DIPLOMARBEIT

Visualisierung von Datenräumen mittels XML / XSL unter Lotus Notes R5 anhand einer dynamisch generierten Sitemap

von

Holger Gimbel
Matrikel-Nummer: 10 20 83 80

Fachbereich Informatik
Studiengang Wirtschaftsinformatik

1. Prüfer: Prof. Dr. Heide Faeskorn - Woyke
2. Prüfer: Oliver Trabert / CONET CONSULTING AG

Fachhochschule Köln
Abt. Gummersbach

Gummersbach, 13. Juni 2000

Inhaltsverzeichnis

Kapitel 1: Vorwort

1.1 Visualisierung von Datenräumen mittels XML / XSL unter Lotus Notes R5 anhand einer dynamisch generierten Sitemap

Die vorliegende Diplomarbeit entstand an der Fachhochschule Köln, Abteilung Gummersbach, im Fachbereich Informatik in Zusammenarbeit mit der Firma CONET CONSULTING AG.

In dieser Diplomarbeit werden die Fragen: „Was ist das?“, „Wer braucht das?“ und „Wie geht das?“ zu XML / XSL zuerst theoretisch und dann praktisch anhand eines Prototyps beantwortet.
Die Diplomarbeit besteht aus dieser Auswertung und einer CD-Rom, die eine Datenbank mit integriertem Lotus Notes[1] Agenten und eine Beispiel-Sitemap enthält.

Unter XML / XSL kann man eine neue Form der Auszeichnungssprachen verstehen, bei der der Inhalt von der Darstellung der Daten getrennt betrachtet wird. Hierbei ergeben sich neue Möglichkeiten im Bereich der Business-to-Business Verbindungen über EDI[2].
In dieser Diplomarbeit untersuche ich allerdings im Wesentlichen die Visualisierung von Datenräumen mittels XML / XSL, die sich in einer Lotus-Notes-Datenbank befinden.

Kapitel 2 erklärt die Grundlagen, die zum Verständnis dieser Diplomarbeit notwendig sind.

[1] Die Bezeichnungen “Notes” und “Lotus Notes” werden innerhalb dieser Arbeit gleichwertig verwendet.

[2] Electronic Data Interchange

Kapitel 3 stellt die Vorgehensweisen vor, wie XML und XSL unter Lotus Notes R5 eingesetzt werden kann, und beschreibt detailliert die Erzeugung der dynamischen Sitemap in XML / XSL.

In Kapitel 4 werden die Programme, auf die die Sitemap aufgesetzt bzw. integriert wurde, sowie die dabei eingesetzten Hilfsmittel vorgestellt.

Zuletzt wird in Kapitel 5 und 6 noch ein Ausblick auf die zukünftige Entwicklung von XML gegeben.

Hinweise auf die in dieser Diplomarbeit verwendeten Literatur werden in eckigen Klammern geschrieben. [Pott 1999] verweist z.B. auf das von Oliver Pott und Gunter Wielage geschriebene Werk „XML – Praxis und Referenz“ vom Markt & Technik Verlag, das 1999 in München veröffentlicht wurde.
URLs werden im Anhang D „Websites“ aufgelistet.

Mein Dank gilt als Erstes meiner betreuenden Professorin Frau Faeskorn-Woyke sowie meinem zweiten Betreuer Herrn Oliver Trabert. Als Nächstes bin ich allen Mitarbeitern der Firma CONET CONSULTING AG, die mich während der letzten Monate bei meiner Arbeit unterstützt haben, zu großem Dank verpflichtet, ganz besonders Herrn Stein, Herrn Radigewski und Herrn Hubrich.
Außerdem möchte ich der Fachhochschule Köln, Abteilung Gummersbach, ein großes Lob aussprechen. Mein Wirtschaftsinformatik Studium empfand ich als äußerst interessant und durch die praxisnahe Ausbildung als besonders aktuell.
Last, but not least, möchte ich mich an dieser Stelle bei meiner Familie und meinen Freunden bedanken, die mich während der ganzen Zeit großartig und verständnisvoll unterstützt haben.

Gummersbach, 13. Juni 2000 Holger Gimbel

1.2 Aufgabenstellung

Im Rahmen der Diplomarbeit sollte neben dieser Ausarbeitung eine Sitemap[3] mit XML und XSL entwickelt werden, die sich dynamisch der Dokumenten-Struktur einer Datenbank des Web-Content-Management Systems WebGate[4] unter Lotus Notes anpasst. WebGate ist ein Produkt der Firma „Innovation Gate GmbH“, die hauptsächlich für die Weiterentwicklung von WebGate zuständig ist. Die Projektierung und Ausführung von Kundenaufträgen wird von der Firma CONET CONSULTING AG durchgeführt. Im Kapitel 4.1 wird das Web-Content-Management System WebGate genauer beschrieben.

Die Sitemap ist als zusätzlicher Service-Navigator gedacht, der in einem eigenen Fenster über den Main-Frame visualisiert wird. Der Inhalt der per Mausklick ausgewählten Dokumente wird dann im Main-Frame geladen und anzeigt. Außerdem erscheint zu jedem Dokument eine Kurzbeschreibung, wenn die Maus über den Link des Dokuments streicht.

Die Ermittlung der zu visualisierenden Daten wird anhand eines Agenten unter Notes realisiert, der zusätzlich die Daten auf Sonderzeichen überprüft und automatisch, je nach gewünschter Sprache, das entsprechende Dokument auswählt. Außerdem muss ein Versions-Check des Browsers stattfinden, denn nicht jeder Internet-Browser ist in der Lage, XML und XSL darzustellen. Zur Zeit kann dies lediglich der Internet Explorer 5 von Microsoft. Netscape hat mit der Beta Version 6 nachgezogen, allerdings wird nur XML und CSS unterstützt. XSL soll erst nach der Verabschiedung des Entwurfs als allgemeiner Standard durch das W3C[5] implementiert werden.

---

[3] Unter einer Sitemap kann man ein Inhaltsverzeichnis verstehen, das wie das Inhaltsverzeichnis dieser Diplomarbeit aufgebaut ist. Es werden alle Überschriften der einzelnen Kapitel angezeigt, zu denen man anschließend per Mausklick gelangt. In diesem Fall handelt es sich um das Inhaltsverzeichnis eines Web-Auftritts.

[4] Siehe Kapitel 4.1

[5] Siehe Kapitel 2.2

1.3 CONET CONSULTING AG

1.3.1 Firmenprofil

Die CONET CONSULTING AG wurde im Jahr 1987 gegründet. Das Leistungsspektrum des Unternehmens reicht von der Unternehmens- und IT-Beratung über die Entwicklung und Anpassung von Standard-Software auf spezielle Erfordernisse bis hin zur Implementierung und Betreuung von IT-Systemen. Weiterhin gehören Training und Support zum Angebot von CONET CONSULTING.

Managementberatung:
Die Beratungsschwerpunkte umfassen unter anderem Organisationsentwicklung, Controlling, Optimierung von Geschäftsprozessen und Kostenrechnung sowie den Aufbau von Management-Informations-Systemen (MIS).

IT-Beratung, IT-Konzepte und Systemlösungen:
Neben der Konzepterstellung werden die organisatorischen Aufgabenstellungen auf geeignete Software-, Hardware- und Kommunikationsressourcen abgebildet. Diese Einzelteile werden anschließend zu einer optimalen Gesamtlösung integriert. Während der gesamten Projektlaufzeit, angefangen bei der Konzeption, über die Unterstützung bei Vorbereitung und Auswertung von Ausschreibungen bis hin zur Begleitung bei der Einführung, erfolgt die Beratung des Kunden.

Anwendungsentwicklung und –integration für administrative und kaufmännische Systeme:
Die Anwendungen werden im Umfeld von Client/Server-Datenbanken realisiert und betreut. Hierbei werden schwerpunktmäßig Netze auf Novell oder Windows NT-Basis, jeweils unter Einbeziehung von Intranet- oder Internet-Technologien, eingesetzt. Es werden aber auch Lösungen mit lokalen Daten auf Einzelarbeitsplätzen entwickelt bzw. integriert.

Kommunikationstechnik:

Um verteilt und vernetzt arbeiten zu können, wird eine leistungsfähige technische Infrastruktur benötigt. Hierfür werden bestehende Netze analysiert und Redesign-Vorschläge entwickelt. Nach der Konzeption und Planung wird die gesamte Infrastruktur für LANs[6] und WANs[7] installiert und in vorhandene heterogene Umgebungen integriert.

Schulung und Betreuung:

Der Kunde wird bei dem effektiven Einsatz seiner neu eingeführten Programme und Systeme mit Schulungen und Support unterstützt, unabhängig davon, ob es sich um Standard-Software oder Individual-Lösungen handelt.
In den fünf Geschäftsbereichen Anwendungsdesign, Industrieprojekte, Logistik & Controlling, Systemintegration sowie Touristik & Technologie betreuen derzeit über 200 Mitarbeiter an den Standorten Hennef, Berlin und Dresden Unternehmen wie Bayer, Henkel, Coca-Cola und viele mehr. Hinzu kommen zahlreiche Kunden aus dem Bereich der öffentlichen Verwaltung wie Bundesministerien und –ämter.

Das Informations- und Qualitätsmanagement wurde nach DIN ISO 9001 zertifiziert, wodurch eine hohe Qualität der Beratungs- und Entwicklungs-leistungen gewährleistet wird.

CONET CONSULTING hat eine Tochterfirma (CONET InfoSys GmbH) in Neubrandenburg und ist an der Innovation Gate GmbH in Ratingen mit 33 % beteiligt. Im Jahr 1999 (Geschäftsjahr April bis März) wurde ein Umsatz von rund 32 Mio. DM erwirtschaftet.

[6] Local Area Network

[7] Wide Area Network

Kapitel 2: Allgemeine Grundlagen

2.1 Lotus Notes / Domino

Lotus Notes gilt als Groupware[8] Plattform mit dem Einsatzgebiet für kooperatives Arbeiten in einem räumlich und zeitlich verteilten Umfeld und basiert auf dem Client / Server-Prinzip. Die Bezeichnung „Domino“ bezieht sich dabei auf die Server-Komponente, während „Notes“ die Client-Seite spezifiziert.
Ein wesentliches Merkmal des Domino-Servers ist das Generieren dynamischer WWW-Seiten. Nach Anforderung durch einen WWW-Client (Browser) wird eine den Datenbank-Inhalt repräsentierende HTML-Seite erzeugt, die Aktualität garantiert und somit Anwendungen wie z.B. Electronic-Commerce ermöglicht.

Lotus Notes / Domino steht für eine Vielzahl von Systemplattformen[9] zur Verfügung.

Notes Datenbanken zeichnen sich durch eine spezielle Datenbankstruktur aus, die als dokumentenorientierte Datenbank bezeichnet wird. In diesen Datenbanken werden Informationen in Form von Dokumenten analog zur Papierwelt erstellt, verwaltet und verarbeitet. Das Dokument bildet die Grundeinheit in der Datenbank analog eines Datensatzes in der Welt der relationalen Datenbanken.[10]
Es kann alle zweidimensional darstellbaren Informationen enthalten wie formatierten und unformatierten Text, Tabellen, Formeln, Grafiken, Bilder oder Sprachinformationen. Dabei wird innerhalb der Notes-Datenbank die Struktur konsequent vom Inhalt getrennt.

---

[8] Groupware beschreibt nicht nur eine Software oder eine Technologie, sondern ein Konzept, welches das gemeinsame Arbeiten von Menschen in den Vordergrund rückt und für diese Arbeit elektronische Werkzeuge zur Verfügung stellt.

[9] Verfügbare Domino-Plattformen sind Windows NT, Windows 95, OS/2, AIX, HP-UX, Solaris, OS/400, ab Version 5.0 auch AS/400, IBM OS/390 und demnächst auch Linux;
Notes Clients werden für Windows 3.1, 95, 98, NT, Mac OS und Unix angeboten

[10] Vgl. Markus Dierker / Martin Sander, Lotus Notes, 1998, S. 26 [Dierker 1998]

2.1.1 Die Gestaltungselemente von Lotus Notes

2.1.1.1 Masken

Masken enthalten die strukturelle Beschreibung eines Dokuments und bilden die Grundlage für die Eingabe und Darstellung von Dokumenten in einer Datenbank.
Das Erscheinungsbild der Maske kann mit Feldern, Hotspots, Grafiken, statischem Text, Tabellen, Schaltflächen, Aktionsleisten, Teilmasken und Layoutbereichen gestaltet werden.

2.1.1.2 Ansichten

Ansichten sind eine tabellenartige Präsentation zur Auswahl, Sortierung und Kategorisierung von Dokumenten. Innerhalb einer Ansicht wird in Spalten der Inhalt eines Feldes oder das Ergebnis einer Formel angezeigt.
Es können auch Filterkriterien für die Auswahl der anzuzeigenden Dokumente festgelegt werden, so dass nur eine bestimmte Untermenge von Dokumenten ausgewählt und visualisiert wird. Im Rahmen dieser Diplomarbeit werden verschiedene schon bestehende Ansichten einer WebGate-Datenbank dazu genutzt, die in der Sitemap zu visualisierenden Dokumente zu selektieren und z.B. gesperrte Dokumente auszufiltern.

2.1.1.3 Navigatoren

Der Navigator ermöglicht die Steuerung der Aktivitäten des Benutzers. Im Navigationsbereich werden die Ordner und Ansichten der Notes-Datenbank dargestellt, so dass der Benutzer Dokumente über den Navigator aus verschiedenen Ansichten auswählen kann.[11]
Der Navigator bietet dabei einen visuellen Index für den Inhalt einer Datenbank und führt den Benutzer bei der Navigation durch die Dokumentstruktur.

[11] Vgl. Uwe Schröter / Stephan Fügner, Domino-Prinzip, 1998, S. 62 f. [Schröter 1998]

2.1.1.4 Aktionen

Aktionen ermöglichen die Realisierung der Funktionalität in einer Notes-Anwendung. Sie sind an Schaltflächen bzw. Optionen im Menü gebunden. Die Schaltflächen wiederum sind an Masken und Ansichten gebunden.

2.1.1.5 Agenten

Ebenso wie Aktionen ermöglichen Agenten die Realisierung der Funktionalität in einer Notes-Anwendung. Sie dienen dazu, bestimmte Abläufe zu automatisieren bzw. auszuführen.
Die Agenten können wie Aktionen manuell oder durch ein Ereignis automatisch ausgelöst werden. Ereignisse können dabei zeitlich gesteuert werden, indem Zeitintervalle für die Ausführung des Agenten vorgegeben werden oder beispielsweise an eine Datenbankänderung gekoppelt sind. Beispiele hierfür könnten das Anlegen, Ändern oder Löschen eines Dokumentes, der Eingang einer Mail usw. sein.

Der in dieser Diplomarbeit eingesetzte Agent wird manuell durch einen Link im Navigationsframe ausgelöst und kann zusätzlich durch den Button „Aktualisieren“ in der Sitemap gestartet werden.

Agenten können durch folgende Möglichkeiten realisiert werden:

- Einsatz von einfachen Aktionen, die vom Notes-System zur Verfügung gestellt werden.
- Erstellen von Formeln, die in der integrierten @Formel-Sprache repräsentiert werden.
- Erstellen von LotusScript-Programmen
- Einsatz von JAVA-Applikationen

2.1.1.6 Seiten

Seiten[12] wurden in Notes-R5 (Release 5) eingeführt, um dem Web-Entwickler die Möglichkeit zu bieten, eine Seite für eine Domino-Webseite zu erstellen, ohne zuvor eine Maske dafür zu definieren oder einen Navigator erstellen zu müssen.

Folgende Komponenten können in eine Seite eingefügt werden:

- Plain-HTML-Sourcecode
- Formatierter Text
- Applets[13]
- Embedded Elements[14]
- Sections[15]
- Gliederungen

Auf Seiten werden folgende Komponenten nicht unterstützt:

- Felder
- Teilmasken[16]
- Layoutregionen[17]

---

[12] Vgl. Raimund Mann, Domino Designer R5, S. 150 f. [Mann 1999]

[13] Einfaches JAVA -Programm, das Bestandteil einer HTML-Seite ist.

[14] Dies kann z.B. eine „Eingebettete Ansicht“ sein.

[15] Eine Section ist ein Bereich, der Objekte, Text und Grafik beinhalten kann. Dieser Bereich kann per Mausklick erweitert, bzw. auf die Überschrift komprimiert werden.

[16] Teilmasken entsprechen vom Aufbau und der Funktionalität her normalen Masken. Sie können allerdings nicht zur Erstellung von Dokumenten verwendet werden. Vgl. [Mann 1999, S. 102]

[17] Layoutregionen geben dem Programmierer die Möglichkeit, Masken und Teilmasken grafisch umfangreicher und anspruchsvoller zu gestalten. Die Layoutregion ist ein vordefinierter Gestaltungsbereich, der in eine Maske oder Teilmaske eingefügt und in ihr gespeichert wird. Sie sind keine eigenständigen Gestaltungselemente und müssen immer in eine Maske oder Teilmaske eingebunden werden.

2.2 Das W3C

Das WWW-Konsortium (W3C) wird vom Laboratory for Computer Science am Massachusetts Institute of Technology (MIT) verwaltet. Seine Aufgabe ist die Ausarbeitung einheitlicher Normen für die Weiterentwicklung des World Wide Web.
Es handelt sich dabei um eine gemeinsame Initiative von MIT, CERN[18] und INRIA[19].
Das US-amerikanische W3C-Zentrum befindet sich am MIT und wird auch von diesem geführt. Das europäische W3C-Zentrum ist am INRIA, dem französischen Institut für Entwicklungen in Computertechnik und Automation, angesiedelt.

Das W3C wurde gegründet, um die gemeinsame Ausarbeitung von Normen für die Entwicklung von Web-Technologien zu unterstützen. Eines der wichtigsten Ziele des W3C ist es, Web-Entwicklern und -Nutzern eine gemeinsame Informationsablagestelle (Global Information Repository) bereitzustellen. Zu diesem Zweck führt das W3C mehrere Web-Sites, auf denen die neuesten Entwürfe in Bezug auf Web-Entwicklungen veröffentlicht werden.

Ein weiteres Ziel des W3C ist die Bereitstellung von Prototyp-Anwendungen, die auf neuen Technologien basieren und in Form von Internet-Entwürfen vorgeschlagen werden.

Bei der Ausarbeitung von Spezifikationen und Normen, die der IETF[20] unterbreitet werden, arbeitet das W3C eng mit seinen Mitgliedsorganisationen zusammen.
Diese Mitgliedsorganisationen zahlen eine Gebühr, die sich nach dem Mitgliedsstatus richtet.

[18] Centre Europeen de Recherches Nucléaires

[19] Institut de Recherche en Informatique et en Automatique

[20] Internet Engineering Task Force

Derzeit zählen zum W3C große Unternehmen wie AT&T, Adobe Systems, America Online, CompuServe, Hewlett Packard, IBM, Lotus Development Corporation, SAP AG, Microsoft Corporation, Mitsubishi Electric Corporation, Deutsche Telekom AG, NEC Corporation, Netscape Communications Corporation, Novell, Oracle Corporation, Silicon Graphics und Sun Microsystems aber auch das DFN (Deutsches Forschungsnetz e.V.) und die Universität Karlsruhe.

2.3 Auszeichnungssprachen

In einer Auszeichnungssprache (Markup-Language) werden verschiedene Befehle definiert, um Informationen optisch und inhaltlich zu strukturieren. Auf dieser Basis können dann Dokumente verfasst und verarbeitet werden.

Es ergeben sich mehrere Hauptprobleme, die eine Auszeichnungssprache lösen muss:

- Weltweit existieren Hunderte von verschiedenen länderspezifischen Zeichen. Zur Darstellung der Zeichen stehen allerdings nur Zeichensätze mit 128 Zeichen (z.B. US-Standard) oder erweiterte Zeichensätze mit 255 Zeichen zur Verfügung. Diese reichen aber bei weitem nicht für die Darstellung der Hunderten verschiedenen Schriftzeichen des z.B. japanischen oder chinesischen Alphabets aus. Es müssen Möglichkeiten gefunden werden, diese Zeichen darzustellen.
- Reine Textdateien reichen zur angemessenen Darstellung von Informationen kaum aus. In ihnen können keine Schriftarten, Überschriften, Fußnoten, Kursiv- oder Fettschrift verwendet werden.
- Daher müssen Möglichkeiten gefunden werden, Dokumente sowohl inhaltlich als auch visuell zu gestalten.
- Außerdem müssen sie Möglichkeiten bieten, Text, Grafik und andere multimediale Elemente darzustellen.

Die Lösung dieser Probleme erfolgt durch die Einführung von vorher festgelegten Auszeichnungen (Markups). Z.B. wird in HTML mit der Auszeichnung <B> definiert, dass der Text in Fettschrift darzustellen ist.

<B>Text in Fettschrift</B>

In HTML hat sich allerdings ein anderer Begriff für Textauszeichnungen durchgesetzt. Diese werden als „Tag" bezeichnet. Die Konvention für Tags besagt, dass der auszuzeichnende Text von einem „Start-„ und „Schluss-Tag" eingeschlossen wird. Allerdings wird eine unkorrekte Verschachtelung der Tags oder ein fehlendes Schluss-Tag oft toleriert.
XML hat in diesem Zusammenhang strengere Konventionen als HTML. Anders als bei HTML muss jedes XML-Tag auch wieder ordnungsgemäß geschlossen werden. Die größere Flexibilität von XML bedingt gleichzeitig eine höhere Genauigkeit in der Syntax der Befehle.

Grundsätzlich wird zwischen zwei verschiedenen Arten von Auszeichnungen unterschieden:

- logische Auszeichnungen
- physische Auszeichnungen

2.3.1 Logische Auszeichnungen

Hierbei handelt es sich um eine inhaltliche Definition des Textes. Bei dem gekennzeichneten Begriff kann man nun festlegen, ob es sich z.B. um eine wichtige Überschrift, eine Fußnote, einen Namen oder eine URL handelt.
HTML kennt einige logische Auszeichnungen (META-Tags), die eine spätere Auswertung des Textes durch z.B. Suchmaschinen erleichtern.
Das folgende Beispiel gibt den Autor eines Dokuments an:

<AUTHOR>John Doe</AUTHOR>

Dieser logisch ausgezeichnete Text impliziert nicht unbedingt eine eigene visuelle Darstellung. Ein Name kann beispielsweise in gleicher Textart und Schriftstärke dargestellt werden wie eine Adresse.
Hauptsächlich verwendet man die logische Auszeichnung dazu, Informationen und deren Strukturen durch EDV-Programme auswertbar zu machen.
Suchmaschinen im Internet könnten dann z.B. gezielt nach Titeln oder Namen suchen. Dadurch wird die Qualität der Hits[21] bei einem Suchvorgang verbessert.

2.3.2 Physische Auszeichnungen

Physische Auszeichnungen ermöglichen es dem Entwickler, Texte visuell zu beschreiben.
Ein als kursiv (Italic) und in Fettschrift (Bold) definierter Text wird auch in dieser Schrift dargestellt.

Beispiel: Dieser Text ist fett und kursiv

Bei der physischen Auszeichnung ergibt sich das Problem der Kompatibilität zu anderen Systemen.
Werden Daten ausgetauscht, kann es vorkommen, dass die Darstellung anders erfolgt als ursprünglich vorgegeben. So wird ein sauber auf das europäische Papierformat DIN A4 ausgerichtetes Dokument anders aussehen, wenn ein Amerikaner dieses auf seinem etwas kürzeren US-Letter-Format ausdruckt.
Auch können Hardware-Unterschiede das Ergebnis auf dem Bildschirm verändern. Bildschirmgrößen, -auflösungen und die unterstützte Anzahl der Farben wurden und werden wohl auch nicht standardisiert werden.

[21] Anzahl der Treffer zu einem Suchbegriff

2.3.3 Semantische Auszeichnungen

Zusätzlich zu den logischen und physischen Auszeichnungen hat sich im Bereich der Internet-Entwicklergemeinde noch ein dritter Begriff durchgesetzt, der für XML eine große Bedeutung besitzt.
Die „semantischen Markups“ beschreiben weder das Layout noch die logische Struktur, sondern sie erlauben Rückschlüsse auf den Inhalt des zwischen den Markups stehenden Textes.

Beispiel: <TELEFONNUMMER>02241123456</TELEFONNUMMER>

Einer späteren Anwendung wird dadurch die Möglichkeit eröffnet, die entsprechenden Felder auszuwerten oder nach einer bestimmten Telefonnummer zu suchen.
In XML werden fast ausschließlich Markups dieses Typs verwendet. [Pott 1999]

2.4 Chronologische Entwicklung der heute eingesetzten Auszeichnungssprachen

Folgende Tabelle stellt die bisherige zeitliche Entwicklung der bis heute eingesetzten Auszeichnungssprachen dar:

Jahr	Bezeichnung	Abkürzung	Beschreibung
1950	Hypertext		Als Theorie entwickelt von Ted Nelson
1969	Generalized Markup Language		Von IBM entwickelt
1986	Standard Generalized Markup Language	(SGML)	Festgelegt im ISO Standard 8879
1989	Hyper Text Markup Language	(HTML)	Entwickelt von Tim Berners-Lee im CERN in Genf
1994	Hyper Text Markup Language 2.0	(HTML 2.0)	Erstmals unter Leitung des W3C als Standard verabschiedet
1994	Cascading Style Sheets 1.0	(CSS 1.0)	Als Ergänzung zu HTML verabschiedet
1996	Hyper Text Markup Language 3.2	(HTML 3.2)	Verabschiedet, nachdem es zu Version 3.0 keine Einigung gab
1996	Extensible Markup Language 1.0	(XML 1.0)	Zunächst als Diskussionsvorschlag verabschiedet
1997	Hyper Text Markup Language 4.0	(HTML 4.0)	Im Dezember als Richtlinie vom W3C verabschiedet
1998	Cascading Style Sheets 2.0	(CSS 2.0)	Weiterentwicklung des bestehenden CSS-Standards
1998	Extensible Markup Language 1.0	(XML 1.0)	Als Standard vom W3C beschlossen
1998	Extensible Style Language 1.0	(XSL 1.0)	Im August vorerst als Arbeitsvorschlag zur Diskussion gestellt
2000	Extensible Hyper Text Markup Language 1.0	(XHTML 1.0)	Im Januar als Richtlinie vom W3C verabschiedet

[Pott 1999, S.31]

Tabelle 1: Chronologische Entwicklung

2.5 Was ist DOM?

Das „**D**ocument **O**bject **M**odel“ ist, wie der Name schon sagt, ein Objektmodell, das den Zweck hat, Dokumente darzustellen, d.h. Dokumente werden durch Objekte im objektorientierten Sinn dargestellt. Das DOM wurde vor allem in Hinblick auf Anwendungen für das World Wide Web entwickelt und deshalb sind die unterstützten Dokumenttypen beliebige XML- oder HTML-Dokumente.
Weitere Zielsetzungen der DOM Spezifikation sind Unabhängigkeit von Plattform und Programmiersprache.
Mögliche Plattformen sind beliebige Betriebssystem-Umgebungen oder virtuelle Umgebungen, wie Browser oder eine Java Virtual Machine. Hieraus ergeben sich als mögliche Sprachen sowohl Programmiersprachen im herkömmlichen Sinn, als auch Skriptsprachen, wie z.B. JavaScript, Java, C++ oder VBScript.

Das DOM bietet in der Basisversion standardisierte Schnittstellen, um die Elementstruktur von XML- oder HTML-Dokumenten auszulesen und zu manipulieren. Eine Erweiterung des DOM sieht außerdem den Zugriff auf das Layout von Dokumenten und ein Event-Modell vor.

2.5.1 Grundlagen zu DOM

2.5.1.1 Standardisierung

Die Entwicklung des DOM wird von der DOM Working Group geleistet, einer Unterabteilung des W3C. Beteiligte Organisationen sind beispielsweise:

- IBM
- Javasoft
- Microsoft
- Netscape
- Novell
- Sun

2.5.1.2 DHTML

Die Entstehung des DOM hat ihren Ursprung in der Entwicklung von Dynamic-HTML (DHTML), das die Nachteile der eingeschränkten Gestaltungs- und Layout-Möglichkeiten von HTML beheben sollte. DHTML beschreibt die Art und Weise, wie in HTML-Seiten eingebettete Skripte arbeiten. Neben dem Zugriff auf die Elementstruktur der HTML-Dokumente ist die Manipulation von Layout-Eigenschaften und die Reaktion auf Events möglich.
Dadurch wird es möglich, Internet-Seiten interaktiver zu gestalten, denn die Inhalte passen sich flexibel den Bedürfnissen des Nutzers an und reagieren auf dessen Eingaben. Dieses wird in erster Linie durch den Einsatz von JavaScript, VisualBasicScript (VBScript) oder ActiveX erreicht.
Der Unterschied zu HTML liegt nun darin, dass nicht für jede Änderung des Inhalts eine neue Seite vom Server angefordert und übertragen werden muss, sondern die vorhandene Seite wird durch den Browser selbst verändert.

Diese Eigenschaften von DHTML treffen auch auf die Sitemap dieser Diplomarbeit zu. Durch den Einsatz von JavaScript-Funktionen in der Sitemap kann der Benutzer die angezeigten Inhalte gezielt steuern. Außerdem findet nur eine einmalige Kommunikation zwischen Server und Browser zwecks Datenübertragung statt. Ist diese abgeschlossen, sind alle benötigten Daten im Browser vorhanden. Änderungen im angezeigten Inhalt des Browsers erfordern dann keine weiteren Server-Zugriffe mehr. In den Kapiteln „3.2.4.4 JavaScript“ und „3.3 Logischer Aufbau der Sitemap und deren Komponenten“ wird später genauer auf dieses Thema eingegangen

2.5.1.3 Dokumentenmanagement

Um die Darstellbarkeit von Web-Inhalten auf Browsern verschiedener Hersteller zu gewährleisten und um gleichzeitig mit der Weiterentwicklung von Web-Technologien über HTML hinaus Schritt zu halten, ist ein allgemein akzeptierter Standard notwendig. Diese Aufgabe erfüllt das DOM.

Die Portabilität zwischen verschiedenen Browsern umfasst folgende Bereiche:

- Skriptsprachen
- Java-Applets
- Stylesheets
- Events

Weitere Anwendungsgebiete für das DOM sind der Einsatz in HTML-/XML-Editoren und in Werkzeugen für die Dokumenten-Konvertierung.

2.5.1.4 Rahmen der Spezifikation

Die DOM-Spezifikation legt genau folgendes fest:

- Ein abstraktes Objektmodell, d.h.
  - Arten von Objekten,
  - ihre Interfaces und
  - die Semantik dieser Interfaces.

Es wird also lediglich festgelegt, wie ein DOM-Client mit einer DOM-Implementierung über Interfaces kommuniziert:

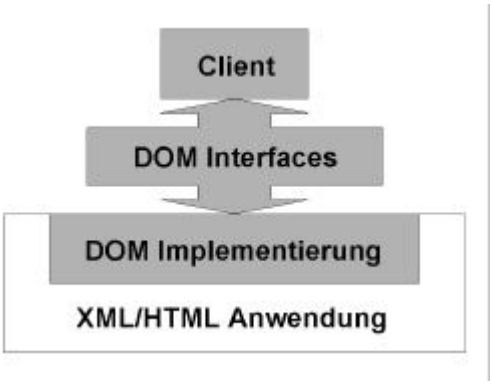


Abbildung 1: DOM Kommunikation

Folgendes wird dagegen nicht festgelegt:

- wie die XML-/HTML-Anwendung die Dokumenten-Objekte bereitstellt, d.h. wie der Client die erste Referenz auf ein Dokumenten Objekt erhält
- ob die Anwendung eine Gültigkeitsprüfung für XML-/HTML-Dokumente durchführt
- wie und ob die Anwendung die Dokumente serialisiert, d.h. in eine Zeichenkette umwandelt
- wie die Anwendung die Objekte im Speicher verwaltet [Helfrich 1999]

2.6 Grundlagen zu XML

2.6.1 XML

XML (E**x**tensible **M**arkup **L**anguage) ist eine erweitere Auszeichnungs-Sprache, die es dem Programmierer erlaubt, eigene Markups zu definieren.
Somit ist es möglich, beliebige Datenstrukturen zu erzeugen, die in semantischen Markups eingeschlossen sind.

Ein anschauliches Beispiel ist z.B. eine Adressenliste.

```
<?xml version="1.0"?>
<ADRESSENLISTE>

    <ADRESSE>
        <NAME>Gimbel</NAME>
        <VORNAME>Holger</VORNAME>
        <STRASSE>Nordring 24</STRASSE>
        <PLZ>51647</PLZ>
        <ORT>Gummersbach</ORT>
    </ADRESSE>

    <ADRESSE>
        .....
    </ADRESSE

</ADRESSENLISTE>
```

Das umschließende Markup <ADRESSENLISTE> bildet hierbei das Wurzelelement des Dokuments. Die Adressen sind durch das Element <ADRESSE> umschlossen, das die Daten mit den entsprechenden Markups <NAME>, <VORNAME>, STRASSE>, <PLZ> und <ORT> enthält.

XML ist somit lediglich als Datencontainer zu verstehen, der die später zu visualisierenden Daten aufnimmt.

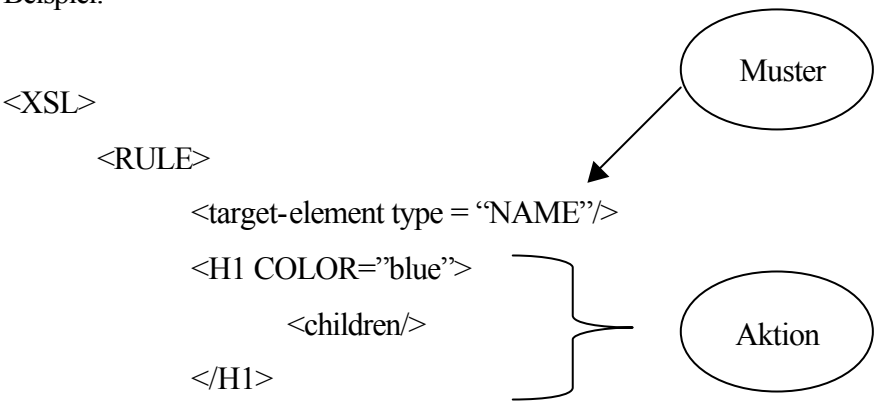
2.6.2 XSL

Für die Visualisierung der in XML vorliegenden Daten ist hauptsächlich die Sprache XSL (E**x**tensible **S**tylesheet **L**anguage) zuständig. Sie ist sozusagen das Salz in der Suppe, denn ihre Aufgaben liegen vor allem in zwei Bereichen:

- Erzeugung einer Sprache, die die XML-Dokumente in andere Formate transformiert (z.B. HTML)
- Bereitstellung eines Vokabulars, das, ähnlich zu den bestehenden CSS (Cascading Stylesheets), den semantischen Markups bestimmte Formatierungen zuweist.

Diese Formatierungen werden durch Konstruktionsregeln (construction rules) erreicht, die sich aus einem Muster und der dazugehörigen Aktion zusammensetzen. Das Muster legt dabei fest, auf welches XML-Element sich die darauffolgende Stilanweisung (Aktion) bezieht. Wurde das durch das Muster beschriebene XML-Element erkannt, so wird dieses durch die Stilanweisung (Aktion) in die angegebene Ausgabeform umgewandelt.

Beispiel:



Hier wurde eine einfache Regel für ein XML-Element erzeugt.
Trifft der XSL-Prozessor in dem vorliegenden XML-Dokument auf ein „NAME"-Element, so erscheint dieses direkt als H1 Überschrift in der Farbe Blau im HTML-Code. XSL bestimmt also das Layout einer Seite und wandelt XML-Dokumente beispielsweise in HTML-fähige Konstrukte um. Allerdings werden nicht automatisch vollständige HTML-Dokumente inklusive der notwendigen Grundstuktur erzeugt. Zusätzlich zu den Regeln müssen noch Tags wie <HTML>, <HEAD> und <BODY> im Stylesheet definiert werden. Im Kapitel „3.2.4.2 Sitmap.xsl" wird diese Vorgehensweise genauer erklärt.

2.6.3 DTD

Die DTD (**D**ocument **T**ype **D**efinition) ist ein zentraler Bestandteil jedes XML-Dokuments.
In ihr werden die Grundregeln für den Aufbau der XML-Daten beschrieben. Außerdem werden in der DTD alle einsetzbaren Markups und deren Attribute definiert. Zusätzlich kann beschrieben werden, wie und in welcher Häufigkeit die einzelnen Elemente ineinander verschachtelt werden dürfen.

Die DTD für das vorherige Adressen-Beispiel würde in etwa so aussehen:

```
<!ELEMENT  ADRESSENLISTE (ADRESSE)*>
<!ELEMENT ADRESSE (NAME, VORNAME, STRASSE, PLZ, ORT)
<!ELEMENT NAME          #PCDATA>
<!ELEMENT VORNAME    #PCDATA>
<!ELEMENT STRASSE      #PCDATA>
<!ELEMENT PLZ           #PCDATA>
<!ELEMENT ORT           #PCDATA>
```

Diese besagt, dass eine Adressenliste Adressen aufnehmen kann. Das Zeichen"*" gibt an, dass eine Adresse beliebig oft (>= 0) in der Adressenliste vorkommen darf.

Die Adresse wiederum enthält je ein Element Name, Vorname, Strasse, Postleitzahl und Ort. Anschließend wird der Datentyp der Elemente definiert. Dieser kann beliebiger Art sein, also auch Multimedia-Files, Bilder, Sounds, PDFs[22] oder nur einfacher Text, wie in diesem Fall.

An dieser Stelle ist es notwendig, noch zwei Begriffe einzuführen, die einem immer wieder begegnen, wenn es um die korrekte Umsetzung der W3C-Spezifikation zu XML geht.

- Wohlgeformtheit
- Gültigkeit

2.6.3.1 Wohlgeformtheit

Ein XML-Dokument ist unter folgenden Bedingungen wohlgeformt:

- Das gesamte Dokument muss ein umschließendes Wurzelelement besitzen (<ADRESSENLISTE> und </ADRESSENLISTE>).
- Alle unbedingt erforderlichen Attribute werden angegeben.
- Die Werte der Attribute entsprechen dem angegebenen Typ und befinden sich im korrekten Wertebereich.

Das XML-Dokument muss also nicht zwingend eine DTD besitzen, um wohlgeformt zu sein.

2.6.3.2 Gültigkeit

Die Überprüfung der Gültigkeit des XML-Dokuments ist neben der Wohlgeformtheit ein weiterer wichtiger Aspekt. Dabei sind drei Kriterien entscheidend:

[22] Portable Document Format (wird von Adobe Acrobat verwendet)

- Es existiert eine DTD (intern oder extern) und sie ist verfügbar.
- Das Dokument entspricht den in der DTD aufgestellten Regeln.
- Das Dokument ist wohlgeformt.

Entsprechend der W3C-Spezifikation zu XML können also XML-Dokumente ohne DTD nicht gültig sein. Dies muss allerdings nicht bedeuten, dass der Browser diese nicht mehr darstellen kann, da eine Überprüfung auf Gültigkeit nicht zwingend durchgeführt werden muss. Dies hängt davon ab, ob ein validierender[23] oder nicht validierender Prozessor (Parser) eingesetzt wird. Der MS-Internet Explorer 5 benutzt beispielsweise einen nicht validierenden Parser.
Es ist aber auf jeden Fall sinnvoll, eine DTD zu erstellen, da besonders bei komplexen Strukturen die Übersichtlichkeit und Wartung verbessert wird. Welche Attribute ein Markup enthält und welche zwingend beschrieben werden müssen, ist dann leicht ablesbar.

2.7 Transformation von XML und XSL zu HTML

Möchte man sich nun die Adressenliste in einem Internet-Browser ansehen, so müssen die Daten z.B. in HTML transformiert werden.

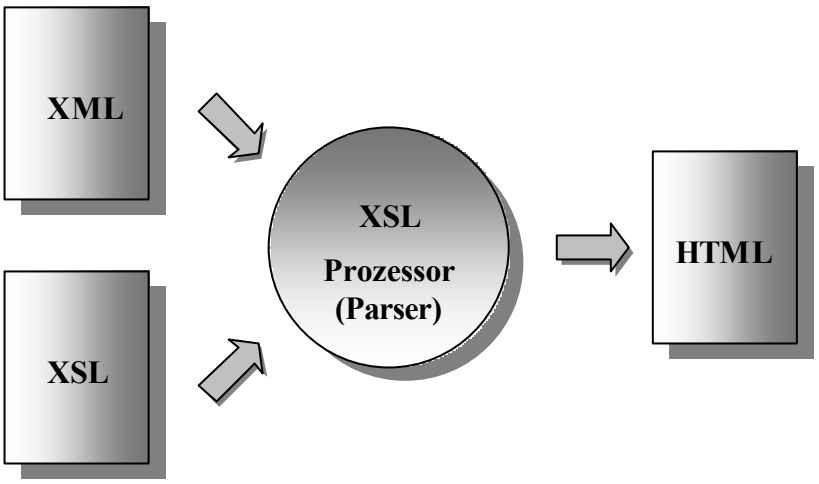


Abbildung 2 : Transformation von XML zu HTML

[23] validieren = überprüfen

Hierzu benötigt man einen XSL-Prozessor (Parser) der den HTML-Code erstellt.

Der Inhalt des neu generierten HTML-Dokuments entstammt den jeweiligen XML-Daten, die visuelle Umsetzung ist in dem zugehörigen XSL-Stylesheet festgelegt.

Es bestehen zwei Möglichkeiten, den Parser zu verwenden:

- serverseitige Integration :
  Die Transformation findet auf dem Server statt, der dann dem Internet-Browser den fertigen HTML-Code sendet. Der Nachteil dabei ist allerdings, dass die Transformation zu Lasten der Serverperformance geht.

- Integration im Browser:
  Ist der Parser im Browser integriert, so schickt der Server die benötigten Dateien, die dann auf dem Client-Rechner transformiert werden.
  Hierzu ist zur Zeit der MS-Internet Explorer 5 und eingeschränkt der Netscape-Communicator 6 bzw. Mozilla in der Lage. Netscape und Mozilla unterstützen momentan nur XML und CSS.

2.8 Wo kann XML eingesetzt werden?

XML kann man als Ausgangssprache für die Weiterverarbeitung ansehen. Die XML-Daten müssen dann den entsprechenden Erfordernissen angepasst werden.
Ein WAP[24]-Handy oder ein PDA[25] ist z.B. nicht in der Lage, größere Datenmengen auf einer Bildschirmseite darzustellen, da das Display ziemlich klein ist. Hier müssen aus den XML-Daten die wichtigsten Informationen herausgefiltert werden, die schließlich auf dem Display erscheinen sollen. Durch die Möglichkeit, einem XML-Dokument eine optionale Beschreibung einer

[24] Wireless Application Protocol

[25] Personal Digital Assistent

Grammatik zuweisen zu können, kann dieses erreicht werden. Eine Applikation kann daraufhin eine strukturelle Überprüfung der Daten durchführen.

Durch diese Flexibilität ist XML prädestiniert als Basis-Format zur Datenspeicherung. Aus diesen Daten können dann fast beliebige andere Dokumenttypen erzeugt werden.

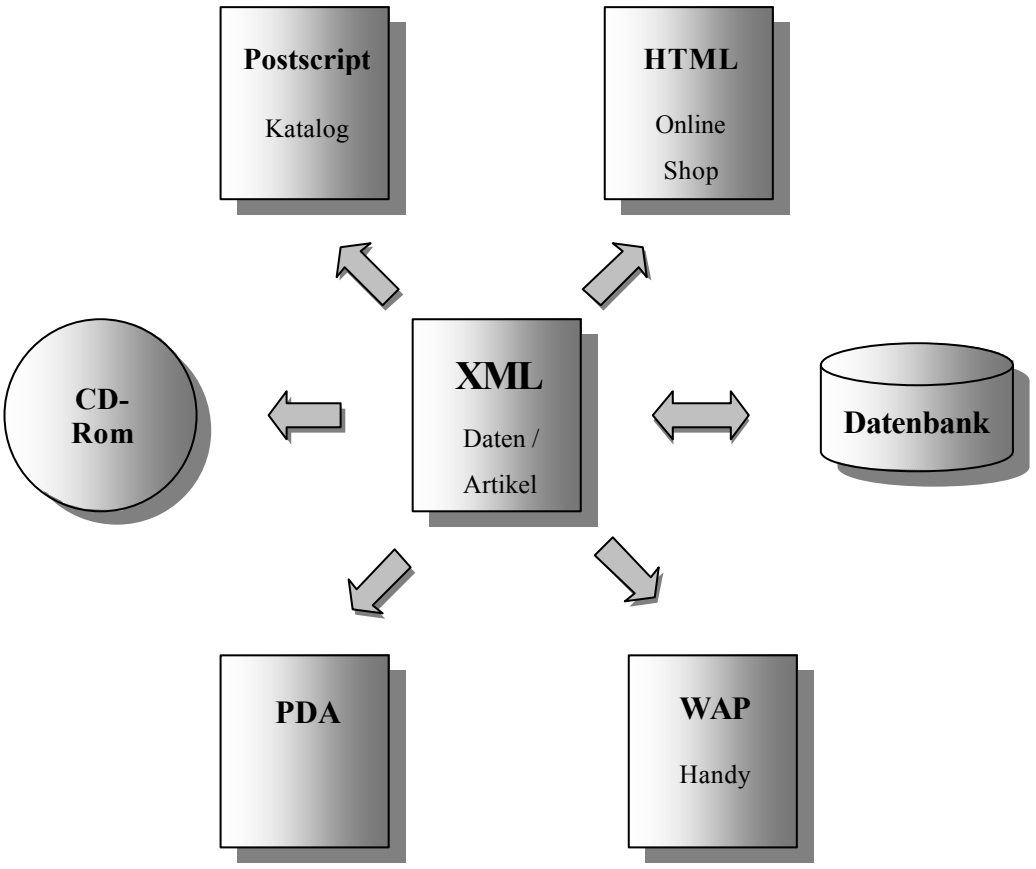


Abbildung 3: Einsatzmöglichkeiten von XML

Ein Vorteil dabei ist, dass die Daten nur einmal erzeugt werden müssen. Einem Unternehmen ist es dann beispielsweise möglich, aufbauend auf den gleichen Daten ihren Online-Katalog, CD-ROM-Katalog oder gedruckten Katalog zu erstellen. Die Doppelarbeit der Konvertierung der Daten fällt dadurch weg.

2.9 Vor- und Nachteile von XML und XSL gegenüber HTML

Vergleicht man die Länge des Codes der XML-Daten und des XSL-Stylesheets mit dem erzeugten HTML-Code, wird man feststellen, dass der HTML-Code kürzer ist. Man kann sich nun fragen, ob sich der Mehraufwand für XML und XSL überhaupt lohnt, denn man kann offensichtlich in HTML viel leichter ansprechende Ergebnisse erzielen. Jedoch bietet XML in Verbindung mit XSL gegenüber HTML mehrere Vorteile:

Ein wesentlicher Vorteil von XML und XSL gegenüber HTML liegt in der Übersichtlichkeit der XML-Daten und des XSL-Stylesheets. Hierbei wird die Strukturierung durch XML strikt von der Visualisierung durch XSL getrennt.
HTML ist sowohl für die Strukturierung als auch für die Visualisierung der Daten zuständig.
Durch die Trennung der Daten von der Visualisierung wird es bedeutend einfacher, die XML-Daten zu warten. Es ist leicht möglich, neue Tags bzw. Daten in das bestehende XML-Dokument hinzuzufügen.
Das gleiche gilt für XSL: Möchte man ein bestehendes Tag mit einer anderen Formatierung versehen (für das Tag <NAME> soll z.B. <H2> anstatt <H1> verwendet werden), so müssten in dem HTML-Code für die Namen alle <H1>-Tags durch <H2>-Tags ersetzt werden. In der XSL-Datei muss lediglich eine Zeile geändert werden, in der die Regel für das „NAME“-Element beschrieben ist.

Ein weiterer großer Vorteil von XML liegt in der Auswertbarkeit der Daten:
Nach der Transformation (im Browser) der XML-Daten und des XSL-Stylesheets zu HTML gehen die reinen XML-Daten nicht verloren. Lässt man sich beispielsweise im Browser den Quelltext der generierten XML-Sitemap anzeigen, so erscheint nicht der transformierte HTML-Code, sondern es wird der vollständige XML-Code, der dynamisch generiert wurde, angezeigt. Dieser kann nun bei Bedarf anhand einer Abfragesprache ausgewertet werden. Hierfür wird zur Zeit die Abfragesprache XQL (**X**ML **Q**uery **L**anguage) entwickelt, mit der es möglich ist, die XML-Daten anhand von Suchbegriffen abzufragen.

Dadurch wird es möglich sein, Webauftritte, sofern sie mit XML erzeugt wurden, auszuwerten. Informationen können von einer Applikation in einer Seite gezielt gefunden und entsprechend bearbeitet werden.

Findet die Transformation von XML und XSL zu HTML auf dem Server statt, so gehen die reinen XML-Daten verloren, da nur der fertige HTML-Code zum Browser übertragen wird. Deshalb ist die Transformation im Browser zu bevorzugen, wenn die Auswertbarkeit der Daten erwünscht ist. Möchte man dies vermeiden, sollte die Transformation auf dem Server stattfinden.

Wie vorher schon erwähnt, wird der XML-Code als Quelltext angezeigt, wenn die Transformation zu HTML im Browser stattfindet. Möchte man sich bei dieser Vorgehensweise den HTML-Code anschauen, kann im XSL-Stylesheet ein zusätzlicher Button eingefügt werden, durch dessen Betätigung dann der HTML-Code in einem Fenster angezeigt wird:

```
<BUTTON ONCLICK="window.alert(document.body.innerHTML);">
HTML anzeigen </BUTTON>
```

Der Nachteil bei der Transformation im Browser ist, dass dem Endbenutzer ein vollständiger Einblick sowohl auf die Datenbasis als auch auf die Realisierung der Visualisierung ermöglicht wird:
Sind in der XML-Datenbasis sensible Daten vorhanden, die durch das XSL-Stylesheet herausgefiltert werden, so sind diese trotzdem im Quelltext sichtbar.
Außerdem werden alle zusätzlich geladenen Dateien, wie z.B. die Stylesheets (XSL und CSS), die Document Type Definition (DTD) oder JavaScripts (JS), im File-Cache-Verzeichnis des Browsers gespeichert. Der Endanwender kann dadurch vollständig nachvollziehen, wie die dargestellte Seite erzeugt wurde.

2.10 XHTML im Vergleich zu XML, HTML und DHTML

HTML kann auch mit XML definiert werden. Hierzu erstellt man eine XML-DTD, die die vorhandenen Tags und Attribute von HTML übernimmt und ihre Bedeutung anhand des existierenden HTML-4-Standards beschreibt.
Dieser Weg wurde bei der Weiterentwicklung von HTML gegangen. Das Ergebnis ist eine Empfehlung des W3C zu XHTML, die unter „http://www.w3.org/TR/xhtml1“ eingesehen werden kann.

XHTML steht für „E**x**tensible **H**yper**t**ext **M**arkup **L**anguage“. Hierbei handelt es sich um die Neudefinition von HTML 4.0 in XML 1.0. Diese neue Fassung erlaubt dem Entwickler, wie bei XML, die Einführung von Erweiterungen bzw. neuen Tags. Die DTDs, die hierzu von XHTML bereitgestellt werden, korrespondieren mit denen von HTML und sind abwärtskompatibel.

Der erste Unterschied zu HTML besteht darin, dass am Anfang eines jeden XHTML-Dokuments ein <!DOCTYPE>-Tag und der Namensraum angegeben werden muss.

Zu XHTML existieren drei DTD´s, in denen die Syntax der Sprache spezifiziert wurde:

- Strict
- Transitional
- Frameset

Strict:

Der Typ „Strict“ wird insbesondere dann benutzt, wenn Cascading Stylesheets (CSS) verwendet werden.

Die Anweisung für diese DTD lautet wie folgt:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" http://www.w3.org/TR/xhtml1/DTD/strict.dtd>

Darauf folgt die Namensraum-Definition[26]:

<html xmlns=http://www.w3.org/TR/xhtml1>

Transitional:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" http://www.w3.org/TR/xhtml1/DTD/transitional.dtd>

<html xmlns=http://www.w3.org/TR/xhtml1>

Diese DTD wird verwendet, wenn die Zielgruppe, also der Endanwender, einen älteren Browser verwendet, der z.B. CSS nicht unterstützt.

Frameset:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" http://www.w3.org/TR/xhtml1/DTD/frameset.dtd>

<html xmlns=http://www.w3.org/TR/xhtml1>

Die Frameset-DTD wird vor allem dann eingesetzt, wenn für die Inhaltsgestaltung Frames verwendet werden.

[26] Vgl. Kapitel „3.2.4.2.1 Exkurs: Namensräume"

Ein einfaches XHTML-Dokument würde wie folgt aussehen:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
http://www.w3.org/TR/xhtml1/DTD/strict.dtd>

<html xmlns=http://www.w3.org/TR/xhtml1>

    <head>
        <title>XHTML</title>
    </head>
    <body>
        <p>Dies ist ein einfaches XHTML-Dokument></p>
    </body>
</html>
```

Der Web-Designer muss bei XHTML im Gegensatz zu HTML darauf achten, dass

- zu jedem Tag auch ein Abschluss-Tag vorhanden ist.
- die Tags korrekt verschachtelt werden (Wohlgeformtheit).
- für Element- und Attributnamen Kleinschreibung verwendet werden muss, da XML zwischen Groß- und Kleinschreibung unterscheidet.
- der Wert eines Attributs innerhalb eines Tags in Anführungszeichen gesetzt wird. (Bsp.:)
- leere Tags durch ein End-Tag in sich selbst beendet werden (Bsp.:
)
- die Verwendung von Scripten und eingebetteten Stylesheets innerhalb von XHTML durch das <script> bzw. <style>-Tag eingeleitet und innerhalb der CDATA-Sektion eingefügt wird

Bsp.:

```
<script>
<![CDATA[
// Hier folgt das Script
]]>
</script>

<style>
<![CDATA[
H1.Ueberschrift     { color:red;}
]]>
</style>
```

Zeichendaten, die innerhalb des CDATA-Abschnitts eingebettet sind, werden durch den Parser direkt ohne Auswertung an die Anwendung weitergeleitet. Hier könnten beispielsweise JavaScripte untergebracht werden, die die Funktionalität der Seite steuern.
Würden diese Zeichendaten ausgewertet, käme es zu Fehlermeldungen, da bestimmte Zeichen unterschiedlich interpretiert werden. Das Zeichen „<“ wird dann beispielsweise als Beginn eines Tags und „&“ als Beginn einer Entity interpretiert.

Wie in Kapitel „2.5.1.2 DHTML“ beschrieben, können durch die Verwendung von Scriptsprachen und Stylesheets innerhalb von HTML-Dokumenten, die Layout-Eigenschaften manipuliert werden. Außerdem ist eine Reaktion auf Events möglich. Der sich daraus ergebene Dokument-Typ wird als DHTML bezeichnet. Dadurch, dass es in XHTML ebenfalls möglich ist, Scripte und Cascading Stylesheets (CSS) einzusetzen, kann man sagen, dass XHTML ebenfalls dynamisch wird. Allerdings ändert sich dadurch die Bezeichnung der Auszeichnungssprache nicht. Man könnte sagen, dass es sich hierbei um

DXHTML (**D**ynamic E**x**tensible **H**yper**t**ext **M**arkup **L**anguage) handelt, wenngleich die Bezeichnung keinem offiziellen Standard entspricht.

XHTML hat gegenüber HTML verschiedene langfristige Vorteile:

Durch die Verwendung einer XML-DTD für HTML können alle Tools eingesetzt werden, die auf XML basieren. Diese können ohne Änderungen auch auf XHTML-Dokumente angewandt werden.
Die Standardisierung auf XML hat den Vorteil, dass die Hersteller von Browsern und anderer Software dazu gezwungen sind, die strengen Regeln für die Wohlgeformtheit und Gültigkeit von XML-Dokumenten zu beachten.
Dadurch werden langfristig solche Dokumente verdrängt, die den HTML-Regeln nicht entsprechen, aber trotzdem vom Browser toleriert werden. Dies wird wahrscheinlich zu schlankeren Browsern führen, da nicht mehr alle möglichen Ausnahmen behandelt werden müssen. Diese schlanken Browser sind aber auch nötig, da in Zukunft vermehrt kleinere Geräte, wie Handys und PDAs Zugriff auf das Internet nehmen.

Kapitel 3: XML und Lotus Notes

Da im Rahmen dieser Diplomarbeit ein möglichst vollständiger Überblick über die Möglichkeiten von XML unter Lotus Notes gegeben werden soll, werden in den folgenden Abschnitten zwei Möglichkeiten beschrieben, wie man XML unter Lotus Notes einsetzen kann.

Beide Möglichkeiten unterscheiden sich insofern voneinander, dass die Erste die Daten im XML-Format in der Notes-Datenbank speichert und für die zweite Möglichkeit die Daten im herkömmlichen Format in der Notes-Datenbank vorhanden sein können.
Für die Erzeugung der Sitemap mittels XML und XSL ist allerdings nur die zweite Vorgehensweise von Bedeutung, da die WebGate-Datenbank, in der die Sitemap integriert wurde, im bisherigen, üblichen Format vorlag.

3.1 Visualisierung mit Masken, Ansichten und Seiten

Die Basis dieses Abschnitts bildet der Online-Artikel von Kyla Town[27] „Exercising XML with Domino Designer“ [Kyla 2000]

Bei der dem Artikel beiliegenden Beispiel-Datenbank handelt es sich um einen Online-Katalog eines fiktiven Autoteileherstellers. In dieser Datenbank wird eine Möglichkeit vorgestellt, wie XML in einer Domino-Applikation integriert werden kann.

[27] Kyla Town gehört dem Lotus-Dokumentations-Team für den Domino-Designer an. Am 01.03.2000 wurde der von ihr verfasste Artikel im Notes.net veröffentlicht, um eine Möglichkeit vorzustellen, wie XML in einer Domino-Applikation integriert werden kann.

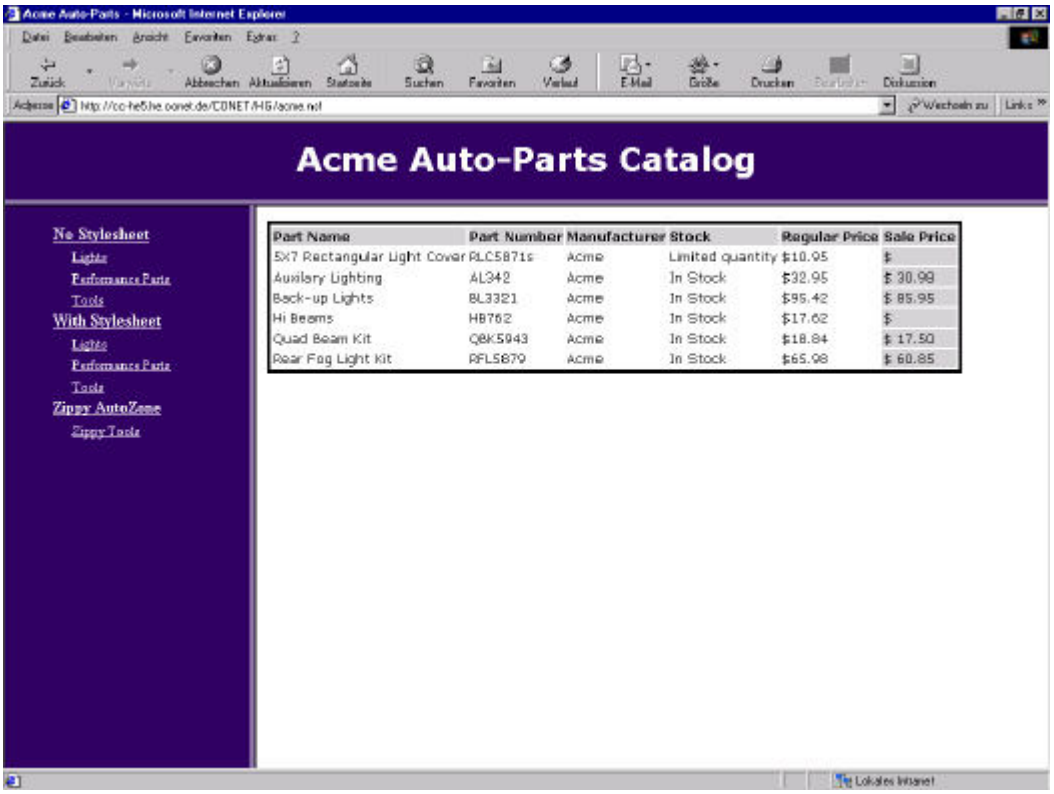


Abbildung 4: XML-Artikelliste

Abbildung 4 zeigt einen Screenshot des Online-Katalogs, bei dem die Artikelliste mit XML und XSL erzeugt wurde.

Im Folgenden werden einige Screenshots von verschiedenen Design-Komponenten des Notes-Designers zu der dem Artikel beigefügten Notes-Datenbank beschrieben, wodurch das Prinzip dieser Vorgehensweise ersichtlich wird.

3.1.1 Masken

Als Erstes sollte es eine Möglichkeit geben, neue Datensätze anlegen zu können. Hierfür werden Masken (Forms) (1) benutzt, in die beispielsweise Text-Felder (2) integriert werden, die die zu speichernden Daten aufnehmen.

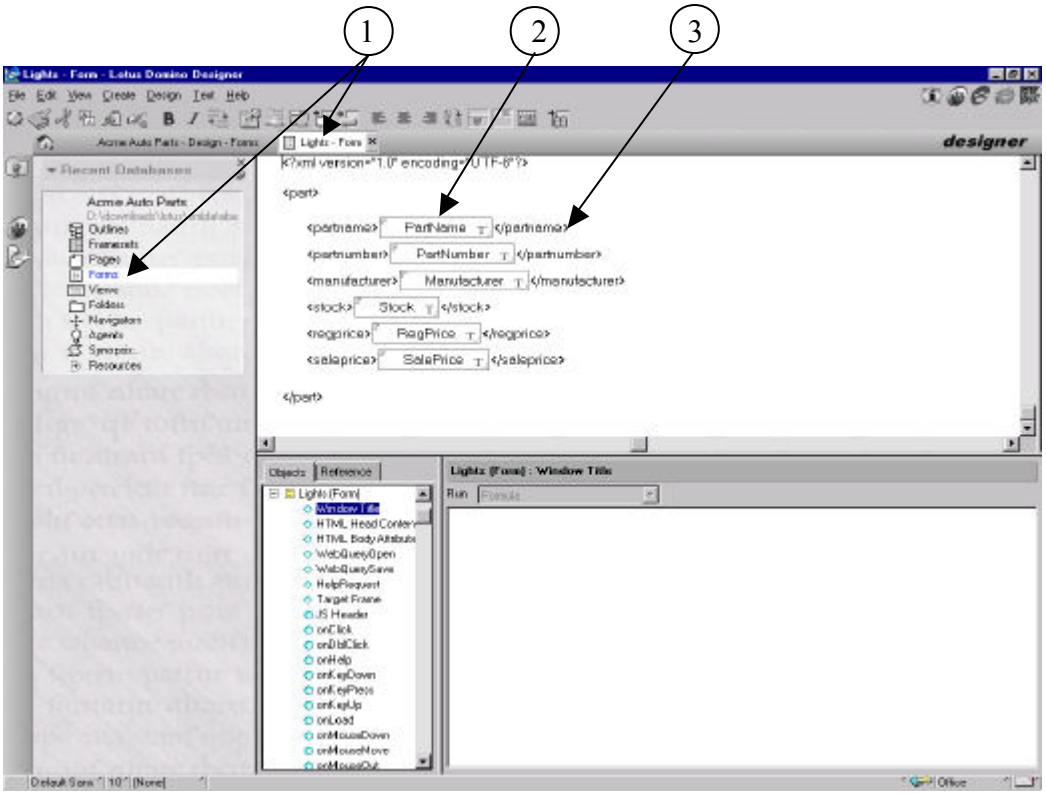


Abbildung 5: Lotus Notes Maske

Wie man sieht, wurden die Felder in die übliche XML-Syntax (3) eingebettet, wodurch ein wohlgeformtes XML-Dokument erzeugt werden kann.

Sowohl bei Masken als auch bei allen anderen folgenden Lotus-Notes Komponenten mit Ausnahme des Agenten muss man darauf achten, dass folgendes Attribut der Komponente aktiviert ist: „Treat content as HTML“

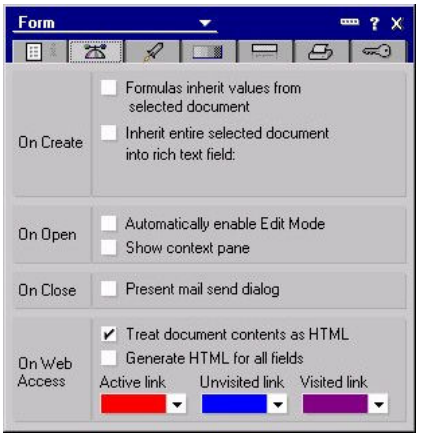


Abbildung 6: Dokument-Eigenschaften

Dieses Attribut wird benötigt, damit die Inhalte unverändert an den Browser weitergegeben werden. Ist diese Funktion nicht aktiviert, würden zusätzlich HTML-Tags in den XML-Code integriert, wodurch dieser nicht mehr wohlgeformt und damit ungültig wäre.

3.1.2 Ansichten

Hat man nun Datensätze angelegt, möchte man diese auch anzeigen bzw. visualisieren können. Ansichten (Views) bieten die Möglichkeit, mehrere Datensätze zu selektieren und diese der Reihe nach anzuzeigen.

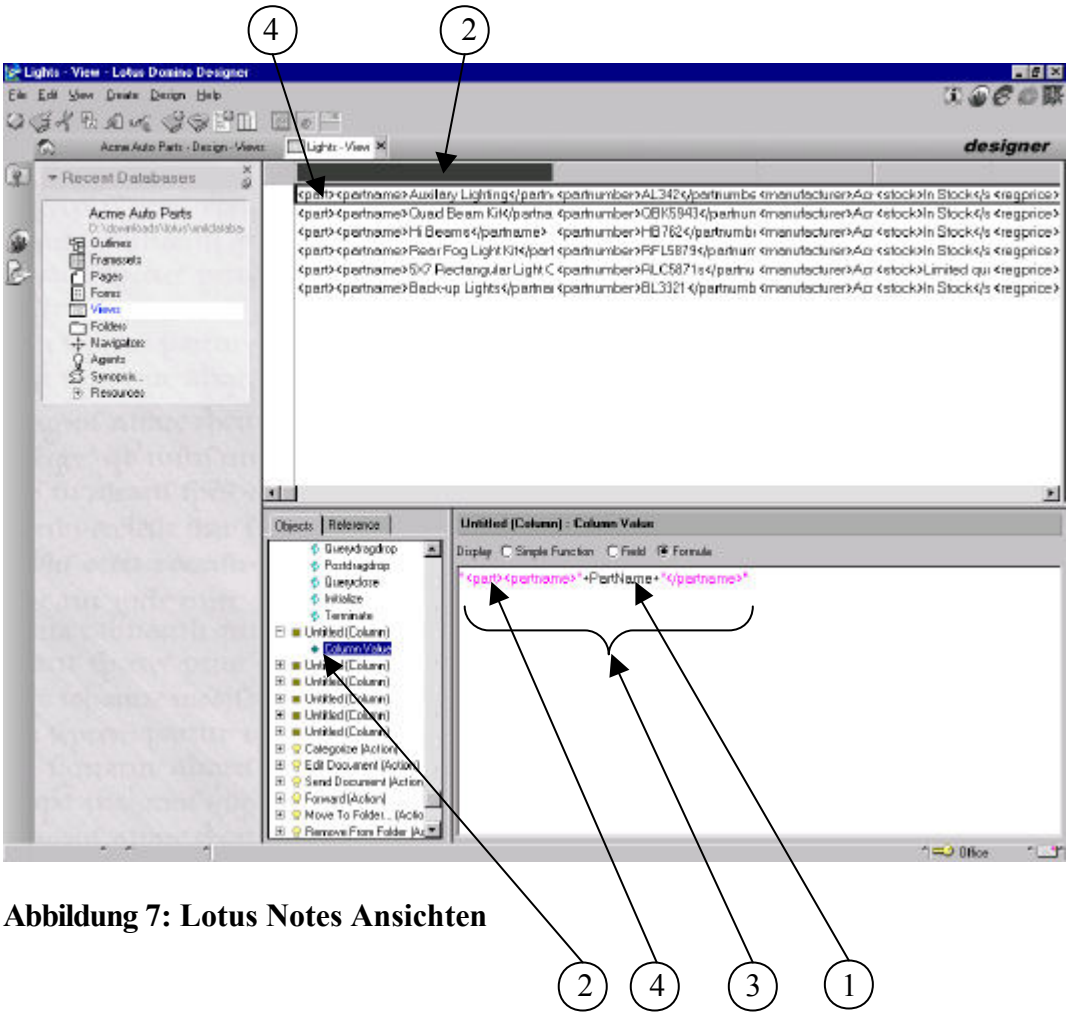


Abbildung 7: Lotus Notes Ansichten

Jedes Element (1) eines Datensatzes wird einer Spalte (2) zugeordnet, in der durch eine Formel (3) der Inhalt der Spalte beschrieben wird. Dabei wird vor und hinter jedes Feld der entsprechende XML-Code als Text eingefügt. Die erste Spalte

nimmt zusätzlich noch das öffnende Markup <part> (4) für den gesamten Datensatz auf. Entsprechend endet in der letzten Spalte die Formel mit </part> um den Datensatz ordnungsgemäß zu schließen, was auf der Abbildung aus Platzgründen nicht zu sehen ist.

3.1.3 Seiten

Als Nächstes wird eine Seite (Page) erstellt, die dann später im Browser angezeigt wird. Dazu muss die Seite ein wohlgeformtes XML-Dokument erzeugen.

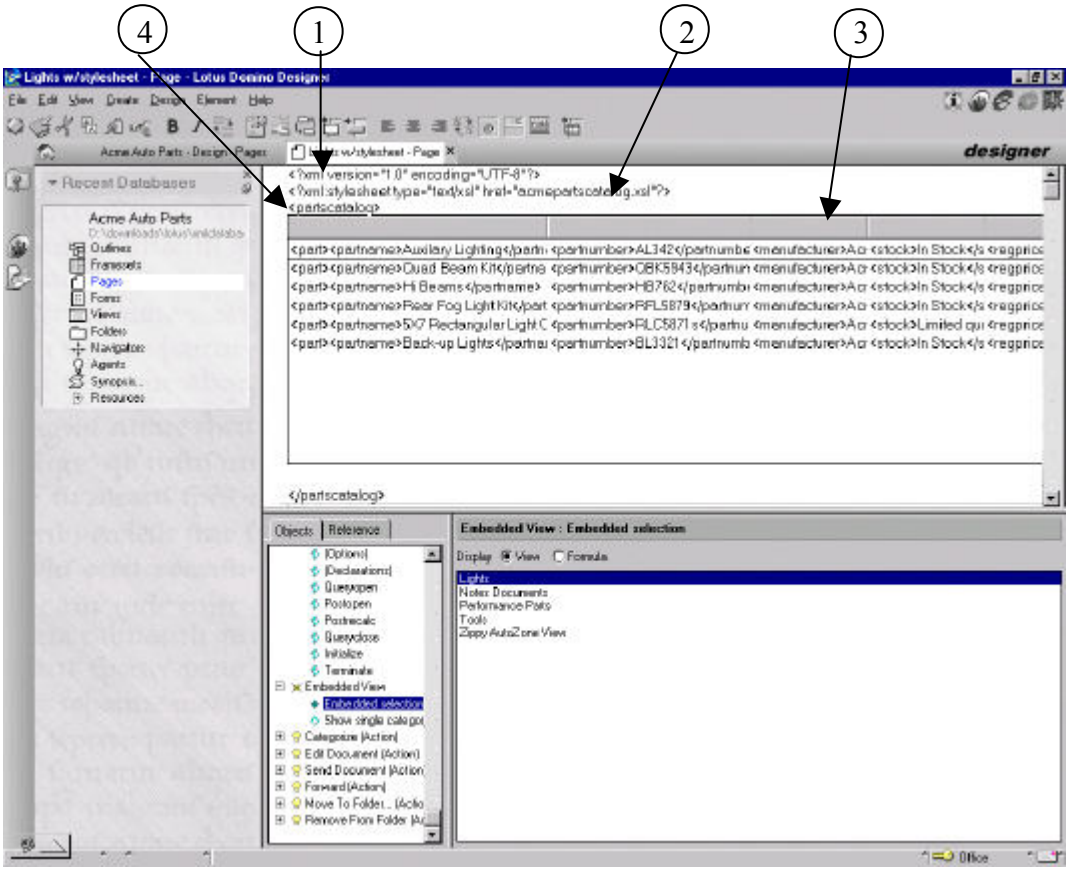


Abbildung 8: Lotus Notes Seiten

Die Zeile (1) „<“?xml version=“1.0“ encoding=“UTF-8“?>“ teilt dem Browser mit, dass es sich bei dem folgenden Code um XML handelt und welcher Zeichensatz benutzt werden soll. UTF-8 steht hier für einen internationalen Zeichensatz.

Anschließend wird das externe Stylesheet „acmepartscatalog.xsl“ (2) geladen.

Die im Kapitel 3.1.2 eingerichtete Ansicht, die die Einzelteile ermittelte, wird nun als „Eingebettete Ansicht“ (3) integriert und von den Markups <partscatalog> und </partscatalog> (4) umschlossen, wodurch die Daten korrekt als Liste implementiert werden.

3.1.4 Stylesheet

Das im Kapitel 3.1.3 genannte Stylesheet (1) kann einfach als Seite in der Datenbank angelegt werden, wobei der Name der Seite dem Filenamen entspricht.

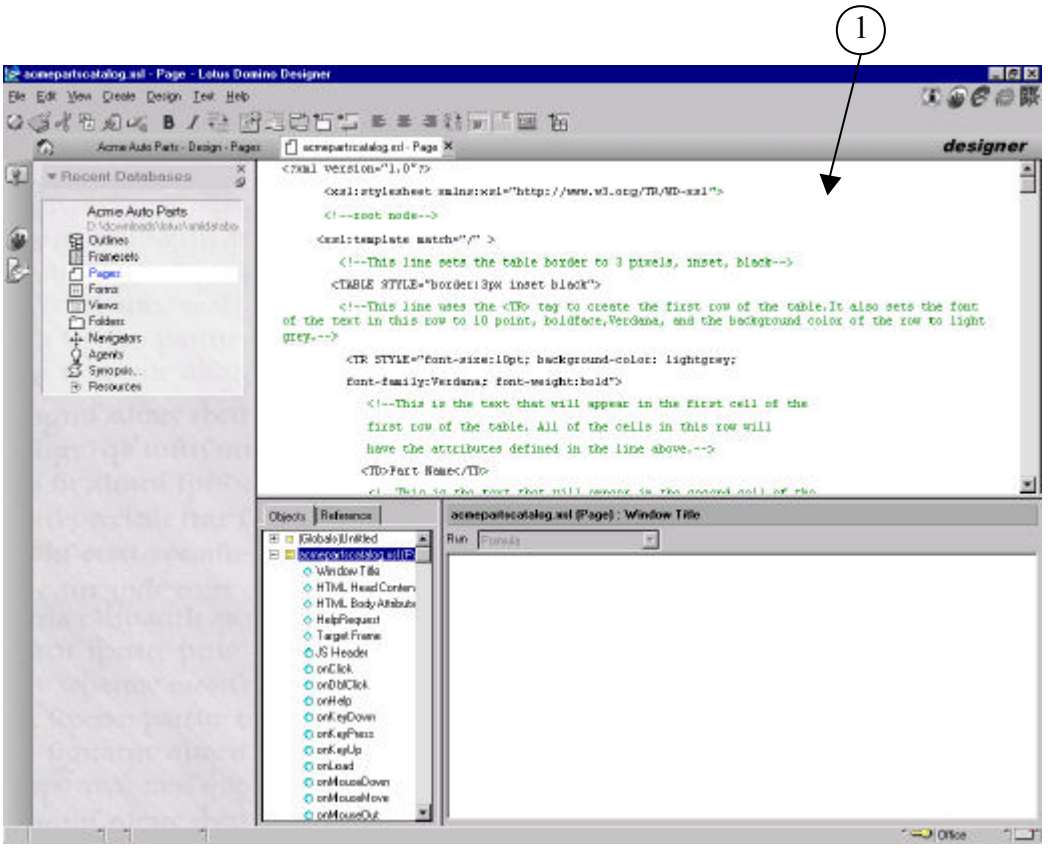


Abbildung 9: Stylesheet innerhalb einer Seite

In diesem Fall wird nur ein wohlgeformtes Dokument erzeugt, das nicht gültig ist. Möchte man ein gültiges Dokument, so muss zusätzlich zum Stylesheet eine DTD angelegt werden.

Da, wie wir bereits wissen, der MS-Internet Explorer 5 einen nichtvalidierenden Parser einsetzt, ist eine DTD nicht zwingend notwendig. (Siehe Kapitel 2.6.3)

3.2 Visualisierung mittels eines Agenten

3.2.1 Der Agent

Der Agent ist ein Kernstück der XML-Sitemap.
Hier wird die Dokumenten-Struktur der Notes-R5 / WebGate Datenbank analysiert und der entsprechende XML-Code generiert. Dabei ist darauf zu achten, dass dieser den Bedingungen der Wohlgeformtheit[28] und Gültigkeit[29] entspricht.
Die hierbei verwendete Programmiersprache ist Lotus-Script.

Im Laufe dieses Abschnitts wird der Agent in einzelne Komponenten aufgeteilt, die anschließend erläutert werden.

Ein Agent unter Lotus Notes besteht in der Regel aus folgenden Komponenten:

- Optionen
- Initialisierung (Hauptprogramm)
- Funktionen

3.2.1.1 Optionen

Dieser Abschnitt findet sich in jeder Lotus-Script-Routine wieder. Hier können bestimmte Optionen festgelegt werden, die dann für die aktuelle Sourceroutine Gültigkeit haben.

```
Option Public
Option Declare
Option Base 1
```

[28] Siehe Kapitel 2.6.3.1

[29] Siehe Kapitel 2.6.3.2

Option Public definiert, dass alle Deklarationen per default *Public* sind.
Option Declare legt fest, dass implizite Variablendeklarationen nicht erlaubt sind. Jede Variable muss dann vor ihrer Verwendung über eine DIM-Anweisung deklariert werden.
Option Base legt den Basiswert für ein Array fest (0 oder 1) [Mann 1999]

3.2.1.2 Initialisierung

Als Erstes werden alle verwendeten Variablen deklariert und initialisiert.
Außerdem werden die benötigten Objektklassen der Lotus-Applikationen für den Agenten bereitgestellt.

```
Sub Initialize

    Dim doc As NotesDocument
    Dim session As New NotesSession
    Dim sPos As String
    Dim Zaehler As Integer
    Zaehler = 0
    Dim i As Integer
    Dim l As Integer
    Dim diff As Integer
    Dim counter As Integer
    Dim Anzahl As Integer
    Dim openordner As Integer
    Dim reachedlayer As Integer
    openordner = 1
    reachedlayer = 1

    Dim db As NotesDatabase
    Dim view As NotesView
    Dim entry As NotesViewEntry
```

```
Dim vc As NotesViewEntryCollection
Set db = session.CurrentDatabase
Set view = db.GetView("Sitemap")
Set vc = view.AllEntries

Dim oViewBereich As notesview
Dim oDocBereich As notesdocument
```

Durch die Verwendung der Klasse „NotesViewEntryCollection“ ist dieser Agent nur unter Notes R5 lauffähig, da diese Klasse erst in diesem Release implementiert wurde.

Außerdem ist zu erwähnen, dass die verwendete WebGate-Ansicht „Sitemap“ als Grundlage für die Ermittlung eines Großteils der benötigten Daten dient. Diese Ansicht ermittelt alle zu visualisierenden Dokumente und schließt z.B. gesperrte Dokumente aus.

Im nächsten Anweisungsblock wird damit begonnen, die erforderlichen Elemente, die eine vollständige XML-Datei benötigt, zu erzeugen. Hierzu können einfache „Print“ Befehle genutzt werden, deren Inhalte direkt in den Quelltext des vom Browser darzustellenden Dokuments geschrieben werden.

```
Print "Content-type: text/xml"
Print "<?xml version='1.0' encoding='ISO-8859-1'?>"
Print "<!DOCTYPE LISTE SYSTEM
'/CONET/HG/Sitemap.nsf/vwFiles/XML1/$FILE/sitemap.dtd'>"
Print "<?xml:stylesheet type='text/xsl'
href='/CONET/HG/Sitemap.nsf/vwFiles/XML1/$FILE/sitemap.xsl' ?>"
Print"<LISTE TYPE='Sitemap'>"
```

Allerdings muss man darauf achten, dass der Default-Header, den der Domino-Server standardmäßig erzeugt, nicht in den Quellcode geschrieben wird, da dies sonst kein gültiges XML-Dokument ergeben würde.

Default-Header:

<HTML>
<!-- Lotus-Domino (Release 5.0.2 (Intl) - 4 November 1999 on Windows NT/Intel) -->
<HEAD>
</HEAD>
<BODY TEXT="000000">

Um dies zu vermeiden, kann mit der Zeile „Print "Content-type: text/xml"“ der Default-Header unterdrückt werden.
Hierauf folgt der Prolog des Dokumentes. Dieser besteht aus der „Processing Instruction“, die die verwendete XML-Version und den Ländercode definiert, und der „Document Type Definition“, die die Struktur der Daten beschreibt. Zusätzlich wird noch ein Stylesheet vom Server angefordert.
Zuletzt wird noch durch das umschließende Markup „<LISTE>“ das Wurzelelement der XML-Datenliste erzeugt.

Der nächste Anweisungsblock benutzt die WebGate Ansicht „vwSetupBereiche“, um die Daten der Hauptbereiche anhand einer Schleife zu ermitteln und in einem Feld „arrBereich()“ abzulegen.

```
Set oViewBereich = db.GetView("vwSetupBereiche")
Set oDocBereich = oViewBereich.getfirstdocument
counter = 1
        While Not oDocBereich Is Nothing
                Redim Preserve arrBereich(counter)
                arrBereich(counter)=oDocBereich.Name(0)
                counter = counter +1
                Set oDocBereich = oViewBereich.getnextdocument(oDocBereich)
        Wend
counter = counter -1
```

Anschließend wird die Anzahl der Dokumente in der zuvor selektierten Ansicht „Sitemap“ ermittelt:

```
Set entry = vc.GetFirstEntry()
        While (Not (entry Is Nothing))
                Zaehler = Zaehler +1
                Set entry = vc.GetNextEntry(entry)
        Wend
        Anzahl = Zaehler
        Neuerzaehler = Zaehler
```

Die Ermittlung der Anzahl der Dokumente ist für die nun folgende dynamische Erzeugung der Felder notwendig, die die Daten der Dokumente aufnehmen sollen. Dadurch wird nur der absolut notwendige Speicherplatz auf dem Domino-Server verbraucht.

```
Redim Preserve xtitel(Anzahl) As String
Redim Preserve xurl(Anzahl) As String
Redim Preserve xebene(Anzahl) As Integer
Redim Preserve xarea(Anzahl) As Integer
```

Diese Felder werden nun mit den entsprechenden Inhalten aus der Ansicht „Sitemap“ gefüllt:

```
Zaehler = 1
        Set entry = vc.GetFirstEntry()
        While (Not (entry Is Nothing))
                Set doc = entry.Document
                sPos=entry.GetPosition(".")
                xtitel(Zaehler) = doc.titel(0)
                xebene(Zaehler) = entry.indentlevel
                xurl(Zaehler)="/"+db.Filepath+"/ContentByKey/"+doc.WG_ID(0)
                +"-"+doc.sprache(0)+"-p"
```

```
            xarea(Zaehler) = GetArea(sPos)

            Set entry = vc.GetNextEntry(entry)
            Zaehler = Zaehler + 1
        Wend
```

Da „Option Base“ gleich 1 gesetzt wurde, beginnt der Zähler mit 1, da das erste Element im Feld die Position 1 besitzt.
Besonders hervorzuheben ist die Funktion „GetPosition“, die zum Dokument den Index in der Hierarchie der Ansicht zurückliefert. Zum Beispiel: 0.1.2
Die Funkion „GetArea“ wird im Kapitel 3.2.1.3 beschrieben.
Das Feld „xtitel“ nimmt den Titel des Dokuments auf, der später als Link-Text verwendet wird.
„xebene“ speichert die Ebene, auf der sich das Dokument in der Indexstruktur befindet. Zum Beispiel 3 für 0.1.2.
Die URL zum Dokument wird in „xurl“ abgespeichert und setzt sich im Wesentlichen aus dem Pfad in der Datenbank, der Art der Ansicht, der eigentlichen einmaligen Dokumenten-ID und dem Ländercode (EN für Englisch und DE für Deutsch) zusammen. Somit kann der Agent für jede Sprache ohne Änderung verwendet werden.
In „xarea“ wird die dazugehörige Hauptebene abgelegt. Also 0 für das obige Beispiel.

Da sich die Kurzbeschreibung zu dem Dokument in einer anderen Ansicht befindet, muss diese getrennt ermittelt werden:

```
Set view = db.GetView("Pages")
Set vc = view.AllEntries
Redim Preserve xkurz(Anzahl+1) As String
Zaehler = 1
Set entry = vc.GetFirstEntry()
While (Not (entry Is Nothing))
        Set doc = entry.Document
```

```
        For i = 1 To Anzahl
                If xurl(i) = "/"+db.Filepath+"/ContentByKey/"+doc.WG_ID(0)+"-
                "+doc.sprache(0)+"-p" Then
                                If doc.Kurzbeschreibung(0) = "" Then
                                xkurz(i) = "Keine Kurzbeschreibung verfügbar!"
                                Else
                                        xkurz(i) = doc.Kurzbeschreibung(0)
                                End If
                                Zaehler = Zaehler + 1
                        End If
        Next
Set entry = vc.GetNextEntry(entry)
Wend
```

Da die WebGate Ansicht „Pages“ mehr Dokumente enthält als die Ansicht „Sitemap“, werden die Dokumente anhand der URL miteinander verglichen und bei Übereinstimmung die Kurzbeschreibung übernommen.

Ein sehr wichtiger Punkt, den es zu berücksichtigen gilt, ist das Vorkommen von Sonderzeichen in den Titeln.
Allerdings sind nicht alle Sonderzeichen kritisch, da nur einige in der Sprache HTML eine besondere Bedeutung haben oder aber standardmäßig nicht in Klarschrift unterstützt werden, wie zum Beispiel die deutschen Umlaute.
Da die Titel später als Link-Text Verwendung finden und ein Internet-Browser in der Regel bestimmte Zeichen ohne Sonderbehandlung nicht richtig darstellen kann, müssen diese so angepasst werden, dass der Browser sie richtig versteht. Deutsche Umlaute bedürfen einer Entity-Deklaration in der Dokument-Typ-Deklaration. Diese wird später im Abschnitt 3.2.4.1 beschrieben.
Zum jetzigen Verständnis muss man erklären, dass durch die Entity-Deklaration zum Beispiel ein „Ä“ durch „Ä“ ersetzt wird. Dieses stellt der Browser dann korrekt als „Ä“ dar. Die Zeichen „<“ und „>“ werden als öffnendes bzw. schließendes Tag interpretiert und müssen durch „<“ und „>“ ersetzt werden.

```
        Dim Suchstring(11) As String
        Dim Entitystring(11) As String
        Dim pos As Long
        Dim Neuerstring As String
        Dim s As Integer

        Suchstring(1) = "&"
        Entitystring(1) = "&amp;"
        Suchstring(2) = "<"
        Entitystring(2) = "&lt;"
        Suchstring(3) = ">"
        Entitystring(3)= "&gt;"
        Suchstring(4) = "ä"
        Entitystring(4) = "&auml;"
        Suchstring(5) = "Ä"
        Entitystring(5) = "&Auml;"
        Suchstring(6) = "ö"
        Entitystring(6) = "&ouml;"
        Suchstring(7) = "Ö"
        Entitystring(7) = "&Ouml;"
        Suchstring(8) = "ü"
        Entitystring(8) = "&uuml;"
        Suchstring(9) = "Ü"
        Entitystring(9) = "&Uuml;"
        Suchstring(10) = """
        Entitystring(10) = "&apos;"
        Suchstring(11) = Chr$(34)
        Entitystring(11) = "&quot;"

For s = 1 To 11
For i = 1 To Anzahl
        pos = Instr (1, xtitel(i), Suchstring(s))
        If pos > 0 Then
```

```
        Neuerstring = ReplaceSubstring(xtitel(i), Suchstring(s), Entitystring(s))
        xtitel(i) = Neuerstring
        End If
Next
Next
```

Die Funktion „InStr“ überprüft, ob sich das gesuchte Zeichen im String befindet.
Die rekursive Funktion „ReplaceSubstring“ übernimmt dann die eigentliche Arbeit des Ersetzens der Zeichen. Diese wird ebenfalls im Kapitel 3.2.1.3 genauer erklärt.
Hierbei ist es wichtig, dass zuerst das „&“ Zeichen ersetzt wird, da zu einem späteren Zeitpunkt ein zuvor eingesetztes „Ä“ zu „Ä“ verändert würde. Diesen Ausdruck wird der Browser mit einer Fehlermeldung quittieren.

An diesem Punkt sind alle notwendigen Vorarbeiten abgeschlossen und es kann nun damit begonnen werden, aus den vorher ermittelten Daten die entsprechende XML-Datenstruktur zu generieren.

Hierfür wurden fünf Regeln entwickelt, die anhand von Vergleichen der Dokumenten-Ebene und der Hauptbereichszugehörigkeit des aktuellen Dokuments mit dem Darauffolgenden, den entsprechenden XML-Code erzeugen.

Hierzu wird zuerst das erste „ORDNER“ Element erzeugt. „ORDNER“ stellen in der Ausgabe die Ordner dar, die sich durch Anklicken öffnen und ihre Inhalte (ORDNER oder DOKUMENT) preisgeben.
Mittels einer Schleife werden dann die Felder der Reihe nach durchlaufen und ausgewertet.
Die Regeln sind mit If- und ElseIf-Bedingungen so verschachtelt, dass höchstens eine Regel pro Dokument angewendet werden kann. Dabei ist es immer sinnvoll, die Abbruchbedingung, also die Regel 1, an den Anfang solcher Abfragen zu legen, um Fehlinterpretationen zu vermeiden. Außerdem würde das letzte Dokument bei den anderen Regeln eine Fehlermeldung erzeugen, da dort nach einem nicht vorhandenen Nachfolgedokument gesucht wird.

```
Print "<ORDNER TYPE="'+arrBereich(1)+"'>"
        For i = 1 To Anzahl

' Regel 1 (Letztes Dokument)

                If i = Anzahl Then
                        Print "<DOKUMENT>"
                        Print "<TITEL>"+xtitel(i)+"</TITEL>"
                        Print "<URL>"+xurl(i)+"</URL>"
                        Print "<KURZ>"+xkurz(i)+"</KURZ>"
                        Print "</DOKUMENT>"

                        If opentopics > 0 Then
                                For l = 1 To opentopics
                                        Print "</ORDNER>"
                                Next
                        End If
```

Sobald die Schleife beim letzten Dokument angelangt ist, wird der entsprechende Datensatz in XML erzeugt und anschließend der geöffnete Teilbaum, je nach Verschachtelung, geschlossen.

```
' Regel 2

        Elseif  ((xebene(i) = xebene(i+1)) And (xarea(i) = xarea(i+1))) Then
                        Print "<DOKUMENT>"
                        Print "<TITEL>"+xtitel(i)+"</TITEL>"
                        Print "<URL>"+xurl(i)+"</URL>"
                        Print "<KURZ>"+xkurz(i)+"</KURZ>"
                        Print "</DOKUMENT>"
```

Die Regel 2 ist für Dokumente zuständig, die sich innerhalb eines Hauptbereichs und auf der gleichen Ebene befinden und einen Nachfolger auf der selben Ebene besitzen.
Bei folgender Indexstruktur

1.1
1.2
1.3

würde die Regel 2 das Dokument zum Index 1.1 und 1.2 bearbeiten.

' Regel 3

```
        Elseif (xarea(i) < xarea(i+1)) Then
                Print "<DOKUMENT>"
                Print "<TITEL>"+xtitel(i)+"</TITEL>"
                Print "<URL>"+xurl(i)+"</URL>"
                Print "<KURZ>"+xkurz(i)+"</KURZ>"
                Print "</DOKUMENT>"

                If opentopics > 0 Then
                        For l = 1 To opentopics
                                Print "</ORDNER>"
                        Next
                        opentopics = 0
                End If
                reachedlayer = reachedlayer + 1

                Print "<ORDNER TYPE="'+arrBereich(reachedlayer)+"'>"
                opentopics = opentopics +1
```

Die dritte Regel wird für den Wechsel von einem zum anderen Hauptbereich verwendet.

Beispiel:

1.1
1.2
2.1

Hier wird wieder das aktuelle Dokument in XML umgewandelt und der Teilbaum, entsprechend der Verschachtelung, geschlossen. Außerdem wird der neue Teilbaum geöffnet.

' Regel 4

```
Elseif ((xebene(i) < xebene(i+1)) And (xarea(i) = xarea(i+1))) Then
    Print "<ORDNER TYPE="'+xtitel(i)+'">"
    Print "<DOKUMENT>"
    Print "<TITEL>"+xtitel(i)+"</TITEL>"
    Print "<URL>"+xurl(i)+"</URL>"
    Print "<KURZ>"+xkurz(i)+"</KURZ>"
    Print "</DOKUMENT>"
    opentopics = opentopics + 1
```

In dieser Regel wird eine neue Unterebene erzeugt, wenn das nächste Dokument eine Ebene tiefer liegt als das Aktuelle.

Beispiel:

3.2
3.2.1

Als Erstes wird ein neuer Teilbaum bzw. Ordner (ORDNER) mit dem Titel des aktuellen Dokuments geöffnet.

Das Dokument 3.2 wird dann als 3.2.1 angelegt, damit das Dokument selbst anklickbar ist. Das Anklicken des Ordners öffnet diesen lediglich, zeigt aber nicht den Inhalt (Content) des Dokuments an.

' Regel 5

```
Elseif ((xebene(i) > xebene(i+1)) And (xarea(i) = xarea(i+1))) Then

    Print "<DOKUMENT>"
    Print "<TITEL>"+xtitel(i)+"</TITEL>"
    Print "<URL>"+xurl(i)+"</URL>"
    Print "<KURZ>"+xkurz(i)+"</KURZ>"
    Print "</DOKUMENT>"
    diff = xebene(i) - xebene(i+1)
    For l = 1 To diff
        Print "</ORDNER>"
        opentopics = opentopics - 1
    Next
```

Die letzte Regel 5 ist für das Schließen eines Ordner zuständig, wenn das nächste Dokument dem gleichen Hauptbereich und einer höheren Ebene als das aktuelle Dokument angehört.

Beispiel:

1.1.1.1
1.2

Nach Erzeugung des XML-Codes für den aktuellen Datensatz wird, entsprechend der Differenz zwischen den Ebenen, der Teilbaum wieder geschlossen.

```
End If
Next
```

Als letzte Anweisung folgt das schließende Markup für das Wurzelelement.

```
Print "</LISTE>"

End Sub
```

3.2.1.3 Funktionen

Die Funktion „ReplaceSubstring“

```
Function ReplaceSubstring(AllStr As String, SearchStr As String, ReplaceStr As String) As String
        Dim position As Integer
        Dim länge As Integer
        Dim vorne As String
        Dim hinten As String
        Dim pos As Integer
        Dim rest As String
        position = Instr(1, AllStr, SearchStr)
        länge = Len(searchstr)
        vorne= Mid (allstr,1, position-1)
        hinten= Mid(allstr, position+länge)
        pos = Instr (1, hinten, SearchStr)

        If pos > 0 Then
                rest = ReplaceSubstring(hinten, SearchStr, ReplaceStr)
                hinten = rest
        End If

        ReplaceSubstring = vorne + ReplaceStr + hinten
End Function
```

Diese Funktion wird für das Abfangen von Sonderzeichen benötigt.
Sie durchsucht einen String nach einer Zeichenkette und ersetzt diese durch eine andere, falls die gesuchte Zeichenkette im String gefunden wird. Da die Funktion rekursiv programmiert wurde, werden bei mehrmaligem Vorkommen der gesuchten Zeichenkette alle durch den neuen String ersetzt.

Die Funktion „GetArea“

```
Function GetArea(Index As String) As Integer
        Dim Position As Integer
        Dim TeilString As Variant
        Position = Instr(1,Index, ".")
        TeilString = Left$(Index, Position -1)
        GetArea = Cint(TeilString)
End Function
```

Diese Funktion liefert die Hauptebene des Dokuments zurück, die zur Erzeugung der XML-Struktur benötigt wird.
So wird zum Beispiel für 1.2.3.4 die Zahl 1 zurückgegeben.

3.2.2 XML-Code

In diesem Abschnitt wird der XML-Code, den der Agent erzeugt, in Auszügen vorgestellt, da dieser sich sonst über mehrere Seiten erstrecken würde.
Der vollständige Code befindet sich auch in der Datei „Sitemap.xml“ auf der beiliegenden CD-Rom und kann im MS-Internet Explorer 5 betrachtet werden. Die Pfade der zusätzlich benötigten Dateien wurden dort so abgeändert, dass die Beispiel-Sitemap auch ohne Lotus Notes Datenbank auskommt, da sich alle Dateien in einem Verzeichnis befinden.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE LISTE SYSTEM
'/CONET/HG/Sitemap.nsf/vwFiles/XML1/$FILE/sitemap.dtd'>
```

```
<?xml:stylesheet type='text/xsl'
href='/CONET/HG/Sitemap.nsf/vwFiles/XML1/$FILE/sitemap.xsl' ?>
<LISTE TYPE="Sitemap">
    <ORDNER TYPE="Das Unternehmen">
        <ORDNER TYPE="Vision | Philosophie">
            <DOKUMENT>
                <TITEL>Vision | Philosophie</TITEL>
                <URL>/CONET\HG\Sitemap.nsf/ContentByKey/HGIL-4HWG6Q-
                DE-p</URL>
                <KURZ>Keine Kurzbeschreibung verfügbar!</KURZ>
            </DOKUMENT>
            <DOKUMENT>
                <TITEL>Gesellschaftliche Verantwortung</TITEL>
                <URL>/CONET\HG\Sitemap.nsf/ContentByKey/HGIL-4HWG7B-
                DE-p</URL>
                <KURZ>Keine Kurzbeschreibung verfügbar!</KURZ>
            </DOKUMENT>
            <DOKUMENT>
                <TITEL>Unternehmensverfassung</TITEL>
                <URL>/CONET\HG\Sitemap.nsf/ContentByKey/HGIL-4HWG8L-
                DE-p</URL>
                <KURZ>Keine Kurzbeschreibung verfügbar!</KURZ>
            </DOKUMENT>
        </ORDNER>
        <ORDNER TYPE="Fakten">
            <DOKUMENT>
                <TITEL>Fakten</TITEL>
                <URL>/CONET\HG\Sitemap.nsf/ContentByKey/HGIL-4HWG6M-
                DE-p</URL>
                <KURZ>Keine Kurzbeschreibung verfügbar!</KURZ>
            </DOKUMENT>
            <DOKUMENT>
                <TITEL>Gesch&auml;ftsstellen</TITEL>
                <URL>/CONET\HG\Sitemap.nsf/ContentByKey/HGIL-4HWG78-
                DE-p</URL>
                <KURZ>Keine Kurzbeschreibung verfügbar!</KURZ>
            </DOKUMENT>
            <DOKUMENT>
                <TITEL>Unternehmensbeteiligungen</TITEL>
                <URL>/CONET\HG\Sitemap.nsf/ContentByKey/HGIL-4HWG73-
                DE-p</URL>
                <KURZ>Keine Kurzbeschreibung verfügbar!</KURZ>
            </DOKUMENT>
            <DOKUMENT>
                <TITEL>Wirtschaftsdaten</TITEL>
                <URL>/CONET\HG\Sitemap.nsf/ContentByKey/HGIL-4HWG6U-
                DE-p</URL>
                <KURZ>Keine Kurzbeschreibung verfügbar!</KURZ>
            </DOKUMENT>
```

```
            <DOKUMENT>
               <TITEL>Vorstand und Aufsichtsrat</TITEL>
               <URL>/CONET\HG\Sitemap.nsf/ContentByKey/HGIL-4HWG7D-
               DE-p</URL>
               <KURZ>Keine Kurzbeschreibung verfügbar!</KURZ>
            </DOKUMENT>
            <DOKUMENT>
               <TITEL>Unsere Kunden</TITEL>
               <URL>/CONET\HG\Sitemap.nsf/ContentByKey/HGIL-4HWG7K-
               DE-p</URL>
               <KURZ>Keine Kurzbeschreibung verfügbar!</KURZ>
            </DOKUMENT>
            <DOKUMENT>
               <TITEL>Das Partnerkonzept</TITEL>
               <URL>/CONET\HG\Sitemap.nsf/ContentByKey/HGIL-4HWG6N-
               DE-p</URL>
               <KURZ>Keine Kurzbeschreibung verfügbar!</KURZ>
            </DOKUMENT>
         </ORDNER>
      <DOKUMENT>
      <TITEL>Geschichte und Entwicklung</TITEL>
      <URL>/CONET\HG\Sitemap.nsf/ContentByKey/HGIL-4HWG7A-DE-
      p</URL>
      <KURZ>Firmenentwicklung CONET CONSULTING ist ein
      unabhängiges IT/DV-Beratungsunternehmen mit über 200 Mitarbeitern an
      insgesamt vier Standorten im Bundesgebiet. Ein wesentlicher Tätigkeitsbereich
      des Unternehmens ist die herstellerneutrale und qualifizierte
      Beratung</KURZ>
      </DOKUMENT>
      </ORDNER>
</LISTE>
```

Wie man sieht, besteht ein Datensatz bzw. Dokument aus folgenden Komponenten:

- TITEL (Link-Text)
- URL (Relativer Pfad zum Dokument)
- KURZ (Kurzbeschreibung zum Dokument)

Diese Komponenten werden mit einem öffnenden Markup <DOKUMENT> und dem schließenden Markup </DOKUMENT> umschlossen. Anschließend werden mit <ORDNER TYPE="Ordnertext"> und </ORDNER> die Dokumente zu einem abgeschlossenen Ordner bzw. Teilbaum zusammengefasst.

Ein Ordner kann aber nicht nur Dokumente aufnehmen. Enthält ein Ordner wiederum einen anderen Ordner, so wird dieser einfach wie der übergeordnete Ordner erzeugt. Dabei ist darauf zu achten, dass ein Ordner an der korrekten Stelle geschlossen wird, damit der logische Aufbau der verschachtelten XML Datenstruktur erhalten bleibt.
Alle Datensätze müssen dann zu einer Liste zusammengefasst werden, indem diese mit den Markups <LISTE TYPE="Sitemap"> und </LISTE> umschlossen werden.

Der Internet Explorer erzeugt nun aus dem vollständigen XML-Code folgendes Ergebnis:

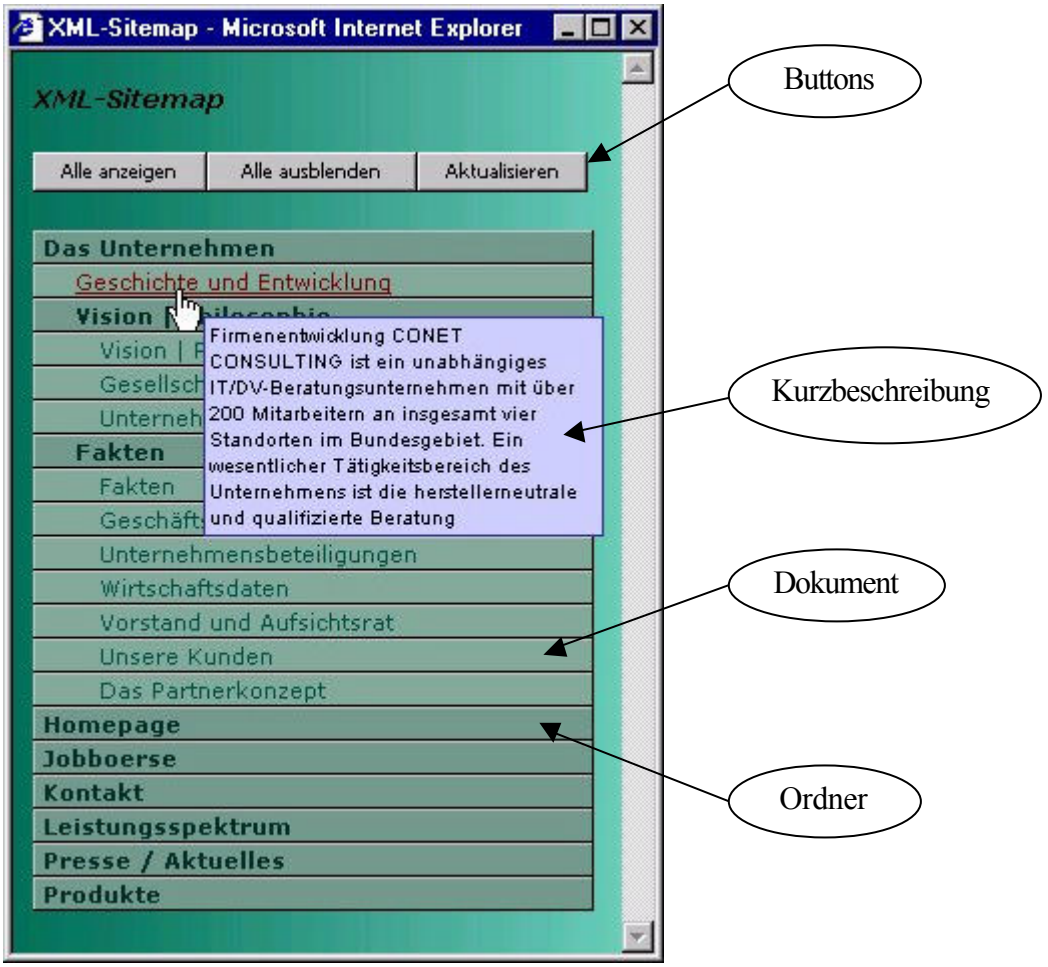


Abbildung 10: XML-Sitemap

Der erste geöffnete Hauptbereich entspricht dabei dem vorher auszugsweise vorgestellten XML-Code.

3.2.3 Der Versions-Check

Der Version-Check ist notwendig, da nicht jeder Browser zur Zeit in der Lage ist, XML-Daten und XSL-Stylesheets darzustellen.

Der MS-Internet-Explorer ab Version 5.0 ist momentan als Einziger dazu fähig.

Diese Abfrage wurde in JavaScript programmiert und befindet sich im Content der Seite, die nach Anklicken des Links „XML-Sitemap“ auf der Homepage geladen wird.

```
<script language="JavaScript">
{
browser_name = navigator.appName;
browser_version = parseFloat(navigator.appVersion);

if (browser_name == "Netscape")
{open("/CONET/HG/Sitemap.nsf/vWFiles/XML1/$FILE/sorry.html","Sorry",
"width=320,height=279,top=5,left=5,location=no,directories=no,status=no,
scrollbars=yes,resizable=0,menubar=no,toolbar=no","mumble");
window.location="../FrameByKey/OTRT-4FFJ6F-EN-p"}

if ((browser_name == "Microsoft Internet Explorer" && browser_version >= 5.0)
|| (navigator.appVersion.indexOf("MSIE 5.0") >= 0))
{open("../Generiere XML?openagent", "Site","width=320,height=279,top=5,
left=5,location=no,directories=no,status=no,scrollbars=yes,resizable=0,
menubar=no,toolbar=no","mumble");
window.location="../FrameByKey/OTRT-4FFJ6F-EN-p"}

else
if (browser_name == "Microsoft Internet Explorer" && browser_version < 5.0)
{open("/CONET/HG/Sitemap.nsf/vWFiles/XML1/$FILE/sorry.html",
"Sorry","width=320,height=279,top=5,left=5,location=no,directories=no,
status=no,scrollbars=yes,resizable=0,menubar=no,toolbar=no","mumble");
window.location="../FrameByKey/OTRT-4FFJ6F-EN-p"}

else
{open("/CONET/HG/Sitemap.nsf/vWFiles/XML1/$FILE/sorry.html",
"Sorry","width=320,height=279,top=5,left=5,location=no,directories=no,
status=no,scrollbars=yes,resizable=0,menubar=no,toolbar=no","mumble");
window.location="../FrameByKey/OTRT-4FFJ6F-EN-p"}
}
</script>
```

Das Skript ermittelt zuerst den Browser-Namen und dessen Version.

Handelt es sich um einen Netscape-Browser oder um einen Internet-Explorer Version 4 oder niedriger, so wird sofort die Seite „Sorry.html“ aufgerufen, die den Benutzer auffordert, mindestens den Internet-Explorer von Microsoft ab Version 5.0 zu verwenden.

Bei dem Microsoft Internet Explorer ist zu beachten, dass eine Version 4, die mittels eines Updates auf 5.0 erneuert wurde, bei einer Standard-Versionsabfrage Version 4 zurückliefert. Allerdings befindet sich im Versions-String die Zeichenkette „MSIE 5.0“, wodurch die richtige Version trotzdem ermittelt werden kann.

Ist der richtige Browser in der korrekten Version vorhanden, so wird ein neues Fenster geöffnet und der Agent gestartet, der die XML-Daten generiert und in dem neuen Fenster visualisiert. Außerdem wird eine Default-Seite im Main-Frame der Homepage angezeigt. Ein anschauliches Beispiel kann im Kapitel „3.4 Implementation in die Homepage“ anhand der CONET-Homepage betrachtet werden.

3.2.4 Die Dateien

Alle benötigten Dateien wurden als File-Attachment in die Notes-Datenbank integriert, wodurch gleichbleibende Pfade garantiert sind.

3.2.4.1 Sitemap.dtd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT LISTE (ORDNER)+>
<!ELEMENT ORDNER (ORDNER | DOKUMENT)+>
<!ELEMENT DOKUMENT (TITEL, URL, KURZ)>
<!ELEMENT TITEL (#PCDATA)>
<!ELEMENT URL (#PCDATA)>
```

```
<!ELEMENT KURZ (#PCDATA)>
<!ATTLIST LISTE TYPE CDATA #REQUIRED>
<!ATTLIST ORDNER TYPE CDATA #REQUIRED>
<!ENTITY auml "ä">
<!ENTITY Auml "Ä">
<!ENTITY uuml "ü">
<!ENTITY Uuml "Ü">
<!ENTITY ouml "ö">
<!ENTITY Ouml "Ö">
```

Da sich die DTD in einer Datei befindet, handelt es sich hier um eine externe Dokument-Typ-Definition.
Die „ELEMENT“ Anweisungen definieren neue Markups, in denen zusätzlich vorgegeben wird, welche Elemente und in welcher Anzahl diese in der Baumstruktur vorkommen dürfen. Das „+“ Zeichen besagt hierbei, dass ein Ordner beliebig viele aber mindestens ein Dokument oder Ordner beinhaltet; bzw. eine Liste mindestens einen Ordner beinhalten muss.
„ATTLIST“ definiert die Attribute „TYPE“, die später im XML-Code den Namen der Liste bzw. des Ordners aufnehmen. „ENTITY“ erzeugt die Entities, die zur Sonderzeichen-Behandlung benötigt werden.

3.2.4.2 Sitemap.xsl

Hierbei handelt es sich um das Stylesheet[30], das für die eigentliche Visualisierung der XML-Daten zuständig ist. Um das Grundprinzip dieses Stylesheets verstehen zu können, ist es notwendig, den Begriff „Namensraum“ bzw. „Namespace“ einzuführen.

[30] Grundlage für dieses Stylesheet war der Online-Artikel von George Young: „DXML: Transformieren eines Inhaltsverzeichnisses aus XML in DHTML“ [Microsoft 2000]

3.2.4.2.1 Exkurs: Namensräume

Solange der Entwickler XML selber programmiert und es innerhalb seines Web-Designs verwendet, gibt es mit den Tags und den Tag-Namen keine Probleme. Er selber kennt und vergibt die Namen. Daher wird es nicht zu der Situation kommen, dass innerhalb eines Dokuments gleiche Tags mit unterschiedlichen Bedeutungen bzw. Attributen verwendet werden. Werden aber externe XML-Dateien geladen, können Konflikte in der Verwendung der Tag-Namen auftreten.

Anfang 1999 hat das W3C XML-Namespaces spezifiziert und standardisiert, um dieses Problem zu lösen. Dadurch wird sichergestellt, dass es nicht zu Konflikten kommt, wenn Tags mit gleichen Namen verwendet werden.
Um dies zu erreichen, muss für jedes Tag ein Namensraum definiert werden, für den er gültig ist.

Beispiel:

<html xmlns:html=http://www.w3.org/TR/REC-html40>

Hier wird ein Namensraum für HTML-Befehle definiert. Werden nun solche Befehle verwendet, müssen diese wie folgt beschrieben werden:

<html:h1>Dieser Text wird in HTML ausgegeben!</html:h1>

Durch die Vergabe von Namensräumen ist es also möglich, verschiedene Web-Sprachen in einem Dokument zu verwenden.
Diese Eigenschaft der Namensräume wird im folgenden Stylesheet genutzt, da XSL- und HTML-Befehle gemeinsam eingesetzt werden.
Dies ist notwendig, da die Sitemap in einem Browser visualisiert werden soll. Dazu müssen die XML-Daten mit Hilfe des Stylesheets zu HTML bzw. DHTML transformiert werden. Der Trick besteht nun darin, dass durch die Erzeugung eines Namensraums für XSL-Tags, zusätzlich zu den XSL-Befehlen auch normale HTML-Tags verwendet werden können.

Das Stylesheet erzeugt mit klassischen HTML-Befehlen die Elemente, die für eine vollständige HTML-Seite notwendig sind wie z.B. der Kopf (<HEAD></HEAD>) oder den Inhaltsteil (<BODY></BODY>), und fügt dann in die entsprechenden Stellen mittels XSL-Befehlen die XML-Daten ein.

3.2.4.2.2 Das Stylesheet

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

Durch die Definition des Namensraumes für XSL-Tags werden diese mit dem Zusatz <xsl:…> von HTML-Tags getrennt.

```
    <xsl:script>
        <![CDATA[

    function RueckeEin(Element)
    {
        var text = "";
        for (var i=2;i<Element;i++) text += "...";
        return text;
    }

]]>
</xsl:script>
```

Die Funktion “RueckeEin” rückt die Textbezeichnung eines Ordners oder den Link eines Dokuments entsprechend der Ebene ein. Dazu werden die Zeichen „...“ in der Hintergrundfarbe vor dem Text bzw. Link geschrieben, damit diese Zeichen unsichtbar bleiben. Je nachdem auf welcher Ebene sich der Ordner bzw. das Dokument befindet, wird dieses entsprechend der Anzahl abzüglich der ersten Ebene wiederholt.

Der nächste Schritt ist die Erzeugung einer Schablone, die die Grundstruktur einer HTML-Seite erstellt und die XML-Daten einfügt:

<xsl:template match="/">

Zusätzlich wird dadurch dem Prozessor die Anweisung gegeben, mit der Transformation am Stamm der XML-Daten zu beginnen, der durch den Schrägstrich angezeigt wird.

```
<HTML>
   <HEAD>
      <TITLE>XML-Sitemap</TITLE>
       <LINK REL="stylesheet" TYPE="text/css"
       HREF="/CONET/HG/Sitemap.nsf/vwFiles/XML1/$FILE/sitemap.css">
       </LINK>
   </HEAD>
```

Innerhalb des Kopfes wird der Titel des Sitemap-Frames definiert und anschließend ein Cascading Stylesheet (CSS) von der WebGate-Datenbank angefordert bzw. geladen.

```
   <BODY bgcolor="#808000">
      <p id="overDiv" style="position:absolute; visibility:hide;z-index:1;"></p>
      <script language="JavaScript"
      src="/CONET/HG/Sitemap.nsf/vwFiles/XML1/$FILE/sitemap.js">
      </script>
```

Der für eine vollständige HTML-Seite notwendige Inhaltsteil (<BODY>) wurde nun geöffnet und die JavaScript-Datei geladen.

```
   <H1>XML-Sitemap</H1>
   <P>
   <BUTTON ONCLICK="ZeigeAlle('SPAN')">Alle anzeigen</BUTTON>
   <BUTTON ONCLICK="BlendeAus('SPAN')">Alle ausblenden</BUTTON>
   <BUTTON ONCLICK="Aktualisiere()">Aktualisieren</BUTTON>
   </P>
```

Nach Erzeugung einer Überschrift werden die Buttons gezeichnet, die es dem Benutzer ermöglichen, alle Knoten des Inhaltsverzeichnisses anzuzeigen oder auszublenden und den Inhalt zu aktualisieren.

Das interessanteste Element in diesem ersten Block ist der nun folgende Teil:

```
        <xsl:apply-templates select="LISTE/ORDNER"></xsl:apply-templates>
    </BODY>
</HTML>
</xsl:template>
```

Er weist den Prozessor an, alle ORDNER-Knoten des XML-Dokuments zu transformieren, die untergeordnete Elemente des LISTE-Stammknotens sind.
An dieser Stelle verlässt der XSL-Prozessor den ersten Block bzw. die erste Schablone und sucht nach einem anderen xsl:template-Block im Stylesheet, der dem select="LISTE/ORDNER"-Attribut entspricht. Dabei handelt es sich um den nächsten xsl:template-Block.

Jeder ORDNER-Knoten, dessen Kontext vom ersten Block an diesen Block übergeben wurde, wird hier verarbeitet. Zunächst wird ein HTML-Textblock (DIV) geöffnet, dem dann das CLASS-Attribut „Ordnermitdocs" verliehen wird.
Anschließend wird die Funktion „RueckeEin" aufgerufen, der das aktuelle Objekt als Parameter mitgegeben wird. Diese schreibt entsprechend der Ebene des Objektes die Zeichen „..." in der Hintergrundfarbe des Textblocks vor den nun folgenden TYPE bzw. die Ordner-Bezeichnung des zu verarbeitenden Knotens. Für das Abrufen von Knotenwerten in XSL wird das xsl:value-of-Element verwendet. Soll der Wert eines Attributs ausgegeben werden, wird „@" an den Anfang des Attributnamens angehängt.

```
<xsl:template match="ORDNER">
    <DIV CLASS="Ordnermitdocs">
            <xsl:eval>RueckeEin(depth(this))</xsl:eval>
            <B CLASS="Ordnermitdocs">
                <xsl:value-of select="@TYPE"></xsl:value-of>
            </B>
    </DIV>
```

Als Nächstes wird ein SPAN-Abschnitt geöffnet, der alle untergeordneten Elemente des aktuellen ORDNER-Elements enthält.
Alle untergeordneten Elemente werden ebenfalls mit Hilfe der Funktion "RueckeEin" eingerückt. Dann werden die Links zu den entsprechenden Dokumenten in der WebGate-Datenbank erzeugt, die innerhalb der <A> und </A>-Tags aus verschiedenen Komponenten zusammengesetzt werden.

Zuerst wird der Ziel-Frame (MainFrame) vorgegeben, in dem der Inhalt des Dokuments angezeigt werden soll. Mit „onMouseOver“ wird eine Funktion „drs“ gestartet, wenn die Maus über den Link streicht. Diese Funktion zeigt die Kurzbezeichnung des aktuellen Dokuments an, die der Funktion als Parameter mitgegeben wird. Dazu muss das Attribut „KBZ“ im Link eingefügt werden, das die Kurzbezeichnung des Dokuments aufnimmt. Schließlich wird dem Link noch die URL zugewiesen, unter der das Dokument zu finden ist und der TITEL eingefügt, der dann als Link-Text im Browser erscheint.

```
<SPAN CLASS="docs">
   <xsl:for-each select="DOKUMENT">
      <DIV>
         <xsl:eval>RueckeEin(depth(this))</xsl:eval>
         <A TARGET="MainFrame"
         onMouseOver="drs(this.KBZ,'Ueberschrift');
         return true;" onMouseOut="nd(); return true;">
         <xsl:attribute name="HREF">
            <xsl:value-of select="URL"></xsl:value-of>
         </xsl:attribute>
         <xsl:attribute name="KBZ">
            <xsl:value-of select="KURZ"></xsl:value-of>
         </xsl:attribute>
         <xsl:value-of select="TITEL"></xsl:value-of>
         </A>
      </DIV>
   </xsl:for-each>
```

Alle untergeordneten ORDNER-Elemente blieben bisher unberücksichtigt, da nur die oberen Ordner mit den zugehörigen Elementen verarbeitet wurden. Die dazu notwendige Rekursion wird im folgenden letzten Teil des Stylesheets realisiert.

```
   <xsl:if test="ORDNER">
      <xsl:apply-templates></xsl:apply-templates>
   </xsl:if>
      </SPAN>
</xsl:template>
</xsl:stylesheet>
```

Unter Verwendung der xsl:if-Bedingung wird die Datenstruktur daraufhin überprüft, ob ein untergeordneter ORDNER-Knoten vorhanden ist. Ist dies der Fall, so wird der Prozessor mit xsl:apply-templates dazu angewiesen, über eine

Schleife wieder direkt in den zweiten Block zu springen und die Daten auszuwerten. Die entsprechende Schachtelung wird dabei automatisch beibehalten.
Dies ist ein Bereich, in dem XSL wirklich herausragt, da die gesamte Rekursion in einer einzigen Anweisung verarbeitet wird.

3.2.4.3 Sitemap.css

```
A:link { text-decoration:none; font-weight:normal; color:"#035649"; }
A:visited { text-decoration:none; font-weight:normal; color:red; }
A:active { text-decoration:none; font-weight:normal; color:red; }
A:hover { text-decoration:underline; color:maroon; }
```

Die ersten vier Zeilen beschreiben das Aussehen der Links in der Sitemap.
Hier werden die Farben eines Links beschrieben, wenn dieser besucht wurde, aktiv ist oder mit der Maus überstrichen wird.

```
BUTTON { font-family:tahoma; font-size:93%; }
```

Hier wird die Schriftart und Schriftgröße des Button-Textes vorgegeben.

```
BODY{background-image:
        url(/CONET/HG/Sitemap.nsf/vwFiles/XML1/$FILE/back.jpg);
        font-family:verdana; font-size:70%; }
```

Im BODY-Tag wird die Hintergrundgrafik geladen und die Schriftart der einzelnen Sitemap-Elemente festgelegt.

```
H1 { font-size:120%; font-style:italic; }
```

H1-Überschriften werden in der entsprechenden Größe und Schriftart (kursiv) angezeigt.

```
DIV {     font-style:normal; cursor:default; background-color:"#81AB9D";
          color:"#81AB9D";
          border-style:solid; border-width:1; border-color:silver black black
          silver; padding:1px; padding-left:4px; width:25em;
          }
```

Das DIV-Tag wird dazu genutzt, alle Elemente (Dokumente), die mit diesem Tag erzeugt wurden, zu umranden und mit einer hellgrünen Hintergrundfarbe zu füllen.

```
DIV.Ordnermitdocs { background-color:"#77998E"; color:"#77998E";
                    cursor:hand; }
```

Dabei muss allerdings zwischen Dokumenten und Ordnern unterschieden werden, da beide mit dem DIV-Tag erzeugt wurden.
DIV.Ordnermitdocs beschreibt die Hauptebenen bzw. Ordner, die Dokumente aufnehmen können. Diese werden in einer dunkelgrünen Farbe dargestellt und es erscheint eine Hand als Mauszeiger, wenn dieser über einem solchen Ordner positioniert ist.

```
B.Ordnermitdocs { color:"#013029"; }
```

Hier wird die Schriftfarbe für die Ordner-Überschrift vorgegeben.

```
SPAN.docs { display:none; }
```

Dokumente werden zusätzlich zum DIV-Tag mit einem SPAN-Tag umschlossen.
Dadurch erreicht man eine zusätzliche Identifizierungsmöglichkeit der Dokumente, da ja Dokumente und Ordner mit einen DIV-Tag versehen wurden.
Jetzt kann gezielt die Sichtbarkeit der Dokumente in der Sitemap gesteuert werden. In der letzten Anweisung werden alle Dokumente ausgeblendet, indem das Display-Attribut der Dokumente per Default auf „none“, also unsichtbar

gesetzt wird. Deshalb werden nach Aufruf der Sitemap nur die Hauptebenen angezeigt.
In der folgenden JavaScript-Datei befinden sich die Funktionen, die durch die Buttons oder einem Mausklick auf einen Ordner aufgerufen werden und die Sichtbarkeit der Dokumente steuern.

3.2.4.4 JavaScript

Der gesamte verwendete JavaScript-Code, bis auf die Versions-Kontrolle, ist in der Datei „Sitemap.js“ zusammengefasst worden.
In ihr befinden sich die Funktionen, die durch die Buttons aufgerufen werden, und die Realisierung der Kurzbeschreibung[31].
Die Button-Funktionen werden nun vorgestellt.

```
function ZeigeAlle(Element)
{
        var AlleElemente = document.all.tags(Element);
        var Anzahl = AlleElemente.length;
        for (var i=0;i<Anzahl;i++) AlleElemente[i].style.display = "block";
}
```

Die Funktion „ZeigeAlle“ ist dafür zuständig, die Sitemap vollständig anzuzeigen, nachdem der Button „Alle anzeigen“ betätigt wurde. Das heißt, alle in einem Ordner befindlichen Ordner und Dokumente werden visualisiert.
Dies geschieht dadurch, dass das Display-Attribut für jedes einzelne Element auf „Block“ also sichtbar gesetzt wird.

---

[31] Unter „http://javascript.internet.com/“ werden freie JavaScripts zu verschiedensten Themengebieten angeboten. Die Visualisierung der Kurzbezeichnung wurde mit einem JavaScript von Erik Bosrup realisiert. [JavaScript 2000] Durch eine Funktion, der der anzuzeigende Text als Parameter mitgegeben wird, wird eine einspaltige Tabelle neben dem Mauszeiger gezeichnet, in der die Kurzbeschreibung eingefügt wird.

```
function BlendeAus(Element)
{
        var AlleElemente = document.all.tags(Element);
        var Anzahl = AlleElemente.length;
        for (var i=0;i<Anzahl;i++) AlleElemente[i].style.display = "none";
}
```

Entsprechend steuert die Funktion „BlendeAus“ die Sichtbarkeit der Dokumente und Ordner nach Betätigen des „Alle ausblenden“ Buttons. Es wird wieder der Grundzustand hergestellt, in dem nur die Hauptebenen angezeigt werden. Bei allen anderen Elementen wird das Display-Attribut wieder auf „None“ also unsichtbar gesetzt.

```
function Aktualisiere()
{
    open("/CONET/HG/Sitemap.nsf/GeneriereXML?openagent",
    "Site","width=320,height=450,top=10,
    left=680,location=no,directories=no,status=no,scrollbars=yes,resizable=0,
    menubar=no,toolbar=no","mumble");
}
```

Ist die Sitemap über einen längeren Zeitraum geöffnet, kann sich die Dokumentenstruktur in der Zwischenzeit geändert haben.
Die Funktion „Aktualisiere“ startet erneut den Agenten, der dann die aktuellen Daten ermittelt und die Sitemap auf den neuesten Stand bringt.

Zusätzlich zu den Buttons kann die Sitemap auch manuell durch einen Maus-Klick auf einen Ordner bedient werden. Die dort enthaltenen Dokumente und Ordner, die sich nur eine Ebene tiefer befinden, werden dann angezeigt.

```
function document.onclick()
{
        var Quelle = window.event.srcElement;
        if ("Ordnermitdocs" == Quelle.className && (Kind =
           ErmittleKind(Quelle,"SPAN")))
        {
        Kind.style.display = ("block" == Kind.style.display ? "none" : "block");
        }
}

function ErmittleKind(Quelle,Element)
{
        var Kinder = document.all;
        for (var i=Quelle.sourceIndex;i<Kinder.length;i++)
                {
                if (Element == Kinder[i].tagName) return Kinder[i];
                }
        return false;
}
```

Die Funktionen „Document.onclick" und „ErmittleKind" arbeiten hierbei zusammen. Zuerst wird überprüft, ob der Benutzer auf ein Element mit dem className „Ordnermitdocs" geklickt hat. Die Funktion „ErmittleKind" wird dann dazu verwendet, einen Verweis auf ein gültiges, untergeordnetes Kind-Element bzw. „false" zurückzugeben, falls kein solches Element vorhanden ist. Ist ein untergeordnetes Element vorhanden, so wird das „display" Attribut anhand der Dreierbedingung auf den Komplementärwert der aktuellen Einstellung gesetzt.

3.3 Logischer Aufbau der Sitemap und deren Komponenten

Betätigt der Benutzer den Link „XML-Sitemap“ im Navigations-Frame, so wird ein leeres Dokument aufgerufen, in dem sich ein JavaScript befindet, dass die Versionsabfrage des Browsers übernimmt.
Da zur Zeit lediglich der MS Internet-Explorer 5 in der Lage ist, XML und XSL darzustellen, wird bei allen anderen Browser-Versionen die HTML-Seite „Sorry.html“ aufgerufen.

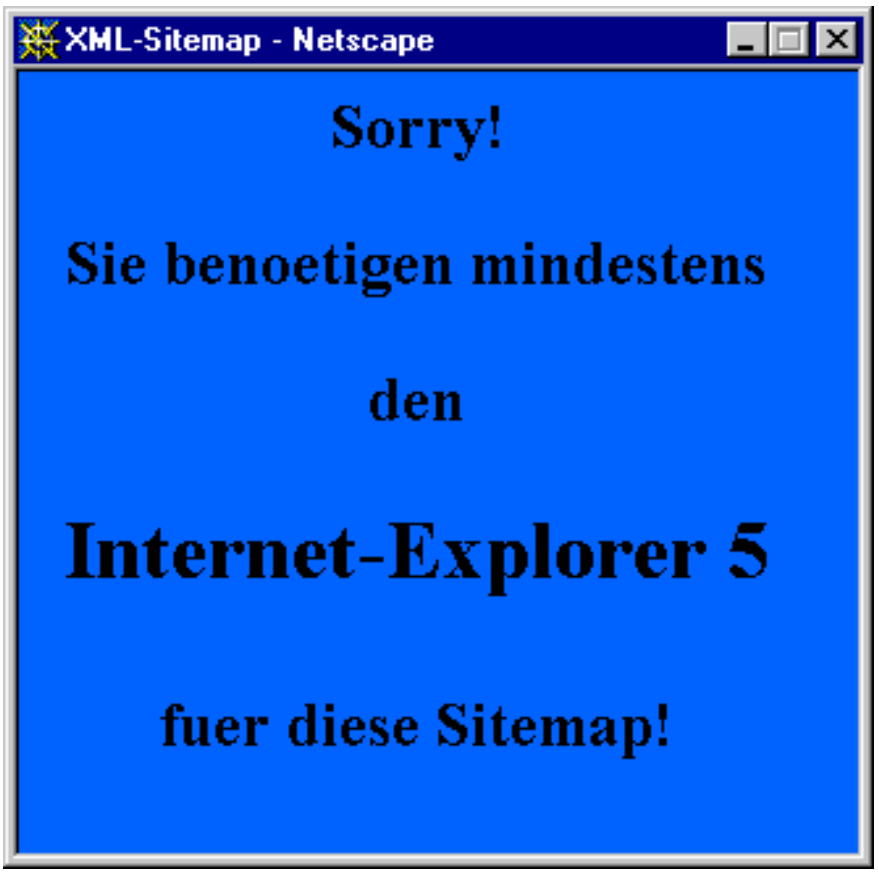


Abbildung 11: Sorry.html

Ist der MS Internet-Explorer 5 vorhanden, wird der Lotus Notes Agent gestartet, der dann, entsprechend der Dokument-Struktur in der WebGate Datenbank, den XML-Code generiert.

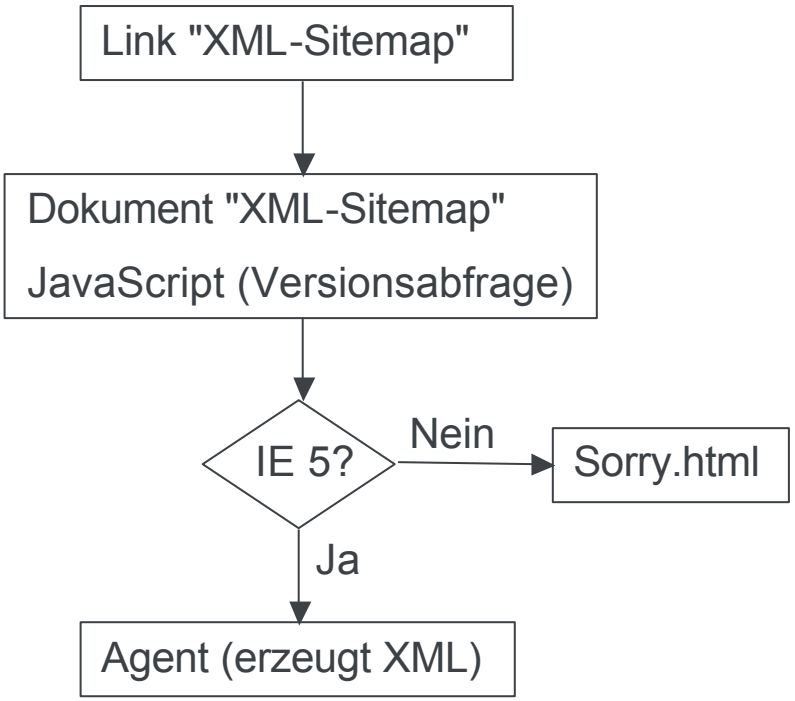


Abbildung 12: Ablaufdiagramm

Nachdem der Browser die XML-Daten erhalten hat, fordert er alle weiteren benötigten Dateien vom Server an.

Als Erstes wird die Document Typ Definition (DTD) und das Stylesheet (XSL) geladen. Dieses benötigt dann die JavaScript Datei und das Cascading Stylesheet (CSS), welches noch die Hintergrundgrafik (JPG) anfordert.

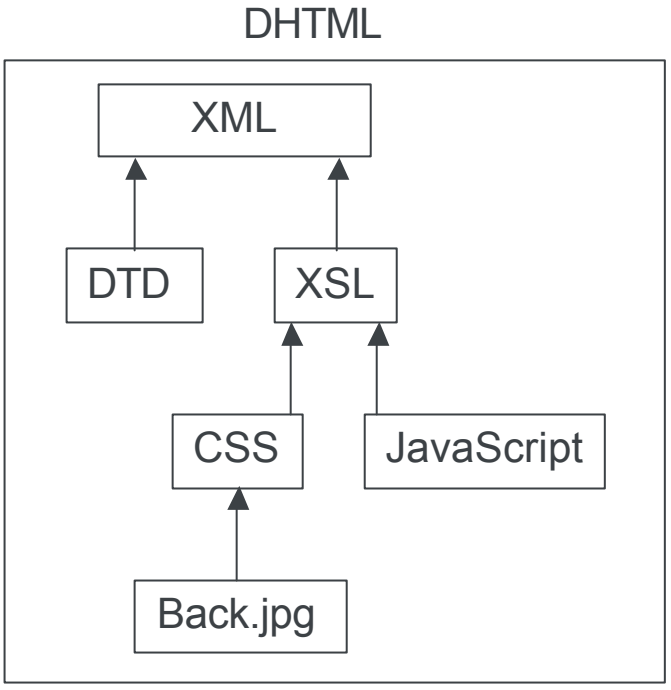


Abbildung 13: Logischer Aufbau

Der XSL-Processor (Parser) transformiert daraufhin XML, XSL und die DTD zu HTML.

Durch die Verwendung der JavaScript-Funktionen und des Cascading Stylesheets (CSS) wird die Sitemap dynamisch, da die Darstellung im Browser nicht mehr statisch ist, sondern sich den Aktionen des Benutzers anpasst. Dadurch werden die einzelnen Sprachkonstrukte nicht nur zu HTML sondern zu dynamischem HTML (DHTML) verbunden.

3.4 Implementation in die Homepage

Die XML-Sitemap wurde in die Produktpalette der Firma CONET CONSULTING AG aufgenommen und wird in die Homepage integriert werden, sobald es mehr als einen Internet-Browser gibt, der XML und XSL visualisieren kann. Abbildung 14 zeigt die Sitemap am Beispiel der CONET Homepage, wodurch dem Leser ein Eindruck über das Gesamterscheinungsbild gegeben wird.

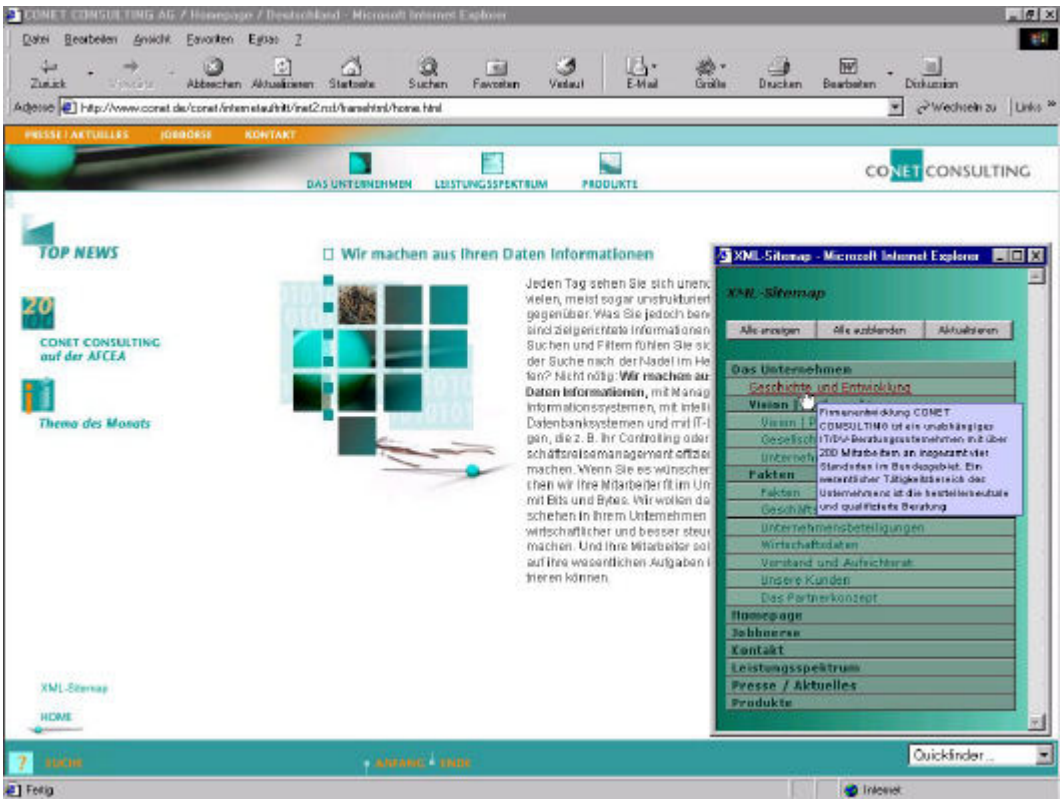


Abbildung 14: Conet Homepage mit Sitemap

Kapitel 4: Eingesetzte Software

4.1 WebGate

Die Basis für „WebGate“ bildet die Groupware Entwicklungsplattform „Lotus Notes“. Grundlegende Informationseinheiten in Notes sind Datenbanken mit den zugehörigen Dokumenten und den darin enthaltenen Feldern. Im Allgemeinen enthält eine Datenbank Informationen zu einem bestimmten Thema, wie eine Pinwand oder ein Diskussionsforum.
Diese kann aber auch einen kompletten Web-Auftritt enthalten.

Dieser Web-Auftritt wurde bisher immer von erfahrenen Redakteuren und Internet-Spezialisten gestaltet, die mit HTML-Editoren einzelne Webseiten erstellen und anschließend die Webserverdaten updaten. Jede Webseitenänderung erfordert also mindestens zwei Spezialisten. Genau hier setzt „WebGate“ an.
In der Regel beträgt der Anteil der Erstellung des Web-Auftritts an den Gesamtkosten nur 10 Prozent. 90 Prozent müssen für die laufende Aktualisierung aufgewendet werden.

„WebGate“ ist ein Web-Content-Management-System, das dem Kunden die Möglichkeit gibt, Websites ohne jegliche HTML-Kenntnisse zu erstellen und zu modifizieren.
Das entsprechende Layout wurde einmal von Internet-Spezialisten erstellt. Danach werden nur noch Inhalte erzeugt und gepflegt, die automatisch in dem einmal definierten Layout dargestellt werden, ohne Programmieraufwand.
Hierdurch lassen sich die Kosten für eine Web-Präsenz entscheidend verringern.

Ein weiterer großer Vorteil von „WebGate“ liegt in der Unterstützung des Autors und des Surfers bei der Navigation durch den Web-Auftritt:

„WebGate“ speichert die Webinhalte in einer hierarchischen Kapitelstruktur, in die Webdokumente eingebettet werden. Dabei bilden die „Hauptdokumente“ die

oberste Ebene, die „Kind-Dokumente“ in beliebiger Tiefe aufnehmen können. Die WebGate-Navigationsstruktur baut sich selbständig aus dieser Kapitelstruktur auf.

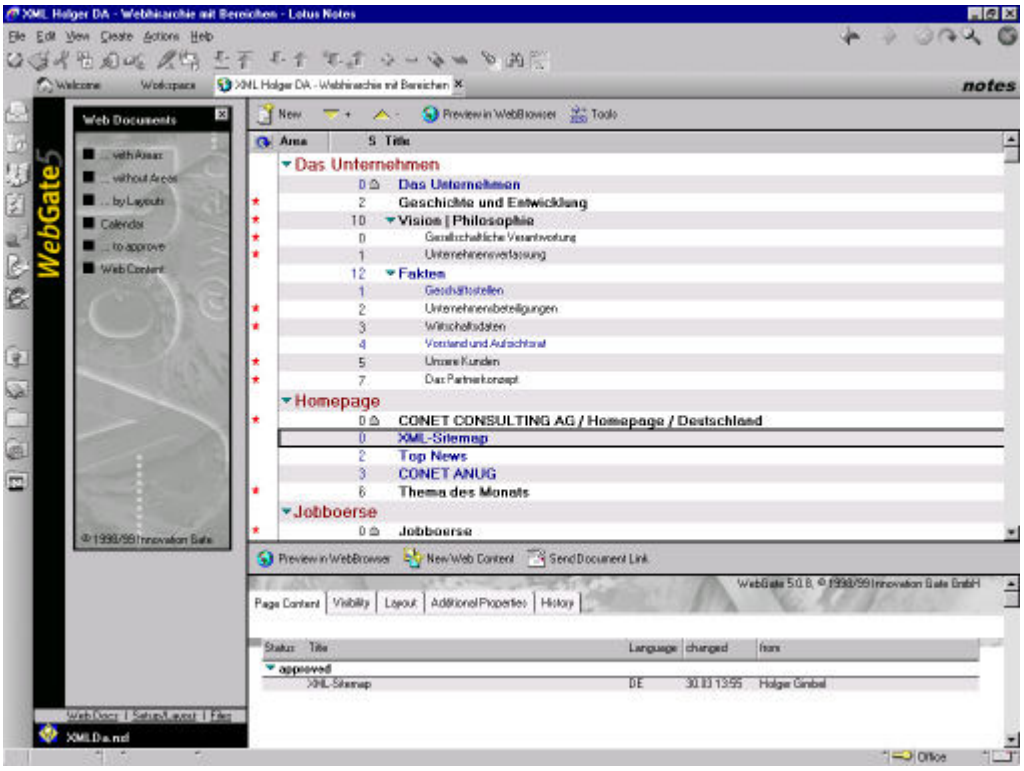


Abbildung 15: Dokumentstruktur unter WebGate

Fordert ein Web-Surfer nun eine bestimmte HTML-Seite an, so wird das Layout mit dem Inhalt „on the fly“ kombiniert und durch Navigationselemente ergänzt.
Hierzu gehört der Navigations-Frame, in dem die aktuelle Kapitelstruktur bereitgestellt wird, und ein Umgebungsnavigator, der dem Surfer überall hin folgt.

„WebGate“ bietet selbstverständlich noch viele andere Features, wie z.B.

- Unterstützung von META-Tags[32] für Suchmaschinen
- Unterstützung mehrsprachiger Websites
- Konfigurierbare Suchfunktion
- Einbindung „externer“ Inhalte über „virtuelle Dokumente“
- Steuerung der Sichtbarkeit von Dokumenten über Zugriffsrechte

[32] vgl. Kapitel 2.3.1

- Redaktionsworkflow mit Freigabefunktion
- Uvm.

Unter „www.Innovationgate.de“ oder „www.youatweb.com“ können weitere Informationen zu „WebGate“ bezogen werden.

4.2 XML-SPY

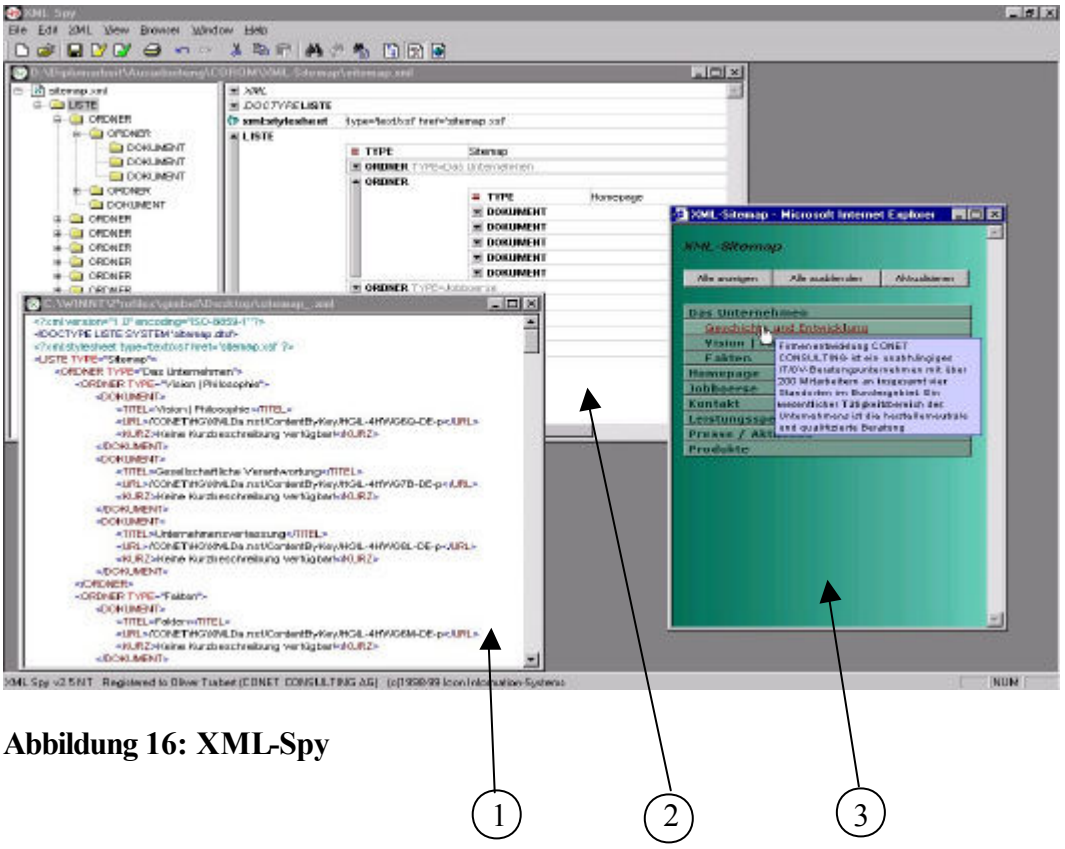


Abbildung 16: XML-Spy

Ein sehr hilfreiches Tool zur Bearbeitung und Entwicklung von XML-Daten und deren Visualisierung mit XSL ist der XML-Spy in der Version 2.5 der Firma „Icon EDV Informations-Systeme GmbH“[33]. Hierbei handelt es sich um einen validierenden XML-Editor, der dem Entwickler drei verschiedene Ansichten auf XML-Dokumente ermöglicht. Der „Enhanced Grid View“ (2) präsentiert die Daten in einer hierarchisch strukturierten Form. Im „Source View“ (1) wird der

[33] www.xmlspy.com ; Hier kann auch eine für 30 Tage gültige Trial-Version bezogen werden.

Source-Code, entsprechend der Syntax, farblich markiert dargestellt. Die integrierte „Browser-View“ (3) nutzt den MS-Internet Explorer 5, um die XML-Daten zu rendern[34]. Diese Daten werden auf Wohlgeformtheit und Gültigkeit überprüft und mit der Dokument Typ Deklaration abgeglichen. Fehler können dadurch leicht erkannt und verbessert werden.

[34] Mit Rendering bezeichnet man den Vorgang, in dem ein Programm oder ein Chip ein dreidimensionales Drahtgittermodell mit Texturen, Licht- und Schatteneffekten ausstattet, um dem menschlichen Auge ein realitätsnahes räumliches Bild zu liefern.

Kapitel 5: Wie sieht die Zukunft aus?

5.1 EDI und XML

Seit mehr als 20 Jahren verwenden Unternehmen Electronic Data Interchange (EDI) zum elektronischen Austausch von strukturierten Handelsdokumenten wie Bestellungen oder Rechnungen. Im Gegensatz zur papiergebundenen Kommunikation soll EDI den Dokumentenaustausch ohne Medienbruch zwischen verschiedenen Systemen ermöglichen. Doch obwohl EDI enorme Einsparungspotentiale zugesprochen werden, nutzen laut Forrester Research[35] nur schätzungsweise 5% der Unternehmen, für die der Einsatz lohnenswert wäre, auch tatsächlich EDI.
Der Hauptgrund liegt darin, dass vor allem kleine und mittelständische Unternehmen (KMU) die erheblichen Setup- und Betriebskosten traditioneller EDI-Lösungen scheuen.

Für die Kommunikation zwischen den Systemen einzelner Unternehmen werden sogenannte Value Added Networks (VANs) genutzt, für die weitere Kosten entstehen.

Der Einsatz von EDI war bisher überwiegend großen Unternehmen vorbehalten. Ein ersten Schritt zur Reduzierung der Kosten solcher Lösungen ist die Nutzung des Internets mit seiner bestehenden Kommunikationsinfrastruktur als Transportmedium. Während bei VANs die Kosten für die Nutzung der Leitungen oft nach der Anzahl der gesendeten Nachrichten oder gesendeten Zeichen berechnet werden, gibt es bei der Nutzung des Internets eine solche Kalkulation nicht. [iX 07/99, S. 127]

[35] www.forrester.com

XML bietet Unternehmen eine Möglichkeit, ein „effizientes EDI“ einzusetzen. XML kommt im EDI-Kontext ohne dediziertes Netzwerk und Übersetzungssoftware aus. Weil sich dieser Ansatz auf dem Internet abspielt, werden Tausende Unternehmen eine effiziente Möglichkeit bekommen, an EDI zu partizipieren.

EDI-Nachrichten sind im Prinzip einfache Zeichenansammlungen. XML dagegen erlaubt die Auszeichnung von einzelnen Zeichen bzw. Zeichenketten, so dass zum Beispiel eine Zeichenkette von den Systemen des Empfängers als Artikelnummer, eine zweite als Liefertermin und eine dritte als Preis interpretiert werden kann.
Jede Anwendung, die XML verstehen kann, kann diese Datenelemente importieren und sie den richtigen Stellen in einer Datenbank zuordnen.

Wenn XML und EDI miteinander kombiniert werden, entstehen Dokumente, die nahtlos von einem Produktivsystem über das World Wide Web in das Produktivsystem des Geschäftspartners fließen können. Damit ist zu erwarten, dass die Anzahl der Unternehmen, die EDI einsetzen, deutlich steigt.

Die Perspektiven von XML gehen aber über EDI weit hinaus. XML ermöglicht den einfachen und plattformunabhängigen Datenaustausch zwischen verschiedenen Applikationen. Da XML ein offenes, textbasiertes Format ist, kann es genauso wie HTML über das World Wide Web kommuniziert werden. Spezielle Anforderungen an Hardware und Netzinfrastruktur bestehen nicht.

Die Möglichkeiten für den Electronic Commerce sind faszinierend: Wenn zum Beispiel Suchprogramme auf XML aufsetzen, können sie nicht nur nach konkreten Wörtern oder Passagen suchen, sondern auch eine Liste aller im Dokument aufgeführten Produkte und Firmen erstellen. Das Klicken auf einen beliebigen Namen könnte die Einblendung relevanter Zusatzinformationen (zum Beispiel Preis, Bezugsort und Produktmerkmale) auslösen.
Übertragen auf die Sitemap dieser Diplomarbeit entspricht dies der Anzeige der Kurzbezeichnung zu jedem Dokument, wenn die Maus den Dokumenten-Link berührt.

Datenbanksysteme, die die zugehörige Abfragesprache XQL unterstützen, sind bereits auf dem Markt (zum Beispiel von der Software AG). [Mattes 1999, S. 104]

5.2 Wohin geht XML?

Die Entwicklung von HTML löste ein wahres Publikationsfeuerwerk aus. Überall wurde begonnen, Dokumente im Web zu publizieren.
Das gleiche Phänomen kann man nun bei XML beobachten.
Daten sind nicht mehr nur mysteriöse Binär-Gebilde sondern sie haben sich zu etwas Gewöhnlichem gewandelt, das Menschen lesen und schreiben können, denn es handelt sich hierbei um gewöhnlichen Text.
Jedermann kann mittels XML eigene Daten erstellen, die Ausgabe gestalten und kontrollieren.
Dadurch können nicht nur mehr Anwender auf Daten zugreifen, es wird auch mehr Daten geben, auf die zugegriffen wird. Dies wird in naher Zukunft eine Daten-Explosion im Web hervorrufen.

Die Entwicklung und Verbreitung von XML hat auch Auswirkungen auf die in unserem täglichen Leben eingesetzte Standard-Software wie Textverarbeitungs-, Tabellenkalkulationsprogramme, E-Mail, Datenbanken und vieles mehr. Das Web wird in diesen Anwendungen verstärkt erschlossen und integriert.

„Vorbei sind die Tage der isolierten, inkompatiblen Anwendung – nun folgt die Zeit des universellen Zugriffs und der gemeinsam genutzten Daten.“
[Goldfarb 1999]

Um dies realisieren zu können, muss XML als Basis für jede Art von Daten etabliert werden. Hieraus lassen sich dann ohne anfallende Doppelarbeit für Erfassung und Konvertierung der Daten nahezu beliebige andere Dokumententypen generieren.

Ein weiterer Aspekt ist die Entwicklung und Einführung von XHTML.

Die Zeiten von HTML 4.0 neigen sich langsam dem Ende zu. Im September 1999 endete die Review-Phase der neuen HTML Version 5.0. Das W3C versah die neue HTML-Version mit einem eigenen Namen: XHTML (E**x**tended **H**yper**T**ext **M**arkup **L**anguage). Diese Namensgebung hat aber auch einen tieferen Hintergrund. Mit der neuen Version von HTML findet eine deutliche Annäherung bzw. eine Verschmelzung zwischen HTML und XML statt. XHTML ist eine XML-konforme Neuformulierung von HTML und kann in allen Browsern, die HTML 4.0 unterstützen, angewendet werden. Die aktuellen Browser von Microsoft und Netscape sollten keine größeren Probleme mit der Darstellung der HTML 4.0-Inhalte haben. Mit der Interpretation der auf XML basierenden Erweiterungen sieht es allerdings anders aus.

Durch den Aufbau auf XML bietet XHTML den Entwicklern einige Vorteile. Die XHTML-Dokumente basieren normalerweise auf dem Medientyp text/html und benötigen dadurch theoretisch keinen neuen Browser. Die neuen Dokumente können zusätzlich auch in speziellen XML-Dokumentformaten (text/xml und application/xml) angeboten und in dafür vorgesehenen XML-Agenten betrachtet werden. XHTML soll nach dem Willen des W3C den nächsten Schritt in der Entwicklung des Internets darstellen. [IW 12/99, S.64]

Das W3C verfolgt bei der Einführung von XHTML im Wesentlichen zwei Ziele: Erweiterbarkeit und Portabilität der Sprache. Ein Grund hierfür ist der verstärkte Einsatz von PDAs und WAP-Handys in der Zukunft für den Zugriff aufs Internet.

Kapitel 6: Schlußwort

„Das Web und das Internet sind für kommerzielle Unternehmungen interessant geworden und haben sich zu einem Massenmedium in der Qualität von Zeitung und Buch entwickelt.
Wer die Zukunft des Web in einigen Jahren vorhersagen will, muss weit über seine heutige Form hinausdenken. Wahrscheinlich werden zunehmend herkömmliche Medien wie Fernseher mit dem Web verbunden. Sie werden Ihren Fernsehapparat anschalten und das Programm mit einem Web-Browser auswählen. Dieser kommuniziert über Telefon oder einem anderem Rückkanal direkt mit den Sendern und ruft vielleicht einen Film als Videostrom über das Internet ab. Auch wenn es noch seltsam klingt – Sie werden eines Tages selbst das Web-basierte Nutzerinterface Ihrer heimischen Kaffeemaschine gestalten können.

Für die nähere Zukunft wird XML die Technologie mit dem schnellsten Wachstum und den meisten Auswirkungen sein. Eine ganze Reihe von Standards zur Auszeichnung von mathematischen oder chemischen Formeln, für Datenbankanbindungen, für Vektorgrafiken etc. befinden sich momentan in der Entwicklung und werden in naher Zukunft auch in Software umgesetzt sein.“
[Tolksdorf 2000]

Anhang A

Quellcode

Sitemap.dtd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT LISTE (ORDNER)+>
<!ELEMENT ORDNER (ORDNER | DOKUMENT)+>
<!ELEMENT DOKUMENT (TITEL, URL, KURZ)>
<!ELEMENT TITEL (#PCDATA)>
<!ELEMENT URL (#PCDATA)>
<!ELEMENT KURZ (#PCDATA)>
<!ATTLIST LISTE TYPE CDATA #REQUIRED>
<!ATTLIST ORDNER TYPE CDATA #REQUIRED>
<!ENTITY auml "ä">
<!ENTITY Auml "Ä">
<!ENTITY uuml "ü">
<!ENTITY Uuml "Ü">
<!ENTITY ouml "ö">
<!ENTITY Ouml "Ö">
```

Sitemap.xsl

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:script>
<![CDATA[

function RueckeEin(Element)
{
        var text = "";
        for (var i=2;i<Element;i++) text += "...";
        return text;
}]]>
</xsl:script>

<xsl:template match="/">
<HTML>
    <HEAD>
        <TITLE>XML-Sitemap</TITLE>
        <LINK REL="stylesheet" TYPE="text/css"
        HREF="/CONET/HG/Sitemap.nsf/vwFiles/XML1/$FILE/sitemap.css"></LINK>
    </HEAD>
    <BODY bgcolor="#808000">
        <p id="overDiv" style="position:absolute; visibility:hide;z-index:1;"></p>
        <script language="JavaScript"
        src="/CONET/HG/Sitemap.nsf/vwFiles/XML1/$FILE/sitemap.js"></script>
        <H1>XML-Sitemap</H1>
```

```
        <P>
            <BUTTON ONCLICK="ZeigeAlle('SPAN')">Alle anzeigen</BUTTON>
            <BUTTON ONCLICK="BlendeAus('SPAN')">Alle ausblenden</BUTTON>
            <BUTTON ONCLICK="Aktualisiere()">Aktualisieren</BUTTON>
        </P>
        <xsl:apply-templates select="LISTE/ORDNER"></xsl:apply-templates>
    </BODY>
</HTML>
</xsl:template>

<xsl:template match="ORDNER">
    <DIV CLASS="Ordnermitdocs">
        <xsl:eval>RueckeEin(depth(this))</xsl:eval>
        <B CLASS="Ordnermitdocs">
            <xsl:value-of select="@TYPE"></xsl:value-of>
        </B>
    </DIV>
    <SPAN CLASS="docs">
        <xsl:for-each select="DOKUMENT">
            <DIV>
                <xsl:eval>RueckeEin(depth(this))</xsl:eval>
                <A TARGET="MainFrame" onMouseOver="drs(this.KBZ,'Ueberschrift'); return
                true;" onMouseOut="nd(); return true;">

                        <xsl:attribute name="HREF">
                            <xsl:value-of select="URL"></xsl:value-of>
                        </xsl:attribute>

                        <xsl:attribute name="KBZ">
                            <xsl:value-of select="KURZ"></xsl:value-of>
                        </xsl:attribute>

                        <xsl:value-of select="TITEL"></xsl:value-of>
                </A>
            </DIV>
        </xsl:for-each>

        <xsl:if test="ORDNER">
            <xsl:apply-templates></xsl:apply-templates>
        </xsl:if>
    </SPAN>
</xsl:template>
</xsl:stylesheet>
```

Sitemap.css

```
A:link { text-decoration:none; font-weight:normal; color:"#035649"; }
A:visited { text-decoration:none; font-weight:normal; color:red; }
A:active { text-decoration:none; font-weight:normal; color:red; }
A:hover { text-decoration:underline; color:maroon; }
```

```
BUTTON { font-family:tahoma; font-size:93%; }
BODY { background-image:url(/CONET/HG/Sitemap.nsf/vwFiles/XML1/$FILE/back.jpg); font-
family:verdana; font-size:70%; }

H1 { font-size:120%; font-style:italic; }

DIV {     font-style:normal; cursor:default; background-color:"#81AB9D"; color:"#81AB9D";
          border-style:solid; border-width:1; border-color:silver black black silver;
          padding:1px; padding-left:4px; width:25em;
           }

B.Ordnermitdocs { color:"#013029"; }
DIV.Ordnermitdocs { background-color:"#77998E"; color:"#77998E"; cursor:hand; }
SPAN.docs { display:none; }
```

Sitemap.js

```
function ErmittleKind(Quelle,Element)
        {
                var Kinder = document.all;
                for (var i=Quelle.sourceIndex;i<Kinder.length;i++)
                {
                        if (Element == Kinder[i].tagName) return Kinder[i];
                }
                return false;
        }

        function document.onclick()
        {
                var Quelle = window.event.srcElement;
                if ("Ordnermitdocs" == Quelle.className && (Kind =
                ErmittleKind(Quelle,"SPAN")))
                {
                        Kind.style.display = ("block" == Kind.style.display ? "none" : "block");
                }
        }
```

```
	function ZeigeAlle(Element)
	{
		var AlleElemente = document.all.tags(Element);
		var Anzahl = AlleElemente.length;
		for (var i=0;i<Anzahl;i++) AlleElemente[i].style.display = "block";
	}

	function BlendeAus(Element)
	{
		var AlleElemente = document.all.tags(Element);
		var Anzahl = AlleElemente.length;
		for (var i=0;i<Anzahl;i++) AlleElemente[i].style.display = "none";
	}


	function Aktualisiere()
	{
	open("/CONET/HG/Sitemap.nsf/Generiere XML?openagent",
	"Site","width=320,height=450,top=10,
	left=680,location=no,directories=no,status=no,scrollbars=yes,resizable=0,menubar=no,to
	olbar=no","mumble");
}
```

// Visualisierung der Kurzbeschreibung:

```
// Original:  Erik Bosrup (erik@bosrup.com)
// Web Site:  http://www.bosrup.com/web/overlib/

// This script and many more are available free online at
// The JavaScript Source!! http://javascript.internet.com

// Main background color (the large area)
// Usually a bright color (white, yellow etc)
if (typeof fcolor == 'undefined') { var fcolor = "#CCCCFF"; }

// Border color and color of caption
// Usually a dark color (black, brown etc)
if (typeof backcolor == 'undefined') { var backcolor = "#333399"; }
```

```
// Text color
// Usually a dark color
if (typeof textcolor == 'undefined') { var textcolor = "#000000"; }

// Color of the caption text
// Usually a bright color
if (typeof capcolor == 'undefined') { var capcolor = "#FFFFFF"; }

// Color of "Close" when using Sticky
// Usually a semi-bright color
if (typeof closecolor == 'undefined') { var closecolor = "#9999FF"; }

// Width of the popups in pixels
// 100-300 pixels is typical
if (typeof width == 'undefined') { var width = "200"; }

// How thick the border should be in pixels
// 1-3 pixels is typical
if (typeof border == 'undefined') { var border = "1"; }



// How many pixels to the right/left of the cursor to show the popup
// Values between 3 and 12 are best
if (typeof offsetx == 'undefined') { var offsetx = 10; }

// How many pixels to the below the cursor to show the popup
// Values between 3 and 12 are best
if (typeof offsety == 'undefined') { var offsety = 10; }

ns4 = (document.layers)? true:false
ie4 = (document.all)? true:false

// Microsoft Check.
if (ie4) {
if (navigator.userAgent.indexOf('MSIE 5')>0) {
ie5 = true;
} else {
ie5 = false; }
} else {
ie5 = false;}
```

```
var x = 0;
var y = 0;
var snow = 0;
var sw = 0;
var cnt = 0;
var dir = 1;
var tr = 1;
if ( (ns4) || (ie4) ) {
if (ns4) over = document.overDiv
if (ie4) over = overDiv.style
document.onmousemove = mouseMove
if (ns4) document.captureEvents(Event.MOUSEMOVE)
}

// Simple popup right
function drs(text) {
dts(1,text);
}

// Caption popup right
function drc(text, title) {
dtc(1,text,title);
}

// Sticky caption right
function src(text,title) {
stc(1,text,title);
}

// Simple popup left
function dls (text) {
dts(0,text);
}

// Caption popup left
function dlc(text, title) {
dtc(0,text,title);
}
```

```
// Sticky caption left
function slc(text,title) {
stc(0,text,title);
}

// Simple popup center
function dcs(text) {
dts(2,text);
}

// Caption popup center
function dcc(text, title) {
dtc(2,text,title);
}

// Sticky caption center
function scc(text,title) {
stc(2,text,title);
}

// Clears popups if appropriate
function nd() {
if ( cnt >= 1 ) { sw = 0 };
if ( (ns4) || (ie4) ) {
if ( sw == 0 ) {
snow = 0;
hideObject(over);
} else {
cnt++;
}}}

// Simple popup
function dts(d,text) {
txt = "<TABLE WIDTH="+width+" BORDER=0 CELLPADDING="+border+"
CELLSPACING=0 BGCOLOR=\""+backcolor+"\"><TR><TD><TABLE WIDTH=100%
BORDER=0 CELLPADDING=2 CELLSPACING=0
BGCOLOR=\""+fcolor+"\"><TR><TD><FONT FACE=\"Arial,Helvetica\"
COLOR=\""+textcolor+"\" SIZE=\"-
2\">"+text+"</FONT></TD></TR></TABLE></TD></TR></TABLE>"
layerWrite(txt);
```

```
dir = d;
disp();
}

// Caption popup
function dtc(d,text, title) {
txt = "<TABLE WIDTH="+width+" BORDER=0 CELLPADDING="+border+"
CELLSPACING=0 BGCOLOR=\""+backcolor+"\"><TR><TD><TABLE WIDTH=100%
BORDER=0 CELLPADDING=0 CELLSPACING=0><TR><TD><SPAN
ID=\"PTT\"><B><FONT
COLOR=\""+capcolor+"\">"+title+"</FONT></B></SPAN></TD></TR></TABLE><TABLE
WIDTH=100% BORDER=0 CELLPADDING=2 CELLSPACING=0
BGCOLOR=\""+fcolor+"\"><TR><TD><SPAN ID=\"PST\"><FONT
COLOR=\""+textcolor+"\">"+text+"</FONT><SPAN></TD></TR></TABLE></TD></TR></T
ABLE>"
layerWrite(txt);
dir = d;
disp();
}

// Sticky
function stc(d,text, title) {
sw = 1;
cnt = 0;
txt = "<TABLE WIDTH="+width+" BORDER=0 CELLPADDING="+border+"
CELLSPACING=0 BGCOLOR=\""+backcolor+"\"><TR><TD><TABLE WIDTH=100%
BORDER=0 CELLPADDING=0 CELLSPACING=0><TR><TD><SPAN
ID=\"PTT\"><B><FONT
COLOR=\""+capcolor+"\">"+title+"</FONT></B></SPAN></TD><TD ALIGN=RIGHT><A
HREF=\"/\" onMouseOver=\"cClick();\" ID=\"PCL\"><FONT
COLOR=\""+closecolor+"\">Close</FONT></A></TD></TR></TABLE><TABLE
WIDTH=100% BORDER=0 CELLPADDING=2 CELLSPACING=0
BGCOLOR=\""+fcolor+"\"><TR><TD><SPAN ID=\"PST\"><FONT
COLOR=\""+textcolor+"\">"+text+"</FONT><SPAN></TD></TR></TABLE></TD></TR></T
ABLE>"
layerWrite(txt);
dir = d;
disp();
snow = 0;
}
```

```
// Common calls
function disp() {
if ( (ns4) || (ie4) ) {
if (snow == 0) {
if (dir == 2) { // Center
moveTo(over,x+offsetx-(width/2),y+offsety);
}
if (dir == 1) { // Right
moveTo(over,x+offsetx,y+offsety);
}
if (dir == 0) { // Left
moveTo(over,x-offsetx-width,y+offsety);
}
showObject(over);
snow = 1;
}
}
// Here you can make the text goto the statusbar.
}

// Moves the layer
function mouseMove(e) {
if (ns4) {x=e.pageX; y=e.pageY;}
if (ie4) {x=event.x; y=event.y;}
if (ie5) {x=event.x+document.body.scrollLeft; y=event.y+document.body.scrollTop;}
if (snow) {
if (dir == 2) { // Center
moveTo(over,x+offsetx-(width/2),y+offsety);
}
if (dir == 1) { // Right
moveTo(over,x+offsetx,y+offsety);
}
if (dir == 0) { // Left
moveTo(over,x-offsetx-width,y+offsety);
}}}

// The Close onMouseOver function for Sticky
function cClick() {
hideObject(over);
sw=0;}
```

```
// Writes to a layer
function layerWrite(txt) {
if (ns4) {
var lyr = document.overDiv.document
lyr.write(txt)
lyr.close()
}
else if (ie4) document.all["overDiv"].innerHTML = txt
}

// Make an object visible
function showObject(obj) {
if (ns4) obj.visibility = "show"
else if (ie4) obj.visibility = "visible"
}

// Hides an object
function hideObject(obj) {
if (ns4) obj.visibility = "hide"
else if (ie4) obj.visibility = "hidden"
}

// Move a layer
function moveTo(obj,xL,yL) {
obj.left = xL
obj.top = yL
}
```

Sorry.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
        <title>XML-Sitemap</title>
</head>
<body bgcolor="#0060FF">
```

```
<H2><p ALIGN="CENTER">Sorry!</p></H2>
<H2><p ALIGN="CENTER">Sie benoetigen mindestens</p></H2>
<H2><p ALIGN="CENTER">den</p></H2>
<H1><p ALIGN="CENTER">Internet-Explorer 5</p></H1>
<H2><p ALIGN="CENTER">fuer diese Sitemap!</p></H2>
</body>
</html>
```

Der Agent

```
Sub Initialize

    Dim doc As NotesDocument
    Dim session As New NotesSession
    Dim sPos As String
    Dim Zaehler As Integer
    Zaehler = 0
    Dim i As Integer
    Dim l As Integer
    Dim diff As Integer
    Dim counter As Integer
    Dim Anzahl As Integer
    Dim openordner As Integer
    Dim reachedlayer As Integer
    openordner = 1
    reachedlayer = 1

    Dim db As NotesDatabase
    Dim view As NotesView
    Dim entry As NotesViewEntry
    Dim vc As NotesViewEntryCollection
    Set db = session.CurrentDatabase
    Set view = db.GetView("Sitemap")
    Set vc = view.AllEntries

    Dim oViewBereich As notesview
    Dim oDocBereich As notesdocument
```

```
Print "Content-type: text/xml"
Print "<?xml version='1.0' encoding='ISO-8859-1'?>"
Print "<!DOCTYPE LISTE SYSTEM
'/CONET/HG/Sitemap.nsf/vwFiles/XML1/$FILE/sitemap.dtd'>"
Print "<?xml:stylesheet type='text/xsl'
href='/CONET/HG/Sitemap.nsf/vwFiles/XML1/$FILE/sitemap.xsl' ?>"
Print"<LISTE TYPE='Sitemap'>"

Set oViewBereich = db.GetView("vwSetupBereiche")
Set oDocBereich = oViewBereich.getfirstdocument
counter = 1
        While Not oDocBereich Is Nothing
                Redim Preserve arrBereich(counter)
                arrBereich(counter)=oDocBereich.Name(0)
                counter = counter +1
                Set oDocBereich = oViewBereich.getnextdocument(oDocBereich)
        Wend
counter = counter -1
Set entry = vc.GetFirstEntry()
        While (Not (entry Is Nothing))
                Zaehler = Zaehler +1
                Set entry = vc.GetNextEntry(entry)
        Wend
        Anzahl = Zaehler
        Neuerzaehler = Zaehler

Redim Preserve xtitel(Anzahl) As String
Redim Preserve xurl(Anzahl) As String
Redim Preserve xebene(Anzahl) As Integer
Redim Preserve xarea(Anzahl) As Integer

Zaehler = 1
        Set entry = vc.GetFirstEntry()
        While (Not (entry Is Nothing))
                Set doc = entry.Document
                sPos=entry.GetPosition(".")

                xtitel(Zaehler) = doc.titel(0)
                xebene(Zaehler) = entry.indentlevel
```

```
                xurl(Zaehler)="/"+db.Filepath+"/ContentByKey/"+doc.WG_ID(0)+"-
                "+doc.sprache(0)+"-p"
                xarea(Zaehler) = GetArea(sPos)

                Set entry = vc.GetNextEntry(entry)
                Zaehler = Zaehler + 1
        Wend

Set view = db.GetView("Pages")
Set vc = view.AllEntries
Redim Preserve xkurz(Anzahl+1) As String
Zaehler = 1
Set entry = vc.GetFirstEntry()
While (Not (entry Is Nothing))
        Set doc = entry.Document
        For i = 1 To Anzahl
                If xurl(i) = "/"+db.Filepath+"/ContentByKey/"+doc.WG_ID(0)+"
                "+doc.sprache(0)+"-p" Then
                        If doc.Kurzbeschreibung(0) = "" Then
                                xkurz(i) = "Keine Kurzbeschreibung verfügbar!"
                                Else
                                        xkurz(i) = doc.Kurzbeschreibung(0)
                        End If
                        Zaehler = Zaehler + 1
                End If
        Next
Set entry = vc.GetNextEntry(entry)
Wend


Dim Suchstring(11) As String
        Dim Entitystring(11) As String
        Dim pos As Long
        Dim Neuerstring As String
        Dim s As Integer

        Suchstring(1) = "&"
        Entitystring(1) = "&amp;"
        Suchstring(2) = "<"
        Entitystring(2) = "&lt;"
```

```
        Suchstring(3) = ">"
        Entitystring(3)= "&gt;"
        Suchstring(4) = "ä"
        Entitystring(4) = "&auml;"
        Suchstring(5) = "Ä"
        Entitystring(5) = "&Auml;"
        Suchstring(6) = "ö"
        Entitystring(6) = "&ouml;"
        Suchstring(7) = "Ö"
        Entitystring(7) = "&Ouml;"
        Suchstring(8) = "ü"
        Entitystring(8) = "&uuml;"
        Suchstring(9) = "Ü"
        Entitystring(9) = "&Uuml;"
        Suchstring(10) = "'"
        Entitystring(10) = "&apos;"
        Suchstring(11) = Chr$(34)
        Entitystring(11) = "&quot;"

For s = 1 To 11
        For i = 1 To Anzahl
        pos = Instr (1, xtitel(i), Suchstring(s))
        If pos > 0 Then
                Neuerstring = ReplaceSubstring(xtitel(i), Suchstring(s), Entitystring(s))
                xtitel(i) = Neuerstring
        End If
        Next
Next

Print "<ORDNER TYPE='"+arrBereich(1)+"'>"
        For i = 1 To Anzahl

' Regel 1 (Letztes Dokument)

        If i = Anzahl Then
                Print "<DOKUMENT>"
                Print "<TITEL>"+xtitel(i)+"</TITEL>"
                Print "<URL>"+xurl(i)+"</URL>"
                Print "<KURZ>"+xkurz(i)+"</KURZ>"
                Print "</DOKUMENT>"
```

```
        If opentopics > 0 Then
                For l = 1 To opentopics
                        Print "</ORDNER>"
                Next
        End If
```

' Regel 2

```
    Elseif  ((xebene(i) = xebene(i+1)) And (xarea(i) = xarea(i+1))) Then
                Print "<DOKUMENT>"
                Print "<TITEL>"+xtitel(i)+"</TITEL>"
                Print "<URL>"+xurl(i)+"</URL>"
                Print "<KURZ>"+xkurz(i)+"</KURZ>"
                Print "</DOKUMENT>"
```

' Regel 3

```
    Elseif (xarea(i) < xarea(i+1)) Then
                Print "<DOKUMENT>"
                Print "<TITEL>"+xtitel(i)+"</TITEL>"
                Print "<URL>"+xurl(i)+"</URL>"
                Print "<KURZ>"+xkurz(i)+"</KURZ>"
                Print "</DOKUMENT>"

                If opentopics > 0 Then
                        For l = 1 To opentopics
                                Print "</ORDNER>"
                        Next
                        opentopics = 0
                End If
                reachedlayer = reachedlayer + 1
                Print "<ORDNER TYPE='"+arrBereich(reachedlayer)+"'>"
                opentopics = opentopics +1
```

' Regel 4

```
    Elseif ((xebene(i) < xebene(i+1)) And (xarea(i) = xarea(i+1))) Then
        Print "<ORDNER TYPE='"+xtitel(i)+"'>"
        Print "<DOKUMENT>"
        Print "<TITEL>"+xtitel(i)+"</TITEL>"
```

```
                Print "<URL>"+xurl(i)+"</URL>"
                Print "<KURZ>"+xkurz(i)+"</KURZ>"
                Print "</DOKUMENT>"
                opentopics = opentopics + 1

' Regel 5

        Elseif ((xebene(i) > xebene(i+1)) And (xarea(i) = xarea(i+1))) Then
                Print "<DOKUMENT>"
                Print "<TITEL>"+xtitel(i)+"</TITEL>"
                Print "<URL>"+xurl(i)+"</URL>"
                Print "<KURZ>"+xkurz(i)+"</KURZ>"
                Print "</DOKUMENT>"
                diff = xebene(i) - xebene(i+1)
                        For l = 1 To diff
                                Print "</ORDNER>"
                                opentopics = opentopics -1
                        Next

        End If
Next
Print "</LISTE>"
End Sub

Function ReplaceSubstring(AllStr As String, SearchStr As String, ReplaceStr As String) As String
        Dim position As Integer
        Dim länge As Integer
        Dim vorne As String
        Dim hinten As String
        Dim pos As Integer
        Dim rest As String
        position = Instr(1, AllStr, SearchStr)
        länge = Len(searchstr)
        vorne= Mid (allstr,1, position-1)
        hinten= Mid(allstr, position+länge)
        pos = Instr (1, hinten, SearchStr)

        If pos > 0 Then
                rest = ReplaceSubstring(hinten, SearchStr, ReplaceStr)
                hinten = rest
```

```
        End If

        ReplaceSubstring = vorne + ReplaceStr + hinten
End Function

Function GetArea(Index As String) As Integer
        Dim Position As Integer
        Dim TeilString As Variant
        Position = Instr(1,Index, ".")
        TeilString = Left$(Index, Position -1)
        GetArea = Cint(TeilString)
End Function
```

Anhang B

Literaturverzeichnis

[Behme 1998]	Henning Behme, Stefan Mintert „XML in der Praxis Professionelles Web-Publishing mit der Extensible Markup Language“ Addison-Wesley, 1998, Bonn
[Dennig 1996]	Jens Dennig, Klaus Gutperlet, Eike G. Rosenow „Lotus Notes R4 Das Kompendium“ Markt & Technik, 1996, München
[Dierker 1998]	Markus Dierker, Martin Sander „Lotus Notes 4.6 und Domino Integration von Groupware und Internet” Addison-Wesley-Longman, 1998, Bonn
[Goldfarb 1999]	Charles F. Goldfarb, Paul Prescod „XML – Handbuch“ Prentice Hall, 1999, Haar
[Helfrich 1999]	Christof Helfrich „Document Object Model Interoperabilität für dynamische Web-Seiten & XML-Anwendungen“ Seminar Dokumentenmanagement im WS 1998/99 Http://www2.informatik.uniwuerzburg.de/staff/wolfram/lehre/9899ws_semdok/dom/ausarbeitung/index.html
[IW 12/99]	Andreas Hitzig „Next HTML“ in Internet World Dezember 1999 Neue Mediengesellschaft, 1999, München

[iX 07/99]	Tim Weitzel, Ralf Kronenberg, Frank Ladner, Peter Buxmann „Die Rückkehr der EDI-Ritter“ in iX Magazin für professionelle Informationstechnik Ausgabe 07/1999 Heise, 1999, Hannover
[Janßen 1993]	Heike Janßen, Manfred Bundschuh „Objektorientierte Software-Entwicklung“ R. Oldenbourg, 1993, München
[Lamprecht 1999]	Stephan Lamprecht „Programmieren für das WWW mit JavaScript, VBScript, XML und SMIL“ Carl Hanser, 1999, München
[Lobin 2000]	Henning Lobin „Informationsmodellierung in XML und SGML“ Springer Verlag, 2000, Berlin
[Mann 1998]	Raimund Mann „Lotus Script Programmieren für Notes“ Computer und Literaturverlag, 1998, Vaterstetten
[Mann 1999]	Raimund Mann „Domino Designer R5 Anwendungsentwicklung mit Lotus Notes“ Computer und Literaturverlag, 1999, Vaterstetten
[Mattes 1999]	Frank Mattes „Electronic Business-to-Business E-Commerce mit Internet und EDI“ Schäffer-Poeschel, 1999, Stuttgart
[Pott 1999]	Oliver Pott, Gunter Wielage „XML Praxis und Referenz“ Markt & Technik, 1999, München

[Schröter 1998]	Uwe Schröter, Stephan Fügner „Das Domino Prinzip Dynamische Generierung interaktiver HTML-Dokumente mit Lotus Notes/Domino“ Dpunkt-Verlag, 1998, Heidelberg
[Seeboerger 2000]	Michael Seeboerger-Weichselbaum „Das Einsteigerseminar XML“ BHV, 1999-2000, Kaarst
[Tolksdorf 2000]	Robert Tolksdorf „HTML & XHTML Informationen aufbereiten und präsentieren im Internet“ Dpunkt, 2000, Heidelberg

Anhang C

Tabellen- und Abbildungsverzeichnis

Anhang D

Websites

[JavaScript 2000]	http://javascript.internet.com/messages/overlib.html
[Microsoft 2000]	http://www.microsoft.com/GERMANY/msdn/artikel/corner.htm
[Kyla 2000]	http://notes.net/today.nsf/lookup/xml_designer
	http://www.conet.de
	http://www.youatweb.com
	http://www.innovationgate.de
	http://www.lotus-dev.net/
	http://www.lnn.com/
	http://www.w3.org/TR/1999/REC-html401-19991224/
	http://www.w3.org/TR/REC-xml#dt-element
	http://www.w3.org/TR/xsl/
	http://www.w3.org/TR/xhtml1/
	http://msdn.microsoft.com/xml/default.asp
	http://msdn.microsoft.com/xml/xslguide/default.asp
	http://www.ct.heise.de/ix/raven/Web/xml/
	http://developerlife.com/
	http://www.devx.com/
	http://www.ibm.com/developer/
	http://www.xml.com/pub
	Http://www2.informatik.uniwuerzburg.de/staff/wolfram/lehre/9899ws_semdok/dom/ausarbeitung/index.html
	http://www.siteexperts.com/
	http://www.teamone.de/selfhtml/sfarchiv/
	http://webreview.com/pub/guides/style/style.html
	http://www.xmltree.com/
	http://www.mozilla.org
	http://www.xmlspy.com

Anhang E

Abkürzungsverzeichnis

CERN	Centre Europeen de Recherches Nucléaires (Schweizer Kernforschungszentrum)
CSS	Cascading Stylesheet
DFN	Deutsches Forschungsnetz e.V.
DHTML	Dynamic HTML
DOM	Document Object Model
EDI	Electronic Data Interchange
HTML	Hyper Text Markup Language
IETF	Internet Engineering Task Force (Die primäre Organisation für die Entwicklung von Internet-Standards)
INRIA	Institut de Recherche en Informatique et en Automatique (Französisches Institut für Entwicklungen in Computertechnik und Automation)
KMU	Klein- und mittelständische Unternehmen
LAN	Local Area Network
Markup	Auszeichnung
MIS	Management-Informations-System
MIT	Massachusetts Institute of Technology
PDA	Personal Digital Assistent
PDF	Portable Document Format
SGML	Standard Generalized Markup Language
Tag	Formatierungsanweisung
URL	Uniform Resource Locator
VAN	Value Added Network
W3C	World Wide Web Consortium
WAN	Wide Area Network
WAP	Wireless Application Protocol

XHTML	Extended HyperText Markup Language
XML	Extensible Markup Language
XQL	XML Query Language
XSL	Extensible Stylesheet Language

Anhang F

Glossar

Applikation	Anwendung, die ausgeführt werden kann.
Browser	Software zum Suchen und Darstellen von Web-Angeboten.
Business-to-Business	Einsatz von Datenverarbeitung und Telekommunikation, um Geschäfte zwischen Unternehmen zu unterstützen.
Button	Schaltfläche, die über Hyperlink eine Aktion auslöst.
Client	Ein Computer, der auf die Dienste eines Netzwerk-Computers, den Server, zugreift.
Content	„Inhalt“ eines Web-Angebots.
Electronic-Commerce	Nutzung elektronischer Datennetze, zum Zwecke des Vertriebs von Produkten.
Entity	Abkürzung, die innerhalb einer DTD definiert werden muss.
Event	Ein Ereignis, das eintreten kann.
Homepage	Elektronisches „Schaufenster“ eines Unternehmens bzw. einer Einzelperson
Internet	Internationales Netzwerk auf TCP/IP-Basis. Es besteht aus mehreren Diensten, darunter E-Mail, ftp, Web und Telnet.
Intranet	Unternehmens-Netzwerk auf TCP/IP-Basis.
Link	Direkt ausführbarer Verweis auf andere Dokumente, zum Beispiel im Internet; Kurzform von Hyperlink.
Pfad	Verzeichnisstruktur, in der das gesuchte Element gefunden werden kann.
Server	Ein Computer, der anderen Computern, den Clients, bestimmte Dienste anbietet.
Sitemap	Inhaltsverzeichnis zu einem Web-Auftritt.

Suchmaschine	Softwareprogramm zur Datensuche im Internet.
Web-Content-Management-System	Programmpaket zur Verwaltung und Speicherung von Web-Auftritten.

Anhang G

Inhalt der CD-Rom

Die CD-Rom, die dieser Diplomarbeit beiliegt, enthält folgende Verzeichnisse und Files:

Verzeichnis **"Diplomarbeit"**

- Diplomarbeit.doc
- Diplomarbeit.htm
- Diplomarbeit.pdf

Verzeichnis **"XML-Sitemap"**

- Sitemap.xml
- Sitemap.xsl
- Sitemap.css
- Sitemap.dtd
- Sitemap.js
- Back.jpg
- Sorry.html
- Readme.txt

Verzeichnis **"LotusNotes"**

- Sitemap.nsf

Verzeichnis **"Vortrag"**

- XML-Vortrag.ppt