

■ Bearbeitung von XML Dokumenten durch
dynamische XSLT Filter

Seminararbeit im 3. Studienjahr

Am Institut für Informatik der Universität Freiburg i.Ue.

Vorgelegt bei:
Prof.Dr.A.Meier

Betreuer:
Hüsemann Stefan

Eingereicht von:
Müller Roman
Avenue du Midi 7
1700 Freiburg
roman.mueller@unifr.ch
026 424 06 94

Inhaltsverzeichnis

1	Einleitung	3
1.1	Problemstellung	3
1.2	Zielsetzung	4
2	Verwendete Technologien	5
2.1	Markupsprachen	5
2.2	XML	6
2.3	XSL	7
2.3.1	XSL Transformation (XSLT)	8
2.3.2	XSL Formatting Objects (XSL-FO)	9
2.3.3	XSL Path Language (XPath)	9
2.4	ASP	10
3	Arbeitsrapport	12
3.1	Grundstruktur	12
3.1.1	reduzieren.asp	12
3.1.2	reduzieren.xsl	14
3.1.3	formread.asp	14
3.1.4	filtergen.xsl	15
3.2	Beispiel der Erstellung eines Filters für IDML Dokumente	16
4	Schlussbemerkungen	18
A	Xsl Stylesheets	20
A.1	Stylesheet reduzieren.xsl	20
A.2	Stylesheet filtergen.xsl	22
B	ASP Seiten	24
B.1	ASP reduzieren.asp	24
B.2	ASP formread.asp	25

Abbildungsverzeichnis

2.1	XML Dokument Beispiel	6
2.2	XSD Dokument Beispiel	7
2.3	Unterteilung von XSL in drei Subkategorien	8
2.4	Anwendung eines XSL Stylesheets auf ein XML Dokument	9
2.5	ASP Beispielcode	10
3.1	Prozessschritte der Anwendung	13
3.2	Zeichenkette Elements	14
3.3	Formular der Elementauswahl	16
3.4	Das XSLT Filter Stylesheet	17

Kapitel 1

Einleitung

1.1 Problemstellung

Die Gesellschaften des 21. Jahrhunderts werden sich immer stärker durch eine netzwerkartige Differenzierung auszeichnen, die vor allem auf der digitalen Informationstechnologie basiert. Die Geschwindigkeit dieses sozio-technologischen Wandels stellt Politik, Wirtschaft und Kultur vor neue Herausforderungen. Sowohl innerhalb der entwickelten Gesellschaften als auch in der sogenannten Dritten Welt.

Ein in diesem Zusammenhang oft verwendeter Ausdruck, die digitale Spaltung, steht symbolisch für eine Problematik mit hoher gesellschaftlicher Brisanz. Gemeint ist zunächst die technologische Kluft zwischen denen, die Vernetzt sind, und jenen, die aus unterschiedlichen Gründen ausgeschlossen sind. Die jüngsten sozio-technologischen Veränderungen, haben schleichend einen gewaltigen Wandel mit sich gebracht. Ein Handy ist heute zur gesellschaftlichen Integration eine Notwendigkeit. Die Kluft bei uns mag vielleicht marginal erscheinen, ist aber vorhanden. Der Verdacht liegt nahe, dass beim Zugang zu den neuen Informationstechnologien eine noch deutlichere Kluft zwischen den industrialisierten Ländern des Nordens und den Ländern der Dritten Welt besteht.

Die humanitären Organisationen sehen in den neuen Informations- und Kommunikationstechnologien einen wichtigen Schritt zu wirtschaftlichem und sozialem Fortschritt der unterentwickelten Ländern. Eine Schlüsselrolle spielen dabei Computernetze wie das Internet.

Für die Organisationen, welche an der Trennlinie zwischen vernetzter und nichtvernetzter Welt arbeiten, sind neue Informationstechnologien von eminenter Bedeutung. Ein wichtiger Schritt für einen effizienten Einsatz der Ressourcen, ist der Austausch von Informationen. Informationen sind jedoch oftmals inhomogen und von komplexer Natur.

Der Datenaustausch zwischen Rechensystemen wird heute meist proprietär realisiert. Es kommen keine einheitlichen Formate und Strukturen zum Einsatz. Um dies zu vereinfachen bedarf es flexibler Datenbeschreibungssprachen, welche eine ausreichende Strukturierungsmöglichkeit bieten und plattformunabhängig sind. Die treibenden Kräfte sind Markupsprachen. Insbesondere XML hat sich als Standard zur Datenbeschreibung etabliert.

1.2 Zielsetzung

Diese Arbeit soll hauptsächlich einen Einblick in XML, XSLT und ASP geben. Diese Technologien werden zunehmend verwendet um den Datenaustausch zwischen Rechensystemen zu realisieren. Eine Einführung in die Markupssprachen bildet die Grundlage zum Verständnis von XML und seiner Teilmenge XSLT.

Auf der Basis dieser Technologien wird ein Datenfilter für XML Dokumente erstellt. Mit dem Filter können Elemente selektiv aus XML Dokumenten entfernt werden. Die Ausgangslage bilden beliebige XML Dokumente, welche dem Nutzer zur Verfügung gestellt werden. Anhand der Elementauswahl des Benutzers, wird mit XSLT ein XSLT Stylesheet generiert. Das generierte XSLT Stylesheet ist der Filter für die XML Dokumente.

Da die Filteranwendung vom Browser unabhängig zu gestalten ist, wird ASP zum Handling der Dokumente und zur Steuerung der Anwendung verwendet. Dies hat den Vorteil, dass man eine klare Kontrolle der Systemumgebung hat und somit Probleme mit Inkompatibilitäten vermieden werden können. Eine kurze Einführung in die Funktionsweise von ASP erleichtert das Verständnis des Beispiels.

Kapitel 2

Verwendete Technologien

2.1 Markupsprachen

Die ersten Ideen zum Konzept des Hypertexts, als Plan zur Überwindung der Beschränkungen und Unzulänglichkeiten des klassischen textbasierten Publikationsmediums Papier datieren zurück bis in die 1950er Jahre. Sie postulieren neben der nichtsequentiellen Organisation des Mediums auch zentrale Begriffe wie Knoten, Link, Anker und Netz. Ziel dieser Überlegungen war es, den auszudrückenden Inhalt von editorielle- und präsentativer Information wie Seitenzahlen, Fußnoten, Paginierung usw. zu trennen. Durch die nichtlineare Organisation soll es dem Leser freigestellt werden, auf welchen Pfaden er sich durch das Dokument bewegt.

Zur Realisierung dieser Bemühungen wird das Dokument mit weiteren Informationen angereichert, die jedoch für den Leser unsichtbar bleiben. Dieser Gedanke reicht zurück bis in die Anfänge des Buchdrucks. Dort sind formatierungsorientierte Auszeichnungssymbole, etwa für Fettdruck oder Unterstreichung, seit jeher bekannt. Vor dem Aufkommen der “what you see is what you get“ (WYSIWYG) Textverarbeitungssysteme waren diese bildlichen Symbole die einzige Möglichkeit zur Kommunikation präsentationsorientierter Information an den Schriftsetzer und Drucker. Jedem Schüler ist bereits ein weiteres Beispiel einer editorielle Auszeichnungssprache bekannt: Die graphischen Korrekturzeichen der Deutschlehrer. Auch sie liefern Informationen über den Inhalt, die nicht Bestandteil des Dokuments sind.

Die Markupsprachen haben ihren Ursprung im Computerschriftsatz. Die ersten Textverarbeitungssysteme basierten auf dem Prinzip der Beschreibung der Formatierung. Der Verfasser muss mit Befehlen beschreiben, wie er sein Dokument formatieren will. Dies nennt man deskriptives Markup.

Die heutigen moderne Textsysteme verwenden jedoch WYSIWYG, prozedurales Markup. Dieses hat gewisse Nachteile. Mit dem Formatieren eines Textes mit einem Textsystem erhält man Abhängigkeiten von demselben. Das Übertragen von Informationen von einem System auf das andere erzeugt oftmals Inkompatibilitäten. Ein Versuch den Problemen entgegenzuwirken, liegt in der Einbindung von Import- und Exportfiltern in Textverarbeitungssystemen. In der Praxis gibt es dennoch oftmals Probleme, selbst zwischen verschiedenen Versionen eines Systems.

Um diesem Problem entgegenzutreten, ist das deskriptive Markup gut geeignet. Denn es weist einer Textpassage nicht einen spezifischen Schriftsatz oder Grösse zu, sondern eine Rolle. Ein Titel wird zum Beispiel einfach als Überschrift ersten Grades identifiziert. Aus dieser Vorgehensweise ergeben sich einfach zu erkennende Vorteile. So können zum Beispiel Überschriften als Inhaltsverzeichnis verarbeitet werden, oder Zitate können zur Generierung einer Bibliografie verwendet werden.

Mit der Standard Generalized Markup Language (SGML) hat die International Organization for Standardization (ISO) einen Standard definiert. Der ISO-Standard definiert SGML als eine Sprache für Dokumentrepräsentation, welche Markup formalisiert und von System- und Verarbeitungsabhängigkeiten löst (vgl. [Pag02]). SGML erlaubt den Austausch von großen und komplexen Datenmengen und vereinfacht den Zugriff auf sie. Zusätzlich zu den Möglichkeiten des deskriptiven Markups benutzen SGML-Systeme ein Dokumentenmodell, welches die Überprüfung der Gültigkeit eines Textelementes in einem bestimmten Kontext erlaubt. Weiterhin enthält SGML Techniken, welche den Benutzer folgende Dinge erlauben: Zusammenbindung von Dateien, zur Erstellung eines zusammenhängenden Dokuments; Einbindung von Illustrationen und Tabellen in Textdateien; Erzeugen von mehreren Versionen eines Dokuments in einer einzigen Datei; Hinzufügen von Kommentaren in die Datei; Kreuzreferenzen.

2.2 XML

XML ist eine vereinfachte Version von SGML [GP99]. Sie beinhaltet die nützlichsten Teile von SGML. Weggelassen wurden die weniger gebräuchlichen und die sehr komplizierten Teile. XML ist eine Metasprache zur Definition von Dokumenttypen. Dokumente, die ähnlich strukturiert sind, gehören demselben Dokumenttyp an. Beispielsweise gehören alle HTML Dokumente dem Dokumenttyp HTML an. Die Idee, dass die Dokumente alle in ihrem Aufbau gewissen Grundmustern folgen, vereinfacht das Handling ungemein. Wenn bei der Erstellung von Dokumenten die Muster eingehalten werden, so sind sie viel allgemeiner einsetzbar, als wenn jedes Dokument eigenen Regeln folgt. Dies ermöglicht es Programme zu schreiben, die die Dokumente automatisch verarbeiten.

```
<?XML version="1.0"?>

  <adresse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <name>Hans Muster</marke>
    <strasse>Musterstrasse</strasse>
    <plz>9999</plz>
    <ort>Musterhausen</ort>
  </adresse>
```

Abbildung 2.1: XML Dokument Beispiel

XML wurde konzipiert, um Daten einfach zu strukturieren. Ganz bewusst wurde die Frage der Darstellung im Sinne von Visualisierung und Formatierung in der XML-Spezifikation nicht bedacht. XML beansprucht dennoch, im

Gegensatz zu binären Formaten, ein von Menschen lesbares Format zu sein. Die Abbildung 2.1 ist ein einfaches Beispiel eines XML Dokumentes.

Zur Definition der Datenstruktur der verschiedenen XML Dokumenttypen wird XSD (XML Schema Definition) verwendet. XSD spielt in XML eine wichtige Rolle. In einer XSD wird die Datenstruktur festgelegt. Zum Beispiel wird festgelegt, welche Elementtypen in den Dokumenten verwendet werden können. Die XSD liefert außerdem Vorgaben dafür, wie Elemente ineinander verschachtelt werden können, und es wird auch angegeben, welche Attribute zu welchen Elementen gehören und welche Attributwerte jeweils zulässig sind. XSD haben grundsätzlich zwei Funktionen: Sie definieren die Strukturen der Dokumenttypen und sind die Referenz zur Programmierung.

Damit ein XML-Parser auf eine XSD zugreifen kann, muß sie entweder auf dem lokalen Rechner verfügbar sein, oder aber sie muß über das Netz abrufbar sein. Im Gegensatz zu SGML erlaubt XML auch Dokumente, die keinen Verweis auf ein Schema enthalten. Stattdessen prüft der XML Parser beim Einlesen das Dokuments auf Wohlgeformtheit.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="adresse">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="strasse" type="xsd:string"/>
      <xsd:element name="plz" type="xsd:decimal"/>
      <xsd:element name="ort" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="land" type="xsd:NMTOKEN" fixed="DE"/>
  </xsd:complexType>
</xsd:schema>
```

Abbildung 2.2: XSD Dokument Beispiel

Das in Abbildung 2.2 dargestellte XSD ist das Schema für das Dokument in Abbildung 2.1 .

2.3 XSL

XSL steht für Extensible Style Language . Wie bereits dargelegt, ist XML darstellungsneutral. In der maschinellen Verarbeitung ist die Präsentation der Daten nebensächlich. Für den Informationsnutzer spielt dagegen die Darstellung eine grosse Rolle. Für diesen Zweck braucht man eine Formatierungssprache, mit der festgelegt werden kann, wie ein XML-Dokument präsentiert werden soll. Seit einigen Jahren ist das CSS (Cascading Style Sheet) Standard zur Formatierung von HTML. Es stellt ein Set von Formatierungsregeln zur Verfügung. XML Dokumente können CSS grundsätzlich verwenden, allerdings werden mit CSS nur ein Teil der Möglichkeiten von XML genutzt.

Eine eigene Formatierungssprache für XML ist der XSL-Standard [W3C02]. Er wurde 1998 vom World Wide Web Consortium als XML Stilsprache freigegeben. Aufgrund des breiten Anwendungsspektrums und der langen Liste von Wünschen an die Leistungsfähigkeit von XSL hat es sich als sinnvoll erwiesen,

XSL in drei Teile zu spalten, die verschiedene funktionale Bereiche abdecken [Dev02].

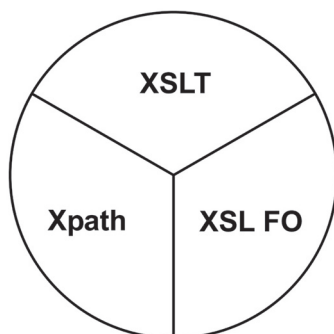


Abbildung 2.3: Unterteilung von XSL in drei Subkategorien

Neben der Umwandlung strukturierter Daten durch XSLT zählt zu XSL auch der Bereich Formatting Objects, die zum Beispiel für Printformate verwendet werden. Der dritte wichtige Teil von XSL ist die XML Path Language, kurz XPath.

2.3.1 XSL Transformation (XSLT)

Eine zentrale Aufgabenstellung bei der Erzeugung eines Endbenutzerdokumentes ist die Überführung der Struktur des Ausgangsdokumentes in die Syntax des Zielformats. Diese Überführung ist Aufgabe von XSLT [Kay01].

XSLT ist eine Programmiersprache, die speziell für die Transformation von XML Dokumenten geschaffen wurde, und ist aus diesem Grund in den meisten Fällen geeigneter als Sprachen wie C oder Java. XSLT beruht auf dem Paradigma der funktionalen Programmierung [Lou94]. Dies macht ein Umdenken in Design und Programmiermethodik erforderlich.

Die Abbildung 2.4 zeigt, wie eine XML Dokument mit einem XSLT Stylesheet vom XSLT Prozessor verarbeitet wird. Das Ergebnis der Verarbeitung kann ein XML-, HTML-, oder Textformat sein. Die Verarbeitung durch den XSLT Parser ist nicht prozedural, die Elemente werden also nicht sequenziell nach ihrem Auftreten im Dokument bearbeitet, sondern sowohl das XSLT Dokument, als auch das XML Dokument werden vom Parser zu DOM Bäumen geparsed. Dieser Ansatz ermöglicht es zu jedem Zeitpunkt der Laufzeit auf alle Elemente und Daten zuzugreifen, und nur das ermöglicht letztendlich umfangreiche Formatierungen.

Der Parser akzeptiert jedes beliebige DOM Dokument und kann somit jeder anderen Anwendung DOM Dokumente weitergeben. Der Prozessor hat während der Ausführung zwei DOM Dokumente gespeichert. Das XML Dokument ist als Baumstruktur mit seinen Elementen verfügbar, das XSLT Stylesheet besteht analog dazu aus einer Baumstruktur der Formatierungsanweisungen. Diese Anweisungen sind sogenannte Template Rules. Wenn der Parser auf eine Regel stößt, durchsucht er das gesamte XML-Dokument nach Elementen, die auf das Suchmuster,- auch Pattern genannt - passen. Sofern der Parser ein Element, auf

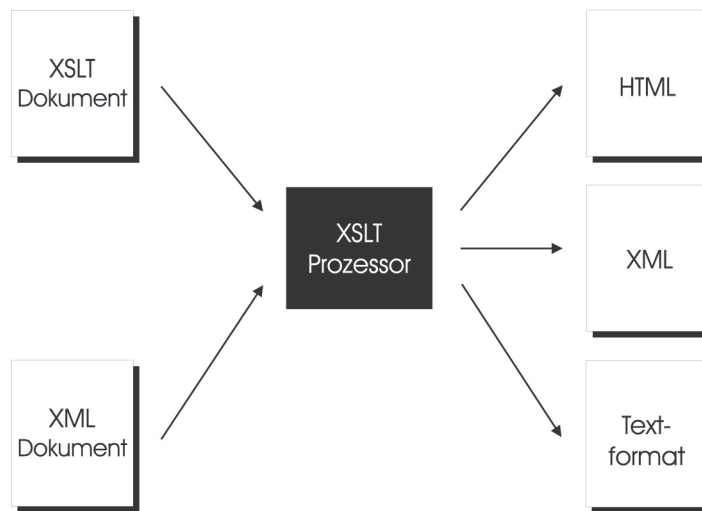


Abbildung 2.4: Anwendung eines XSL Stylesheets auf ein XML Dokument

das der Pattern zutrifft, findet, fügt er das Template dem Ergebnisbaum hinzu. Der ganze Verarbeitungsprozess besteht aus einem Suchen von Elementen und einem Anfügen der entsprechenden Templates an den Ergebnisbaum. Um Daten in das Ergebnis zu übernehmen, müssen diese explizit angegeben und verarbeitet werden. Wenn für Elemente des bearbeiteten XML Dokumentes keine Templates vorhanden sind, kommen sie im Output auch nicht vor.

2.3.2 XSL Formatting Objects (XSL-FO)

Ein Parser wandelt wie im vorherigen Abschnitt beschrieben ein XML Dokument in ein Ergebnisbaum um. Dieser Ergebnisbaum wird in einem zweiten Schritt auf ein Medium ausgegeben. Dieser Formatierungsprozess ist unabhängig vom Transformationsprozess und wird durch einen sogenannten Formatter durchgeführt. Mit Hilfe der Objektformatierungsspezifikationen kann das Layout einer Seite bestimmt werden [W3C02]. Die Spezifikationen unterstützen sowohl Seitenbeschreibung von Browserdokumenten wie auch von Drucksachen. XSL-FO wird nicht weiter ausgeführt, da sie nicht Bestandteil der Anwendung in Kapitel 3 ist.

2.3.3 XSL Path Language (XPath)

XPath bietet die Möglichkeit, DOM Bäume zu durchsuchen und einzelne Elemente (oder Gruppen von Elementen) in Dokumenten vom Typ Text oder XML, zu adressieren. XPath ist nicht direkt anwendbar, sondern dient als Grundlage für weitere Anwendungen. Alle Suchmuster in XSLT sind XPath Syntax.

Die Adressierung der einzelnen Elemente des DOM Baums erfolgt bei XPath mittels sogenannter Location Paths. Location Paths wählen nach bestimmten Kriterien Knoten aus dem Baum aus. So können sehr differenziert Teile des Baums selektiert werden. Ein Location Path wird immer innerhalb eines vorher definierten Kontextes ausgewertet. Zum Kontext gehören:

- ein Kontext- oder Ausgangsknoten
- eine Kontextgrösse, die angibt, aus wievielen Knoten ausgewählt wird
- eine Kontextposition, die die Position des Kontextknotens angibt
- eine Menge an definierten Variablen
- eine Menge an Funktionen des im Kontext gültigen Namespaces.

Zur Unterstützung dieser Hauptfunktion unterstützt XPath auch Grundfunktionen zur Manipulation von Zeichenketten, Nummern und Booleans. XPath benutzt einen kompakten, nicht XML Syntax. Dies erleichtert die Nutzung innerhalb von Attribut Werten.

2.4 ASP

ASP steht für Active Server Pages (Aktive Server-Seiten) und ist eine extrem mächtige Erweiterung der Möglichkeiten von Websites [Wel99]. Am Anfang des WWW bestand Sites nur aus einzelnen statischen Seiten, die untereinander verlinkt waren. Was ihnen jedoch fehlte war die Interaktivität. Es entstanden mehrere Systeme, die zwar nicht mit HTML zusammen arbeiten, aber den Webserver um wichtige Funktionen erweitern. Das bekannteste System ist CGI, eine genormte Schnittstelle zwischen dem Webserver und den CGI-Programmen.

Die Lösung dieser schwerfälligen Implementation von Funktionen ist ASP. ASP ist keine Programmier- oder Scriptsprache, sondern eine Technologie von Microsoft, welche es erlaubt, serverseitiges Scripting direkt in HTML Dateien einzubinden. Die am häufigsten verwendeten Scriptsprachen sind VBScript und JScript. Man kann jedoch im Prinzip jede beliebige Sprache verwenden.

ASP wird benutzt um beispielsweise Suchtreffer einer Datenbankabfrage an den aufrufenden Browser zu senden, Cookies zu schreiben und zu lesen oder wie in Kapitel drei DOM Bäume zu manipulieren. Es sind nicht wie bei CGI separate Scripts erforderlich, die in speziellen Verzeichnissen abgelegt werden.

ASP ist ein Produkt von Microsoft und somit für den Web Server von Microsoft optimiert. Es existieren Implementationen auf dem Apache Server, verwendet wird ASP in der Regel aber auf dem Internet Information Server von Microsoft.

```
<html>
<head>
<title>Die aktuelle Zeit</title>
</head>
<body>
<Es ist jetzt <% = Time %><p>
</body>
</html>
```

Abbildung 2.5: ASP Beispielcode

In Abbildung 2.5 sieht man, wie ASP-Code in eine HTML-Seite eingebunden wird. Alles was zwischen den `<% %>` steht, wird als ASP-Code interpretiert und auf dem Server ausgeführt.

Bei herkömmlichen HTML Seiten beschränkt sich der Webserver auf reine File Server Funktionen. Wird eine Seite angefordert, so liefert der Server via HTTP-Protokoll die HTML-Datei an den Browser des Anwenders. Bei ASP Seite wird dagegen einiges mehr verarbeitet [Kra99].

1. Der Browser fordert eine ASP-Seite an.
2. Der Webserver öffnet die ASP-Seite und liest den Inhalt.
3. Die HTML Blöcke werden unverändert an den Browser geschickt.
4. Findet der Server jedoch ASP-Skriptblöcke, dann führt er den Programmcode aus. Das Ergebnis wird in den HTML-Strom integriert.

ASP-Code ermöglicht jedoch nicht nur den HTML-Inhalt dynamisch anzupassen, er kann generell steuern, welche Daten an den Client geschickt werden. Überdies ist ASP in der Lage nahezu alle Ressourcen eines Webserver einzubinden.

Kapitel 3

Arbeitsrapport

3.1 Grundstruktur

Wie in der Zielsetzung beschrieben, ist ein dynamischer Datenfilter für XML Dokumente gesucht. Der Filter kann unerwünschte Elemente in einem XML Dokument entfernt. Zunächst wird ein Überblick über den Aufbau der gesamten Anwendung gegeben (Abbildung 3.1). Anschliessend werden die einzelnen Teile näher betrachtet.

Der Ausgangspunkt für den Filter ist ein beliebiges XML Dokument, mit zumindest allen gewünschten Elementen. Das ASP `reduzieren.asp` liest das XML Dokument und das Stylesheet `reduzieren.xsl` als DOM Objekte ein und übergibt diese dem MSXML Parser. Das Resultat des Parsers wird als Objekt dem ASP zurückgegeben. Es ist eine Zeichenkette aller Elemente und ihrer Hierarchieebenen in der richtigen Reihenfolge. Doppelte Elemente werden entfernt. Die Zeichenkette wird umgeformt und die Elemente werden der Reihe nach ausgelesen. Für jedes Element wird eine Checkbox erstellt. Entsprechend ihrer Ebene werden sie in der Ausgabe eingerückt. Die Ausgabe ist ein Formular, welches zum Client gesendet wird.

Mit dem Formular wird auf dem Client der Filter zusammengestellt. Als Vorgabe sind alle Elemente markiert. Um ein Element wegzufiltern muss es demarkiert werden. Wenn ein Element demarkiert ist, werden ebenso die Kind-elemente, auch wenn sie markiert sind, weggefiltert.

Die Formulardaten übernimmt das ASP `formread.asp`. Es überprüft zu Beginn, ob die Daten zulässig sind. Ist dies nicht der Fall, wird zum Formular zurückgekehrt, ansonsten werden die Formulardaten einem DOM Objekt übergeben. Das ASP liest das Stylesheet `formgen.xsl` als Objekte ein und übergibt es mit dem Objekt der Formulardaten an den MSXML Parser. Als Resultat erhält man eine Datei vom Typ `xsl`, welche den eigentlichen Filter darstellt.

Die XSL Datei kann auf alle XML Dokumente vom Typ der Ausgangsdatei angewendet werden.

3.1.1 `reduzieren.asp`

Das ASP Dokument hat die Aufgabe die Ausgangsdaten einzulesen und zu transformieren. Das Resultat ist ein Formular, welches zum Client geschickt wird.

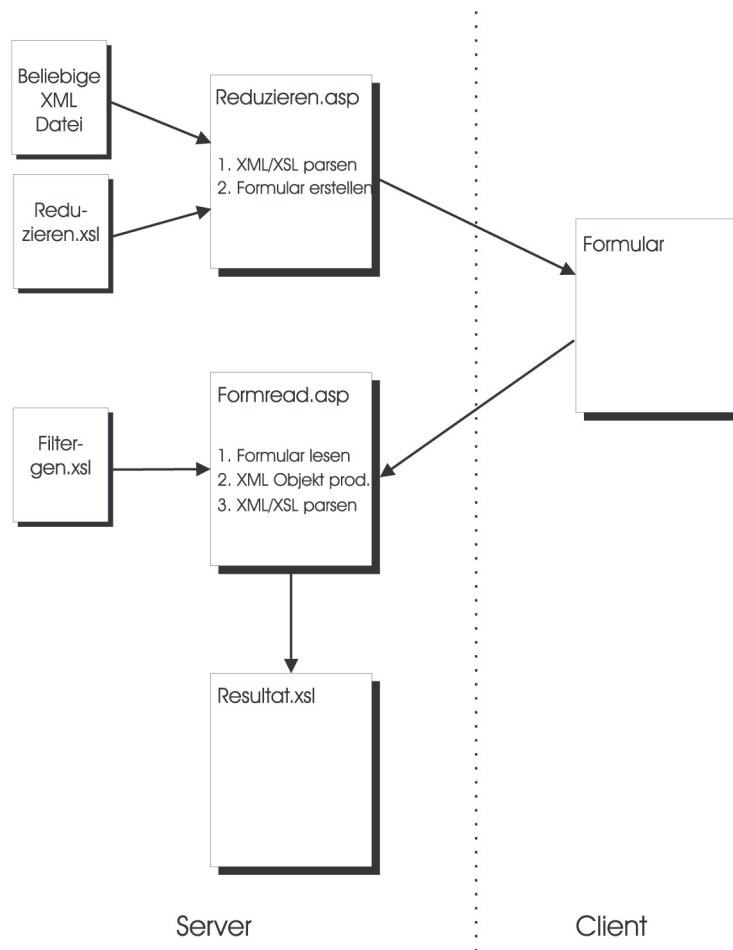


Abbildung 3.1: Prozessschritte der Anwendung

Im ersten Scriptblock des Dokumentes (Anhang B.1) werden Formulardaten gelesen und zugewiesen. Zur Benutzer- und Sessionidentifikation sind dies die Variablen ID und Session ID. Der Pfad des XML Dokumentes wird als XMLFile abgelegt. Die benötigten DOM Objekte werden erstellt. Mit dem Befehl load werden XSL und XML Dokument in das entsprechende Objekt geschrieben. Falls die gelesenen Daten nicht valid sind, wird ein Fehler ausgegeben.

Der Befehl transformNode übergibt dem MSXML Parser das xmldoc und das xsl doc Objekt und gibt das Resultat als eine Zeichenkette zurück. Mit den nachfolgenden Befehlen wird der relevante Kettenabschnitt definiert.

Die HTML Tags fließen unverändert in den Outputstream. Mit den verdeckten Formularfeldern werden die später benötigten Variablen mitgegeben. Im nachfolgenden Abschnitt wird mittels einer Schleife für jedes Element eine Checkbox, welche entsprechend der Ebene eingerückt ist, erstellt.

3.1.2 reduzieren.xsl

Reduzieren.xsl ist ein XSLT Stylesheet zur Transformation von XML Dokumenten. Wenn man im Anhang A.1 die Struktur des Stylesheets betrachtet, erkennt man die in der Theorie beschriebene Templatestruktur. Im Body des Tags Stylesheet sind drei Templates. Das erste Template, welches keinen Namen, sondern nur ein match besitzt, wird, als erstes bearbeitet. Denn der Parser sucht im XSL Baum immer das erste Template, welches auf ein XML Knoten passt. In diesem Fall wurde der ganze Baum selektiert.

Zum weiteren Verständnis des Codes muss das Konzept der funktionalen Programmierung näher betrachtet werden. Die funktionale Sicht macht keinen Unterschied zwischen einem Programm, einer Prozedur oder einer Funktion. Sie unterscheidet jedoch immer zwischen Eingabe- und Ausgabewerten. Alle Operationen können als Funktion betrachtet werden. Im Falle eines Programms ist x als Eingabe und y als Ausgabe zu sehen. Eine der grössten Umstellungen zu sequenziellen Sprachen: Es existiert keine Speicherzuweisung. Somit muss das Konzept der Variable, ausser als Namen für einen Wert, eliminiert werden.

Der Befehl “xsl:variable“ sollte somit verständlich sein. Dem Namen elements wird der Resultatwert des Rumpfes zugewiesen. Im Rumpf wird, mit den ineinander geschachtelten for-each Befehlen, der XML Baum durchlaufen. Der Selektor von for-each wählt jeweils alle Kindelemente aus. Hat ein Element einen Wert, so wird dieser mit dem value-of Befehl ausgegeben. Der gesamte Rückgabewert ist eine zusammengesetzte Zeichenkette aus den Ebenen, den Elementen und Trennzeichen (Abbildung 3.2).

§Elements

```
1,activities;2,activity;3,ID;4,assignOrg;3,ID;4,assignOrg;4,uniqID;...
```

Abbildung 3.2: Zeichenkette Elements

Mit dem Parameter “elements“ wird das Template sort aufgerufen. Das Template nimmt das erste Element der Zeichenkette und vergleicht, ob im Rumpf der Zeichenkette das Element nochmals vorhanden ist. Ist dies der Fall, so wird das erste Element mit der gefundenen Sequenz ersetzt. Eine Sequenz besteht aus einem Element und seinen nachfolgenden Subelementen. Die Funktion wird rekursiv aufgerufen bis die Zeichenkette durchgearbeitet ist. Die sortierte Kette wird dem Template output übergeben. Die einzige Funktion des Templates output ist, eine Zeichenkette auszugeben.

3.1.3 formread.asp

Mit dem ASP Dokument werden die Formulardaten zurückgelesen. In einem ersten Schritt wird überprüft ob die selektierte Elementkombination korrekt ist. Ist dies nicht der Fall, so wird die Zeichenkette korrigiert.

Die Funktion Merge generiert als erstes zwei Dom Objekte. Der DOM Baum des XSL Dokumentes wird in das Objekt geschrieben. Die Baumstruktur des zweiten Objektes muss mit ASP geschrieben werden. Sie enthält die korrigierte

Zeichenkette. Die Objekte werden geparkt und als Resultat erhält man den fertigen ASP Filter in Form eines XSL Stylesheets.

3.1.4 filtergen.xsl

Mittels dreier Templates wird ein XSL Output generiert, der eigentliche Filter.

Mit dem ersten Template "root" wird dem Namen "string" der Wert des Elementes "string" übergeben. Nachfolgend wird der Kopf des XSL Filters geschrieben und das Template "einstieg" mit dem Parameter "string" aufgerufen.

Das Template "einstieg" liest die Ebene des ersten Elements. Falls im Rumpf ein weiteres Element mit der gleichen Ebene existiert, wird die Zeichenkette bei diesem Element geteilt. Mit dem zweiten Teil wird das Template rekursiv aufgerufen, bis kein weiteres Element mit der gleichen Ebene existiert. Der erste Teil, bis zu einem Element mit gleicher Ebene, wird dem Template "bearbeitet" übergeben.

In diesem Template wird getestet, ob die übergebene Zeichenkette aus einem oder mehreren Elementen besteht. Falls die Kette nur ein Element besitzt, werden mit "out:" XSL Befehle in den Output geschrieben. Die XSL Befehle müssen nicht geschachtelt werden, da eine Zeichenkette mit nur einem Element, ein Blatt eines Dokuments repräsentiert. Wenn eine Zeichenkette mehrere Elemente hat, wird das Template "einstieg" mit den Subelementen als Parameter aufgerufen. Der Aufruf wird mit mehreren Outbefehlen umschlossen. Damit erreicht man eine Schachtelung.

3.2 Beispiel der Erstellung eines Filters für IDML Dokumente

The screenshot shows a web browser window titled 'reduzieren.xsl - Microsoft Internet Explorer'. The main content area is titled 'Auswahl der Knoten' and contains the following text: 'Die Knoten die sie nicht markieren, werden weggefiltert, ebenso ihre Kindelemente.' Below this text is a list of checkboxes for selecting XML nodes. The checked elements are: activities, activity, ID, title, personInvolved, assignOrg, uniqID, termsAssist, person, startDate, and endDate. The unchecked elements are: funding, fundingOrg, and role. At the bottom of the form is a button labeled 'Uebergabe'.

Abbildung 3.3: Formular der Elementauswahl

Auf der Basis eines einfachen IDML-Dokumentes wird ein Filter erstellt. Das Dokument enthält nur wenige Elemente, ist aber vollständig und gültig.

Auf der Startseite kann mittels eines Pull Down Menüs das XML Dokument ausgewählt werden. Durch drücken des Buttons "Übergeben" startet sich die Anwendung. Im Hintergrund bereitet die ASP Seite `reduzieren.xsl` das Formular (Abbildung 3.3) auf. Die genaue Funktionsweise ist in Kapitel 3.1 erläutert.

Das Formular hat für jedes Element eine Checkbox. Entsprechend der Reihenfolge und der Hierarchieebene sind die Elemente angeordnet. Die unerwünschten Elemente können demarkiert werden. Im Beispiel sind die Elemente `assignOrg`, `funding`, `fundingOrg` und `role` nicht ausgewählt. Sie werden somit weggefiltert.

Durch abschicken des Formulars, wird die Auswahl dem Server übergeben. Das `formread.asp` liest die Formulardaten und überprüft die Auswahl auf ihre Gültigkeit. In diesem Beispiel erscheint die Meldung, dass das Element `termsAssist` zusätzlich weggefiltert wird. Da das Oberelement `funding` nicht markiert wurde, wird automatisch auch das Unterelement mitgefiltert. Es besteht jedoch die Möglichkeit zur Elementauswahl zurückzukehren. Falls die Auswahl in Ordnung ist, kann dem neuen Filter ein Namen gegeben werden. Das Resultat ist ein XSLT Stylesheet, das auf dem Server gespeichert wird.

Im Resultat, dem XSLT Stylesheet in Abbildung 3.4, ist ersichtlich, dass für

```

<?xml version="1.0" encoding="UTF-16" ?>
- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes" />
  <!-- ***** -->
  <!-- Stylesheet generiert mit filtergen.xsl -->
  <!-- ***** -->
- <xsl:template match="/">
- <xsl:for-each select="child::node()[name()='activities']">
  - <xsl:copy>
    <xsl:copy-of select="@*" />
  - <xsl:for-each select="child::node()[name()='activity']">
    - <xsl:copy>
      <xsl:copy-of select="@*" />
    - <xsl:for-each select="child::node()[name()='ID']">
      - <xsl:copy>
        <xsl:copy-of select="@*" />
      - <xsl:for-each select="child::node()[name()='uniqID']">
        - <xsl:copy>
          <xsl:copy-of select="@*" />
          <xsl:copy-of select="text()" />
        </xsl:copy>
      </xsl:for-each>
    </xsl:copy>
  </xsl:for-each>
  <xsl:for-each select="child::node()[name()='title']">
  - <xsl:copy>
    <xsl:copy-of select="@*" />
    <xsl:copy-of select="text()" />
  </xsl:copy>
  </xsl:for-each>
  <xsl:for-each select="child::node()[name()='personInvolved']">
  - <xsl:copy>
    <xsl:copy-of select="@*" />
  - <xsl:for-each select="child::node()[name()='person']">
    - <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:copy-of select="text()" />
    </xsl:copy>
  </xsl:for-each>
  - <xsl:for-each select="child::node()[name()='startDate']">
    - <xsl:copy>
      <xsl:copy-of select="@*" />
      <xsl:copy-of select="text()" />
    </xsl:copy>
  </xsl:for-each>
  - <xsl:for-each select="child::node()[name()='endDate']">
    - <xsl:copy>

```

Abbildung 3.4: Das XSLT Filter Stylesheet

jedes Element ein Kopierbefehl eingefügt ist. Falls ein Element ein Blatt ist, also keine Kindelemente besitzt, hat es einen Befehl zum Kopieren des Elementwertes und seiner Attribute. Hat das Element jedoch Subelemente, so wird nur ein Befehl zum Kopieren der Attribute zu finden sein.

Wird dieser Filter nun auf ein Dokument des Typs IDML angewendet, so werden nur die gewünschten Elemente kopiert.

Kapitel 4

Schlussbemerkungen

Der Dokumententyp IDML ist ein wichtiger Schritt zur Vereinheitlichung des Datenaustausches zwischen humanitären Organisationen. Die Strukturierung der Daten ermöglicht eine bessere Nutzung der Ressourcen und erleichtert dadurch die Zusammenarbeit zwischen verschiedenen Organisationen. Die vorliegende Arbeit erlaubt es, auf einfache Weise, Elemente aus einem Dokument zu entfernen. Ohne Kenntnisse von XML oder XSL produziert der Benutzer webbasiert ein XSL Stylesheet als Filter.

Das Entwicklungsziel war ein Filter für IDML. Es war jedoch möglich die Stylesheets flexibel zu gestalten, so dass alle XML Dokumente bearbeitet werden können.

Erst mit den genaueren Kenntnissen war es möglich, die Struktur des Filters zu definieren. Das Studium verschiedener Technologien zeigte, dass XSL am besten geeignet ist um Manipulationen an XML Dokumenten durchzuführen. XSL hat jedoch gewisse spezifische Eigenschaften der funktionalen Programmierung, welche ein Umdenken in der Programmierlogik erfordern.

Durch eine gezielte Kombination von verschiedenen Technologien erhält man eine relativ einfach strukturierte Anwendung. Mit XSL werden die Daten transformiert und die Ausgabe formatiert. Das Handling der Dokumente und die Steuerung des Parsers sind in ASP realisiert.

Literaturverzeichnis

- [Dev02] Web Developer's. *XSLT, XPath and XSL Formatting Objects*. Web Developer's, (<http://www.wdvl.com/Authoring/Languages/XSL/>), 2002.
- [GP99] Charles F. Goldfarb and Paul Prescod. *XML Handbuch*. Prentice Hall, 1999.
- [Kay01] Michael Kay. *XSLT Programmer's Reference*. Wrox Press, 2001.
- [Kol02] Christian Koller. *Dateiattribute unter ASP auslesen*. Aspheute, (<http://www.aspheute.com/artikel/20000523.htm>), 2002.
- [Kra99] Jörg Krause. *Active Server Pages*. Addison Wesley, 1999.
- [Lam98] Leslie Lamport. *Latex: a document preparation system*. Addison Wesley, 1998.
- [Lou94] Kenneth C. Loudon. *Programmiersprachen, Grundlagen, Konzepte, Entwurf*. Thomson Publ., 1994.
- [Mar00] Benoit Marchal. *XML By Example*. QUE), 2000.
- [Mic02] Microsoft. *XML Web Services*. Microsoft, (<http://msdn.microsoft.com/library/>), 2002.
- [Pag02] Cover Pages. *SGML and XML as (Meta-) Markup Languages*. W3, (<http://xml.coverpages.org/sgml.html>), 2002.
- [Rot01] Gunther Rothfuss. *Content Management mit XML*. Springer, 2001.
- [W3C02] W3C. *The Extensible Stylesheet Language Reference*. W3C, (<http://www.w3.org/Style/XSL/>), 2002.
- [Wel99] Tobias Weltner. *Active Server Pages lernen und beherrschen*. Microsoft Press, 1999.
- [XML02a] XML101. *XML Examples*. XML101, (<http://www.xml101.com/>), 2002.
- [Xml02b] Xmlpitstop. *XML Examples*. Xmlpitstop, (<http://www.xmlpitstop.com/default.asp?DataType=XMLEXAMPLES>), 2002.
- [Zan01] Matt Zandstra. *Jetzt lerne ich PHP 4*. Markt und Technik, 2001.

Anhang A

Xsl Stylesheets

A.1 Stylesheet reduzieren.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<!--
----- Stylesheet reduzieren -----
Autor:    Mueller Roman, geschrieben im Rahmen einer Seminararbeit
Funktion: Die Elemente eines beliebigen XML Dokumentes werden mit
ihrer
          Hierarchieebene als Zeichenkette in den Output geschrieben.
          Die Reihenfolge bleibt erhalten und doppelte Elemente werden
          eliminiert.
Input:    Valides XML Dokument
Output:   Resultatzeichenkette des Stylesheets.
          Soll der Output ein valides XML Dokument sein, muss das
          <xsl:output> attribut "omit-xml-declaration" auf "no" gesetzt
          werden und der Templateaufruf "sort" im Root Template mit einem
          Elementtag umschlossen werden.
-----
-->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" encoding="utf-8" omit-xml-declaration="yes" />
<!--
----- Template root -----
Funktion: Bei Durchlauf des XML Dokumentes werden die Elemente mit
ihrer Ebene, der Reihenfolge nach, in den String "elements" aufgenommen.
          Das Template "sort" wird mit der Variable "elements" aufgerufen.
-----
-->
<xsl:template match="/">
  <xsl:variable name="elements">
    <xsl:for-each select="child::*">1,<xsl:value-of select="concat(name(),'')"/>
    <xsl:for-each select="child::*">2,<xsl:value-of select="concat(name(),'')"/>
    <xsl:for-each select="child::*">3,<xsl:value-of select="concat(name(),'')"/>
    <xsl:for-each select="child::*">4,<xsl:value-of select="concat(name(),'')"/>
    <xsl:for-each select="child::*">5,<xsl:value-of select="concat(name(),'')"/>
    <xsl:for-each select="child::*">6,<xsl:value-of select="concat(name(),'')"/>
  </xsl:for-each></xsl:for-each></xsl:for-each></xsl:for-each></xsl:for-each>
  </xsl:variable>
  <xsl:call-template name="sort">

```

```

        <xsl:with-param name="elements" select="$elements"/>
    </xsl:call-template>

</xsl:template>

<!--
----- Template sort -----
Funktion: Nimmt das erste Element einer Zeichenkette und vergleicht, ob
im Rumpf der Zeichenkette das Element nochmals vorhanden ist.
Ist dies der Fall, so wird das erste Element mit der gefundenen
Sequenz ersetzt. Eine Sequenz besteht aus einem Element und
seinen nachfolgenden Subelementen.
Die Funktion wird rekursiv aufgerufen bis die Zeichenkette
durchgearbeitet ist.
-----
-->

<xsl:template name="sort">

    <xsl:param name="elements"/>
    <xsl:param name="first"/>
    <xsl:variable name="aktiv_u" select="substring-before($elements, ';' )"/>
    <xsl:variable name="aktiv" select="concat($aktiv_u, ';' )"/>
    <xsl:variable name="aktiv_n" select="substring-before($aktiv, ';' )"/>
    <xsl:variable name="aktiv_e" select="substring-after($aktiv, ';' )"/>
    <xsl:variable name="rest" select="substring-after($elements, ';' )"/>

    <xsl:choose>
        <xsl:when test="$rest = ''">
            <xsl:call-template name="output">
                <xsl:with-param name="out_elements" select="concat($first,$aktiv)"/>
            </xsl:call-template>
        </xsl:when>

        <xsl:when test="contains($rest,$aktiv)">
            <xsl:variable name="auswahl_f" select="substring-before($rest,$aktiv)"/>
            <xsl:variable name="auswahl_f_u" select="concat($auswahl_f,$aktiv)"/>
            <xsl:variable name="auswahl_r" select="substring-after($rest,$auswahl_f_u)"/>

            <xsl:variable name="auswahl_r_schnittpunkt_n" select="number($aktiv_n)-1"/>
            <xsl:variable name="schnittpunkt" select="string($auswahl_r_schnittpunkt_n)"/>

            <xsl:variable name="auswahl_r_f" select="substring-before($auswahl_r,$schnittpunkt)"/>
            <xsl:variable name="auswahl_r_a" select="substring-after($auswahl_r,$auswahl_r_f)"/>

            <xsl:choose>
                <xsl:when test="$auswahl_r_a = ''">
                    <xsl:call-template name="sort">
                        <xsl:with-param name="elements" select="concat($aktiv,$auswahl_r,$auswahl_f)"/>
                        <xsl:with-param name="first" select="$first"/>
                    </xsl:call-template>
                </xsl:when>

                <xsl:otherwise>
                    <xsl:call-template name="sort">
                        <xsl:with-param name="elements" select="concat($aktiv,$auswahl_r_f,$auswahl_f,$auswahl_r_a)"/>
                        <xsl:with-param name="first" select="$first"/>
                    </xsl:call-template>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:when>

        <xsl:otherwise>
            <xsl:call-template name="sort">
                <xsl:with-param name="elements" select="$rest"/>
                <xsl:with-param name="first" select="concat($first,$aktiv)"/>
            </xsl:call-template>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>

```

```

-->
----- Template reduzieren -----
Funktion: Schreibt den Parameter "out_elements" als Zeichenkette
        in den Output.
-----
-->
<xsl:template name="output">
  <xsl:param name="out_elements"/>
  <xsl:variable name="out" select="$out_elements"/>

  <xsl:value-of select="$out"/>
</xsl:template>
</xsl:stylesheet>

```

A.2 Stylesheet filtergen.xsl

```

<?xml version="1.0" encoding="utf-8"?>
<!--
----- Stylesheet Filtergen -----
Autor:   M\{u}ller Roman, geschrieben im Rahmen einer Seminararbeit

Funktion: Die Zeichenkette im Element string wird verarbeitet. Anhand
        der Daten der Zeichenkette wird ein XSL Stylesheet generiert.

Input:   Valides XML Dokument mit einem Element string

Output:  XSL Stylesheet das als Filter auf ein XML angewendet werden kann
-----
-->

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:out="test.xsl">
  <xsl:output method="xml" indent="yes" encoding="utf-8"/>
  <xsl:namespace-alias stylesheet-prefix="out" result-prefix="xsl"/>

  <xsl:template match="/">
    <xsl:variable name="string" select="string"/>
    <xsl:variable name="laufnummer" select="number(1)"/>

    <out:stylesheet version="1.0" >
      <out:output method="xml" indent="yes"/>

      <xsl:comment>*****</xsl:comment>
      <xsl:comment> Stylesheet generiert mit filtergen.xsl</xsl:comment>
      <xsl:comment>*****</xsl:comment>

      <out:template match="/">
        <xsl:choose>
          <xsl:when test="$string = ''">
            <xsl:comment>*****</xsl:comment>
            <xsl:comment>Keine Filterwerte</xsl:comment>
            <xsl:comment>*****</xsl:comment>
          </xsl:when>
          <xsl:otherwise>
            <xsl:call-template name="einstieg">
              <xsl:with-param name="einstieg_p" select="$string"/>
            </xsl:call-template>
          </xsl:otherwise>
        </xsl:choose>
      </out:template>
    </out:stylesheet>
  </xsl:template>

```

```

<xsl:template name="einstieg">
  <xsl:param name="einstieg_p"/>

  <xsl:variable name="string" select="$einstieg_p"/>
  <xsl:variable name="nummer" select="substring-before($string,',' )"/>

  <xsl:choose>
    <xsl:when test="contains(substring-after($string,$nummer),$nummer)">
      <xsl:variable name="string_o_n" select="substring-after($string,$nummer)"/>
      <xsl:variable name="string_b_o_n" select="substring-before($string_o_n,$nummer)"/>
      <xsl:variable name="string_b" select="concat($nummer,$string_b_o_n)"/>

      <xsl:call-template name="bearbeiten">
        <xsl:with-param name="bearbeiten_p" select="$string_b"/>
      </xsl:call-template>

      <xsl:call-template name="einstieg">
        <xsl:with-param name="einstieg_p" select="substring-after($string,$string_b)"/>
      </xsl:call-template>
    </xsl:when>

    <xsl:otherwise>
      <xsl:call-template name="bearbeiten">
        <xsl:with-param name="bearbeiten_p" select="$string"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="bearbeiten">
  <xsl:param name="bearbeiten_p"/>

  <xsl:variable name="string" select="$bearbeiten_p"/>
  <xsl:variable name="string_a" select="substring-before($string,',' )"/>
  <xsl:variable name="string_element_a" select="substring-after($string_a,',' )"/>
  <xsl:variable name="string_nummer_a" select="substring-before($string_a,',' )"/>

  <xsl:variable name="string_nummer_n" select="substring-before(substring-after($string,concat($string_a,',' )),',' )"/>
  <xsl:variable name="string_n" select="substring-after($string,concat($string_a,',' )"/>

  <xsl:choose>
    <xsl:when test="$string_nummer_n = ''">
      <xsl:for-each select="child::node() [name()='{$string_element_a}']">
        <out:copy>
          <out:copy-of select="@*"/>
          <out:copy-of select="text()"/>
        </out:copy>
      </xsl:for-each>
    </xsl:when>

    <xsl:when test="number($string_nummer_n) > number($string_nummer_a)">
      <xsl:for-each select="child::node() [name()='{$string_element_a}']">
        <out:copy>
          <out:copy-of select="@*"/>
          <xsl:call-template name="einstieg">
            <xsl:with-param name="einstieg_p" select="$string_n"/>
          </xsl:call-template>
        </out:copy>
      </xsl:for-each>
    </xsl:when>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>

```

Anhang B

ASP Seiten

B.1 ASP reduzieren.asp

```
<html> <head> <title>reduzieren.xsl</title> </head>

<%
'Variablen zur Identifizierung der Session und des
Benutzers DIM ID, Session_ID

ID = Request("ID") Session_ID = Request("Session_ID")

'Der Pfad des zu bearbeitenden XML Dokumentes wird
eingelesen Dim XmlFile XmlFile = Request("XmlFile")

'Die Dom Objekte xmldoc, und xsl doc werden generiert Dim
xmldoc, xsl doc, newdoc

Set xmldoc = Server.CreateObject("MSXML2.DOMDocument") Set
xsl doc = Server.CreateObject("MSXML2.DOMDocument")

xmldoc.Load XmlFile

if xmldoc.parseerror.errorcode <> 0 then
    Response.Write "Error loading XML Document :" & "<BR>"
    Response.Write "-----" & "<BR>"
    Response.Write "Error Code : " & xmldoc.parseerror.errorcode & "<BR>"
    Response.Write "Reason : " & xmldoc.parseerror.reason & "<BR>"
    Response.End
End If

xsl doc.Load Server.MapPath("xsl/reduzieren.xsl")

if xsl doc.parseerror.errorcode <> 0 then
    Response.Write "Error loading XSL Document :" & "<BR>"
    Response.Write "-----" & "<BR>"
    Response.Write "Error Code : " & xsl doc.parseerror.errorcode & "<BR>"
    Response.Write "Reason : " & xsl doc.parseerror.reason & "<BR>"
    Response.End
End If%>

<%
Dim strTree, treeLength

strTree = (xmldoc.transformNode (xsl doc)) treeLength =
InStrRev(strTree, "") 'Die nachfolgende 0 kann durch Die
relevante Startpunkt 'einer Zeichenkette ersetzt werden.
strTree = Right(strTree, treeLength - 0) treeLength =
InStrRev(strTree, "") 'Die nachfolgende 0 kann durch Die
relevante Endpunkt 'einer Zeichenkette ersetzt werden.
strTree = Left(strTree, treeLength - 0)
treeLength = InStrRev(strTree, "")%>
```

```

<body bgcolor="#999999"> <p><font face="Arial, Helvetica,
sans-serif" size="+3"> <b><font color="#CCCCCC">Auswahl der
Knoten</font></b></font></p> <hr noshade> <p><font
face="Arial, Helvetica, sans-serif" size="2"
color="#FFF0CC">Die Knoten
    die sie nicht markieren, werden weggefiltert, ebenso ihre Kindelemente.</font></p>

<table> <tr>
    <form method="post" action="formread.asp" id=form1 name=form1>
</tr>

<%
Dim strTemp, varLength, varTiefe, test1

varLength = treeLength

' Die Werte ID, Session_ID, und strTree werden hidden \"{u}begeben.%>
    <input type=hidden value=<%= strTree%> name=strTree>
    <input type=hidden value=<%= ID%> name=ID>
    <input type=hidden value=<%= Session_ID%> name=Session_ID>

<%
' Von jedem Element der Zeichenkette wird die Tiefe und der
Namen gelesen ' und eine Checkbox erstellt.
While varLength > 0

    strTemp = Left(strTree, InStr(strTree, ";")-1)
    strTemp = Replace(strTemp, ",", ";")
    strTree = Right(strTree, varLength - (InStrRev(strTemp, ";") + 1))
    varLength = InStrRev(strTree, ";")
    varTiefe = Left(strTemp,1)%>

    <tr>
    <td width="25%"><%=></td>
    <%
    While varTiefe > 1%>

    <td width="15%">&nbsp;</td>
    <%
        varTiefe = varTiefe - 1
    Wend%>

    <td width="15%"><input type="checkbox" checked name="Fields" value=<%= strTemp & " "%>
    <%If Left(strTemp,1) = 1 Then%>OnClick="document.forms.form1.Result[0].checked=true;"<%End If%>>
    <%= (Right(strTemp,InStrRev(strTemp, ";")- 2))%></td>
    </tr>

<%
Wend
%>
</table> <br><br><br> <center><input center="left"
type="submit" value="Uebergeben" name="submitButton">
</body> </html>

```

B.2 ASP formread.asp

```

<%
DIM Session_ID, ID Session_ID = Request("Session_ID") ID =
Request("ID")

Dim Selection, Tree, Element, Ebene, TElement, TEbene,
Gefiltert, Result

'Zeichenkettenkorrekturen Tree = Request("strTree") Tree =
Trim(Tree & "1,ende;") Selection = Request("Fields")
Selection = Replace(Selection, ",", "k") Selection =
Replace(Selection, ";", " ") Selection = Replace(Selection,
"k", ";") Selection = Trim(Selection & ";") Selection =
Trim(Selection & "1,ende;")

```

```

'Algorithmus zur Elimination der ung\{u}ltigen Elemente
Schreibberechtigung = Cint("1")
While ((Tree > "") or (Selection > ""))
  IF ((Tree > "") or (Selection > "")) then

    IF ((Selection = "1,ende;") or (Tree = "1,ende;")) then
      End if

    Ebene = (Left(Selection, 1))
    Element = Left(Selection, InStr(Selection, ";")-1)
    Element = Trim(Element & ";")
    TEbene = Left(Tree, 1)
    TElement = Left(Tree, InStr(Tree, ";")-1)
    TElement = Trim(TElement & ";")

    IF Element = TElement then

      If Cint(TEbene) <= Schreibberechtigung then
        Result = Result & Element
        Schreibberechtigung = Cint(Ebene) + 1

        TreeLength = InStrRev(Selection, "")
        ElementLength = InStrRev(Element, "")
        Selection = Trim(Right(Selection, (TreeLength - ElementLength)))
        TreeLength = InStrRev(Tree, "")
        TElementLength = InStrRev(TElement, "")
        Tree = Trim(Right(Tree, (TreeLength - TElementLength)))

      ELSE ' Wenn keine
        Gefiltert = Gefiltert & Element

        TreeLength = InStrRev(Selection, "")
        ElementLength = InStrRev(Element, "")
        Selection = Trim(Right(Selection, (TreeLength - ElementLength)))
        TreeLength = InStrRev(Tree, "")
        TElementLength = InStrRev(TElement, "")
        Tree = Trim(Right(Tree, (TreeLength - TElementLength)))
      END IF

    ELSE ' Wenn kein Treffer

      If Cint(TEbene) < Schreibberechtigung then
        Schreibberechtigung = Cint(TEbene)

        TreeLength = InStrRev(Tree, "")
        TElementLength = InStrRev(TElement, "")
        Tree = Trim(Right(Tree, (TreeLength - TElementLength)))

      ELSE ' Wenn keine

        TreeLength = InStrRev(Tree, "")
        TElementLength = InStrRev(TElement, "")
        Tree = Trim(Right(Tree, (TreeLength - TElementLength)))
      END IF

    END IF

  End IF
Wend

TreeLength = InStrRev(result, "")
ElementLength = InStrRev("1,ende;", "")
result = Trim(Left(result, (TreeLength - ElementLength)))

%>
<HTML> <HEAD> </HEAD>

```

