttypen bzw. eigener Auszeichnungssprachen verwendet.

Solche mittels XML definierte Auszeichnungssprachen sind beispielsweise XHTML, SVG, WML, MathML oder SMIL und werden auch als XML-Derivate[214] [Sht03f] bezeichnet .

Beim Erstellen eines XML-Dokuments geht es zunächst nur einmal darum, die benötigten Daten sinnvoll zu strukturieren und vollständig zu beschreiben. Ein solches Dokument besteht dabei aus bestimmten erlaubten Elementen, Attributen, Wertzuweisungen und Regeln zur Verschachtelung von Elementen. Welche Elemente, Attribute und Verschachtelungsmöglichkeiten in einer XML-gerechten Auszeichnungssprache erlaubt sind, wird in der dazugehörigen Dokumenttypdefinition (DTD)[215] geregelt.

Um XML-Dokumente, im Gegensatz zu SGML-Dokumenten, auch losgelöst von einer solchen DTD verarbeiten zu können, wurde auf bestimmte Freiheiten verzichtet.

---

[214] S.a. 6 .
[215] S.a. 5.2. bzw. 2.4 .

So kann innerhalb eines XML Dokuments der Prolog[216] leer sein. Um die sich aus der Auszeichnung ergebene Baumstruktur konstruieren zu können, genügt es einem in den Browser integrierten *XML-Parser*, wenn das XML-Dokument die Grundregeln der XML-Syntax[217] einhält. Man spricht hier auch von einem *wohlgeformten* XML-Dokument[218]. Im Gegensatz zu einem solchen wohlgeformten Dokument, muss ein *gültiges* Dokument eine Instanz einer DTD[219] darstellen [Vgl. HäPeS00, S.447].

Da XML ein Universalkonzept für Datenspeicherung darstellt, ist es nicht auf das Internet und WWW beschränkt [Vgl. Sht03d].

Im Hinblick auf den Softwaremarkt sorgte dies für Aufregung. Da es sich bei XML um ein standardisiertes Dateiformat handelt, steht nun nicht mehr das Softwareprodukt als Einheit aus anbieterspezifischem Dateiformat und Benutzeroberfläche, sondern lediglich die im Vergleich zur Konkurrenz überlegene Gestaltung einer Benutzeroberfläche im Vordergrund. Durch die universelle Anwendbarkeit von XML, spielt es keine Rolle ob es sich um Software für Textverarbeitung, Tabellenkalkulation, Datenbank, Grafik, Musik oder gar Computerspiele handelt. So hat der Microsoft Konzern bereits damit begonnen sein Office Paket komplett auf XML-basierte Dateiformate umzustellen. Sollte sich die zum Erstellen von Dokumenten benutzte Software konsequent an den XML-Standard halten und nachweisbare Dokumenttypdefinitionen verwenden, so könnten die in verschiedenen Anwendungen erzeugten Dokumente zwischen konkurrierenden Anwendungen problemlos portiert werden [Vgl. Sht03d].

In XML sind die Daten und das Layout strikt voneinander getrennt. Somit lässt sich aus dem gleichen XML-Datenbestand eine Webseite, Druckvorlage oder gar eine akustische Ausgabe der Daten erzeugen [Vgl. Sht03d].

Die Vorraussetzung für eine konkrete Darstellung der Daten ist eine erfolgreiche Analyse der Struktur eines XML-Dokuments. Hierfür wurden XML-Parser entwickelt, welche XML-Strukturen auslesen, analysieren und nachgeschach-

[216] S.a. 5.1.2 .
[217] S.a. 5.1.1 .
[218] S.a. 5.1.1 .
[219] S.a. 5.2 .

telter Software zur Verfügung stellen. Ein Parser kann in die Umgebung eines Web-Servers eingebunden und/oder im Browser (Client-Seite) integriert sein [Vgl. Sht03d].

Ein XML-Dokument kann mit jedem beliebigen Texteditor erstellt werden (reiner Unicode-Text) und wird mit der Endung ".xml" abgespeichert.

5.1.1 XML-Syntax

Die Syntaxregeln von XML sind sehr einfach und dürfen nicht verletzt werden. Beim Erstellen eines *wohlgeformten* XML-Dokuments müssen demnach folgende wichtige Regeln eingehalten werden [Vgl. Duen03]:

- Alle XML Elemente müssen einen Endtag besitzen oder mit /> enden
- Groß- und Kleinschreibung wird unterschieden
- Alle XML Elemente müssen richtig geschachtelt sein
- XML Dokumente müssen ein Wurzelement enthalten
- Attributwerte müssen in Anführungszeichen stehen
- Kommentare müssen wie in HTML in <!-- .....-- > geschrieben werden

5.1.2 Prolog[220]

Ein Prolog ist der Teil eines XML-Dokuments, der vor dem Start-Tag des ersten Elements erscheint. Ein Prolog besteht in der Regel aus einer XML-Deklaration und einer Dokumenttypdeklaration. Es besteht jedoch auch die Möglichkeit nur eine oder auch keine der beiden Deklarationen anzugeben. Insofern ist ein Prolog zum Erstellen eines XML-Dokuments nicht zwingend erforderlich.

5.1.2.1 XML-Deklaration

```
<?xml version="[Versionsangabe]"  encoding="[Kodierungs-
Deklaration] standalone="[Standalone-Deklaration]" ?>
```

In die [*Versionsangabe]* ist einzutragen, welche XML-Version verwendet wird.

[220] Dieses Kapitel basieret auf [Duen03] .

Die *Kodierungs-Deklaration* zeigt an, welche Zeichenkodierung verwendet wird.

Beispiel : `UTF-8, ISO-8859-1, ....`

Die *Standalone-Deklaration* kann entweder den Wert "`yes`" oder "`no`" beinhalten. Bei der Angabe "`yes`", wird dem Parser mitgeteilt, dass keine externe DTD-Untermenge[221] verwendet wird und es keine zu berücksichtigenden externen Parameter-Entities[222] gibt. Die Angabe "`no`" würde dagegen angeben, dass eventuell eine externe Deklaration vorhanden ist.

5.1.2.2 Dokumenttyp-Deklaration

```
<!DOCTYPE [Wurzelelement] SYSTEM "[Angabe der zu verwendenden externen DTD]" >
```

Eine Dokumenttyp-Deklaration gibt an, in welcher Datei sich die zu verwendende externe Dokumenttypdefinition (DTD)[223] befindet.

Neben der Verwendung einer externen DTD kann auch eine Dokumenttypdeklaration innerhalb des Dokuments vorgenommen werden. Hierzu wird die DTD innerhalb der Dokumenttypdeklaration in eckigen Klammern angegeben. Somit ergibt sich folgende allgemeine Form einer Dokumenttypdeklaration mit einer internen DTD.

```
<!DOCTYPE [Wurzelelement] SYSTEM "[Angabe der zu verwendenden externen DTD]" [
//-- interne DTD --//
] >
```

Sollte eine interne und externe DTD vorhanden sein, so fasst der Parser diese zusammen und behandelt sie als eine Gesamt-DTD. Hierbei wird allerdings zuerst die interne DTD eingelesen und es gilt die Regel, dass bei Mehrfachdeklaration immer die zuerst eingelesene gültig ist.

---

[221] S.a. 5.2 .
[222] S.a. 2.3 .
[223] S.a. 5.2 .

5.1.3 Verarbeitung eines XML-Dokuments

Die Verarbeitung eines XML-Dokuments übernimmt der in den Browser integrierte *XML-Parser*. Er hat dabei die Aufgabe zu überprüfen ob das Dokument wohlgeformt[224] und eventuell gültig (valide) ist. Sollte das Dokument wohlgeformt sein, so kann es im Browser als Baumstruktur dargestellt oder mit Hilfe der vom XML-Parser zur Verfügung gestellten Software in der entsprechenden Form dargestellt werden.

XML-Parser existieren in allen gängigen Programmiersprachen. Es wird dabei zwischen SAX (**S**imple **A**PI for **X**ML) und DOM-Parsern unterschieden.

Sax-Parser arbeiten ereignisorientiert wohingegen DOM-Parser das komplette XML-Dokument in den Speicher laden. Der DOM-Parser betrachtet alle Elemente eines XML-Dokuments als Objekte. Diese werden in den Speicher geladen und können nur dort verändert werden. Er ermöglicht einer Skriptsprache [225]das Layout, den Inhalt und die Struktur eines XML-Dokuments zu verändern [Vgl. AnCr03, S.11].

5.1.4 Beispiel

Das Beispiel in Abbildung 44 zeigt die Ausgabe eines wohlgeformten XML Dokuments in einem Browser (IE 6). Die Darstellung entspricht zugleich auch dem Quellcode. Ob das Dokument gültig ist, kann angesichts einer im Prolog fehlenden Dokumenttypdeklaration nicht bestimmt werden.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
-<Person>
  -<Name>
     <Forename>Thorsten</Forename>
     <Surname>Schädler</Surname>
   </Name>
 </Person>
```

Abbildung 44 : Einfaches XML-Dokument

[224] S.a. 5.1.1 .
[225] S.a. 4.2.2 .

5.2 DTD

XML ist die Sprache zum Verfassen von Dokumenttypdefinitionen (DTD). DTDs spielen in XML eine zentrale Rolle. Mit ihrer Hilfe wird es einem XML-Parser ermöglicht zu überprüfen, ob ein XML-Dokument gültig (valide) ist oder nicht.

Eine DTD besteht in der Regel aus einer Elementtypdeklaration und einer Attributdeklaration. Daneben besteht auch noch die Möglichkeit der Deklaration von Entities[Vgl. Duen03].

5.2.1 Elementtyp-Deklaration

In einer Elementtypdeklaration werden die Art und Struktur der in einem XML- bzw. SGML-basierten Dokument enthaltenen Elemente angegeben [Vgl. Duen03].

Syntax : `<! ELEMENT [Elementname] ([Inhaltsmodell])>`

5.2.2 Attributtyp-Deklaration

Bei der Deklaration von Attributen wird jedes Attribut einem Typ zugeordnet. Hierbei wird jedes Attribut einem der insgesamt elf Typen von Attributen zugeordnet [Vgl. Duen03].

Allerdings werden neun dieser Typen zu einer Gruppe zusammengefasst, sodass folgende drei Gruppen von Attributen entstehen [Vgl. Duen03] :

1. der Zeichenkettentyp (Schlüsselwort CDATA)
2. der Aufzählungstyp
3. Token -Typ (Zusammenfassung mehrerer Attributtypen)

Syntax :
```
<!ATTLIST [Elementname]
                [Attribut] [Typ] >
```

5.2.3 Deklaration eines Entities

Bevor ein Entity verwendet werden kann , muss ihm in einer Deklaration innerhalb einer DTD ein Name und der entsprechende Inhalt zugewiesen werden.
Für jedes Entity gibt es einen Namen (z.B. Mfg) und einen entsprechenden Inhalt (hier: Mit freundlichen Grüßen). In einer Deklaration werden dabei einem jeden Entity ein Name und dessen Inhalt zugewiesen. Nach der Deklaration kann der Entity-Name als ein Kürzel im Dokument verwendet werden, und der Parser setzt im Dokument an der Stelle des Entity, dessen Inhalt [Vgl. Duen03].

Entities werden zudem in drei Kriterien unterteilt [Vgl. Duen03] :

- allgemeine Entities vs. Parameter Entities
- geparste vs. Nicht-geparste Entities
- interne vs. externe Entities

Aus diesen drei Kriterien ergeben sich theoretisch acht verschiedene Arten von Entities. Tatsächlich gibt es jedoch nur diese fünf verschiedenen Arten [Vgl. Duen03]:

1. Interne geparste allgemeine Entities
2. Interne geparste Parameter Entities
3. Externe geparste allgemeine Entities
4. Externe geparste Parameter Entities
5. Externe nicht-geparste allgemeine Entities

Syntax : `<! Entity [Entity-Name] “[Inhalt]“>`

5.2.4 Beispiel

Das Beispiel in Abbildung 45 zeigt die Darstellung eines mit der entsprechenden DTD (Code 35) überprüften XML-Dokumentes innerhalb des MS Internet Explorers.

Bei diesem XML-Dokument handelt es sich folglich um ein wohlgeformtes[226] und gültiges[227] Dokument.

```
<!ELEMENT Person (Name)>
<!ELEMENT Name (Forename, Surname)>
<!ELEMENT Forename (#PCDATA)
<!ELEMENT Surname (#PCDATA)
```

Code 35 : Person.dtd

Abbildung 45 zeigt wie das XML-Dokument aus Abbildung 44 unter Verwendung der DTD aus Code 35 im MS Internet Explorer dargestellt wird.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Person (View Source for full doctype...)>
-<Person>
 -<Name>
    <Forename>Thorsten</Forename>
    <Surname>Schädler</Surname>
   </Name>
```

Abbildung 45 : XML-Dokument mit DTD

5.3 Erweiterungen / Ergänzungen von XML

Wie für HTML gibt es auch für XML verschiedene Erweiterungen bzw. Ergänzungen mit welchen ein XML-Dokument beispielsweise formatiert, transformiert oder vernetzt werden kann.

5.3.1 XSL

Da XML lediglich für den Inhalt und Struktur von Daten konzipiert wurde, benötigt man eine Sprache zur Formatierung solcher XML-Dokumente. Die vom

[226] S.a. 5.1.1 .
[227] S.a. 5.1 .

W3C hierfür empfohlene und als Standard eingesetzte XML-basierte Sprache ist die e**X**tensible **S**tylesheet **L**anguage (XSL).

XSL besteht dabei aus zwei wichtigen Komponenten. Die erste Komponente ist XSLT, welche zur Transformation von XML Daten in andere XML Daten verwendet wird. Die andere Komponente wird zur Formatierung von XML Daten verwendet und nennt sich XSLFO[228] [Vgl. Sht03a].

Zur Formatierung von XML Dokumenten besteht neben XSL allerdings auch noch die Möglichkeit der Verwendung der bereits erläuterten CSS[229] [Vgl. Sht03a].

In diesem Kapitel soll daher ein etwas intensiverer Einblick in die verschiedenen Techniken für Klarheit sorgen.

5.3.1.1 XSLT

XSLT steht für XSL-Transformation und hat die Aufgabe Elemente einer XML-basierten Sprache in eine andere XML-gerechte Sprache zu transformieren. Somit besteht also auch die Möglichkeit der Transformation von XML-Daten in HTML, sodass jeder Browser die Daten anzeigen kann. Hierbei können auch Skriptsprachen[230] und CSS eingesetzt werden, wodurch in der Praxis des öfteren CSS anstatt XSLT verwendet wird. Der Einsatz von CSS zur Formatierung von XML-Dokumenten ist allerdings angesichts der Vorteile von XSLT nur für einfache XML-Dokumente zu empfehlen [Vgl. Sht03a].

Der große Vorteil von XSLT gegenüber CSS besteht in der Verwendung der XSLT-Elemente, mit denen es beispielsweise möglich wird, Objekte wie z.B. Personen automatisch nach dem Namen zu sortieren, bestimmte Objekte auszuwählen, Tabellen zu erzeugen und dergleichen [Vgl. Sht03a].

XSLT benötigt einen in den Browser integrierten *XSLT-Parser*, der die Transformationsanweisungen ausführt [Vgl. Sht03a].

---

[228] S.a. 5.3.1.2 .
[229] S.a. 4.1 .
[230] S.a. 4.2.2 .

5.3.1.1.1 XSLT-Prozessor (Parser)

Ein XSLT-Parser ist ein Software Modul. Ein solcher XSLT-Parser kann in den Browser des Client oder auf dem Web Server integriert sein. Ein solches Modul ist seit Version 5 in den Internet Explorer integriert. Die Integration eines solchen Moduls im Server hat den Vorteil, dass eine Transformation in HTML auf dem Server stattfindet und anschließend die Datei in dieser transformierter Form an den Client geschickt wird. Somit ist es unerheblich was für ein Browser zum Darstellen verwendet wird. Für den Apache Web Server ist dies beispielsweise Xalan [Vgl. Sht03a].

5.3.1.1.2 Transformation von XML-Daten in HTML[231]

Für eine Transformation eines XML-Dokuments in ein HTML-Dokument werden die XSLT Elemente benötigt. Im nun folgenden Beispiel werden zur Formatierung bzw. Transformation eines XML-Dokuments (Code 36 – Person.xml), die XSLT-Elemente `stylesheet`, `template`, `for-each`, `sort` und `value-of` innerhalb der XSLT-Datei (Code 37- Person.xsl) verwendet.

So wird in diesem Beispiel das `sort`-Element dazu verwendet in der mittels XSLT erstellten Tabelle (vgl. Abbildung 46) die Nachnamen automatisch in alphabetisch geordneter Reihenfolge auszugeben. Eine solche automatische Anordnung wäre mit HTML und CSS dagegen nicht möglich gewesen.

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="Person.xsl" ?>
<!DOCTYPE Person SYSTEM "Person.dtd">
<Body>   -- root element of the xml file
<Headline>This is a headline</Headline>
<Person>
 <Name>
  <Forename>Thorsten</Forename>
  <Surname>Schädler</Surname>
 </Name>
 <Name>
  <Forename>Wella</Forename>
  <Surname>Design</Surname>
 </Name>
 <Name>
```

[231] Basierend auf [Sht03a] und [Sht03b].

```
  <Forename>Michi</Forename>
  <Surname>Mustermann</Surname>
 </Name>
 <Name>
  <Forename>Coca</Forename>
  <Surname>Cola</Surname>
 </Name>
</Person>
</Body>
```

Code 36 : Person.xml

In Code 37 ist der entsprechende Formatierungscode für das XML-Dokument aus Code 36 dargestellt.

```
   -- XML Declaration
<?xml version="1.0" encoding="iso-8859-1"?>
   -- the root element of a XSL Stylesheet including a
      xsl namespace
<xsl:stylesheet version="1.0
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   -- define a template for translation in a HTML file
<xsl:template match="Body">     --  match the Body element
                                    of the XML file
 <html>
 <head>
 </head>
 <body bgcolor="lightyellow" style="margin-left:200px;margin-
  top:100px">
   <h1 style="color:blue; font-family:Viner Hand ITC">
     <xsl:value-of select="Headline" /></h1>
   <table border="3" width="60%" bgcolor="white"
     style="color:firebrick">
   <tr align="center" bgcolor="orange"
  style="color:black">
   <td><u><b>Surname</b></u></td>
   <td><u><b>Forename</b></u></td>
   </tr>
       -- sorts the inputs of surname in an alphabetical
          order using the order attribute with the value
          ascending and the datatype attribute with the
          value text
  <xsl:for-each select="Person/Name">
  <xsl:sort select="Surname" order="ascending"
  datatype="text" />
  <tr align="center">
    <td valign="top"><xsl:value-of select="Surname"
  /></td>
     <td><xsl:value-of select="Forename" /></td>
   </tr>
  </xsl:for-each>
 </table>
```

```
 body>
 </html>
 </xsl:template>                    -- end of template
</xsl:stylesheet>                   -- end of stylesheet
```

Code 37 : Person.xsl

Abbildung 46 zeigt die Darstellung des mittels XSLT (Code 37) formatierten XML-Dokument aus Code 36 im Netscape Navigator.

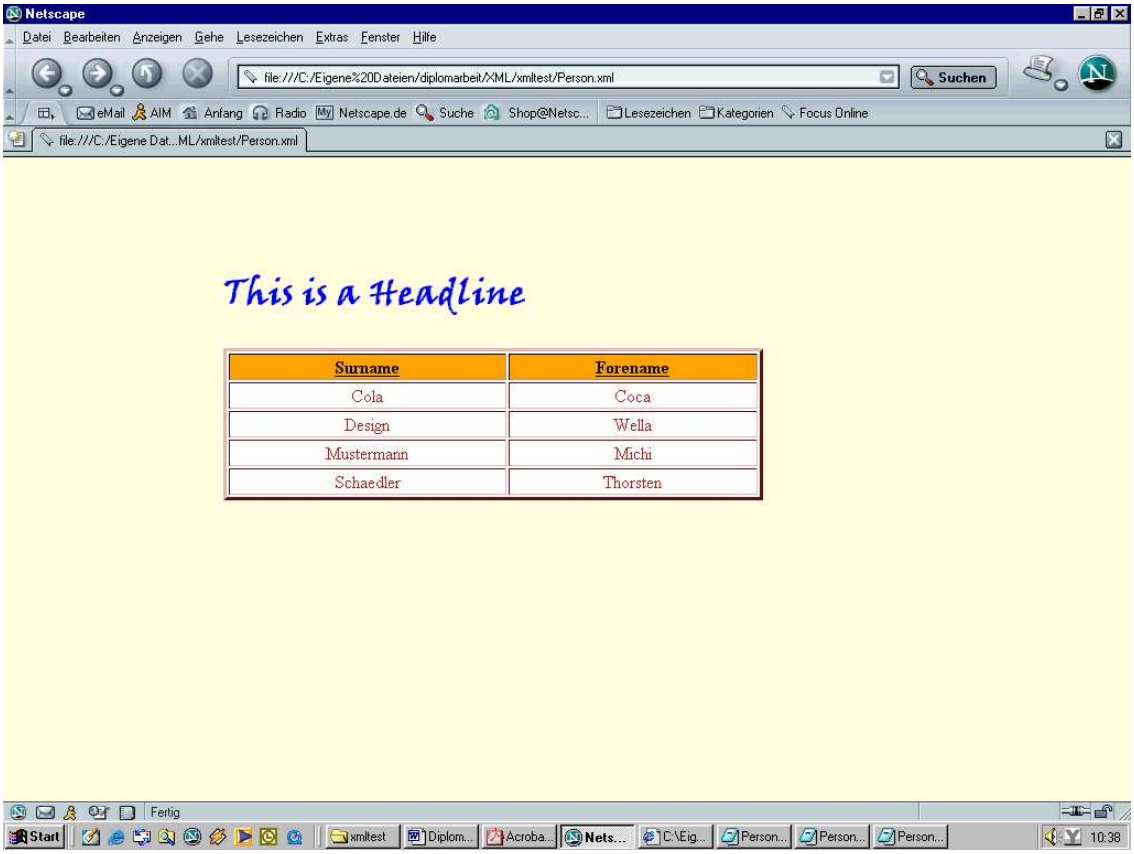

Abbildung 46 : Formatiertes XML-Dokument

Die Darstellung eines mittels XSLT transformierten XML-Dokuments ist im Internet Explorer 6 und Netscape Navigator 7.1 problemlos möglich, da in diesen Browsern ein XML-Parser[232] bzw. ein XSLT-Parser[233] integriert ist.

[232] S.a. 5.1.3 .
[233] S.a. 5.3.1.1.1 .

In Opera 7.21 als auch dessen Vorgänger wurde ein solcher XML-Parser bzw. XSLT-Parser jedoch nicht standardmäßig integriert und daher lässt sich die Darstellung eines XML-Dokuments als Baumstruktur oder als formatierte Tabelle im Opera-Browser nicht realisieren.

5.3.1.1.3 XPath

Der primäre Zweck von XPath besteht darin, Teile wie Elemente oder Gruppen von Elementen eines XML-Dokuments zu adressieren bzw. auszuwählen. XPath ist eine Art Hilfssprache, welche erforderlich ist, damit XSLT seine Aufgaben wahrnehmen kann. XPath hat dabei mit der Adressierung von Daten, Definition logischer Ausdrücke und dem bereitstellen zusätzlicher Funktionen drei wichtige Aufgaben zu erfüllen [Vgl. Sht03c].

Innerhalb der XSLT-Stylesheets kann mit XPath die Übersetzung der XML-Ausgangsdaten in den Ergebnisbaum (z.B. HTML) genauer kontrolliert und dabei zusätzliche Features bereitgestellt werden. Hierzu stehen die XPath-Funktionen zur Verfügung welche im Normalfall einen Wert zurückliefern [Vgl. Sht03e].

XPath Funktionen bestehen aus einem Funktionsnamen, gefolgt von runden Klammern, in denen die benötigten Argumente stehen [Vgl. Sht03e].

XPath Funktionen kommen meistens innerhalb von Wertzuweisungen an Attribute von XSLT-Elementen zum Einsatz. Da diese Wertzuweisungen selbst schon in (doppelten) Anführungszeichen stehen, muss für eine Zeichenkette die innerhalb einer XPath-Funktion übergeben wird, einfache Anführungszeichen verwendet werden [Vgl. Sht03e].

5.3.1.2 XSLFO

Eine weitere Möglichkeit ein XML Dokument zu formatieren bzw. zu präsentieren, besteht in der Verwendung der XSL Formatting Objects (XSLFO). Hierzu stellt XSLFO mehrere Objekte zum Erzeugen von Tabellen und Formularen, zum Formatieren von Überschriften, Absätzen und dergleichen zur Verfügung.

5.3.2 XML Linking

Um XML Dokumente miteinander zu verbinden wurde die XML-basierte Sprache XML Linking eingeführt. XML Linking besteht zum Einen aus einer Sprache zur Definition von Verweisen (XLink) und zum Anderen aus einer Sprache für die Adressierung von Verweiszielen (XPointer) [Vgl. HäPeS00, S.448].

5.3.2.1 XLink

Mit der XML Linking Language (XLink) besteht die Möglichkeit einfache als auch erweiterte Links zu erzeugen.

Einfache Links entsprechen der Funktionalität eines HTML Verweises[234], wohingegen die erweiterten Links eine Reihe wesentlicher Neuerungen mit sich bringen. So kann beispielsweise ein erweiterter Link auf mehrere Ziele verweisen [Vgl. HäPeS00, S.448].

„Einfache Links bieten eine Kurzschreibweise für eine häufige Art von Link, nämlich einen ausgehenden Link mit genau zwei teilnehmenden Ressourcen (worunter die HTML-artigen `A`- und `IMG`-Links fallen). Da einfache Links eine geringere Funktion als erweiterte Links bieten, haben sie keine spezielle innere Struktur.“ [Vgl. W3Ce03a]

5.3.2.2 XPointer

Die XML Pointer Language (XPointer) ist eine Erweiterung von XPath[235] .Es kann wie XPath Elemente, Attribute und Attributwerte adressieren und darüber hinaus auch Bereiche sowie Anfangs- und Endpunkte eines Bereichs ansprechen [Vgl. eds03].

„Diese Erweiterung ermöglicht, auch innerhalb von Elementen Teile eines XML-Dokuments zu adressieren. Wie diese Funktion für die Verlinkung genutzt werden kann, legt *XLink* fest“ [eds03].

---

[234] S.a. 3.5.1 .
[235] S.a. 5.3.1.1.3 .

5.3.3 XML-Namensräume[236]

In einem XML-Dokument kann es vorkommen gleiche Elemente oder Attribute mit unterschiedlichen Bedeutungen zu verwenden. Wenn der Parser mit Hilfe der entsprechenden DTD[237] prüft ob das XML-Dokument gültig ist, kann er die Elemente nicht unterscheiden. Somit wäre es aufgrund dieser Mehrdeutigkeit völlig unklar was ein solches Element nun enthalten dürfte und es käme zu Namenskonflikten (name collisions). Um solche Namenskonflikte zu vermeiden wurde 1999 vom W3C eine Empfehlung in Form der XML-Namensräume herausgegeben.

XML-Namensräume sind eine Zusammenstellung von Namen die in XML-Dokumenten als Namen für Elementtypen und Attributnamen verwendet werden.

Namensräume werden mit einem eindeutigen Bezeichner voneinander unterschieden. Für das XML-Namensraumkonzept werden hierfür URIs (Uniform Resource Identifier) verwendet, welche jedoch nicht auf ein tatsächlich existierendes Dokument verweisen müssen. Um Namensräume zu deklarieren wird das `xmlns`-Attribut verwendet. Eine Namensraumdeklaration stellt dabei die Verbindung zwischen *Namensraumpräfix* und *Namensraumnamen* her und gilt nur für das Element in dem es angegeben wurde.

Eine solche Namensraumdeklaration ermöglicht dabei auch die Verwendung von HTML-Code in einem XML-Dokument.

Wird innerhalb eines Elements bzw. des Wurzelelements die Deklaration `xmlns:html="[verwendeter Namensraum]"` angegeben, so kann HTML-Code direkt in XML-Code verwendet werden. Dabei muss innerhalb der Namensraumdeklaration (hier: das Element `<Transformation>`) jedem Element, das in HTML transformiert werden soll das Präfix `html:` vorangestellt werden.

Code 38 zeigt eine solche Transformation.

---

[236] Basierend auf [wat03] .
[237] S.a. 2.4 bzw. 5.2 .

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<?xml-stylesheet  type="text/css"?>
   -- use of the html namespace
<Transformation xmlns:html="http://www.w3.org/TR/REC-html40">
<html:body style="background-color:lightyellow">
<html:form style="margin-left:300px;margin-top:200px">
<html:h1>Using XML Namespaces</html:h1>
<html:table border="2" >
<html:tr>
<html:td>Namespace</html:td>
</html:tr>
<html:tr>
<html:td><html:a style="color:red" href="Person.xml">a simple
Link without using XML Linking Language</html:a></html:td>
</html:tr>
</html:table>
</html:form>
</html:body>
</Transformation>
```

Code 38 : HTML in einem XML-Dokument

Abbildung 47 zeigt die Darstellung von Code 38 im MS Internet Explorer. Der Netscape Navigator kann diese Form der Darstellung allerdings nicht erzeugen.

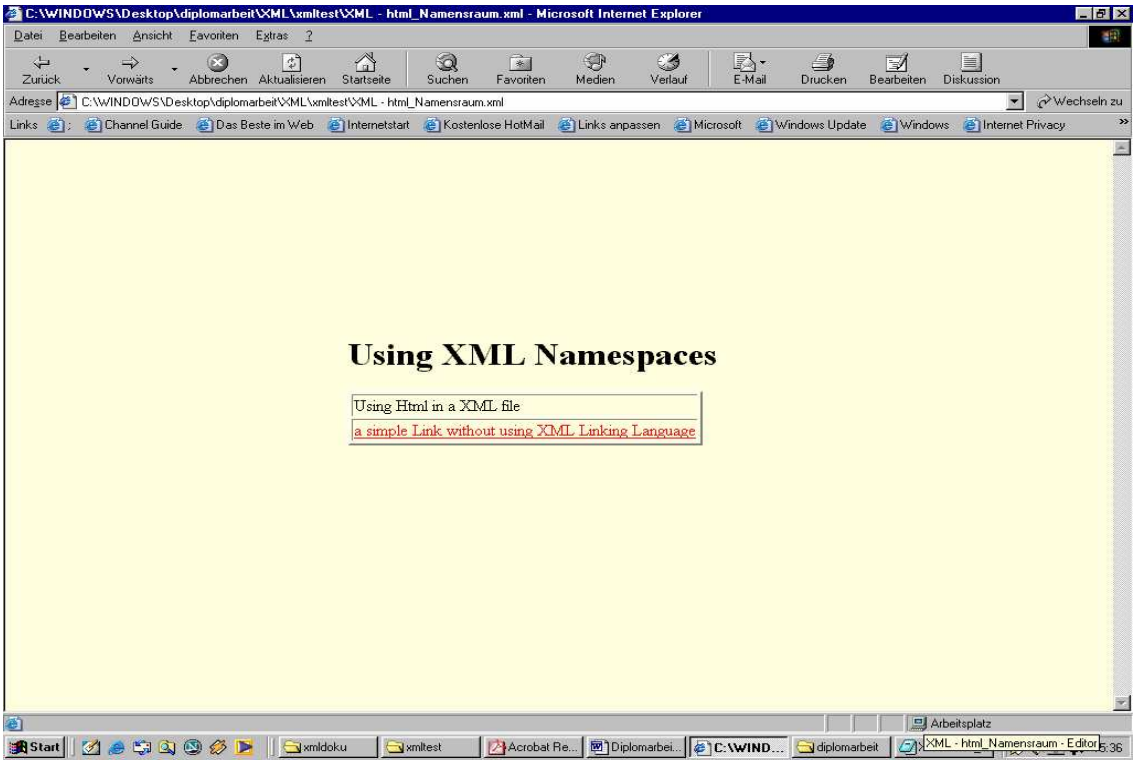

Abbildung 47 : HTML in einem XML-Dokument

5.3.4 Einsatz von Skriptsprachen in XML

Skriptsprachen können in einem XML-Dokument unter Verwendung des Namensraums[238] `xmlns:html="[verwendeter Namensraum]"` und einem damit verbundenen `html:` Prafix eingebettet werden. Mit Hilfe dieses Namensraums kann in einem XML-Dokument das `script`-Element in gleicher Weise wie in einem HTML-Dokument verwendet werden. Somit kann auch innerhalb eines XML-Dokuments mittels Object Rexx ein OLE Objekt[239] erzeugt und verwendet werden.

Code 39 soll dies anhand eines Beispiels verdeutlichen.

```
<?xml version='1.0' ?>
<?xml-stylesheet type="text/css" ?> -- use of CSS in this file
  -- XML rootelement ORexx using the html namespace
<ORexx xmlns:html="http://www.w3.org/TR/REC-html40">
 -- XML element Headline - formated by using CSS
<Headline style="color:red; font-size:24pt">Demonstration of
 using Object Rexx in a XML file </Headline>
<html:br/>
<file>XML</file>                          -- XML element file
<html:body bgcolor="lightyellow"/>
<html:br/>
<script>Object Rexx</script>              -- XML element script
<html:br/>
<html:table border="2">
<html:tr>
<html:td>Sale :</html:td>
<html:td><html:input id="plus" type="text"/></html:td>
</html:tr>
<html:tr>
<html:td>Cost :</html:td>
<html:td><html:input id="minus" type="text"/></html:td>
</html:tr>
<html:tr>
<html:td><html:input type="button" value="Excel" onclick="call
   Excel" language="Object Rexx"/></html:td>
</html:tr>
</html:table>
          <!--Beginning of Object Rexx script -->
<html:script type="text/Object Rexx" >
   -- routine Excel invoked by pressing the "Excel" button
::routine Excel public
    -- Instantiation of the Excel object
EO = .OLEObject~New("Excel.Application")
```

[238] S.a. 5.2.4 .
[239] S.a. 4.2.2.2.6 .

```
EO~Visible = .True                          -- Excel is visible
EO~WorkBooks~Add                            -- adds a new workbook
 -- value of the “plus” input field is saved in the variable x
x=plus~value
 -- value of the “minus” input field is saved in the variable y
y=minus~value
z=x-y
EO~Cells(1,A)~value = Sale      -- Cell A1 gets the value Sale
EO~Cells(3,A)~value = Cost      -- Cell A3 gets the value Cost
EO~Cells(5,A)~value = Profit    -- Cell A5 gets the value Profit
EO~Cells(1,C)~value = x         -- Cell C1 gets the value of x
EO~Cells(3,C)~value = y         -- Cell C3 gets the value of y
EO~Cells(5,C)~value = z     -- Cell C5 gets the value of z
</html:script>
            <!-- End of Object Rexx script -->
</ORexx>
```

Code 39 : Object Rexx in einem XML-Dokument[240]

Im Unterschied zu HTML ist es hier allerdings zwingend erforderlich in Zeile 25 das `language`-Attribut in Verbindung mit dem Event-Handler[241] zu verwenden.

Abbildung 48 zeigt abschließend eine Ausführung des Object Rexx Skripts aus Code 39 im MS Internet Explorer.

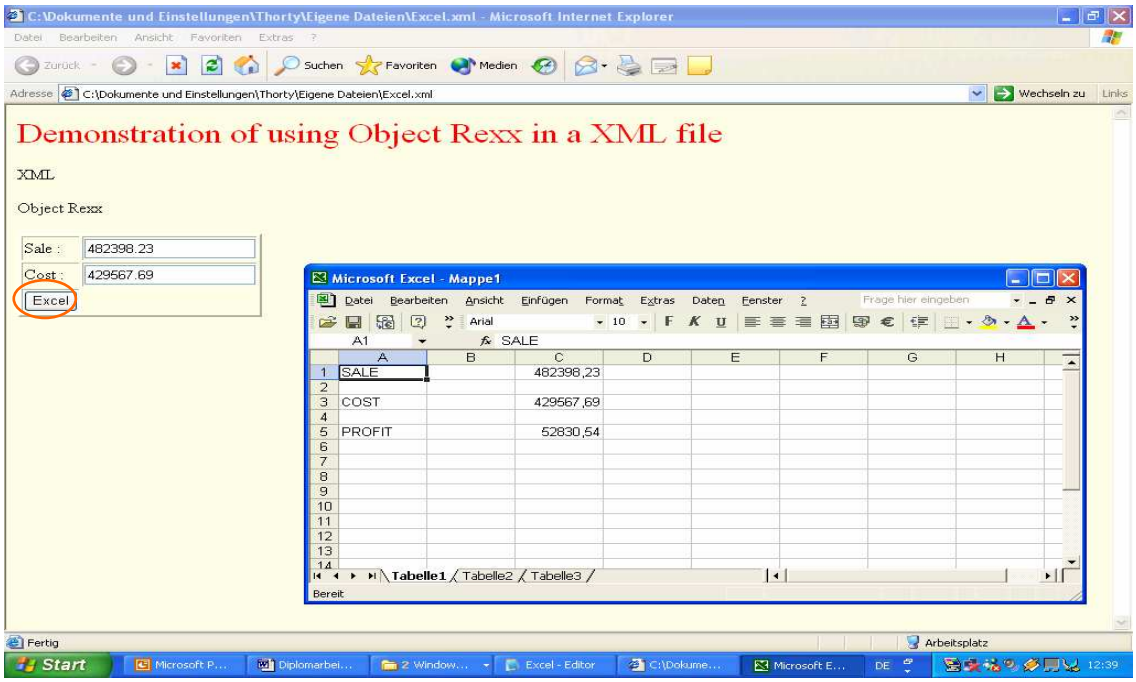

Abbildung 48 : Object Rexx in einem XML-Dokument

[240] Basierend auf [PeeL03] .
[241] S.a. 3.6.1 .

5.4 XML Schema

Bisher konnten XML-Dokumente mittels einer DTD[242] auf Gültigkeit überprüft werden. Mit Einführung der XML-Namensräume[243] und den XML-Derivaten[244] [Sht01f] und einer damit verbundenen steigenden Komplexität eines XML-Dokuments, wurde es zunehmend schwerer mittels einer DTD ein XML-Dokument auf Gültigkeit zu prüfen. So wurde 1999 vom W3C begonnen eine XML Schema Definitionssprache zu entwickeln, mit der solche komplexeren Applikationen überprüft werden konnten.

XML Schema wurde ursprünglich von Microsoft vorgeschlagen und ist seit dem 2. Mai 2001 eine offizielle W3C Empfehlung [Vgl. W3s03ca], welche in drei Teile (Einführung,[245] Strukturen[246], Datentypen[247]) unterteilt ist .

Der Zweck eines XML Schemas (**.**xsd) ist es die Bestandteile eines XML Dokumentes zu definieren. Aus diesem Grund wird ein Dokument das einem gewissen Schema entspricht auch oft als „Instanz“ bezeichnet. Weder Instanzen noch Schemata müssen per se als Dokument existieren. „Sie können genauso gut Datenströme aus Bytes sein, Felder in einem Datenbanksatz oder eine Sammlung von XML-Infoset Informationseinheiten.“[W3Ce03b]. Jedes Element im Schema enthält dabei ein Präfix (üblicherweise `xsd:` oder `xs:`) welches über die Deklaration `xmlns:[Päfix]="http://www.w3.org/2001/XMLSchema"` beim `schema`-Element mit dem XML Schema Namensraum assoziiert ist [Vgl. W3Ce03b].

XML Schemas sind die Nachfolger der DTDs und werden diese wohl auch bald ersetzen. Hierfür sprechen mehrere Gründe. XML Schemas sind erweiterbar für künftige Anforderungen, reichhaltiger und nützlicher als DTDs, in XML geschrieben und unterstützen Datentypen als auch Namensräume[248]. So können beispielsweise eigene Datentypen definiert und verwendet werden, Daten zwi-

[242] S.a 2.4 bzw. 5.2. .
[243] S.a. 5.3.3.
[244] S.a. 6 .
[245] http://www.edition-w3c.de/TR/2001/REC-xmlschema-0-20010502/ , Abruf am 2003-10-12.
[246] http://www.edition-w3c.de/TR/2001/REC-xmlschema-1-20010502/ , Abruf am 2003-10-12.
[247] http://www.edition-w3c.de/TR/2001/REC-xmlschema-2-20010502/ , Abruf am 2003-10-12.
[248] S.a 5.3.3 .

schen verschiedenen Datentypen ausgetauscht werden oder Schemas mit dem XML-DOM manipuliert als auch mit XSLT[249] transformiert werden [Vgl. W3s03ca].

In XML Schema wird grundlegend zwischen komplexen und einfachen Typen unterschieden [Vgl. W3Ce03b].

Komplexe Typen dürfen Elemente und Attribute enthalten. Diese Typen werden mit dem `complexType` Element definiert und enthalten eine Menge von Elementdeklarationen, Elementreferenzen und Attributdeklarationen. Elemente werden dabei mit dem `element`-Element und Attribute mit dem `attribute`-Element deklariert [Vgl. W3Ce03b].

Es gibt vier Arten von komplexen Elementen [Vgl. W3s03cb]:

- Leere Elemente
- Elemente die nur andere Elemente beinhalten
- Elemente die nur Text beinhalten
- Elemente die andere Elemente und Text beinhalten

Beispiel :

```
<xsd:element name="Body" type="bodytype"/>
 <xsd:complexType name="bodytype">
    <xsd:element name="Person" type="persontype"/>
 </xsd:complexType>
 <xsd:complexType name="persontype">
    <xsd:sequence>
      <xsd:element name="Forename" type="xsd:string"/>
      <xsd:element name="Surname" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="number" type="xsd:int"/>
 </xsd:complexType>
```

Einfache Typen dagegen können lediglich Text beinhalten. Der Text kann aus vielen verschiedenen Typen bestehen. Es kann sich dabei um vordefinierte Typen wie `string, int, date, boolean` usw. handeln, welche in der XML Schema Definition enthalten sind oder um selbst definierte bzw. abgeleitete Typen [Vgl. W3s03cc].

---

[249] S.a. 5.3.1.1 .

Syntax : `<[Präfix]:element name="[Elementname]" type="[Typname]">`

Beispiel : `<xsd:element name="Headline" type="xsd:string"]>`

Neue einfache Typen hingegen werden mit dem `simple-type` Element definiert und benannt. Um den bestehenden Basistyp anzugeben wird das `restriction`–Element und dessen `base`-Attribut verwendet. Innerhalb dieses Elements werden dann die Facetten angegeben, welche den Datentyp genauer spezifizieren [Vgl. W3Ce03b].

So wird im Folgenden Beispiel ein neuer, aus dem `string`-Datentyp abgeleiteter, einfacher Datentyp mit Namen Bundesland definiert und mit Hilfe der Enumerationsfacette (`enumeration`) spezifiziert.
XML Schema definiert insgeamt 15 Fassetten. Durch die Verwendung von einer oder auch mehrer Fassetten können die erlaubten Werte eines einfachen Datentyps eingeschränkt werden [Vgl. W3Ce03b].

Beispiel[250] *:*
```
<xsd:simpleType name="Bundesland">
      <xsd:restriction base="xsd:string">
          <xsd:enumeration value="Baden-Württemberg"/>
               <xsd:enumeration value="Bayern"/>
               <xsd:enumeratin value="Berlin"/>
               <!-- usw. -->
               <xsd:enumeration value="Thüringen"/>
          </xsd:restriction>
  </xsd:simpleType>
```

Innerhalb eines XML-Dokuments wird genau wie bei verwenden einer DTD, eine Referenz auf das zu verwendende Dokument eingebettet. Diese Referenz muss dabei im Wurzelelement des XML-Dokuments verankert werden [Vgl. W3s03cd].

Syntax :
```
<[Wurzelelement] xmlns:[Präfix]="Namensraum"
      [Präfix]:schemaLocation="[Namensraum][Dokumentname].xsd">
```

Beispiel :
```
<Person xmlns:xsd="http://www.w3.org/XML-Schema"
      xsd:schemaLocation="http://www.W3schools.com person.xsd">
```

[250] Entnommen aus [W3Ce03b] .

6. XML-Derivate

Unter XML-Derivaten versteht man Auszeichnungssprachen, welche auf Basis von XML definiert wurden. Dementsprechend ist HTML ein SGML-Derivat, da es mittels SGML definiert wurde. So gibt es vom W3C mehrere veröffentlichte XML-Derivate, von denen einige im Folgenden kurz vorgestellt und veranschaulicht werden [Vgl. Sht03f].

6.1 XHTML

XHTML steht für e**X**tensible **HTML**. Die erste Version wurde am 26 Januar 2000 vom W3C verabschiedet.

„XHTML ist eine Familie aktueller und zukünftiger Dokumenttypen und Module, die HTML 4 nachbilden, eine Untermenge davon bilden bzw. es erweitern. Die XML basierten Dokumenttypen aus der XHTML-Familie sind letztlich darauf ausgelegt, in Kombination mit auf XML basierenden Benutzerprogrammen eingesetzt zu werden“ [W3Ce03c].

6.1.1 XHTML 1.0

In der XHTML-Familie ist XHTML 1.0 der erste Dokumenttyp. Dieser Dokumenttyp ist „eine Neuformulierung der drei HTML 4 Dokumenttypen als Anwendungen von XML 1.0.“ [W3Ce03c].

„Es ist für die Verwendung als Sprache für Inhalte vorgesehen, die sowohl zu XML konform sind, und die auch, falls sie einigen einfachen Richtlinien folgen, in zu HTML 4 konformen Benutzerprogrammen eingesetzt werden können. Entwickler, die ihre Inhalte auf XHTML 1.0 umstellen, werden die folgenden Vorteile erkennen.“ [W3Ce03c] :

1.) XHTML-Dokumente sind konform zu XML. Als solche können sie ohne weiteres mit Standard-XML-Werkzeugen angezeigt, bearbeitet und validiert werden.

2.) XML-Dokumente können so formuliert werden, dass sie in bereits existierenden, zu HTML 4 konformen Benutzerprogrammen ebenso gut oder besser funktionieren als zuvor, ebenso wie in neuen, zu XHMTL 1.0 konformen Benutzerprogrammen.

3.) XHTML-Dokumente können Applikationen (z.B. Skripte und Applets) einsetzen, die auf dem HTML Document Object Model (DOM) oder dem XML-DOM basieren.

4.) Falls sich die XHTML-Familie weiterentwickelt, ist es wahrscheinlicher, dass Dokumente, die konform zu XHTML 1.0 sind, innerhalb und zwischen unterschiedlichen XHTML-Umgebungen kombiniert werden können.

In der Evolution des Internet stellt die XHTML-Familie den nächsten Schritt dar. Entwickler von Inhalten, die heute auf XHTML umsteigen, können in die XML-Welt vordringen und die dortigen Vorteile ausnützen. Dabei kann ein Entwickler darauf vertrauen, dass der Inhalt vorwärts- als auch rückwärtskompatibel ist [Vgl. W3Ce03c].

Da XHTML voll kompatibel zu den anderen XML-Derivaten ist, kann es in diese als Dateiinsel eingebunden werden und umgekehrt. Ferner besteht aufgrund der gemeinsamen XML-basierten Grundlage die Möglichkeit Skriptsprachen zu vereinheitlichen [Vgl. MüNe02, S.387f].

Da es sich bei XHTML um eine XML-Anwendung handelt, müssen einige Änderungen vorgenommen werden [Vgl. W3Ce03c] .

In diesem Kapitel werden daher die wichtigsten Unterschiede zwischen HTML 4 und XHTML 1.0 angesprochen [Vgl. MüNe02, S.388ff].

1. Neben dem `text/html` Mime Typ, können in einem XHTML-Dokument auch die Mime Typen `text/xml`, `application/xml` sowie `application/xhtml+xml` verwendet werden.
2. Ein XHTML-Dokument kann die Endung .htm, .html, .xhtml oder .xml erhalten. Bei .html oder.htm wird der HTML -Parser zum verarbeiten

des Dokumentes verwendet. In den beiden anderen Fällen kommt der XML-Parser zum Einsatz.

3. In einem XML-Dokument ist eine XML-Deklaration[251] enthalten.
4. XHTML verwendet einen anderen Dokumenttyp.
5. Im `html`-Element sollte der Namensraum für XHTML angegeben werden. Hierzu wird das Attribut `xmlns` innerhalb des einleitenden `<html>` Tags verwendet.
6. In XHTML muss ein HTML-Dokument zwingend das Grundgerüst mit den Elementen `html`, `head` und `body` angegeben werden.
7. XHTML unterscheidet strikt nach Groß- und Kleinschreibung.
   „Das XML-Dokumentobjektmodell spezifiziert, dass Element- und Attributnamen so zurückgegeben werden, wie sie angegeben wurden. In XHTML 1.0 werden Elemente und Attribute in Kleinbuchstaben angegeben“[W3Ce03c].
8. Alleinstehende Tags müssen am Ende die Zeichenfolge „/>“ enthalten.
9. Alle Elemente müssen ein Anfangs- und Endtag verwenden.
10. Allen Attributen muss explizit ein Wert zugewiesen werden, welcher in Anführungszeichen stehen muss.
11. Die Definition eines Ankers[252] muss zusätzlich das `id`-Attribut enthalten.
12. Der Inhalt von `script-` und `style`-Elementen muss in einen mit CDATA markierten Abschnitt (`<![CDATA` ..nicht maskierter Inhalt..`]]`) eingehüllt werden. Diese Abschnitte werden vom XML-Parser erkannt und erscheinen als Knoten im DOM [Vgl. W3Ce03c].

6.1.2 Modularisierung von XHTML

Unter Modularisierung wird das Konzept des Abspeckens und Erweiterns von XHTML bezeichnet [Vgl. MüNe02, S.399].

„XHTML-Modularisierung ist eine Aufteilung von XHTML 1.0 – und durch Verweise auch von HTML 4 – in eine Sammlung von abstrakten Modulen, die be-

[251] S.a. 5.1.2.1 .
[252] S.a. 3.5.1 .

stimmte Funktionstypen anbieten. Diese abstrakten Module werden in dieser Spezifikation unter Verwendung von XML-Dokumenttyp-Deklarationen implementiert; eine Implementierung mit XML-Schema[253] wird erwartet. Die Regeln zur Definition der abstrakten Module und zur Implementierung mit XML-DTDs werden ebenfalls in diesem Dokument definiert. Diese Module können miteinander und mit anderen Modulen kombiniert werden, um Teilmengen von XHTML oder erweiterte Dokumenttypen zu schaffen, die sich als Teil der Familie von XHTML-Dokumenttypen auszeichnen.“ [W3Ce03d]

Mit *XML Basic* steht dabei ein Dokumenttyp zur Verfügung, welcher die Module einschließt, die mindestens erforderlich sind um ein Dokumenttyp der XHTML Muttersprache zu sein [Vgl. W3Ce03e].

Es stellt sozusagen die einfachste Version eines XHTML Dokuments dar, welche durch hinzufügen von abstrakten XHTML-Modulen erweitert werden kann.

Unter einem abstrakten Modul wird dabei „eine Einheit einer Dokumenttyp-Spezifikation korrespondierend mit einer bestimmten Art von Inhalt, korrespondierend mit einem Markup-Konstrukt, die diese bestimmte Art widerspiegelt“, verstanden [W3Ce03d].

Tabelle 15 stellt die abstrakten XHTML-Module kurz vor. Welche Elemente und Attribute die entsprechenden Module beinhalten kann unter [W3Ce03d] oder [W3C03b] nachgelesen werden.

Modul	Beschreibung
Structure	Definiert die hauptsächlichen Strukturelemente für XHTML.
Text	Definiert die Basis-Textcontainer-Elemente, Attribute und ihr Inhaltsmodell.
Hypertext	Stellt das Element zur Verfügung, das zur Definition von Hyperlinks zu anderen Ressourcen benutzt wird.

[253] S.a. 5.2.5 .

List	Stellt listenorientierte Elemente zur Verfügung.
Object	Stellt Elemente für die Einbindung von allgemeinen Objekten zu Verfügung.
Presentation	Definiert Elemente, Attribute und ein minimales Inhaltsmodell für einfache darstellungsorientierte Auszeichnungen.
Edit	Definiert Elemente und Attribute zum Redigieren von Texten.
Bidirectional Text	Definiert ein Element für die Definition der Schreibrichtung des Elementinhalts.
Basic Forms	Stellt Formularelemente in eingeschränkter Form zur Verfügung.
Forms	Stellt alle Formularfähigkeiten von HTML 4.0 zur Verfügung.
Basic Tables	Stellt Tabellenelemente in eingeschränkter Form zur Verfügung.
Tables	Stellt Tabellenelemente zur Verfügung, auf die nicht visuelle Benutzerprogramme besser zugreifen können.
Image	Stellt die Grundlage für die Einbindung von Bildern dar und kann in einigen Implementationen unabhängig von client-seitigen Image Maps benutzt werden.
Client–side Image Map	Stellt Elemente für client-seitige Image Maps zur Verfügung. Es setzt das Image-Modul (oder ein anderes Modul, das das `img`-Element enthält) voraus.
Server–side Image Map	Unterstützt Auswahl und Übertragung von Bildkoordinaten. Es setzt das Image-Modul (oder ein anderes Modul, das das `img`-Element enthält) voraus.
Frames	Stellt Elemente für Frames zur Verfügung.
Target	Fügt das `target`-Attribut zu den Link-

	definierenden Elementen hinzu. Es kann in Dokumente eingebaut werden die in Frames auftreten, und auch in Dokumente, die `target` verwenden, um ein neues Fenster zu öffnen.
Iframe	Definiert ein Element für Inline Frames.
Intrinsic Events	Eingebaute Ereignisse sind Attribute einiger Elemente. Diese Elemente zeichnen sich dadurch aus, dass ein bestimmtes Ereignis eintritt, wenn der Benutzer eine bestimmte Handlung ausführt. Die Attribute werden *nur dann* der Attributmenge der jeweiligen Elemente hinzugefügt, wenn die Module, die diese Elemente definieren, auch ausgewählt wurden.
Metainformation	Definiert ein Element, das Information innerhalb des deklarativen Teils eines Dokuments beschreibt (in XHTML das `head`-Element).
Scripting	Definiert Elemente, die Informationen bezüglich ausführbarer Programme oder bezüglich der fehlenden Unterstützung für die Ausführung solcher Programme enthalten.
Style Sheet	Definiert ein Element zur Deklaration von internen Stylesheets.
Style Attribute	Definiert das `style`-Attribut und aktiviert die Style-Attributsammlung.
Link	Definiert ein Element, mit dem Links zu externen Ressourcen definiert werden können.
Base	Definiert ein Element, mit dem ein Basis-URI definiert werden kann, bezüglich dessen relative URIs im Dokument aufgelöst werden.

Tabelle 15 : Liste der XHTML-Module[254]

[254] Nach [W3Ce03d] .

Die Module Structure, Text, Hypertext und List gehören zu den so genannten Kernmodulen (Core Moduls), welche notwendigerweise in jedem zur XHTML-Familie konformen Dokumenttyp enthalten sein müssen [Vgl. W3Ce03d].

Daneben gibt es noch das Name Identification– und das Legacy Modul, welche jedoch missbilligt werden, und insofern nicht in Tabelle 15 enthalten sind [Vgl. W3Ce03d].

XHTML Basic besteht dabei aus den Modulen Structure, Text, Hypertext, List, Basic Forms, Basic Tables, Image, Object, Metainformation, Link und Base [Vgl. W3Ce03e].

6.1.3 XHTML 1.1 - Modulbasiertes XHTML

Diese W3C Empfehlung vom 31.5.2001 definiert einen neuen voll funktionsfähigen XHTML Dokumenttyp mit umfangreichen semantischen Eigenschaften , der auf dem Modulrahmenwerk als auch den Modulen basiert, die in der „Modularisierung von XHTML“ definiert sind. Somit enthält der XHTML1.1 Dokumenttyp keine der missbilligten Funktionen von XHTML 1.0 oder HTML 4. Im Wesentlichen stellt dieser Dokumenttyp eine Neuformulierung von XHTML 1.0 *Strict* (reines Markup ohne Layoutspezifikationen) unter Verwendung von XHTML Modulen dar. Insofern fehlen diesem Dokumenttyp viele Fähigkeiten wie beispielsweise das Darstellen von Frames, die in anderen Dokumenttypen der HTML Familie (z.B. XHTML-Frames) enthalten sind. Diese Fähigkeiten sind allerdings als Module verfügbar [Vgl. W3Ce03f].

Der XHTML 1.1 Dokumenttyp umfasst dabei die Module aus Tabelle 15 mit Ausnahme der Module Frames, Target und Iframe. Allerdings verwendet XHTML 1.1 zusätzlich noch das Ruby Annotation Modul [Vgl. W3Ce03f].

6.2 WML[255]

WML steht für **W**ireless **M**arkup **L**anguage und wurde in einer Zusammenarbeit des WAP Forums und des W3C´s entwickelt.

Aufgrund der Einschränkungen wie Speicherplatz und Datenübertragungsrate als auch kleinen Displays, konnte HTML nicht zur Darstellung von Informationen auf mobilen Endgeräten wie einem Handy oder einem PDA (Personal Digital Assistent) verwendet werden. Um solche Informationen aus dem Internet (normaler Server) auf die mobilen Endgeräte übertragen zu können, wurde das Wireless Application Protocol (WAP) entwickelt [Vgl. KoRP03a].

Die Anfrage eines WML-Dokuments (reine ASCII Text Datei mit der Endung .wml [Vgl. KoRP03b]) von einem mobilen Endgerät muss dabei über ein Gateway erfolgen. Ein solches Gateway ermöglicht die Kommunikation zwischen einem mobilen Endgerät und dem WAP Angebot im Internet [Vgl. KoRP03a].

Die Verwendung von WML verursachte allerdings einige Probleme, da die Seiten in HTML als auch in WML vorliegen mussten.

Wie in HTML können auch WML-Dokumente dynamisch mit Hilfe von Web Technologien wie CGI-Skripten, ASP[256] oder JSP[257] erzeugt werden. Somit sind auch für mobile Endgeräte Datenbankabfragen u.a. möglich [Vgl. KoRP03a].

Um ein WML-Dokument am PC betrachten zu können wird ein fähiger WAP Browser benötigt. Winwap stellt einen solchen WAP-Browser kostenpflichtig zur Verfügung. Zum Testen kann eine kostenlose und 30 Tage gültige Testversion des WAP-Browsers WinWap3 Pro herunter geladen werden.[258] Des Weiteren ist Opera, wenn auch beschränkt, in der Lage WML-Dokumente anzuzeigen. Der Internet Explorer 6 hingegen zeigt das selbe Dokument lediglich als Baumstruktur.

---

[255] Basierend auf [KoRP03] .
[256] S.a. 4.2.10 .
[257] S.a. 4.2.9 .
[258] http://www.winwap.org/winwap/download.html, Abruf am 2003-10-16.

6.2.1 Aufbau eines WML-Dokuments

Zu Beginn eines WML-Dokuments wird ein entsprechender XML-Prolog[259] angegeben. Anschließend wird der gesamte Inhalt der Datei innerhalb des `wml`-Elements, welches auch als *Deck* bezeichnet wird, notiert. Innerhalb dieses `wml`-(Wurzel-)Elements werden die einzelnen *Karten* `<cards>` verwendet [Vgl.KoRP03c]. Eine solche Karte dient als Container für den Inhalt in Form von Grafiken, Texten usw.[Vgl. KoRP03d] .

Wird ein WML-Dokument von einem mobilen Endgerät angefordert, werden alle Karten vom WAP-Server heruntergeladen [Vgl. W3s03d].

Eine solches `card`-Element verfügt dabei über die drei Attribute `id`, `title` und `newcontext` [Vgl.KoRP03d].

Das `id`-Attribut wird dabei wie in HTML verwendet. Mit dem `title`-Attribut wird der Titel der Karte definiert, welcher auf verschiedenste Weisen (Titelleiste, Bookmarktext) vom mobilen Endgerät verwendet werden kann. Durch das `newcontext`-Attribut wird der WAP-Browser-Context in einen wohldefinierten Zustand zurückgesetzt. Hat das Attribut den Wert `true`, so werden alle Variableninhalte im Browser-Context gelöscht und auch die History zurückgesetzt. Nimmt das Attribut den Wert `false` an, bleibt alles wie es ist [Vgl. KoRP03d].

Innerhalb einer solchen Karte können anschließend HTML-Elemente[260] dazu verwendet werden um Text, Tabellen, Grafiken als auch Verweise zu definieren. In einem WAP-Browser lassen sich allerdings nur WBMP-Grafikformate einbetten bzw. darstellen. Herkömmliche BMP Formate können mit Hilfe geeigneter Programme in WMBP-Formate umgewandelt werden [Vgl. KoRP03c].

6.2.2 Eventhandling in WML

In WML werden vier verschieden Ereignisse unterschieden. Die Ereignisse „`onenterforward`“, „`onenterbackward`“ und „`ontimer`“ werden bei Bedarf

[259] S.a. 5.1.2 .
[260] S.a. 3.5 .

innerhalb des einleitenden <card> Tags als Attribut angegeben oder über das <onevent> Tag in die Karte eingebunden. Die Events stehen dann nur innerhalb dieser Karte zur Verfügung [Vgl. KoRP03f].

„Das onevent-Tag <onevent> und </onevent> bindet einen bestimmten Task an ein Ereignis, welches über das type-Attribut des <onevent> Elements angegeben wird.“ [KoRP03f]

„Das onenterforward Event tritt ein, wenn eine Card oder ein Deck über einen go-Task, also einen Link, oder direkt durch die Eingabe der URL aufgerufen wird.“ [KoRP03f]

„Der onenterbackward-Event tritt ein, wenn eine Card oder ein Deck über die History angesprochen wird.“ [KoRP03f]

„Der ontimer-Event tritt ein, wenn ein vordefinierter Timer (Stopuhr) abgelaufen ist.“[KoRP01d]. Ein solcher Timer wird mittels des <Timer /> Elements definiert und durch die Attribute name und value spezifiziert. Das value Attribut gibt an (in Zehntel Sekunden) wie lange der Timer laufen soll, ehe ontimer das Ereignis auslöst bzw. die entsprechende Datei aufruft [Vgl. KoRP03f].

Das onpick Event wird hingegen als Attribut im einleitenden option-Tags[261] eingesetzt und tritt ein, wenn die entsprechende Option vom Anwender ausgewählt wird [Vgl. KoRP03f].

Um ein Event für das gesamte Deck und somit für alle Karten verfügbar zu machen, wird das template-Element benötigt.

[261] S.a. 3.5.5.3 .

6.2.3 WMLScript

Im Gegensatz zu JavaScript[262], Object Rexx[263] oder anderen Skriptsprachen, welche in ein HTML-Dokument eingebettet werden können, darf WMLScript nicht in das WML-Dokument eingebettet sein [Vgl. KoRP03g].

„WMLScript muss immer in einer separaten WMLScript-Datei implementiert werden“[KoRP01g] und wird über eine Funktion in der WML-Datei aufgerufen.

In der WMLScript-Datei werden die verschiedenen WMLScript Funktionen definiert, weshalb sie auch „*Compilation Unit*“ genannt wird [Vgl. KoRP03g].

In WMLScript wird zwischen lokalen-, externen- und Bibliotheksaufrufen unterschieden. In den sogenannten Bibliotheken sind verschiedene Funktionen definiert. Von einer Funktion wird in WMLScript immer ein Wert zurückgegeben [Vgl. KoRP03h].

Opera kann zwar zur Darstellung von WML-Dokumenten verwendet werden, ist allerdings nicht in der Lage ein solches WMLScript auszuführen. Mit WinWap 3.1 Pro dagegen lässt sich eine solches Skript problemlos verwenden.

6.2.4 Beispiel

Das folgende Beispiel soll nun den Zusammenhang zwischen den Karten in einem WML-Dokument, darin enthaltenen Events und einer WMLScript-Datei verdeutlichen. In Code 40 wird ein solches WML-Dokument dargestellt.

```
<?xml version="1.0"?>
<wml>
-- first card for Input is defined
 <card id="in" title="INPUT">
  <strong> number 1 :</strong>
  <input type="text" name="n1" size="5"/>  -- input of number 1
  <br/> <br/>
  <strong> number 2 :</strong>
  <input type="text" name="n2" size="5"/>   --input of number 2
  <br/><br/>
  -- invokes the MULTI function in the calculate.wmls file by
```

[262] S.a. 4.2.2.1 .
[263] S.a. 4.2.2.2 .

```
     pressing the “Multiplication” link
  <anchor> Multiplication <go href="calculate.wmls#MULTI()"/>
  </anchor>
  <br/> <br/>
  -- invokes the DIV function in the calculate.wmls file by
     pressing the “Division” link
  <anchor> Division <go href="calculate.wmls#DIV()"/>
  </anchor>
 </card>
-- second card for presenting the result of the calculation
 <card id="out" title="OUTPUT" ontimer=”calculate.wml”>
  <i><u>The result is :</u></i>
  <strong>$(RESULT)</strong>
      <timer value=”50”/>      -- timer is set to 5 seconds
 </card></wml>
```

Code 40 : calculate.wml

Code 41 zeigt die zu Code 40 gehörende WMLScript-Datei.

```
-- definition of the MULTI function to multiplicate the both
   input values
Extern function MULTI()
{
-- hand over the value of  number 1 and 2 with the method
   getVar()of the WML-Browser library
  var x = WMLBrowser.getVar("n1");
  var y = WMLBrowser.getVar("n2");
  var z = x * y;
-- set the z Variable (result)into the second card (id=”out”)
   and present it in another page
  WMLBrowser.setVar("RESULT",z);
  WMLBrowser.go("calculate.wml#out");
}
Extern function DIV()
{
--  definition of the DIV function to divide the both input
    values is in the same way as before – with one exception*
….
var z = x / y;  *
….
}
```

Code 41 : calculate.wmls

Abbildung 49 zeigt die Darstellung von Code 40 im WinWapBrowser (Karte1).

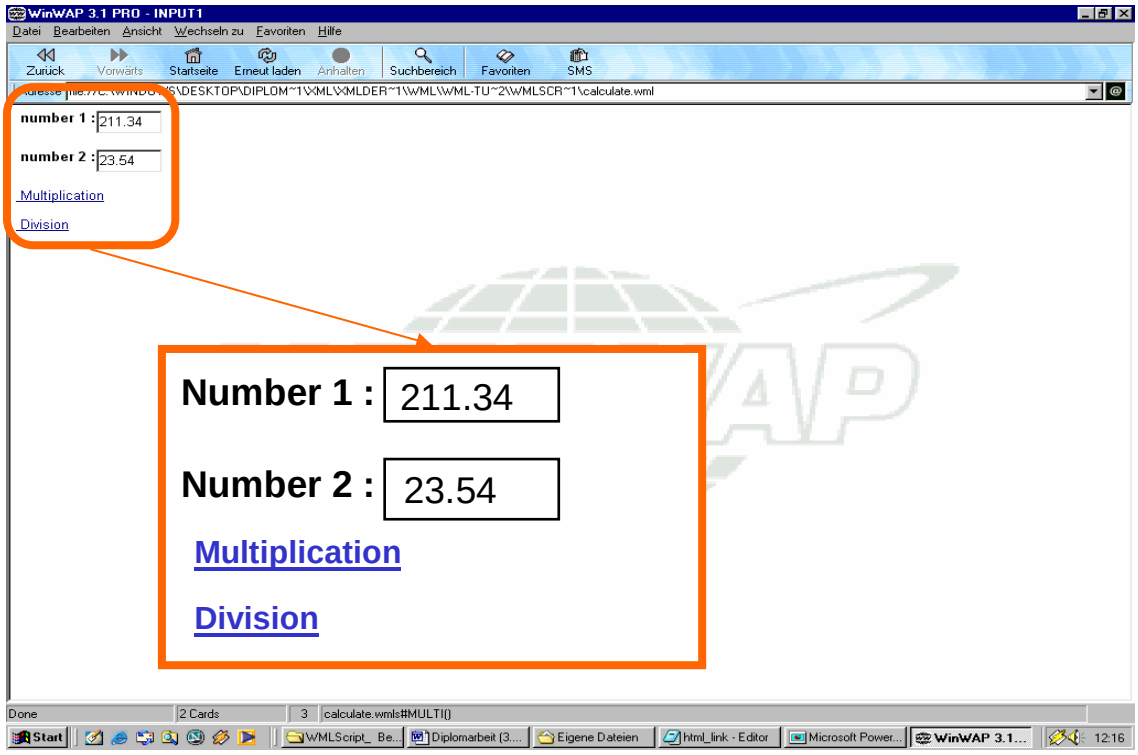

Abbildung 49 : calculate.wml im WinWap Browser

Abbildung 50 zeigt die Darstellung nach einer Multiplikation der beiden vorab eingegeben Zahlen in Karte 2.

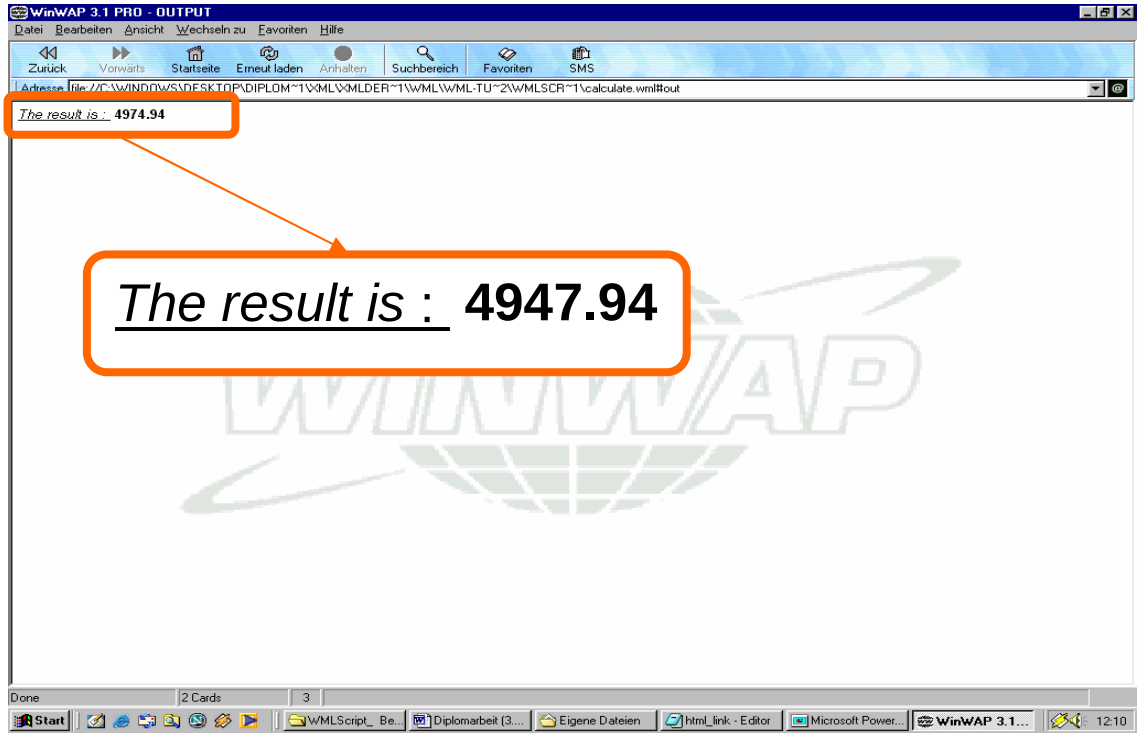

Abbildung 50 : calculate.wml#out im WinWap Browser

In diesem Beispiel wird in der WML Datei calculate.wml (Code 40) eine Karte für den Input (`id="in"`) und eine für den Output (`id="out"`) verwendet. Innerhalb der Input Karte werden zum Einen die Eingaben des Anwenders entgegengenommen und zum Zweiten über Verweise die gewünschten Funktionen aufgerufen. Diese Funktionen sind in der Datei calculate.wmls (Code 41) definiert.

Die Präsentation des Ergebnisses erfolgt anschließend in einer neuen Seite (Karte mit `id="out"`) im WAP Browser. In dieser zweiten Karte, wird auch ein `ontimer` Event[264] verwendet. Dieses Event wird dazu verwendet um nach fünf Sekunden wieder zur Datei calculate.wml (Code 40) zurückzukehren, damit eine erneute Berechnung durchgeführt werden kann.

6.3 SMIL[265]

Ein weiteres „XML-Derivat“ [Sht01f] ist die **S**ynchronized **M**ultimedia **I**ntegration **L**anguage (SMIL). SMIL steht derzeit als W3C Empfehlung in der Version 2.0 (7.August 2001)[266] zur Verfügung.

SMIL ist eine Sprache zum Erstellen von audiovisuellen Präsentationen, welche auf dem lokalem PC oder über das Internet präsentiert werden können.

In einer SMIL-Datei können mehrere verschiedene Dateitypen verwendet werden. Dabei kann angegeben werden ob die Dateitypen gleichzeitig oder in einer zeitlichen Abfolge verwendet werden sollen. Neben diesen Dateitypen können auch Kontrollbutton oder Links zu anderen SMIL-Präsentationen enthalten sein [Vgl. W3s03aa].

SMIL-Präsentationen können entweder in einer separaten SMIL-Datei gespeichert und anschließend mit Hilfe eines SMIL-Players abgespielt werden, oder in ein HTML-Dokument eingebettet werden [Vgl. W3s03aa].

---

[264] S.a. 6.2.2 .
[265] Basiernd auf [W3s03a] .
[266] http://www.w3.org/TR/2001/REC-smil20-20010807/ .

6.3.1 SMIL-Elemente

Zum Erstellen von Präsentationen gibt es verschiedene SMIL-Elemente. Ein SMIL-Element dient als Container für verschiedene weitere Elemente. So können die verschiedenen SMIL-Elemente miteinander kombiniert werden.

Im Rahmen dieser Arbeit werden die Elemente Paralell `par` [Vgl. W3s03ab], Sequence `seq` [Vgl. W3s03ac], und `media` [Vgl. W3s03ad], angesprochen. Mit Hilfe des `seq`-Elements können dessen Kinderelemente nacheinander am Bildschirm angezeigt werden. Dabei kann der Beginn, die Dauer und die Anzahl der Wiederholungen mit Hilfe der Attribute `beginn`, `dur` und `repeatcount` festgelegt werden [Vgl. W3s03ab]. Als `media`-Elemente können u.a. `video`, `audio`, `animation` und `text` verwendet werden, welche wiederum durch Attribute spezifiziert werden [Vgl. W3s03ad],.

6.3.2 Separate SMIL-Datei

Eine separate SMIL-Datei (.smil) wird über das `smil`-Element definiert. Innerhalb des `body`-Elements wird anschließend unter Verwendung der SMIL-Elemente der Inhalt und Ablauf der Präsentation definiert [Vgl. W3s03aa].

Code 42 zeigt eine solche separate SMIL-Datei.

```
<smil>     -- root element of a smil file
<body>
  <seq repeatCount="indefinite">
    <img src="image1.jpg" dur="3s" />
    <img src="image2.jpg" dur="5s" />
  </seq>
</body>
</smil>
```

Code 42 : separate SMIL-Datei[267]

Um eine solche SMIL-Datei (.smil) präsentieren zu können muss allerdings ein SMIL-Player auf dem System installiert sein.

---

[267] Nach [W3s03aa] .

6.3.3 SMIL in einem HTML-Dokument[268]

Seit dem Internet Explorer 5.5 ist es Dank einer HTML-Erweiterung möglich SMIL-Elemente in einem HTML-Dokument zu verwenden. Diese Erweiterung nennt sich ***TIME*** und steht für ***T**imed **I**nteractive **M**ultimedia **E**xtensions*.

Insofern kann jede SMIL-Präsentation als HTML-Dokument über das Internet präsentiert werden oder am lokalen PC ablaufen. Hierzu bedarf es allerdings einiger Modifikationen des HTML-Dokuments.

Damit der Internet Explorer die SMIL-Elemente erkennt, muss der „time“ Namensraum[269] im einleitenden `<html>` Tag definiert werden .

Syntax : `<html` **`xmlns:y="urn:schemas-microsoft-com:time"`** `>`

Anstatt y kann auch irgend ein anderer Buchstabe oder ein Wort verwendet werden. Dieser frei wählbare Buchstabe (oder auch Wort) muss allerdings auch für die später definierte Klasse verwendet werden.

Um diesen „***time***“ Namensraum zu importieren, muss im Kopfteil des HTML-Dokuments das **`<?import namespace=“y“ implementation=“#default #time2“>`** Element hinzugefügt werden. Anschließend wird die Klasse “y“ durch **`<style> .y {behavior:url (#default #time2)} </style>`** definiert.

Dies ist nötig um die verschiedenen SMIL-Elemente in ein HTML-Dokument verwenden zu können. Hierzu müssen die zu verwendenden SMIL-Elementen dieser Klasse zugeordnet werden.

```
<html xmlns:y="urn:schemas-microsoft-com:time" >
  <head>
    <?import namespace= “y“ implementation=“#default#time2“>
    <style> .y {behavior: url(#default#time2)} </style>
  </head>
  <body>
-- use of the sequence-element for making a presentation
```

[268] Basiernd auf [W3s03ae] .
[269] S.a. 5.2.4 .

```
<y:seq repeatCount="1">
 <img  src="[Path.file]" class="y" width="200"  height="150"
   dur="4s"/>
 <h1 class="y" dur="4s">Enjoy the movie</h1>
 <y:video src="[Path.file]" class= "y" width="200" height="150"
  />
 <h1 class="y" dur="5s">I hope you enjoyed the movie - Have a
    nice day</h1>
</y:seq>
</body>
</html>
```

Code 43 : Verwenden von SMIL in einem HTML-Dokument

In Code 43 wird das `seq`-Element dazu verwendet um eine Präsentation im Browser des Anwenders zu erstellen. Dabei wird zuerst das mit dem `src`-Attribut referenzierte Bild vier Sekunden im Browser angezeigt. Vier Sekunden später wird dieses Bild durch den Text “Enjoy the movie”, der ebenfalls vier Sekunden am Bildschirm angezeigt, ersetzt. Daran anschließend startet der Windows Media Player in einer Größe von 150 x 200 Pixel den mit `src="[Path.file]"` referenzierten Film. Ist der Film zu Ende, wird der Text „I hope you enjoyed the movie“ fünf Sekunden lang im Bildschirm angezeigt und die Präsentation ist beendet.

6.4 SVG[270]

SVG steht für **S**calable **V**ector **G**raphics und wird zum Beschreiben von zweidimensionalen Grafiken und grafischen Applikationen in XML verwendet.
An der Entwicklung von SVG waren unter anderem Firmen wie Sun Microsystems, IBM, Adobe, Apple und Kodak beteiligt.

Version 1.1[271] wurde am 14. Januar 2003 als W3C Empfehlung veröffentlicht.
Seit dem 15. Juli 2003 liegt Version 1.2[272] als vorläufiger Working Draft zum Nachlesen zur Verfügung.

---

[270] Basierend auf [W3s03b] .
[271] http://www.w3.org/TR/2003/REC-SVG11-20030114/, Abruf am 2003-11-03.
[272] .http://www.w3.org/TR/2003/WD-SVG12-20030715/, Abruf am 2003-11-03.

Um SVG-Grafiken in einem (neueren) Browser betrachten zu können wird allerdings ein Plug-In benötigt. Ein solches Plug-In ist beispielsweise der Adobe SVG-Viewer[273], welcher kostenlos aus dem Internet herunter geladen werden kann.

Weitere SVG-Tools findet man unter http://www.w3.org/Graphics/SVG/SVG-Implementations.htm8#viewer .

6.4.1 Vorteile von SVG

SVG bietet mehrere Vorteile gegenüber anderen Grafikformaten wie GIF oder JPEG [Vgl. W3s03ba].

1. SVG Grafiken verlieren nicht an Qualität wenn sie vergrößert werden, und können somit auch bei jeder Auflösung mit einer guten Qualität im Bildschirm angezeigt oder ausgedruckt werden.
2. jedes Element und jedes Attribut kann animiert werden.
3. SVG Grafiken sind kleiner und komprimierbarer.
4. SVG Grafiken sind ein offener Standard und ergänzen sich zu anderen W3C-Standards wie DOM oder XSL.
5. SVG Dateien können mit vielen verschiedenen und kostenlosen Tools wie beispielsweise Notepad betrachtet als auch verändert werden.
6. Texte in einer SVG Grafik können ausgewählt werden.
7. Nach Texten in einer SVG Grafik kann gesucht werden.

6.4.2 Separates SVG-Dokument

Code 44 zeigt das grundsätzliche Aussehen eines SVG-Dokuments (**.**svg).

```
<?xml version="1.0" standalone="no"?>     -- XML Declaration
<!DOCTYPE ......!>                        -- DTD
<svg  width="..."  height="..." >         -- SVG
....
</svg>
```

Code 44 : Separates SVG-Dokument[274]

Im einleitenden <svg> Tag kann mit den Attributen width und height die Größe der Grafik angegeben werden [Vgl. W3s03bb].

Innerhalb des einleitenden und abschließenden <svg> Tag werden die verschiedenen SVG-Elemente wie Rechtecke, Kreise, Ellipsen, Linien, Texte usw. als auch so genannte SVG-Filter, mit welchen diese Formen und Texte animiert werden können, angegeben [Vgl. W3s03bc]. SVG-Filter müssen innerhalb des defs-Elements mittels des filters-Elements definiert und mit dem id-Attribut eindeutig referenziert werden. In jedem SVG-Element können dabei mehrere Filter verwendet werden [Vgl. W3s03bd].

6.4.3 SVG in einer Web-Seite

SVG-Grafiken können mittels eines embed- oder object-Elements[275] in ein HTML/XHTML-Dokument eingebettet werden [Vgl. W3s03bb].

6.4.4 Beispiel

Code 45 zeigt ein HTML-Dokument, in welche zwei SVG-Dateien eingebettet sind. Eine Grafik enthält zwei Links und die andere Grafik ist eine Animation.

```
<html>
<head>
</head>
<body bgcolor="lightyellow" margin-left="200pt">
<h1 align="center">This is a demonstration of using SVG in a
HTML file</h1> <br><br>
 --embeds the svg file animation into the html file
<embed style="margin-left:550pt" width="90%" height="20%"
   src="animation.svg">
<p align="center">You can use the links below by pressing the
graphic symbol.</p>
  --embeds the svg file link into the html file
<embed  style="margin-left:200px" width="30%" height="20%"
   src="links.svg">
</body></html>
```

Code 45 : SVG in einem HTML-Dokument

[273] http://www.adobe.com/svg/viewer/install/ , Abruf am 2003-11-04 .
[274] Nach [W3s03bb] .
[275] S.a. 3.5.3 .

Abbildung 51 zeigt die Darstellung von Code 45 im MS Internet Explorer.

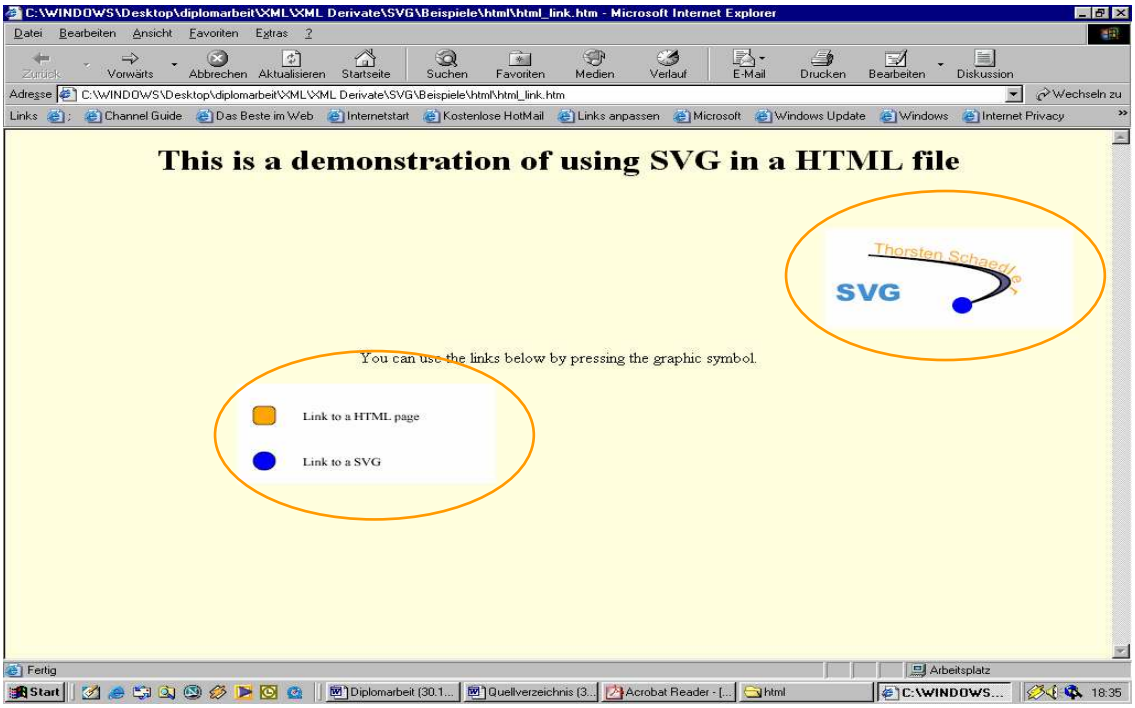

Abbildung 51 : SVG in einem HTML-Dokument

Code 46 zeigt den Quelltext der in Code 45 eingebetteten Animation.

```
<?xml version="1.0" standalone="no"?>
<svg>   --root element of the svg file
<defs>  --includes some SVG Filters used for animations
<animateMotion  xlink:href="#circle" dur="6s" path="M0 0 Q203
16 85 55" fill="freeze"/>
<animateColor  xlink:href="#circle" dur="6s" from="orange"
to="blue" attributeName="fill" fill="freeze"/>
<animate  xlink:href="#circle" dur="6s" from="5" to="2" attrib-
uteName="r" />
</defs>
 -- definition of the circle- and sickle object and text
<g transform="translate(-90,-50)">
<text x="100" y="130" style="font-family:Arial Black;font-
size:26;stroke:none;fill:rgb(70,140,215)">SVG</text>
<path id="sickle"  d="M130 80 Q333 96 215 135 Q315 96 130 80"
style="stroke:rgb(00,00,00);strokewidth:2;fill:rgb(90,90,139)"/
>
<text style="alignment-baseline:text-bottom;baseline-shift:2;
font-family:Arial;font-size:16;fill:orange">
<textPath xlink:href="#sickle" startOffset="4">Thorsten Schaed-
ler</textPath>
</text>
<circle id="circle" cx="130" cy="80" r="9" style="fill:red;
 stroke:none" />
```

```
</g>
</svg>
```

Code 46 : animation.svg[276]

Code 47 zeigt den Quelltext der in Code 45 eingebetteten SVG Grafik „Links„.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="320px" height="170px"> --root element
 -- definition of the link “Link to a html page”
  <a xlink:href=”W3C.html"
     xlink:title="HTML">
    <rect x="15" y="25" ry="5" width="20" height="20"
    style="fill:orange; stroke:black;" />
  </a>
 -- definition of the link “Link to a SVG”
  <a xlink:href="filter.svg"
     xlink:title="SVG">
    <rect x="15" y="75" ry="15" width="20" height="20"
    style="fill:blue; stroke:black;" />
  </a>
 --text of the both links
<text x="60" y="40">Link to a HTML page</text>
<text x="60" y="90">Link to a SVG</text>
</svg>
```

Code 47 : links.svg[277]

Abschließend zeigt Abbildung 52 noch die Verwendung des Adobe SVG-Viewers 3.0.1 innerhalb des HTML-Dokuments. Um Zugriff auf den SVG-Viewer zu erlangen, muss hierfür innerhalb eines eingebetteten SVG-Dokuments die rechte Maustaste gedrückt werden.

[276] Basiernd auf [HeSp03a] .
[277] Basierend auf [HeSp03b] .

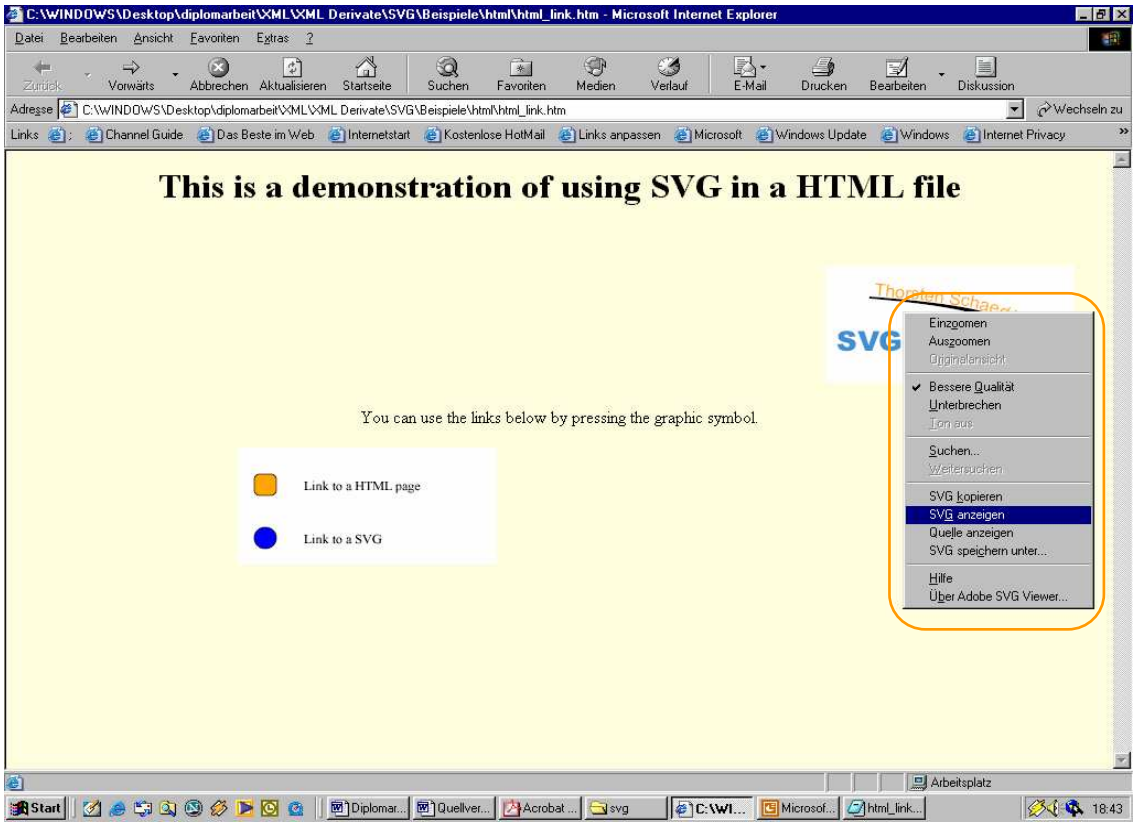

Abbildung 52 : Adobe SVG-Viewer

6.5 MathML

Mit der **Math**ematical **M**arkup **L**anguage (MathML) steht ein weiteres XML Derivat zur Verfügung, mit welchem es möglich ist mathematischen Text zu erstellen und in Webseiten einzubinden.

Mit MathML besteht somit die Möglichkeit mathematische Notationen in gleicher weise wie normale Texte über das WWW zu veröffentlichen.

Seit dem 21. Februar 2001 ist MathML eine W3C Empfehlung. Version 2.0[278] wurde am 23. Oktober 2003 veröffentlicht .

[278] http://www.w3.org/ TR/MathML2/, Abruf am 2003-11-28.

7. XML Web-Services

Unter XML Web-Services versteht man weltweit verteilte unabhängige Internet Anwendungen, welche klar definierte Aufgaben zu erfüllen haben. XML Web-Services unterscheiden sich dabei von bisherigen Diensten (Anwendungen) dadurch, dass sie auf automatische Benutzung und nicht auf die Benutzung durch Menschen ausgerichtet sind. Sie arbeiten als lose gekoppelte Systeme über das Internet zusammen und ermöglichen so neue Wege der Zusammenarbeit im Internet. Dieser Ansatz verwendet dabei die Internetstandards SOAP[279], WSDL[280], UDDI[281], XML[282] und HTTP („offener" Port 80). Einzelne Services können so verteilt über das Internet an den verschiedensten Standorten eines Unternehmens als auch bei (un-)bekannten Zulieferern, Kunden oder Partnern ablaufen [Vgl. nwc03].

Bei verteilten Anwendungen wurde bisher die elektronische Kommunikation zwischen Client und Server meistens per CORBA, RMI oder DCOM realisiert.
CORBA verwendet dabei ein eigenes Protokoll, welches einen eigenen Port benötigt. Allerdings werden aus Sicherheitsgründen viele Ports durch Firewalls gesperrt, wodurch einige Kommunikationen nicht möglich sind. Bei RMI müssen sowohl Server- als auch Client-Anwendungen in Java implementiert sein [Vgl. nwc03].

Durch den automatisierten Austausch von verschiedensten Web Services entstehen in der Kombination komplexe Web Anwendungen. Ein Web Service kann dabei eine bestehende Anwendung kapseln, sich aus bestehenden Web Service Anwendungen zusammensetzen oder es werden neue Anwendungen direkt als Web Service konzipiert [vgl. nwc03].

Um eine Web Service Anwendung zu erstellen, müssen zuerst die unternehmenseigenen Web Services gebildet werden, anschließend über UDDI nach

[279] S.a. 7.2 .
[280] S.a. 7.3 .
[281] S.a. 7.4.
[282] S.a. 5 .

Web Services anderer Unternehmen gesucht werden und zuletzt werden die fremden als auch eigenen Web Services zu einer Web Service Anwendung zusammengesetzt [Vgl. nwc03].

Dabei muss weiterhin auf die Ausfallsicherheit, Zugriffssicherheit auf Anwendungen und Daten sowie die Performance geachtet werden. Dabei ist zu beachten, dass in entsprechende Vereinbarungen die Pflichten eines Anbieters von Web Services geklärt sind [Vgl. nwc03].

Im Rahmen dieser Arbeit werden nun die drei Internetstandards SOAP, WSDL un UDDI vorgestellt und gezeigt wie diese mittels XML definiert werden.

7.1 Schichtenmodell der Web Services

Bevor die einzelnen Web Services Komponenten (Internetstandards) näher betrachtet werden, soll das Schichtenmodell aus Abbildung 53 den Zusammenhang dieser Komponenten verdeutlichen.

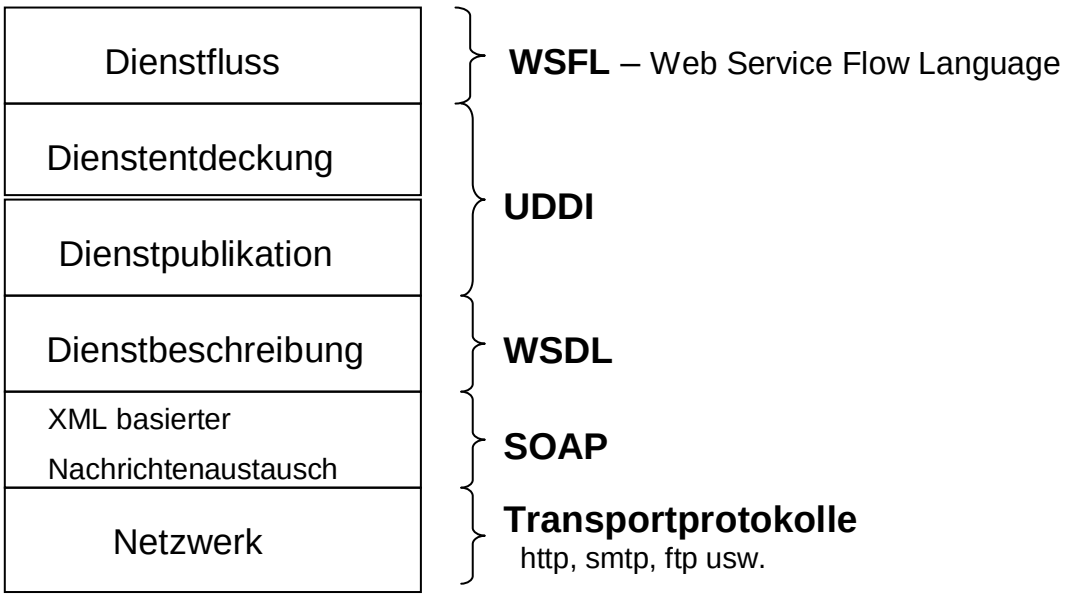

Abbildung 53 : WS-Schichtenmodell[283]

[283] Nach [ReCH03, S. 2] .

7.2 SOAP

Die Entwicklung des **S**imple **O**bject **A**ccess **P**rotcol wurde 1998 von Microsoft und anderen Firmen begonnen. Eine erste Version wurde 1999 von Microsoft über die Internet Engineering Task Force (IETF) veröffentlicht, woraufhin sich Firmen wie SAP und IBM dieser Entwicklungsgruppe anschlossen.

Am 8. Mai 2000 wurde das SOAP (Version 1.1)[284] zu einer W3C Note . Die am 24. Juni 2003 veröffentlichte und aus drei Teilen (Primer[285], Messaging Framework[286], Adjuncts[287]) bestehende Version 1.2 ist dagegen eine W3C Empfehlung und beinhaltet einige Änderungen, welche unter http://www.w3.org/TR/2003/REC-soap12-part0-20030624/#L4697 detailliert aufgeführt werden.

Dieses Kapitel soll lediglich einen Einblick in SOAP Version 1.2 geben und dessen wichtigste Punkte ansprechen.

SOAP ist ein leichtgewichtiges XML basiertes Protokoll für den Austausch von Informationen in einer dezentral verteilten Umgebung [Vgl. W3C03c].

„SOAP is a simple XML based protocol to let applications exchange information over http" [W3s03e].

7.2.1 SOAP und HTTP

Da SOAP ein Kommunikationsprotokoll ist, beschreibt es lediglich wie die Inhalte und XML-Daten übertragen werden sollen. Client und Server regeln dabei mit SOAP einen entfernten Methodenaufruf (RPC), welcher Parameter und Ergebnisse in XML kodiert und überträgt [Vgl. FrUl03, S.4].

Die Übertragung wird durch Transportprotokolle wie HTTP, FTP, SMTP oder TCP geregelt, wobei bevorzugt HTTP eingesetzt wird [Vgl. FrUl03, S.4].

---

[284] http://www.w3.org/TR/2000/NOTE-SOAP-20000508, Abruf am 2003-12-03 .
[285] http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, Abruf am 2003-12-03 .
[286] http://www.w3.org/TR/2003/REC-soap12-part1-20030624/, Abruf am 2003-12-03.
[287] http://www.w3.org/TR/2003/REC-soap12-part2-20030624/, Abruf am 2003-12-03.

Der Einsatz von HTTP bringt dabei einige Vorteile mit sich. Zum Einen wird es von nahezu jedem Rechner der an das Internet angebunden ist verwendet. Mittels HTTPS kann eine einigermaßen sichere Verbindung aufgebaut werden und SOAP selbst muss sich um das Thema Sicherheit nur wenig kümmern. Um die Sicherheit zu erhöhen kann SOAP eine Erweiterung in Form von WS-Security verwenden. Des Weiteren verwendet http den Port 80, welcher normalerweise nicht durch Firewalls gesperrt ist [Vgl. nwc03].

Da SOAP auf XML basiert und somit reinen Text darstellt, ist es unabhängig von Betriebssystemen, Programmiersprachen und Objektmodellen und kann somit verschiede Plattformen wie .NET und Java verbinden [Vgl. nwc03].

7.2.2 Struktur[288]

Eine SOAP Nachricht besteht aus einem vorgeschriebenen `SOAP-Envelope`, einem optionalen `SOAP-Header` und einem zwingend erforderlichen `SOAP-Body`, welcher die Nachricht enthält [Vgl. W3C03c].

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
envelope" env:encodingStyle="http://www.w3.org/2003/05/soap-
encoding " >
       <env:Header>
            <x:[Header-Block Element 1] xmlns:x="[URI]" ……>
                  ..............
            </x:[Header-Block Element 1]>
            <y:[Header-Block Element 2] xmlns:y="[URI]" …… >
                 ..............
           </y:[Header-Block Element 2]>
       </env:Header>
       <env:Body>
          <env:Fault>
            <!--Two or more child element information items -->
          </env:Fault>
              <x:[Body child-element 1] xmlns:x="[URI]" ……>
                    ..............
              </x:[Body child-element 1]>
              <y:[Body child-element 2] xmlns:y="[URI]" …… >
              </y:[Body child-element 2]>
      </env:Body>
</env:Envelope>
```

Code 48 : Struktur einer SOAP-Nachricht (Version 1.2)

[288] Basierend auf [W3C03c] und [W3C03d] .

Abbildung 54 zeigt die bildhafte Darstellung einer SOAP-Nachricht.

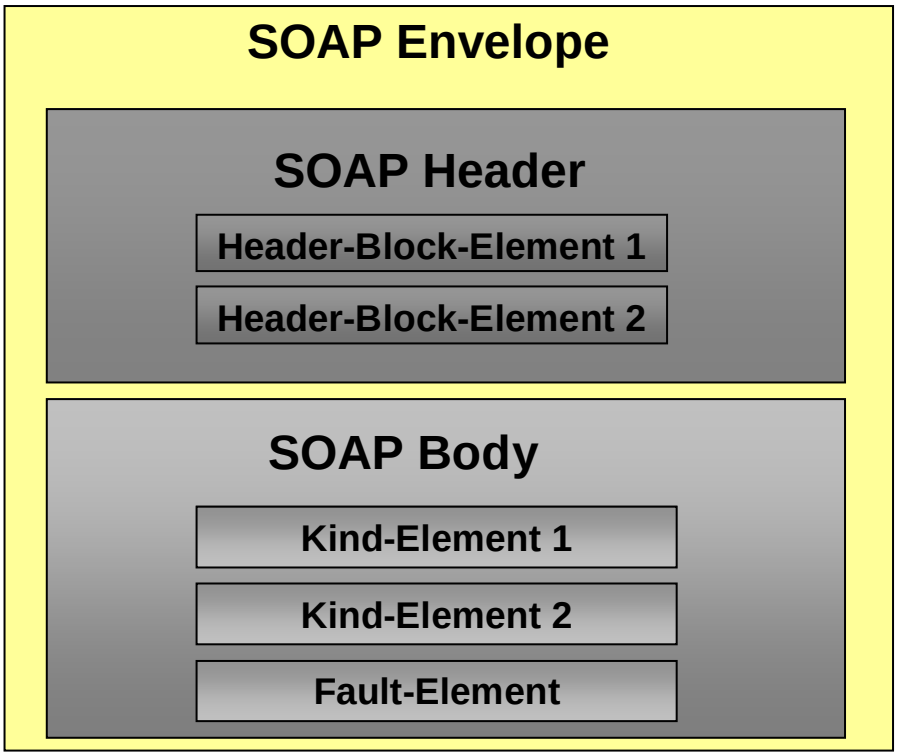

Abbildung 54 : Bildhafte Darstellung einer SOAP-Nachricht[289]

7.2.2.1 Das Envelope-Element

Das `Envelope`-Element ist das oberste Element einer SOAP-Nachricht und entspricht dem Wurzelelement eines XML-Dokuments[290]. Somit legt es den Start als auch das Ende einer SOAP Nachricht fest. Es enthält ein optionales `Header`-Element und ein zwingend erforderliches `Body`-Element.

Die im `Envelope`-Element angegebenen Namensräume[291] identifizieren die `Envelope Version` und den `Encoding-Style` einer SOAP-Nachricht. Um den `Encoding-Style` einer SOAP-Nachricht anzugeben wird das `encodingStyle`-Attribut verwendet. Diesem Attribut können die Werte (Namensräume) `http://www.w3.org/2003/05/soap-encoding`, `http://example.org/encoding` oder `http://www.w3.org/2003/05/soap-envelope/encoding/none` zugeordnet werden.

[289] Nach [W3C03d] .
[290] S.a 5 .
[291] S.a. 5.3.3 .

7.2.2.2 Das Header-Element

Das optionale `Header`-Element ist ein Erweiterungsmechanismus, mit welchem einer SOAP-Nachricht Merkmale angehängt werden können.

Die direkten Kinderelemente des `Header`-Elements werden „*Header-Blocks*“ genannt. Ein solcher „Header-Block“ enthält dabei Informationen die zum verarbeiten einer Nachricht relevant sind. So kann beispielsweise ein Routing Block erstellt werden, welcher den Weg der SOAP Nachricht festlegt.

Das `Header`-Element kann das `encodingStyle`-Attribut, das `mustUnderstand`-Attribut, das `role`-Attribut und das `relay`-Attribut verwenden.

7.2.2.3 Das Body-Element

Das `Body`-Element enthält die Informationen, welche an den Empfänger gesendet werden sollen. Es ist genau wie das `Header`-Element ein Kinderknoten des `Envelope`-Elements.

Das `Body`-Element muss einen Namensraum aus `http://www.w3.org/2003/05/soap-envelope` beinhalten und kann daneben auch noch das `encoding Style`-Attribute verwenden.

7.2.2.4 Das Fault-Element

Das `Body`-Element beinhaltet das optionale `Fault`-Element, welches zur Fehlerbeschreibung innerhalb einer SOAP-Nachricht verwendet wird. Diese Fehlerbeschreibung wird mit Hilfe von bis zu fünf verschiedenen Kinderelementen vorgenommen. Diese Kinderelemente sind das vorgeschriebene `Code`- und `Reason`-Element als auch die optionalen `Node`- , `Role`- und `Detail`-Elemente.

7.3 WSDL

WSDL steht für **W**eb **S**ervivces **D**ecription **L**anguage und wurde im Jahr 2001 von den Firmen Ariba, IBM und Microsoft entwickelt.

WSDL ist ein XML-Format zum Beschreiben von Netzwerkservices als eine Menge von so genannten Endpoints, die mit Nachrichten (Messages) arbeiten, welche dokument-orientierte oder prozedur-orientierte Informationen enthalten. Operationen und Nachrichten werden dabei zuerst abstrakt beschrieben und anschließend an ein konkretes Netzwerkprotokoll und Nachrichtenformat gebunden [Vgl. W3C03e].

7.3.1 Entwicklung

Das W3C veröffentlichte am 15.03.2001 WSDL 1.1[292] als W3C Note. Am 9 Juli 2002 wurde Version 1.2 in einer ersten Fassung als ein W3C Working Draft veröffentlicht. Diese Fassung wurde mehrfach überarbeitet. So folgten am 14. Januar 2003 als auch am 3. März 2003 überarbeitete Versionen. Seit dem 11. November 2003 ist es als W3C Working Draft in Version 2.0[293] zu begutachten.

7.3.2 Struktur [294]

Ein WSDL Dokument ist eine Menge von Definitionen. Das `definitions`-Element bildet dabei das Wurzelelement und definiert die XML-Namensräume[295], welche innerhalb des WSDL-Dokuments gebraucht werden.

Es enthält sechs verschiedene Kinderelemente, welche die Services definieren. Diese Kinderelemente sind das `types`-, `message`-, `portType`-, `binding`-, `port`- und `service`--Element.

Code 49 zeigt die Grundstruktur eines WSDL-Dokuments.

```
<definitions>
     <types>...</types>
     <message>...</message>
     <portType>...</portType>
     <binding>...</binding>
     <service>
       <port>...</port>
     </service>
```

[292] http://www.w3.org/TR/wsdl/, Abruf am 2003-12-01 .
[293] http://www.w3.org/TR/wsdl20/, Abruf am 2003-12-04 .
[294] Basierend auf [W3C03e] und [W3C03f] .
[295] S.a. 5.3.3 .

```
</definitions>
```

Code 49 : Grundstruktur eines WSDL-Dokuments

7.3.2.1 Das Types-Element

Das `types`-Element beinhaltet bzw. definiert die Datentypen, welche für den Nachrichtenaustausch zwischen Service und dem Benutzer benötigt bzw. verwendet werden. Um die maximale Interoperabilität und Plattformunabhängigkeit zu erreichen werden hierfür XML Schemas[296] verwendet.

Das `types`-Element wird allerdings nur benötigt, falls komplexe Datentypen verwendet werden, da einfache Datentypen wie `string`, `float`, `double` etc. bereits von XML Schema[297] zur Verfügung gestellt werden.

7.3.2.2 Das Message-Element

Das `message`-Element besteht aus einem oder mehreren so genannten *Parts* (`<part.../>`), welche zum Beschreiben des logisch abstrakten Inhalts einer Nachricht verwendet werden.

7.3.2.3 Das PortType-Element

Das `portType`-Element definiert eine Menge von Operationen. Es erhält mit dem `name`-Attribut einen einmaligen Namen und enthält ein oder mehrere `operation`-Elemente. Jedes `operation`-Element enthält eine Kombination aus `input`- und `output`-Elementen.

Es können lediglich vier Operationen beschrieben werden.

- *One-Way* : Eine Nachricht vom Benutzer zum Service, wobei das `<input>` Element das abstrakte Nachrichtenformat spezifiziert.
- *Request-Response* : Eine Nachricht vom Benutzer zum Service (`request`) und eine vom Service zum Benutzer (response).

[296] S.a. 5.4 .
[297] S.a. 5.4 .

Das `input-` und `output`-Element spezifizieren das abstrakte Nachrichtenformat für die Anfrage und die Antwort. Das `fault`-Element spezifiziert die abstrakte Nachrichtenformat für beliebige Fehlermeldungen.

- *Solicit-Response :* Im Unterschied zur Request-Response Operation beginnt der Service die Kommunikation.
- *Notification :* Eine Nachricht vom Service zum Benutzer, wobei das `output`-Element das abstrakte Nachrichtenformat spezifiziert.

7.3.2.4 Das Binding-Element

Das `binding`-Element definiert für jede Operation eines `PortTypes` das Nachrichtenformat und das verwendete Protokoll. Im Allgemeinen wird SOAP[298] als Protokoll verwendet. Es ist allerdings auch möglich HTTP GET/POST oder auch MIME als Protoll zu verwenden.

7.3.2.5 Das Port-Element

Das `port`-Element definiert einen einzelnen Endpunkt indem es einem `binding`-Attribut eine Adresse zuordnet.

7.3.2.6 Das Service-Element

Das `service`-Element fasst eine Gruppe zusammengehöriger Ports zusammen.

Beim Übergang zu WSDL 1.2 bzw. 2.0 gab es diesbezüglich einige kleinere Änderungen. So unterstützte Version 1.1 eine Überladung an Operationen, was in Version 1.2 beseitigt wurde. Ferner wurden `PortTypes` in `Interfaces` und `Ports` in `Endpoints` umbenannt [Vgl. W3C03g].

[298] S.a. 7.2 .

7.4 UDDI

UDDI entstand aus einer Zusammenarbeit von IBM, Microsoft und Ariba. Heute sind mehr als 300 Firmen Mitglied dieser Community. Mit UDDI steht ein plattformunabhängiger und offener Rahmen (**U**niversal) zur Verfügung, welcher es ermöglicht einen mittels WSDL definierten Web Service zu beschreiben (**D**escription), Unternehmen zu finden (**D**iscovery) und die Services zusammenzuschließen (**I**ntegration).

Im Folgenden soll nun ein kleiner Einblick in die Funktionsweise von UDDI gegeben werden.

7.4.1 UDDI Datenbanken

UDDI unterscheidet zwei unterschiedliche Arten von Datenbanken. Zum Einen eine zentrale UDDI Datenbank und zum Zweiten private UDDI Datenbanken.

In einer solchen Datenbank bzw. Registry werden dabei drei verschiedene Informationstypen abgelegt [Vgl. FiBrT03, S.20f].

- *White Pages* ermöglichen einem Service-Anbieter Informationen über sich und die entsprechende Kontaktperson einem Service-Nutzer bereitzustellen.

- *Yellow Pages* sind sozusagen das Branchenverzeichnis in welchem eine Zuordnung des Unternehmens zu bestimmten Industriebereichen erfolgt.

- *Green Pages* enthalten die technischen Informationen des Web Service. Eine technische Information ist beispielsweise ein WSDL Dokument.

7.4.1.1 UDDI Business Regisrty

Um eine UDDI Anfrage weltweit zur Verfügung zu stellen, gibt es eine zentrale UDDI Business Registry (UBR) welche physisch bei Microsoft, IBM und Ariba vorhanden ist und in der täglich sämtliche Änderungen untereinander repliziert

werden. Somit sind bei allen drei Anbietern dieser UBR alle aktuellen Daten vorhanden [Vgl. WalsP03, S.4].

Für den Zugriff auf diese Datenbank stellen diese Anbieter ein Web Interface zur Verfügung, mit welchem die Suche, Einträge und alles andere von jedermann vorgenommen werden kann [Vgl. WalsP03, S.4] .

7.4.1.2 Private UDDI Datenbanken

Neben der UBR gibt es auch noch die Möglichkeit private UDDI Datenbanken aufzusetzen. UDDI.org gibt hierfür fünf verschiedene Szenarien an, auf welche im Rahmen dieser Arbeit jedoch nicht näher eingegangen wird [Vgl. WalsP03, S.5].

- E-marketplace UDDI
- Portal UDDI
- Partner Katalog UDDI
- Internet Enterprise Application Integration UDDI
- Test UDDI

7.4.2 Das UDDI Grundmodell

Bevor in Kapitel 7.4.3 auf die einzelnen Elemente näher eingegangen wird, soll Abbildung 55 den Zusammenhang dieser Elemente veranschaulichen.

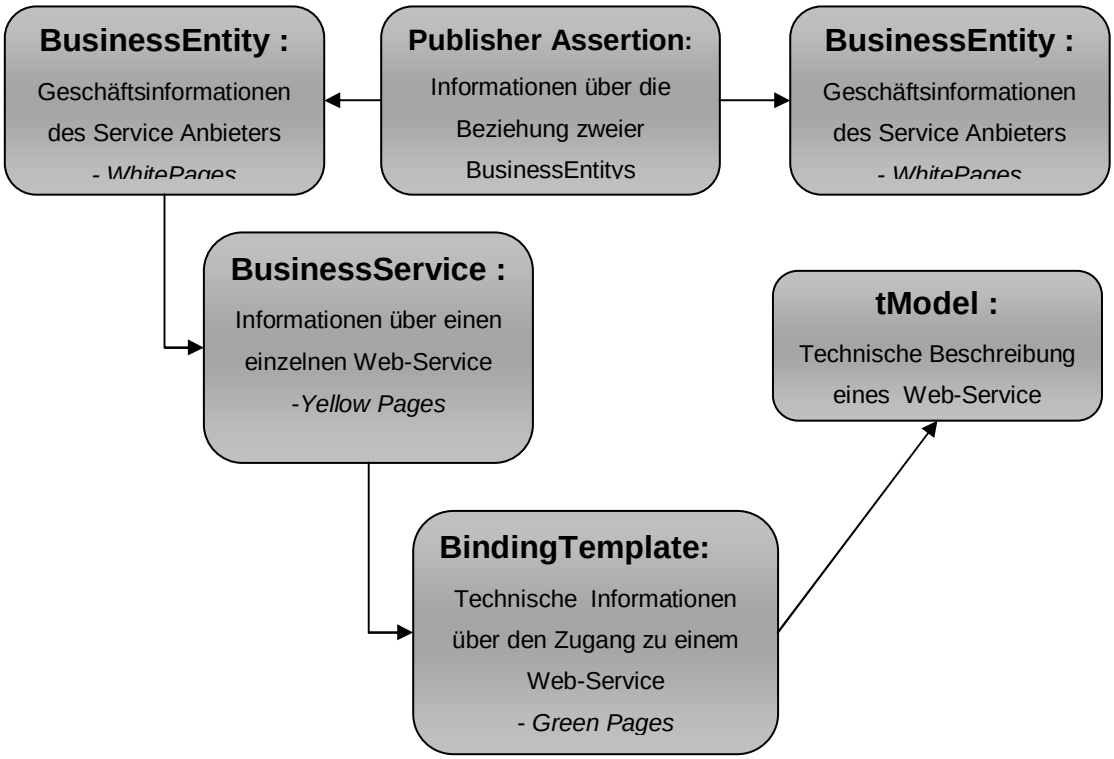

Abbildung 55 : Das UDDI Grundmodell[299]

7.4.3 Datenstruktur – XML Schema[300]

Mit dem `BusinessEntitiy` und dem `tModel` gibt es in einer UDDI Datenbank zwei zentrale Datenstrukturen die für die komplette Datenhaltung verantwortlich sind [Vgl. WalsP03, S.7]. Darüber hinaus gibt es noch ein `publisherAssertion`-Element mit welchem Beziehungen zwischen verschiedenen `BusinessEntitys` beschrieben werden.

7.4.3.1 Das BusinessEntitiy

Das `BusinessEntitiy` ist ein XML-(Wurzel-)Element bzw. Datensatz in welchem sämtliche relevanten Daten eines Web-Service und eines Unternehmens enthalten sind [Vgl. WalsP03, S.9]. Ein `BusinessEntity`-Element besteht im Allgemeinen aus mehreren Elementen und hat die in Code 50 dargestellte allgemeine Struktur [Vgl. WalsP03, S.9].

---

[299] Nach [OaUD03, S.30] .
[300] Basiert neben den sonstigen Quellangaben auf [OaUD03, S.32-49] .

```
<BusinessEntitiy businessKey="…" operator="…" authorized
   Name="…" >
      <Name>...<Name>
      <discoveryURLs>
             <discoveryURL useType="…">...</diccoveryURL>
       </discoveryURLs>
      <description>...</description>
      <contacts>
              <contact>...</contact>
               ....
      </contacts>
      <BusinessServices>
              <BusinessService>...</BusinessService>
                ..................
      </BusinessServices>
      <identifierBag>
               <keyedReference tModelKey="…" keyname="…" key
                   Value="…" />
      </identifierBag>
      <categoryBag>
               <keyedReference tModelKey="…" keyname="…" key
                   Value="…" />
      </categoryBag>
</Business Entitiy>
```

Code 50 : Allgemeine Struktur eines `BusinessEntity`-Elements

7.4.3.2 Das BusinessService-Element

Wie in Code 50 dargestellt, ist ein `BusinessService`-Element Bestandteil eines `BusinessEntity`-Elements. Es hält wiederum mehre Elemente welche Daten enthalten, mit welchen ein Web-Service näher beschrieben wird.

Code 51 zeigt die allgemeine Struktur eines solchen `BusinessService`-Elements [Vgl. OaUd03, S.39ff], [Vgl. WalsP03, S.10].

```
<BusinessService serviceKey="…" businessKey="…">
       <name>...</name>
       <description>...</description>
       <BindingTemplates>
          <BindingTemplate>...<BindingTemplate>
          …
       </BindingTemplates>
       <categoryBag>...</categoryBag>
</BusinessService>
```

Code 51 : Allgemeine Struktur eines `BusinessService`-Elements

7.4.3.3 Das BindingTemplate-Element

Das `BindingTemplate`-Element ist die Komponente welche die genauen Angaben enthält wie der Web-Service angesprochen werden kann. `BindingTemplate`-Elemente speichern dabei wie ein Web-Service aufgerufen wird und enthalten Referenzen auf so genannte `tModele`.

In Code 52 ist die allgemeine Struktur eines solchen `BindingTemplate`-Elements dargestellt [Vgl. OaUD03, S.42].

```
<BindingTemplate bindingKey="..." serviceKey="...">
        <description>...</description>
        <accessPoint>.....</accessPoint>*
        <hostingRedirector>.....</hostingRedirector>*
        <tModelInstanceDetails>….</tModelInstanceDetails>
        <categoryBag >….</categoryBag>
</BindingTemplate>
```

Code 52 : Allgemeine Struktur eines `BindingTemplate`-Elements

* In einem `BindingTemplate`-Element muss entweder das `accessPoint`-Element oder das `hostingRedirector`-Element vorhanden sein.

7.4.3.4 Das tModel

Ein `tModel` ist eine technische Spezifikation innerhalb der UDDI Registry, wie beispielsweise ein WSDL-Dokument in welchem der Service technisch beschrieben ist [Vgl. FiBrT03, S.21].

Ein `tModel` besteht im Prinzip lediglich aus einem Namen, einem Schlüssel (`tModelkey`), einer optionalen Beschreibung und einer URL. Es wird in allen Bereichen wie Kategorisierung, Typenspezifikation, Informationsspezifikation usw. gebraucht und taucht in beinahe jedem UDDI Element auf [Vgl. WalsP03, S.7].

UDDI verwendet das `tModel`, um Metadaten über einen Web Service auszutauschen.

Code 53 stellt die allgemeine Struktur eines `tModels` dar [Vgl. WalsP03, S.7], [Vgl. OaUd03, S.48].

```
<tModel  tModelKey="..." operator="..." authorizedName="..."
    <name>....</name>
    <description>...</description>
    <overviewDoc>
         <description>...</description>
         <overviewURL>...</overviewURL>
    </overviewDoc>
    <identifierBag>
          <keyedReference/>
           ...
    </identifierBag>
    <catergoryBag>
          <keyedReference/>
           ...
    </categoryBag>
</tModel>
```

Code 53 : Allgemeine Struktur eines `tModel`

7.4.3.5 Das categoryBag-Element

Um angesichts eines ständigen Wachstums und ständigen Erweiterungen und Entwicklungen neuer Web-Services eine effektive Suche zu ermöglichen, wurden in der ersten Version von UDDI drei Methoden zur Kategorisierung anhand von einzelnen `tModels` implementiert [Vgl. WalsP03, S.7f].

- *North America Industry Classification System (NAICS)*
  Klassifizierung von Industrie und Unternehmen anhand ihrer Branche

- *Universal Standard Products and Services Classification (UNSPSC)*
  Klassifizierung von Industrie und Unternehmen anhand der von ihnen angebotenen Produkte und Dienstleistungen.

- *ISO 3166*
  Klassifizierung von Industrie und Unternehmen anhand ihrer geographischen Lage.

Eine solche Kategorisierung wird mittels des `categoryBag`-Elements vorgenommen, in welchem Referenzen auf das entsprechende `tModel` abgelegt

werden. Dies wird über das `keyedReference`-Element und dessen Attribute `keyName`, `keyValue` und `tModelKey` erreicht, wobei der Wert des `tModelKey` vom UDDI Operator zugewiesen wird und auf ein existierendes `tModel`-Element verweisen muss [Vgl. WalsP03, S.8].

Syntax : `<keyedReference keyName="..." keyValue"..."`
`tModelkey="uuid:....." />`

7.4.3.6 Das identifierBag-Element

Ein `identifierBag`-Element entspricht dem Schema eines `categoryBag`-Elements mit dem Unterschied dass es nicht zur Kategorisierung sondern zur Identifizierung eines Web-Services bzw. Anbieters stattfindet [Vgl. WalsP03, S.8].

7.4.3.7 PublisherAssertions

Bei `PublisherAssertions` handelt es sich um eine Komponente welche die Beziehung zwischen Unternehmen beschreibt. Diese Komponente gibt es erst seit UDDI 2.0 .

Die Komponente bzw. das Element besteht aus den zwingend erforderlichen Elementen `fromKey`, `toKey`, `keyedReference`. `FromKey` gibt dabei den Schlüssel `(businessKey des BusinessEntity)` von Unternehmen eins an und `toKey` den Schlüssel des zweiten Unternehmens. Mit `keyedReference` wird der Typ der Beziehung beschrieben [Vgl. ReCh03, S.5].

Die allgemeine Struktur einer `publisherAssertion`-Komponente ist in Code 54 dargestellt.

```
<publisherAssertion>
      <fromKey>......</fromKey>
      <toKey>...... ....</toKey>
      <keyedReference tModelKey="..." keyName="..." key
        Value="..." / >
</publisherAssertion>
```

Code 54 : Struktur einer `publisherAssertion`- Komponente

7.4.4 Operationen

UDDI Operationen können in zwei verschiedene Kategorien aufgeteilt werden.

Zum Einen in die *Publishing API*, welche für das Eintragen, Ändern und Löschen aller Einträge verantwortlich ist und zum Zweiten in die *Inquiry API* die für das Suchen und Finden von `BusinessEntities` und `BusinessServices` verantwortlich ist [Vgl. WalsP03, S.13].

8. Zusammenfassung und Ausblick

Dem Leser dieser Diplomarbeit sollte nun der Zusammenhang zwischen SGML, HTML und XML klar sein und er sollte auch deren Vor- und Nachteile kennen. Hierzu wurde in Kapitel 2 ein Einblick in den geschichtlichen Hintergrund und die Struktur dieser Sprachen gegeben.

In Kapitel 3 wurde anschließend auf das Thema HTML näher eingegangen. HTML ist derzeit immer noch die wichtigste und am weitesten verbreitete Technologie um Informationen im Worl Wide Web darzustellen. HTML ist für Themen wie Web-Design, Web-Engineering und das Erstellen von E-Commerce Anwendungen von zentraler Bedeutung. Aus diesem Grund wurden in Kapitel 3 wichtige HTML-Elemente wie Verweise, Tabellen, Formulare und Frames, mit welchen die Basis für solche E-Commerce Anwendungen realisiert werden können, angesprochen und erläutert.

HTML ist im eigentlichen Sinne eine Sprache mit welcher lediglich der Inhalt und die logische Struktur eines Dokuments definiert werden soll. Jedoch entwickelten sich im Laufe der Zeit immer wieder neue HTML-Elemente, welche zur Darstellung (Layout) von Inhalten verwendet werden. Um dieser Verwässerung der Elemente entgegenzuwirken, wurden vom W3C die so genannten Cascading Style Sheets (CSS) eingeführt. CSS werden verwendet um ein solches Dokument entsprechend optisch aufzuwerten. Dabei gibt es verschiedene Möglichkeiten ein SGML– bzw. XML-basiertes Dokument mit Hilfe von CSS darzustellen. Diese Möglichkeiten wurden daher in Kapitel 4.1 beschrieben.

Um ferner in einem HTML-Dokument Interaktionen zwischen Client und Server realisieren zu können wurden verschiedene Web-Technologien und deren Einsatzmöglichkeiten anhand von Beispielen veranschaulicht. Ein besonderes Augenmerk kam dabei der Programmiersprache Object Rexx zugute, welche sich durch ihre Einfachheit und Vielseitigkeit auszeichnet. Die meisten Skriptbeispiele in Kapitel 3, 4 und 5 wurden daher mit Hilfe von Object Rexx realisiert und sollten die Einsatzmöglichkeiten als auch die Zusammenhänge zwischen HTML bzw. XML, CSS und Skriptsprachen verdeutlichen. Besonders hervorzu-

heben ist dabei die Möglichkeit der Verwendung vordefinierter Objekte bzw. deren Eigenschaften und Methoden, mit welchen sich beispielsweise Windows Anwendungen automatisieren lassen oder Zugriffe auf HTML- oder XML-Dokumente ermöglicht werden.

Dabei wurden auch einige wichtige Unterschiede zwischen den Browsern MS Internet Explorer, Netscape Navigator und Opera angesprochen. So ist beispielsweise lediglich der MS Internet Explorer ein Windows Script Host (WSH) und erlaubt als einziger das Arbeiten mit Object Rexx. Allerdings arbeitet der Augsburger Student Tobias Specht im Rahmen seiner Diplomarbeit derzeit an einer Lösung, welche dieses Problem beseitigen soll. Diese Lösung nennt sich BSFWebScripting und basiert auf den Technologien BeanScriptingFramework (BSF) und LiveConnect.

Daneben kennt der Netscape Navigator das `all`-Objekt nicht und kann daher DHTML auf Basis des `all`-Objekts nicht verarbeiten. Aus diesem Grunde sollte standardmäßig dieses `all`-Objeklt nicht verwendet werden und stattdessen der Einsatz der Methoden `getElementById`(), `getElementByName`() oder `getElementByTagame`() des `document`-Objekts erfolgen.

Darüber hinaus wurden auch andere Sprachen wie PHP oder Perl vorgestellt und deren serverseitigen Einsatz in einem HTML-Formular demonstriert. Um diese Beispiele mit Hilfe von Abbildungen veranschaulichen zu können, wurde der kostenlose Apache Web Server 1.3.6 eingesetzt. Dieser wurde dabei lokal auf einem Windows 98 Betriebssystem installiert und verwendet.

Daran anschließend wurden weitere serverseitige Technologien wie JSP, ASP und RSP vorgestellt.

Kapitel 5 gab anschließend einen Einblick in die e**X**tensible **M**arkup **L**anguage und den XML-basierten Sprachen XSL, XML-Namensräume, XML Linking und XML Schema.

Anschließend wurden die XML-basierten Auszeichnungssprachen XHTML, SMIL und SVG näher betrachtet und anhand von Beispielen praktisch erläutert.

Den Abschluss dieser Arbeit bilden die zunehmend an Bedeutung nehmenden XML Web-Services, welche auf der Basis von XML erstellt werden und somit im Vergleich zu bisherigen Anwendungen bzw. Diensten den Vorteil der Standardisierung in Form von SOAP, WSDL, UDDI und HTTP in sich bergen.

Anhang

Code 55 stellt den Quelltext des für Code 8 verwendeten HTML-Dokuments dar.

```
<html>
<head>
<title></title>
<meta name="author" content="Thorsten Sch&auml;dler">
</head>
<body>
   -- form
<form style="background-color:lightyellow;width:50%"><br>
<h1>Select a name for the group</h1>  -- headline
<input type="text" id="name"><br>   -- textfield for groupname
  -- table
<table border="3">
<tr><td><h4 align="center">Name</h4></td>
<td><h4>Extension</h4></td></tr>
<tr>
<td>Ink Number 1:<input type="text" id="1"></td>
<td><input type="text" id="e1" size="5"></td>
</tr>
<tr>
<td>Ink Number 2:<input type="text" id="2"></td>
<td><input type="text" id="e2" size="5"></td>
</tr>
<tr>
<td>Ink Number 3:<input type="text" id="3"></td><td><input ty-
pe="text" id="e3" size="5"></td>
</tr>
</table><br>
<table border="4" >
<tr>
<td>Add the new group into your start menu:</td><td> <input
type="button" value="Add" onclick="call addgroup"></td>
</tr>
<tr>
<td>Delete the group:</td><td> <input type="button"
value="Delete" onclick="call deletegroup"></td>
<tr>
</table>
 -- end of table
<br>
<input type="reset" value="new">             -- "Reset"-button
</form>
 -- end of the form
</body></html>
```

Code 55 : HTML-Code für Code 8

Code 56 stellet den Quelltext des für Code 9 bzw. Abbildung 14 verwendeten HTML-Dokuments dar.

```
<html>
<head>
<title>Browser-usage</title>
</head>
 -- the background of the html file is lightyellow
<body bgcolor="lightyellow">
 -- the Object Rexx routine “Browser” is started by pressing
    the “Browser?”-button
<input type="button" value="Browser?" onclick="call Browser">
</body>
</html>
```

Code 56 : HTML-Code für Code 9

Quellverzeichnis

[AdWe03]	Addison-Wesley: Apache Grundlagen – open source library. http://www.addison-wesley.de/media_remote/katalog/bsp/3827320399bsp.pdf, Abruf am 2003-10-14.
[AnCr03]	Anthes Christina: HTML und XML. Proseminararbeit, 2002. http://www.dbis.informatik.uni-frankfurt.de/TEACHING/DB-Seminar /2002_SS/Ausarbeitung/XMLHTML.pdf, Abruf am 2003-08-29.
[AshD03a]	Ashley David : Apache Mod_Rexx_Interface Package, \Mod_Rexx-2.0.0\Apache\docs\Readme.html http://oss.software.ibm.com/developerworks/projects/modrexx, 2002, Abruf am 2003-12-09.
[AshD03b]	Ashley David : REXX Server Pages. \Mod_Rexx-2.0.0\Apache\docs\RSP.html. http://oss.software.ibm.com/developerworks/projects/modrexx, 2002, Abruf am 2003-12-09.
[AshD03c]	Ashley David : Available REXX Variables. Mod_Rexx-2.0.0\Apache\docs\RexxVars.html http://oss.software.ibm.com/developerworks/projects/modrexx, 2002, Abruf am 2003-12-09.
[AshD03d]	Ashley David : Mod_Rexx-2.0.0 . \Mod_Rexx-2.0.0\Apache\rexscripts\test.rex http://oss.software.ibm.com/developerworks/projects/modrexx, 2002, Abruf am 2003-12-09.
[ATvN04]	ATvirtual.Net: Apache Module. http://www.atvirtual.net/server/module.html, Abruf am 2004-01-27.
[BePiW03a]	Berger R., Pirklbauer M., Wollendorfer M. : Servlets, JSP, Java Beans, EJB`s - Servlets. Seminararbeit 19.April.2002. http://www2.iicm.edu/cguetl/education/projects/javatech2002/HTML_Version/node3.html, Abruf am 2003-09-18.
[BePiW03b]	Berger R., Pirklbauer M., Wollendorfer M. : Servlets, JSP, Java Beans, EJB`s – Java Server Pages. Seminararbeit, 2002-04-19. http://www2.iicm.edu/cguetl/education/projects/javatech2002/HTML_Version/node4.html, Abruf am 2003-09-20.
[BePiW03c]	Berger R., Pirklbauer M., Wollendorfer M. : Servlets, JSP, Java Beans, EJB`s – Java Beans. Seminararbeit , 2002-04-19. http://www2.iicm.edu/cguetl/education/projects/javatech2002/HTML_Version/node5.html, Abruf am 2003-09-20.
[BüMe03]	Büsching A., Meyer D. : DSSSL. Seminararbeit. Universität Bremen, 1999. http://www-rn.informatik.uni bremen.de/lehre/ sgml/www/ refe-rate/dsssl/dsssl.ps, Abruf am 2003-09-12.
[DSZ03]	Datenschutzzentrum : ActiveX. http://www.datenschutzzentrum.de/selbstdatenschutz/safer/browser/actcntnt/activex.htm, Abruf am 2003-11-08.

[Duen03]	Dünhölter Kuno : XML Syntax. Version 2.0 (vom 1.9.98). http://members.aol.com/xmldoku/syntax.htm, Abruf am 2003-10-27.
[eds03]	XPointer – Lexikon. http://www.eds.schema.de/doku/html-deu/lex/begriff/xpointer.htm, Abruf am 2003-11-07.
[En03]	Engenhausen Jan : Rexx on Windows – The power of ActiveX/OLE Automation. http://www.share.org/proceedings/sh96/data/S8312.pdf, Abruf am 2003-09-19.
[FiBrT03]	Fichtner Brunswig Tanja : W3C Standards I: SOAP, WSDL, UDDI. Seminararbeit, Technische Universität München, SS2003. http://www3.informatik.tu-muenchen.de/lehre/SS2003/HSEMbayer/Ausarbeitung2.pdf, Abruf am 2003-12-02.
[Fla01]	Rony G. Flatscher: Unterlagen zu Einführung in die prozedurale und objektorientierte Programmierung (Object Rexx). Seminar WS 01/02. Einheit 15+16.
[Fla03a]	Rony G. Flatscher: Automatisierung von Windows Anwendungen - Automatisierung von Windows Anwendungen (1). http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/Automatisierung_01.pdf, Abruf am 2003-01-12.
[Fla03b]	Rony G. Flatscher: Automatisierung von Windows Anwendungen - Automatisierung von Windows Anwendungen (2). http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/Automatisierung_02.pdf, Abruf am 2003-01-12.
[Fla03c]	Rony G. Flatscher: Automatisierung von Windows Anwendungen - Automatisierung von Windows Anwendungen (3). http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/Automatisierung_03.pdf, Abruf am 2003-01-12.
[Fla03d]	Rony G. Flatscher: Automatisierung von Windows Anwendungen - Automatisierung von Windows Anwendungen (4). http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/Automatisierung_04.pdf, Abruf am 2003-01-12.
[Fla03e]	Rony G. Flatscher: Automatisierung von Windows Anwendungen - Automatisierung von Windows Anwendungen (5). http://www.wiwi.uni-augsburg.de/wi3/2002ws/Automatisierung/Automatisierung_05.pdf, Abruf am 2003-01-12.
[Fla03f]	Rony G. Flatscher : BSF Web Scripting with Rexx – Architecture and Application Examples. http://wi.wu-wien.ac.at/rgf/rexx/orx14/2003_orx14_A_WebBScripting_by2.pdf, May 2003, Abruf am 2003-12-21.
[Fla03g]	Rony G. Flatscher : The Augsburg Version of BSF4rexx. http://wi.wu-wien.ac.at/rgf/rexx/orx14/orx14_bsf4rexx-av.pdf, May 2003, Abruf am 2003-12-20.
[Fla03h]	Rony G. Flatscher : Java Bean Scripting with Rexx. http://wi.wu-wien.ac.at/rgf/rexx/orx12/JavaBeanScriptingWithRexx_orx12.pdf, April 2001 , Abruf am 2003-11-26.

[FrUl03]	Frank Ulrich : Hauptseminar – Web Services und verteilte Datenbankanbindungen – SOAP, 2003-05-31. www.fmi.uni-passau.de/~ehrlich/ Hauptseminar/ Hauptseminar.pdf, Abruf am 2003-12-12.
[gbn03]	ASP.NET – QuickStart Tutorial. http://de.gotdotnet.com/QuickStart/aspplus/default.aspx?url=/quickstart/aspplus/doc/whatisaspx.aspx, Abruf am 2003-10-17.
[GrüF03]	Grützmacher Felix :Kurzeinführung in Javascript, Ausarbeitung zum Seminar vom 26. April 2002. http://homepages.fh-giessen.de/~hg10851/seminar.pdf, Abruf am 2003-09-28.
[HäPeS00]	H. Häckelmann, J.P. Petzold, S. Strahringer : Kommunikationssysteme – Technik und Anwendung. Springer, 2000.
[HelF03]	Florian Helmecke: Automation of Windows Applications with Object Rexx, Diplomarbeit, Universität Augsburg, 2003. http://wi.wu-wien.ac.at/rgf/diplomarbeiten/2003_Helmecke_A/Master Thesis_AutomationOfWindowsApplications WithObjectRexx.pdf, Abruf am 2003-06-12.
[HeSp03a]	Helma Spona: Animation. http://www.s-v-g.net/bspAnim.php, Abruf am 2003-11-14.
[HeSp03b]	Helma Spona: Hyperlinks. http://www.s-v-g.net/beispiele/hyperlinks.svg, Abruf am 2003-11-14.
[IBM03a]	IBM: Object Rexx for Windows Reference Version 2.1, SH12-6725-00. http://www-3.ibm.com/software/awdtools/obj-rexx/library.html,2001, Abruf am 2003-09-12.
[IBM03b]	IBM: Object Rexx for Windows: OODialog Method Reference Version 2.1, SH12-6727-00. http://www-3.ibm.com/software/awdtools/obj-rexx/library.html, 2001, Abruf am 2003-09-12.
[KnSW98]	Knobloch Frank, Seeboerger-Weichselbaum Michael : Java – Das Einsteigerseminar. Bhv, 1998, 3.Auflage.
[KoRP03]	Korte Ralf Peter :WML-Tutorial, April 2001 (Version 01.0422) . http://www.wml-tutorial.de/inhalt.html, Abruf am 2003-09-15.
[KoRP03a]	Korte Ralf Peter :WML-Tutorial -Allgemeines zu WML un WAP. http://www.wml-tutorial.de/w01.html, Abruf am 2003-09-15.
[KoRP03b]	Korte Ralf Peter :WML-Tutorial – Grundlagen WML-Programmierung http://www.wml-tutorial.de/w02.html, Abruf am 2003-09-15.
[KoRP03c]	Korte Ralf Peter :WML-Tutorial - Grundgerüst einer WML-Datei. http://www.wml-tutorial.de/w03.html, Abruf am 2003-09-15.

[KoRP03d]	Korte Ralf Peter :WML-Tutorial - Die Card. http://www.wml-tutorial.de/w05.html, Abruf am 2003-09-15.
[KoRP03e]	Korte Ralf Peter :WML-Tutorial - Grafiken in WML. http://www.wml-tutorial.de/w07.html, Abruf am 2003-09-15.
[KoRP03f]	Korte Ralf Peter :WML-Tutorial - Ereignisbehandlung. http://www.wml-tutorial.de/w12.html, Abruf am 2003-09-15.
[KoRP03g]	Korte Ralf Peter :WML-Tutorial – Grundlagen WMLScript-Programmierung. http://www.wml-tutorial.de/wso1.html, Abruf am 2003-09-15.
[KoRP03h]	Korte Ralf Peter :WML-Tutorial – Funktionen. http://www.wml-tutorial.de/inhalt.html, Abruf am 2003-09-15.
[KrWeG03]	Kronsbein M., Weinert T., Gutweiler C. :PHP zum Nachschlagen. SYBEX, 2 Auflage, 2003.
[Kurz03]	Kurzweil Andreas :WSH – Definieren von COM Interfaces. Seminararbeit, WU Wien, SS2002. http://www.wu-wien.ac.at/usr/h96a/h9651692/wi_sem.pdf, Abruf am 2003-09-02.
[msAN03]	Microsoft ASP.NET. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/anchoraspdotnet.asp, Abruf am 2003-12-28.
[MSJS03]	Microsoft: JScript -Tutorium, 1997. http://www.aspgerman.com/iishelp/jscript/htm/js0.htm, Abruf am 2003-08-17.
[msnd03]	Microsoft .Net Development. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/netdevanchor.asp, Abruf am 2003-12-26.
[MSVBS03]	Microsoft: VBScript – Tutorium, 1997. http://www.aspgerman.com/iishelp/VBScript/htm/vbstutor.htm, Abruf am 2003-08-17.
[msWA03]	Microsoft : Web Services with ASP.NET. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspnet/html/asp02222001.asp, Abruf am 2003-12-28.
[MüEl03]	Eliane Müller : Interaktion Java und JavaScript. http://fgb.informatik.unibas.ch/fgb/archive/ss01/java_technologien/material/online/lekt14/interaktion.pdf, Abruf am 2003-12-12.
[MüNe01]	Münz S., Nefzger W.: HTML 4.0 Handbuch, Franzis, 3. Auflage, 2001.
[MüNe02]	Münz S., Nefzger W.: HTML & Web-Publishing Handbuch, Franzis, 2002.

[Net03a]	Netscape Communications Corp. : Core JavaScript Guide 1.5, Chapter 1 – JavaScript Overview, 2000-09-28. http://devedge.netscape.com/library/manuals/2000/javascript/1.5/guide/intro.html#1009367, Abruf am 2004-01-26.
[Net03b]	Netscape Communications Corp. : Core JavaScript Guide 1.5, Chapter 8 – Details of the Object Model, 2000-09-28. http://devedge.netscape.com/library/manuals/2000/javascript/1.5/guide/obj2.html#101380, Abruf am 2004-01-26.
[Net03c]	Netscape Communications Corp.: Core JavaScript Guide 1.5, Chapter 7 - Working with Objects, 2000-09-28. http://devedge.netscape.com/library/manuals/2000/javascript/1.5/guide/obj.html#1008302, Abruf am 2004-01-26.
[Net03d]	Netscape Communications Corp.: Core JavaScript Guide 1.5, Chapter 9 – LiveConnect Overview, 2000-09-28. http://devedge.netscape.com/library/manuals/2000/javascript/1.5/guide/lc.html#1008305, Abruf am 2004-01-26.
[nwc03]	Networkcomputing : Web stets zu Diensten. http://www.networkcomputing.de/heft/solutions/sl-2002/sl_0502_52.htm, Ausgabe 5, 2002, Abruf am 2003-08-07.
[OaUD03]	Oasis : UDDI Version 3.0.1. UDDI Specification, 2003-10-14 http://www.uddi.org/pubs/uddiv-3.0.1-20031014.htm, Abruf am 2003-20-12.
[PeeL03]	Peedin Lee: Excel_demo – Session_8332.ppt. Automating Microsoft Excel via Object Rexx OLE. http://pragmaticlee.safedataisp.net/zipfiles/Excel_demo.zip, Abruf am 2003-11-23.
[ReCH03]	Prof. Dr. Ch. Reich : UDDI. http://www.informatik.fh-furtwangen.de/~reich/ XMLWebServi-ces/ index.html, Abruf am 2003-12-15.
[Sche00]	Ignaz Schels Jun., Ignatz Schels : Hacker Basics –Programmieren lernen. Markt+Technik, 2000.
[Sht03a]	Selfhtml : Grundlagen von XSL und XSLT, 2001 http://selfhtml.teamone.de/xml/darstellung/xslgrundlagen.htm, Abruf am 2003-10-28.
[Sht03b]	Selfhtml : XSLT Elemente, 2001. http://selfhtml.teamone.de/xml/darstellung/xsltelemente.htm, Abruf am 2003-10-28.
[Sht03c]	Selfhtml : Xpath Syntax. http://selfhtml.teamone.de/xml/darstellung/xpathsyntax.htm, 2001, Abruf am 2003-09-30.
[Sht03d]	Selfhtml : Einfühung in XML. http://selfhtml.teamone.de/xml/intro.htm, 2001, Abruf am 2003-09-30.
[Sht03e]	Selfhtml :Xpath Funktionen. http://www.netzwelt.com/selfhtml/xml/darstellung/xpathfunktionen.htm, 2001, Abruf am 2003-09-30.

[Sht03f]	Selfhtml : XML und XML Derivate. http://selfhtml.teamone.de/intro/technologien/xml.htm#derivate, 2001, Abruf am 2003-09-28.
[Sht03g]	Selfhtml: Modul CGI : Funktionen für die CGI Programmierung http://selfhtml.teamone.de/cgiperl/module/cgi.html, Abruf am 2003-09-14.
[Sht03h]	Selfhtml: Einführung in das Arbeiten mit Modulen. http://selfhtml.teamone.de/cgiperl/module/intro.htm, Abruf am 2003-09-14.
[Sht03i]	Selfhtml : ASP- Active Server Pages. http://selfhtml.teamone.de/intro/technologien/asp.htm, Abruf am 2003-10-12.
[Sht03j]	Selfhtml: Grundsätzliches zu Perl. http://selfhtml.teamone.de/cgiperl/sprache/intro.htm, Abruf am 2003-09-12.
[SpeT03a]	Specht Tobias : Using Web-Browser as Application Platform. http://bsfws.berlios.de/, 2003-01-20, Abruf am 2003-11-26.
[SpeT03b]	Specht Tobias : Browser als Anwendungsplattform – Entwicklung eines Browser Skriptsprachen Interface (Handout). http://bsfws.berlios.de/handout_2002-12-17.html, 2002-12-17, Abruf am 2003-12-19.
[SpeT03c]	Specht Tobias: Mail vom 15.2.2004.
[Stae03]	Stärk Jochen : PHP Tutorial, 2001. http://www.usegroup.de/software/phptutorial/was_ist_php.html, Abruf am 2003-09-10.
[Sun03]	The Java WebServices Tutorial , 25.7.2003. http://java.sun.com/webservices/tutorial.html,, Abruf am 2003-10-10.
[TuKr02]	Turowski Klaus, Krammer Andreas: Beiblattsammlung zur Vorlesung Web Engineering, SS2002.
[W3C02]	W3C: HTML 4.0.1 Specification. W3C Recommendation 24 December 1999. http://www.w3.org/TR/1999/REC-html401-19991224, Abruf am 2002-11-14.
[W3C03a]	W3C : DOM – Document Object Model. http://www.w3.org/DOM/, Abruf am 2003-09-17.
[W3C03b]	W3C: Modularization of XHTML™, 2001-04-10. http://www.w3.org/TR/2001/REC-xhtml-modularization-20010410, Abruf am 2003-10-12.
[W3C03c]	W3C: SOAP 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508, 08.05.2000, Abruf am 2003-12-03.

[W3C03d]	W3C : SOAP Version 1.2 Part 0 : Primer. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, 24.6.2003, Abruf am 2003-12-03.
[W3C03e]	W3C: WSDL 1.1 . http://www.w3.org/TR/wsdl/, 15.01.2001, Abruf am 2003-12-01.
[W3C03f]	W3C : WSDL 1.2 . http://www.w3.org/TR/2003/WD-wsdl12-20030303, 03.03.2003, Abruf am 2003-12-01.
[W3C03g]	W3C : WSDL 2.0 Part 1 – Core Language. http://www.w3.org/TR/2003/WD-wsdl20-20031110, 10.11.2003, Abruf am 2003-12-04.
[W3C04]	W3C: Cascading Style Sheets, level 2 revision 1. CSS 2.1 Specification, W3C Working Draft 15 September 2003. http://www.w3.org/TR/2003/WD-CSS21-20030915, Abruf am 2004 -01-26.
[W3Ce03a]	W3C: XML Linking Language (XLink) Version 1.0, Deutsche Übersetzung, 2002-06-26. http://www.edition-w3c.de/TR/2001/REC-xlink-20010627/, Abruf am 2003-10-12.
[W3Ce03b]	W3C : XML Schema Teil 0: Einführung, Deutsche Übersetzung. http://www.edition-w3c.de/TR/2001/REC-xmlschema-0-20010502/, Abruf am 2003-10-12.
[W3Ce03c]	W3C : XHTML 1.0 (Zweite Auflage), 01.08.2002, Deutsche Übersetzung. http://www.edition-w3c.de/TR/2002/REC-xhtml1-20020801/, Abruf am 2003-10-12.
[W3Ce03d]	W3C: Modularisierung von XHTML, 07.01.2003. http://www.edition-w3c.de/TR/2001/REC-xhtml-modularization-20010410, Abruf am 2003-10-12.
[W3Ce03e]	W3C : XHTML Basic – Überarbeitete Fassung, 20.12.2002. http://www.w3.org/TR/2000/REC-xhtml-basic-20001219, Abruf am 2003-10-12.
[W3Ce03f]	W3C : XHTML 1.1 - Modulbasiertes XHTML, 07.01.2003. http://www.edition-w3c.de/TR/2001/REC-xhtml11-20010531, Abruf am 2003-10-12.
[W3Ce04]	W3C : Cascading Style Sheets, Level 2, Deutsche Übersetzung. http://www.edition-w3c.de/TR/1998/REC-CSS2-19980512, 1998-05-12, Abruf am 2004-01-26.
[W3Ce04a]	W3C: Pseudo-Elemente und Pseudo-Klassen. http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap05.html#heading-5.10, 1998-05-12, Abruf am 2004-01-26.
[W3ce04b]	W3C: Erkannte Medientypen. http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap07.html#heading-7.2.1, 1998-05-12, Abruf am 2004-01-26.
[W3Ce04c]	W3C: @import-Regel. http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap06.html#heading-6.3, 1998-05-12, Abruf am 2004-01-26.

[W3Ce04d]	W3C: @media-Regel. http://edition-w3c.de/TR/1998/REC-CSS2-19980512/kap07.html#heading-7.2.1, 1998-05-12, Abruf am 2004-01-26.
[W3s03a]	W3schools: SMIL-Tutorial. http://www.w3schools.com/smil/default.asp, Abruf am 2003-11-03.
[W3s03aa]	W3schools: SMIL-Tutorial - SMIL files. http://www.w3schools.com/smil/smil_files.asp, Abruf am 2003-11-03.
[W3s03ab]	W3schools: SMIL-Tutorial – SMIL in Parallel. http://www.w3schools.com/smil/smil_parallel.asp, Abruf am 2003-11-03.
[W3s03ac]	W3schools: SMIL-Tutorial – SMIL Sequence http://www.w3schools.com/smil/smil_seq.asp,Abruf am 2003-11-03.
[W3s03ad]	W3schools: SMIL-Tutorial – SMIL Media Elements. http://www.w3schools.com/smil/smil_media.asp,Abruf am 2003-11-03.
[W3s03ae]	W3schools: SMIL-Tutorial – SMIL in HTML. http://www.w3schools.com/smil/smil_html.asp ,Abruf am 2003-11-03.
[W3s03b]	W3schools: SVG-Tutorial. http://www.w3schools.com/svg/default.asp, Abruf am 2003-11-03.
[W3s03ba]	W3schools: SVG-Tutorial – Introduction to SVG. http://www.w3schools.com/svg/svg_intro.asp, Abruf am 2003-11-03.
[W3s03bb]	W3schools: SVG-Tutorial – SVG Structure. http://www.w3schools.com/svg/svg_structure.asp, Abruf am 2003-11-03.
[W3s03bc]	W3schools: SVG-Tutorial – SVG Filters. http://www.w3schools.com/svg/svg_filters_intro.asp, Abruf am 2003-11-03.
[W3s03bd]	W3schools: SVG-Tutorial – SVG Gaussian Blur. http://www.w3schools.com/svg/svg_filters_gaussian.asp , Abruf am 2003-11-03.
[W3s03ca]	W3scools: Schema-Tutorial – Introduction to XML Schema. http://www.w3schools.com/schema/schema_intro.asp,Abruf am 2003-10-10.
[W3s03cb]	W3scools: Schema-Tutorial – XSD Complex Elements. http://www.w3schools.com/schema/schema_complex.asp,Abruf am 2003-10-10.
[W3s03cc]	W3scools: Schema-Tutorial – XSD Simple Elements. http://www.w3schools.com/schema/schema_simple.asp,Abruf am 2003-10-10.
[W3s03cd]	W3scools: Schema-Tutorial – XSD - The <schema> Element. http://www.w3schools.com/schema/schema_schema.asp,Abruf am 2003-10-10.

[W3s03d]	W3schools: WAP-Tutorial. http://www.w3schools.com/wap/wap_basic.asp, Abruf am 2003-09-15.
[W3s03e]	W3schools: SOAP-Tutorial. http://www.w3schools.com/soap/soap_intro.asp, Abruf am 2003-12-12.
[W3s03g]	W3schools : ASP.NET-Tutorial. http://www.w3schools.com/aspnet/aspnet_vsasp.asp, Abruf am 2003-11-12.
[WalsP03]	Walsweer Pieter: Das Universal Description,Discovery and Integration. http://iug.uni-paderborn.de/iug/lehre/ws03/pg_WNG/UDDI.pdf, Abruf am 2003-12-15.
[wat03]	Wir-age : tutorial - XML-Namensräume (NSP). www.wir-age.de/home/tutorials/xml/nsp.pdf, Abruf am 2003-10-18.
[WinB04]	Winkler B. : Active Server Pages. http://www.rz.uni-bayreuth.de/lehre/mmi/vorlesung/node43.html, 2003-10-29, Abruf am 2004-01-23.
[WinJ03]	Winkler Jan : HTML-World – Einführung JavaScript. http://www.html-world.de/program/js_1.htm, Abruf am 2003-09-13.